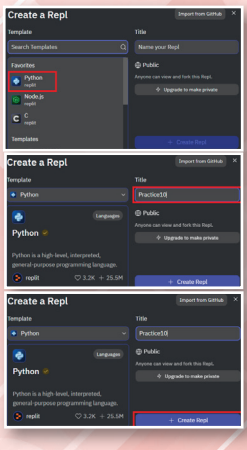
тольяттинский
государственный
университет

О.М. Гущина, А.В. Герасимов

СРЕДСТВА ПРОГРАММНОЙ РАЗРАБОТКИ НА ЯЗЫКЕ PYTHON: ВЕБ-ПРИЛОЖЕНИЯ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И МАШИННОЕ ОБУЧЕНИЕ

Учебно-методическое пособие



Тольятти
Издательство ТГУ
2025

Министерство науки и высшего образования
Российской Федерации
Тольяттинский государственный университет

О.М. Гущина, А.В. Герасимов

СРЕДСТВА ПРОГРАММНОЙ РАЗРАБОТКИ НА ЯЗЫКЕ PYTHON: ВЕБ-ПРИЛОЖЕНИЯ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И МАШИННОЕ ОБУЧЕНИЕ

Учебно-методическое пособие

Тольятти
Издательство ТГУ
2025

УДК 004.43(075.8)+004.8(075.8)
ББК 32.973.2я73+16.6я73
Г 981

Рецензенты:
д-р пед. наук, доцент, заведующий кафедрой «Прикладная информатика» Тольяттинской академии управления *Н.Б. Стрекалова*;
д-р техн. наук, доцент, профессор кафедры прикладной математики и информатики Тольяттинского государственного университета *С.В. Мкртычев*.

Г 981 Гущина, О.М. Средства программной разработки на языке Python: веб-приложения, искусственный интеллект и машинное обучение : учебно-методическое пособие / О.М. Гущина, А.В. Герасимов. – Тольятти : Издательство ТГУ, 2025. – 363 с. – ISBN 978-5-8259-1754-2.

Учебно-методическое пособие содержит теоретические представления и практические рекомендации по использованию языка программирования Python в задачах программной разработки и машинного обучения.

Предназначено для использования в работе со студентами всех направлений подготовки, всех специальностей, всех форм обучения (в том числе с использованием ДОТ) и может быть полезно студентам, профессорско-преподавательскому составу высших учебных заведений, а также любому желающему получить знания по языку программирования Python.

УДК 004.43(075.8)+004.8(075.8)
ББК 32.973.2я73+16.6я73

Рекомендовано к изданию научно-методическим советом Тольяттинского государственного университета.

ISBN 978-5-8259-1754-2

ВВЕДЕНИЕ

Учебно-методическое пособие «Средства программной разработки на языке Python: веб-приложения, искусственный интеллект и машинное обучение» предназначено для студентов и всех тех, кто хочет получить базу теоретических знаний и практических навыков в области разработки программных продуктов на языке программирования Python.

В данной работе представлено описание основ анализа данных, которые включают описание базовых структур языка Python, основ объектно ориентированного программирования и веб-разработки на нем. Дается также представление об искусственном интеллекте и машинном обучении с примерами реализации на языке Python. Каждый модуль завершается контрольными вопросами и предлагаемыми практическими заданиями с методическими рекомендациями для их выполнения.

Цель пособия заключается в предоставлении теоретических и практических рекомендаций по использованию языка программирования Python в задачах программной разработки и машинного обучения.

Задачи

1. Дать общее представление о языке программирования Python.
2. Рассмотреть основы разработки программного продукта.
3. Показать практику применения языка программирования Python для разработки веб-приложений и решения задач машинного обучения.
4. Проверить знания материала с помощью контрольных вопросов и выполнения практических заданий.

Первый модуль пособия дает общее представление о языке программирования Python и принципах объектно ориентированного подхода. Второй модуль рассматривает основы веб-разработки от создания программного продукта до его публикации. Третий модуль дает общее представление об искусственном интеллекте и машинном обучении с примерами реализации решений на языке программирования Python.

В результате изучения учебно-методического пособия обучающийся должен

✓ *знать*:

– базовые структуры языка Python;
– принципы работы с файлами, модулями и функциями;
– основные понятия ООП;
– основы веб-разработки;
– основные задачи систем искусственного интеллекта;
– примеры работы с системами глубокого обучения;

✓ *уметь*:

– создавать программные продукты на языке Python;
– осуществлять настройку веб-приложения;
– применять простейшие алгоритмы машинного обучения;

✓ *владеть навыками*:

– разработки от простых приложений до веб-приложений;
– применения простейших алгоритмов машинного обучения в задачах систем искусственного интеллекта.

Модуль 1. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON

Тема 1. Базовые структуры языка Python и их обработка

Общее представление о языке Python

Python – интерпретируемый, высокоуровневый, динамически типизированный язык программирования, который был разработан в конце 1980-х годов Гвидо ван Россумом. Python был создан как язык для обучения программированию, но с тех пор он стал одним из самых популярных языков программирования в мире.

Python может быть использован для широкого спектра задач, включая следующие.

Автоматизация задач: Python – мощный инструмент для написания скриптов и автоматизации задач. Это может быть особенно полезно для работ с большими объемами данных, обработки изображений, создания отчетов и других повседневных задач.

Разработка веб-приложений: Python часто используется для создания веб-сайтов и веб-приложений. Фреймворки, такие как Django и Flask, делают разработку веб-приложений более простой и быстрой.

Научные и математические вычисления: Python имеет множество библиотек для научных и математических вычислений, таких как NumPy, SciPy и Pandas. Они позволяют выполнять сложные вычисления, работу с массивами данных, создание графиков и многое другое.

Разработка игр: Python может использоваться для создания игр. Библиотеки, такие как Pygame и PyOpenGL, позволяют создавать 2D и 3D игры.

Машинное обучение и искусственный интеллект: Python является одним из наиболее популярных языков программирования для машинного обучения и искусственного интеллекта. Библиотеки, такие как TensorFlow, Keras и PyTorch, позволяют создавать сложные модели машинного обучения и производить анализ данных.

Разработка мобильных приложений: Python может использоваться для создания мобильных приложений. Фреймворки, такие как Kivy и BeeWare, позволяют создавать кроссплатформенные приложения для Android и iOS.

Python является очень гибким и универсальным языком программирования, который может использоваться для решения широкого спектра задач. Он обладает множеством преимуществ, таких как простота, легкость в изучении и использовании, а также широкий выбор библиотек и фреймворков.

Достоинства Python

• Простой и легкий в изучении и использовании. Имеет широкий спектр библиотек и фреймворков, облегчающих разработку приложений и повышающих производительность.

• Обладает высокой скоростью разработки благодаря своей простоте и удобству синтаксиса.

• Интерпретируемый язык, и это означает, что нет необходимости компилировать его программы, что существенно ускоряет процесс разработки и отладки.

• Имеет мощную систему управления памятью, что помогает избежать проблем с утечками памяти.

Недостатки Python

• Может быть более медленный, чем некоторые другие языки программирования, такие как C++ или Java.

• Динамически типизированный язык, что может привести к некоторым проблемам с типами данных.

• Интерпретируемый язык, что может привести к снижению производительности приложений, особенно при работе с большими объемами данных.

Изучение языка программирования Python может помочь в развитии навыков логического мышления, решения проблем и алгоритмического мышления. Эти навыки могут быть полезными для решения разнообразных задач как в программировании, так и в повседневной жизни.

Python имеет множество преимуществ, которые делают его удобным для разработки веб-приложений.

Простота языка. Python имеет чистый и понятный синтаксис, который облегчает понимание кода и ускоряет процесс разработки. Python не требует знания множества дополнительных языков, таких как HTML, CSS, JavaScript, и др., для разработки веб-приложений. Вместо этого Python может использоваться вместе с веб-фреймворками, такими как Django или Flask, которые облегчают создание веб-приложений.

Мощность языка. Python имеет множество библиотек, которые облегчают разработку веб-приложений. Библиотеки, такие как Requests, BeautifulSoup, и Pandas, предоставляют разработчикам множество инструментов для работы с данными, сетевыми протоколами и многими другими аспектами веб-разработки. Кроме того, Python имеет мощные инструменты для обработки данных, такие как NumPy, SciPy, и Matplotlib, которые могут быть полезны при создании веб-приложений, связанных с научными исследованиями и анализом данных.

Кроссплатформенность. Python может быть запущен на различных операционных системах, включая Windows, macOS, и Linux. Это делает его удобным для разработки веб-приложений, которые могут быть запущены на различных платформах.

Активное сообщество. Python имеет большое и активное сообщество разработчиков, которые создают новые библиотеки, фреймворки и инструменты для улучшения процесса разработки. Сообщество также предоставляет множество ресурсов для обучения и поддержки, таких как документация, форумы и онлайн-курсы.

Высокая скорость разработки. Python позволяет создавать веб-приложения быстрее, чем многие другие языки программирования. Python имеет понятный синтаксис и множество инструментов, которые облегчают создание веб-приложений. Кроме того, Python имеет множество библиотек, которые облегчают работу с базами данных, сетевыми протоколами и многими другими аспектами веб-разработки.

Python является удобным языком программирования для разработки веб-приложений. Он имеет простой и понятный синтаксис, мощные инструменты и библиотеки, кроссплатформенность, активное сообщество и высокую скорость разработки. Разработка

веб-приложений на Python с использованием Django может быть полезна для создания различных типов приложений, таких как интернет-магазины, социальные сети, корпоративные порталы и т. д.

Основные понятия языка Python

Переменные в Python – именованные области памяти, которые используются для хранения значений. В Python переменные не требуется объявлять явно – они создаются автоматически при первом присваивании значения.

Пример на рис. 1 показывает описание процесса создания переменной и присваивание ей значения.

```
x = 5 # создание переменной x и присваивание ей значения 5
y = "Hello world" # создание переменной y и присваивание ей значения "Hello world"
z = True # создание переменной z и присваивание ей значения True
```

Рис. 1. Описание процесса создания переменной

Для того чтобы вывести значение переменной на экран, можно использовать функцию print(), описание которой представлено на рис. 2.

```
x = 5
print(x) # выводит значение переменной x на экран
```

Рис. 2. Описание работы функции print()

Переменные в Python могут изменять свои значения в процессе выполнения программы. С переменными в Python можно выполнять различные математические операции, а также операции сравнения и логические операции.

Пример на рис. 3 показывает описание сложения двух переменных и вывод результата.

```
x = 5
y = 3
z = x + y
print("Сумма чисел", x, "и", y, "равна", z) # вывод на экран нескольких значений с помощью функции print()
```

Рис. 3. Сравнение двух переменных

Пример на рис. 4 позволяет продемонстрировать применение логического «и».

```
x = True
y = False
z = x and y # логическое "и" между переменными x и y, результат
присваивается переменной z
print(z) # выводит значение переменной z на экран
```

Рис. 4. Применение логического «и»

При использовании операций с переменными в Python результат может зависеть от типа данных переменных. Например, при делении двух целых чисел результат будет также целым числом, а при делении числа с плавающей точкой на целое число результат будет иметь тип float. Также важно следить за порядком выполнения операций и использовать скобки при необходимости.

В Python для ввода и вывода данных используются функции input() и print() соответственно. Функция input() позволяет получить данные от пользователя, а функция print() используется для вывода данных на экран.

Пример на рис. 5 показывает правильную запись ввода и вывода данных.

```
name = input("Введите ваше имя: ") # получение данных от пользователя с
помощью функции input()
print("Привет,", name) # вывод приветствия на экран с помощью функции print()
```

Рис. 5. Правильная запись ввода/вывода данных

Пример на рис. 6 показывает запись вывода нескольких переменных, используемых в программе.

```
x = 5
y = 3
z = x + y
print("Сумма чисел", x, "и", y, "равна", z) # вывод на экран нескольких значений с
помощью функции print()
```

Рис. 6. Запись вывода нескольких переменных

При использовании функции input() все введенные данные будут иметь тип str. Поэтому необходимо преобразовывать данные к нужному типу (например, int или float) перед их использованием

в вычислениях. Также важно обращать внимание на правильность ввода данных пользователем и обрабатывать возможные ошибки при вводе.

Строки в Python – последовательность символов, заключенных в кавычки. Они могут содержать любые символы, включая буквы, цифры и специальные символы. Строки в Python – упорядоченные последовательности символов, где каждый символ имеет свой индекс. Индексы начинаются с 0 для первого символа в строке и увеличиваются на 1 для каждого следующего символа.

В примере на рис. 7 показывается, как происходит создание и вывод строки.

```
s = "Hello, world!« # создание строки
print(s) # вывод строки на экран
```

Рис. 7. Создание и вывод строки

Строки в Python могут быть изменяемыми и неизменяемыми. Неизменяемые строки не могут быть изменены после создания, а изменяемые строки могут быть изменены.

Пример на рис. 8 показывает описание изменяемой и неизменяемой строк:

– строка «Hello, world!» сохраняется в переменной s1;
– попытка изменить первый символ строки приводит к ошибке, так как строка неизменяемая;
– строка «Hello, world!» сохраняется в переменной s2;
– функция replace() используется для замены символов в строке;
– измененная строка сохраняется в той же переменной s2;
– функция print() используется для вывода измененной строки на экран.

```
# неизменяемая строка
s1 = "Hello, world!"
s1[0] = "h" # ошибка строка не может быть изменена
# изменяемая строка
s2 = "Hello, world!"
s2 = s2.replace("o", "0") # замена символов в строке
print(s2) # вывод измененной строки на экран
```

Рис. 8. Описание изменяемой/неизменяемой строки

При работе со строками в Python необходимо учитывать их особенности, такие как неизменяемость строк, использование кавычек для определения начала и конца строки, а также различные методы для работы со строками в Python. Кроме того, форматирование строк и использование регулярных выражений могут значительно упростить и ускорить обработку строк в Python.

Код программы на языке Python (рис. 9) показывает обращение к символам строки. Этот код создает переменную string и присваивает ей значение «Hello, World!». Он выводит отдельные символы этой строки на экран, используя индексы символов в квадратных скобках.

```
string = "Hello, World!"
print(string[0])   # Output: H
print(string[1])   # Output: e
print(string[2])   # Output: l
print(string[-1])  # Output: !
```

Рис. 9. Обращение к символам строки

В этом примере мы выводим первый символ, используя индекс 0, второй символ – используя индекс 1, и т. д. Также используем отрицательный индекс –1 для обращения к последнему символу строки. В результате выполнения этой программы на экране появятся символы H, e, l и ! – каждый на отдельной строке – соответственно первый, второй, третий и последний символы строки «Hello, World!».

Код, представленный на рис. 10, демонстрирует форматирование строк: создает переменные name и age и присваивает им значения «John» и 30 соответственно. Он выводит отформатированную строку, используя метод format().

```
name = "John"
age = 30
print("Меня зовут {} и мне {} лет".format(name, age))
```

Рис. 10. Форматирование строк

Метод format() используется для вставки значений переменных внутрь строки. В примере выше используются фигурные скобки {} внутри строки для обозначения мест, где должны быть вставлены значения переменных. Передаем значения переменных name

и age в метод format() в качестве аргументов. При выполнении метод format() заменяет фигурные скобки соответствующими значениями переменных и формирует новую строку в результате. В данном случае метод format() заменит {} на name и age, чтобы получилась строка «My name is John and I'm 30 years old.».

На рис. 11 показан пример, описывающий методы строк. Методы строк позволяют выполнять различные операции с символьными данными, такие как изменение регистра символов, замена подстрок на другие подстроки и многое другое.

```
string = "Hello, world!"
print(string.upper())  # выводит "HELLO, WORLD!"
print(string.lower())  # выводит "hello, world!"
print(string.replace("Hello", "Hi"))  # выводит "Hi, world!"
```

Рис. 11. Описание методов строк

Код программы на языке Python создает переменную string и присваивает ей значение «Hello, World!». Он вызывает несколько методов на этой строке и выводит результат их работы на экран:

– метод upper() используется для преобразования всех символов строки в верхний регистр. В данном случае мы вызываем метод upper() на переменной string и выводим результат на экран с помощью функции print(). В результате выполнения этой операции на экране появится строка «HELLO, WORLD!»;

– метод lower() используется для преобразования всех символов строки в нижний регистр. В данном случае мы вызываем метод lower() на переменной string и выводим результат на экран с помощью функции print(). В результате выполнения этой операции на экране появится строка «hello, world!»;

– метод replace() используется для замены одной подстроки на другую в строке. В данном случае мы вызываем метод replace() на переменной string и передаем два аргумента – подстроку, которую нужно заменить, и подстроку, на которую нужно заменить. В результате выполнения этой операции на экране появится строка «Jello, World!» – символ H в слове «Hello» был заменен на символ J.

Циклы в Python

Циклы в Python используются для повторения определенного блока кода несколько раз. В Python есть два основных типа циклов: for и while.

Цикл for используется для перебора элементов в последовательности, такой как строка, список или кортеж. Цикл for выполняет блок кода для каждого элемента в последовательности. Рассмотрим пример, представленный на рис. 12:

– список [1, 2, 3, 4, 5] сохраняется в переменной numbers;
– цикл for используется для перебора элементов в списке;
– переменная number принимает значение каждого элемента списка в каждой итерации цикла;
– функция print() используется для вывода элемента на экран.

```
# цикл for для перебора элементов в списке
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print(number) # вывод элемента на экран
```

Рис. 12. Описание работы for

Цикл while используется для повторения блока кода, пока определенное условие истинно. Цикл while проверяет условие перед каждой итерацией цикла. Пример на рис. 13 демонстрирует работу цикла while:

– переменная i инициализируется значением 1;
– цикл while используется для вывода чисел от 1 до 5;
– условие i <= 5 проверяется перед каждой итерацией цикла;
– функция print() используется для вывода числа на экран;
– значение переменной i увеличивается на 1 в каждой итерации цикла.

```
# цикл while для вывода чисел от 1 до 5
i = 1
while i <= 5:
    print(i) # вывод числа на экран
    i += 1 # увеличение значения переменной i на 1 в каждой итерации
```

Рис. 13. Описание работы while

При работе с циклами в Python необходимо учитывать их особенности, такие как использование range() для создания последовательностей, использование break, continue и pass для управления циклами, а также принципы работы циклов и их эффективность. При правильном использовании циклов в Python можно ускорить выполнение программ и реализовать более сложные задачи.

В примере на рис. 14 мы используем цикл for для перебора элементов списка fruits, содержащего три элемента – «apple», «banana» и «cherry». Переменная fruit – это временная переменная, которая принимает значение каждого элемента списка fruits на каждой итерации цикла. Используем функцию print() для вывода значения этой переменной на экран. В результате выполнения этой программы на экране появятся три строки, содержащие значения элементов списка fruits – «apple», «banana» и «cherry».

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Рис. 14. Описание цикла for для перебора элементов

В примере на рис. 15 мы используем цикл while для вывода чисел от 0 до 4. Переменная count инициализируется значением 0, затем на каждой итерации цикла выводим значение переменной и увеличиваем ее на 1. Цикл while используется для повторения одного и того же действия, пока заданное условие истинно. В данном случае используется цикл while, чтобы повторять вывод значения переменной i на экран до тех пор, пока i меньше 5. На каждой итерации цикла значение переменной i увеличивается на 1 с помощью оператора +=. Это позволяет нам постепенно увеличивать значение переменной i на каждой итерации, пока оно не достигнет значения 5. В результате выполнения этой программы на экране появятся пять строк, содержащих значения переменной i от 0 до 4.

```
count = 0
while count < 5:
    print(count)
    count += 1
```

Рис. 15. Описание цикла while

В примере на рис. 16 используем цикл for для перебора элементов списка fruits, содержащего три элемента – «apple», «banana» и «cherry». Здесь используется цикл for для перебора всех элементов списка и вывода их на экран. Однако если встречается элемент «banana», цикл прерывается.

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    if fruit == "banana":
        break
    print(fruit)
```

Рис. 16. Описание цикла for для перебора элементов

Ключевое слово break используется для прерывания цикла, в котором оно находится. В данном случае мы используем оператор if для проверки, является ли текущий элемент списка fruits равным «banana». Если это так, используем break, чтобы прервать выполнение цикла. Если элемент не равен «banana», используем функцию print() для вывода значения этого элемента на экран. Прерывание цикла может быть полезным в различных ситуациях, например, когда нужно найти определенный элемент в списке или выполнить действие до определенного момента. При этом если условие, которое вызывает прерывание, не выполняется, цикл будет продолжаться до конца.

Условия в Python

Условия в Python используются для выполнения различных действий в зависимости от того, выполняется определенное условие или нет. В Python условия проверяются с помощью операторов сравнения, таких как ==, !=, <, >, <= и >=.

Условные операторы в Python позволяют программисту управлять потоком выполнения программы в зависимости от выполнения определенных условий. В Python есть два основных типа условных операторов: if и else. Также существует оператор elif, который позволяет проверять несколько условий.

Пример на рис. 17 показывает код применения условного оператора if:

– переменная x инициализируется значением 5;
– условие x > 0 проверяется;
– если условие истинно, то выполняется блок кода, который выводит строку на экран.

```
# условный оператор if
x = 5
if x > 0:
   print("x положительное число")
```

Рис. 17. Код применения условного оператора if

Пример на рис. 18 показывает, как применяется оператор if-elif:

– переменная x инициализируется значением -5;
– условие x > 0 проверяется;
– если условие ложно, то выполняется блок кода, который выводит другую строку на экран.

```
# условный оператор if-else
x = -5
if x > 0:
   print("x положительное число")
else:
   print("x отрицательное число")
```

Рис. 18. Пример применения оператора if-elif

Пример на рис. 19 демонстрирует описание оператора if-elif-else:

– переменная x инициализируется значением 0;
– условие x > 0 проверяется;
– если условие ложно, то проверяется следующее условие x == 0;
– если и это условие ложно, то выполняется блок кода, который выводит другую строку на экран.

```
# условный оператор if-elif-else
x = 0
if x > 0:
   print("x положительное число") # вывод строки на экран
elif x == 0:
   print("x равно нулю") # вывод строки на экран
else:
   print("x отрицательное число") # вывод строки на экран
```

Рис. 19. Пример применения оператора if-elif-else

Условные операторы if, else и elif позволяют проверять различные условия и выполнять соответствующие действия.

Пример на рис. 20 показывает код программы на языке Python, который демонстрирует описание оператора if-else.

```
num = int(input("Введите число: "))
if num > 0:
   print("Число", num, "больше нуля")
else:
   print("Число", num, "меньше или равно нулю")
```

Рис. 20. Пример применения оператора if-else

Этот код запрашивает у пользователя ввод числа, затем использует оператор if-else, чтобы проверить, является ли введенное число больше нуля или меньше или равно нулю. В зависимости от результата проверки программа выводит соответствующее сообщение на экран.

Оператор if-else используется для выполнения разных действий в зависимости от того, выполняется заданное условие или нет. В данном случае мы используем оператор if для проверки, является ли введенное число больше нуля. Если это так, используем функцию print() для вывода сообщения, что число больше нуля. В противном случае используем оператор else, чтобы выполнить другое действие – вывод сообщения, что число меньше или равно нулю. В результате выполнения этой программы на экране появится одно из двух сообщений, в зависимости от того, какое число было введено пользователем.

Оператор if-else может использоваться для выполнения различных действий в зависимости от любого условия, которое можно выразить в виде логического выражения. Этот оператор может также использоваться для проверки других условий, таких как равенство, неравенство, сравнение строк и т. д. Кроме того, можно использовать множественные операторы if-else, чтобы проверять несколько условий и выполнять соответствующие действия в зависимости от выполнения каждого условия.

В примере на рис. 21 используется оператор elif для проверки, является ли значение переменной x отрицательным, равным нулю

или положительным. В зависимости от результата проверки программа выводит соответствующее сообщение на экран.

```
x = 5
if x < 0:
    print("x отрицательное число")
elif x == 0:
    print("x равно нулю")
else:
    print("x положительное число")
```

Рис. 21. Описание оператора elif для проверки

Оператор elif является сокращением от else if. Он используется для проверки дополнительных условий, если первое условие (определенное оператором if) не выполняется. В данном случае нами используется оператор if для проверки, является ли значение переменной x отрицательным. Если это так, используем функцию print() для вывода сообщения, что «x отрицательное число». Если значение переменной x не отрицательное, переходим к следующему условию с помощью оператора elif. Здесь мы проверяем, равно ли значение переменной x нулю. Если это так, используем функцию print() для вывода сообщения, что x равно нулю. Если ни одно из первых двух условий не выполняется, используем оператор else, чтобы выполнить другое действие – печать сообщения «x положительное число».

Оператор elif может использоваться для проверки нескольких дополнительных условий и выполнения соответствующих действий в зависимости от их выполнения. Если ни одно из условий не выполняется, можно использовать оператор else, чтобы выполнить действие по умолчанию.

Коллекции в Python

В Python существует несколько типов коллекций, которые позволяют хранить множество объектов в одном месте.

Коллекции (collections) – структуры данных в Python, которые позволяют хранить и обрабатывать данные в виде коллекций. Коллекции в Python включают в себя кортежи, списки, словари и множества.

Кортежи в Python – упорядоченные коллекции объектов, которые могут содержать объекты разных типов. Кортежи являются неизменяемыми, то есть элементы кортежа не могут быть изменены, добавлены или удалены.

Для работы с кортежами используются следующие операторы:
– my_tuple = (1, 2, 3, 4, 5) # создание кортежа;
– print(my_tuple[0]) # обращение к элементу кортежа по индексу;
– print(len(my_tuple)) # определение длины кортежа.

В примере на рис. 22 мы создаем кортеж dimensions, который содержит два элемента – ширину и высоту экрана. Программа выводит на экран информацию о кортеже. Для доступа к элементам кортежа используется оператор индексации [] и соответствующие индексы. Значения элементов выводятся на экран вместе с соответствующими сообщениями.

```
dimensions = (1920, 1080)
print("The screen resolution is", dimensions[0], "x", dimensions[1])
```

Рис. 22. Создание кортежа

Программа выводит информацию о кортеже dimensions, используя оператор индексации для доступа к его элементам. Кортежи являются неизменяемыми объектами, поэтому их элементы не могут быть изменены после создания.

Списки – упорядоченные коллекции объектов, которые могут содержать объекты разных типов. Списки могут быть изменяемыми, то есть элементы списка могут быть изменены, добавлены или удалены.

Для работы со списками используются следующие операторы:
– my_list = [1, 2, 3, 4, 5] # создание списка;
– print(my_list[0]) # обращение к элементу списка по индексу;
– my_list[0] = 6 # изменение элемента списка по индексу;
– my_list.append(6) # добавление элемента в конец списка;
– del my_list[0] # удаление элемента из списка;
– print(len(my_list)) # определение длины списка.

Пример на рис. 23 показывает, как создается список numbers, который содержит 11 элементов. Программа выводит на экран информацию о списке, используя несколько функций и методов:

– функция len() используется для определения количества элементов в списке numbers. Результат выводится на экран;
– для доступа к первому элементу списка используется оператор индексации [] и индекс 0. Значение этого элемента выводится на экран;
– для доступа к последнему элементу списка используется отрицательный индекс -1. Значение этого элемента выводится на экран;
– функция sum() используется для нахождения суммы всех элементов в списке numbers. Результат выводится на экран;
– для нахождения среднего значения всех элементов в списке сумма всех элементов делится на количество элементов. Результат выводится на экран;
– с помощью оператора in проверяется, содержится ли число 5 в списке numbers. Если да, то метод count() используется для подсчета количества вхождений элемента 5 в списке. Результат выводится на экран. Если число 5 не содержится в списке, выводится другое сообщение.

```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
print("The list contains", len(numbers), "elements")
print("The first element is", numbers[0])
print("The last element is", numbers[-1])
print("The sum of the elements is", sum(numbers))
print("The average of the elements is", sum(numbers) / len(numbers))
if 5 in numbers:
    print("The number 5 appears", numbers.count(5), "times in the list")
else:
    print("The number 5 does not appear in the list")
```

Рис. 23. Создание списка

Программа выводит информацию о списке numbers, используя различные функции и методы, и выполняет несколько простых операций на этом списке.

Словари в Python – неупорядоченные коллекции объектов, которые хранятся в виде пар «ключ-значение». Ключи словаря должны быть уникальными, а значения могут быть любого типа данных. Словари могут быть изменяемыми, то есть значения могут быть изменены, добавлены или удалены.

Для работы со словарями используются следующие операторы:

– my_dict = {“name”: “John”, “age”: 30, “city”: “New York”} # создание словаря;
– print(my_dict[«name»])# выводит «John» # доступ к значению по ключу;
– my_dict[«age»] = 31 # изменение значения по ключу;
– my_dict[«gender»] = «male» # добавление новой пары «ключ-значение»;
– del my_dict[«city»] # удаление пары «ключ-значение»;
– print(len(my_dict)) # определение количества элементов в словаре.

На рис. 24 представлен код программы «Словари», который создает словарь person, содержащий информацию о человеке: его имени, возрасте и городе проживания. Программа выполняет несколько операций на словаре и выводит результаты на экран:

– функция print() используется для вывода на экран содержимого словаря person;
– используя оператор индексации [], значение ключа «age» в словаре person изменяется на 31. Новый ключ «job» и его значение «Developer» добавляются в словарь person;
– с помощью цикла for и метода items() проходится по всем ключам и значениям в словаре person и выводит их на экран в формате «ключ-значение».

```
person = {"name": "John", "age": 30, "city": "New York"}
print("Person:", person)

person["age"] = 31
person["job"] = "Developer"
print("Updated person:", person)

for key, value in person.items():
    print(key + ":", value)
```

Рис. 24. Код программы «Словари»

Программа выполняет несколько операций на словаре person, изменяя его содержимое и выводя информацию о ключах и значениях на экран. Словари представляют собой неупорядоченные кол-

лекции пар «ключ-значение», которые могут быть использованы для хранения и обработки структурированных данных.

Множества в Python – неупорядоченные коллекции уникальных элементов. Множества могут быть изменяемыми, то есть элементы могут быть добавлены или удалены.

Для работы с множествами используются следующие операторы:

– my_set = {1, 2, 3, 4, 5} # создание множества;
– my_set.add(6) # добавление элемента в множество;
– my_set.remove(1) # удаление элемента из множества;
– print(3 in my_set) # проверка наличия элемента в множестве;
– print(len(my_set)) # определение количества элементов в множестве.

На рис. 25 показан программный код, который создает два множества – set1 и set2, содержащие несколько элементов. Программа выполняет несколько операций на множествах и выводит результаты на экран:

– функции print() используются для вывода на экран содержимого множеств set1 и set2;
– метод union() используется для нахождения объединения множеств set1 и set2. Результат выводится на экран;
– метод intersection() используется для нахождения пересечения множеств set1 и set2. Результат выводится на экран;
– метод difference() используется дважды – для нахождения разности set1 – set2 и set2 – set1. Результаты выводятся на экран.

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

print("Set 1:", set1)
print("Set 2:", set2)

print("Union:", set1.union(set2))
print("Intersection:", set1.intersection(set2))
print("Difference (set1 - set2):", set1.difference(set2))
print("Difference (set2 - set1):", set2.difference(set1))
```

Рис. 25. Пример описания множеств

Программа выполняет несколько операций на множествах set1 и set2, используя соответствующие методы, и выводит результаты на экран. Множества представляют собой неупорядоченные коллекции уникальных элементов, которые могут быть использованы для выполнения различных операций над элементами.

Тема 2. Файлы, модули и функции в Python

Работа с файлами в Python

Работа с файлами позволяет считывать данные из файлов и записывать данные в файлы. В Python используются различные методы для работы с файлами, которые позволяют открывать файлы, считывать данные из файлов, записывать данные в файлы, закрывать файлы и так далее.

Для открытия файла в Python используется функция open(). Она принимает два аргумента: имя файла и режим открытия файла.

Выделяют следующие режимы открытия файла:

«r» – открыть файл для чтения (по умолчанию);

«w» – открыть файл для записи, содержимое файла удаляется, если файл не существует, то создается новый файл;

«x» – открыть файл для записи, но только если файл не существует;

«a» – открыть файл для записи в конец файла, содержимое файла не удаляется;

«b» – открыть файл в бинарном режиме;

«t» – открыть файл в текстовом режиме (по умолчанию);

«+» – открыть файл для чтения и записи.

Для чтения данных из файла в Python используется метод read(). Этот метод считывает все содержимое файла и возвращает его в виде строки.

Для записи данных в файл в Python используется метод write(). Этот метод записывает данные в файл.

После того как работа с файлом завершена, его необходимо закрыть, чтобы освободить ресурсы. Для закрытия файла в Python используется метод close().

В табл. 1 представлены примеры записи программного кода для основных операций, которые можно производить с файлами.

Таблица 1

Основные операции с файлами

Операция	Пример записи кода программы
Открытие файла	file = open(«test.txt», «r»)
Чтение данных из файла	file = open(«test.txt», «r») content = file.read() print(content) file.close()
Запись данных в файл	file = open(«test.txt», «w») file.write(«Hello, World!») file.close()
Закрытие файла	file = open(«test.txt», «r») content = file.read() file.close()

Код (рис. 26) показывает пример программы для чтения содержимого файла и вывода его на экран.

```
# Открываем файл для чтения
filename = "example.txt"
with open(filename, "r") as file:
    # Считываем содержимое файла в переменную content
    content = file.read()

# Выводим содержимое файла на экран
print("File content:")
print(content)
```

Рис. 26. Чтение и вывод содержимого файла

Эта программа использует функцию open() для открытия файла с именем «example.txt» в режиме чтения («r»). Она использует оператор with, который автоматически закрывает файл после окончания работы с ним и позволяет избежать проблем с утечкой ресурсов. Внутри блока with программа считывает содержимое файла в переменную content с помощью метода read(). Программа выводит содержимое файла на экран с помощью функции print(). Со-

держимое выводится в виде одной строки, так как метод read() сохраняет все содержимое файла в одну строку.

Код (рис. 27) показывает программу для записи строки в файл. Эта программа использует функцию open() для открытия файла с именем «example.txt» в режиме записи («w»). Она использует оператор with, который автоматически закрывает файл после окончания работы с ним и позволяет избежать проблем с утечкой ресурсов. Внутри блока with программа записывает строку «Hello, world!» в файл с помощью метода write(). Программа выводит сообщение об успешной записи в файл с помощью функции print().

```
# Открываем файл для записи
filename = "example.txt"
with open(filename, "w") as file:
    # Записываем строку в файл
    file.write("Hello, world!")

# Выводим сообщение об успешной записи в файл
print("String has been written to the file.")
```

Рис. 27. Запись строки в файл

Программа позволяет быстро и легко записать строку в файл. Если файл с указанным именем уже существует, то его содержимое будет заменено новой строкой. Если файл не существует, то он будет создан.

Код (рис. 28) показывает программу, которая считывает содержимое файла построчно и выводит его на экран.

```
filename = "example.txt« # Открываем файл для чтения
with open(filename, "r") as file: # Считываем содержимое файла построчно и
выводим на экран
    for line in file:
        print(line.strip())
```

Рис. 28. Считывание содержимого файла

Эта программа использует функцию open() для открытия файла с именем «example.txt» в режиме чтения («r»). Она использует оператор with, который автоматически закрывает файл после окончания работы с ним и позволяет избежать проблем с утечкой ресурсов.

Внутри блока with программа использует цикл for для считывания содержимого файла построчно. Каждая строка считывается в переменную line, которая затем выводится на экран с помощью функции print(). Метод strip() используется для удаления символов переноса строки (\n) в конце каждой строки. Программа заканчивает работу, когда все строки файла были прочитаны.

Программа позволяет быстро и легко прочитать и вывести на экран содержимое файла построчно. Это может быть полезно, например, для обработки больших текстовых файлов, где требуется построчное чтение.

Код (рис. 29) показывает программу, которая считывает содержимое файла, изменяет его и записывает обратно в файл.

```
filename = "example.txt"
with open(filename, "r") as file:
    content = file.read()    # Считываем содержимое файла в переменную content
modified_content = content.replace("world", "Python") # Изменяем содержимое файла
with open(filename, "w") as file: # Открываем файл для записи
    file.write(modified_content) # Записываем измененное содержимое в файл
print("File has been modified.") # Выводим сообщение об успешной записи в файл
```

Рис. 29. Действия с содержимым файла

Программа использует функцию open() для открытия файла с именем «example.txt» в режиме чтения («r»). Она использует оператор with, который автоматически закрывает файл после окончания работы с ним и позволяет избежать проблем с утечкой ресурсов. Внутри блока with программа считывает содержимое файла в переменную content с помощью метода read(). Программа изменяет содержимое файла, заменяя все вхождения слова «world» на «Python». Измененное содержимое сохраняется в переменную modified_content. После этого программа открывает файл для записи («w») и использует оператор with для автоматического закрытия файла. Она записывает измененное содержимое в файл с помощью метода write(). Программа выводит сообщение об успешной записи в файл с помощью функции print().

Программа позволяет быстро и легко изменить содержимое файла, сохранить изменения и вывести сообщение об успешной записи.

Функции в Python

Функция – это блок кода, который может принимать входные аргументы и возвращать результаты. Функции в Python – блоки кода, которые могут быть вызваны из других частей программы. Функции в Python позволяют разбивать большие программы на более мелкие и легко управляемые блоки кода. Функции упрощают написание кода, позволяют избежать дублирование кода и повторное использование кода.

Определение функции состоит из имени функции, списка аргументов в скобках и блока кода, который выполняется при вызове функции. В Python функция объявляется с помощью ключевого слова def, за которым следует имя функции, а затем скобки, в которых указываются входные аргументы. Тело функции начинается с отступа и может содержать любой допустимый код Python. Функция может возвращать результаты с помощью ключевого слова return. Она вызывается путем указания ее имени и передачи аргументов в скобках.

Функции могут принимать аргументы. Аргументы могут быть обязательными или необязательными. Обязательные аргументы должны быть переданы функции. Необязательные аргументы имеют значения по умолчанию и могут быть не переданы функции.

На рис. 30 представлены пример функции с обязательными аргументами и пример функции с необязательными аргументами.

Пример функции с обязательными аргументами:

```
def greet(name):
    print("Hello, " + name)

greet("Alice")
```

Пример функции с необязательными аргументами:

```
def greet(name, message="Hello"):
    print(message + ", " + name)

greet("Alice")
greet("Bob", "Hi")
```

Рис. 30. Описание функции с обязательными/необязательными аргументами

Функции в Python могут быть вызваны из других частей программы с помощью имени функции и передачи необходимых аргументов.

На рис. 31 представлены примеры описания использования функций в программе. Программа использует три разные функции: factorial(), is_prime() и get_primes_until(). Она запрашивает у пользователя число, вычисляет его факториал с помощью функции factorial(), а затем находит список простых чисел до этого числа с помощью функций is_prime() и get_primes_until() и выводит их на экран.

```python
# Функция для вычисления факториала числа
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

# Функция для проверки, является ли число простым
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

# Функция для получения списка простых чисел до заданного числа
def get_primes_until(n):
    primes = []
    for i in range(2, n+1):
        if is_prime(i):
            primes.append(i)
    return primes

# Пример использования функций
num = int(input("Введите число: "))
fact = factorial(num)
print(f"Факториал числа {num} равен {fact}")

primes = get_primes_until(num)
print(f"Простые числа до {num}: {primes}")
```

Рис. 31. Примеры описания использования функций

Представим пошаговое объяснение, что делает каждая строка кода в этой программе:

– объявляет функцию factorial(), которая принимает один аргумент n и вычисляет его факториал;
– проверяет, является ли n равным 0;
– возвращает 1, если n равен 0;
– возвращает n умноженное на факториал (n - 1), если n не равен 0;
– объявляет функцию is_prime(), которая принимает один аргумент n и проверяет, является ли n простым числом;
– возвращает False, если n меньше или равен 1;
– используя оператор for, программа перебирает числа от 2 до квадратного корня из n + 1;
– проверяет, делится ли n на i без остатка, и возвращает False, если делится;
– возвращает True, если n является простым числом;
– объявляет функцию get_primes_until(), которая принимает один аргумент n и возвращает список простых чисел до n;
– создает пустой список primes;
– используя оператор for, программа перебирает числа от 2 до n + 1;
– проверяет, является ли i простым числом с помощью функции is_prime(), и добавляет его в список primes, если является;
– возвращает список простых чисел;
– запрашивает у пользователя число и преобразует его в целое число;
– вычисляет факториал числа num с помощью функции factorial() и сохраняет результат в переменную fact;
– выводит сообщение о том, что был вычислен факториал числа num и его значение;
– находит список простых чисел до num с помощью функции get_primes_until() и сохраняет его в переменную primes;
– выводит список простых чисел до num на экран.

Программа демонстрирует использование различных функций в Python для выполнения различных задач. Она объявляет три функции: factorial(), is_prime() и get_primes_until(), которые выполняют различные действия, и использует эти функции для вычисления факториала числа, проверки числа на простоту и нахождения списка простых чисел.

В Python функции могут быть определены внутри других функций. Эти функции называются **вложенными функциями** или **внутренними функциями**. Вложенные функции могут быть использованы для улучшения организации кода, а также для повышения безопасности и сокрытия функций от других частей программы.

Вложенные функции определяются так же, как и обычные функции, с помощью ключевого слова def, но внутри другой функции.

На рис. 32 представлен пример программы, демонстрирующей работу вложенных функций, то есть определение функции внутри других функций. Программа использует вложенную функцию inner_func() внутри внешней функции outer_func(). Она устанавливает значение переменной x в 10 внутри outer_func(), а затем вызывает inner_func(), которая увеличивает значение x на 5 и выводит его на экран. Затем outer_func() также выводит значение x на экран.

```
def outer_func():
    x = 10

    def inner_func():
        nonlocal x
        x += 5
        print(f"Значение x внутри вложенной функции: {x}")

    inner_func()
    print(f"Значение x внутри внешней функции: {x}")

outer_func()
```

Рис. 32. Описание вложенных функций

Опишем пошаговое объяснение, что делает каждая строка кода в этой программе:

– объявляет внешнюю функцию outer_func();
– устанавливает значение переменной x равной 10;
– объявляет вложенную функцию inner_func();
– объявляет переменную x как нелокальную, чтобы ее значение можно было изменять внутри вложенной функции;
– увеличивает значение переменной x на 5;

– выводит значение переменной x внутри вложенной функции;
– вызывает вложенную функцию inner_func();
– выводит значение переменной x внутри внешней функции;
– вызывает внешнюю функцию outer_func().

В этой программе вложенная функция inner_func() объявляется внутри внешней функции outer_func(). Поскольку inner_func() находится внутри outer_func(), она имеет доступ к переменным, объявленным внутри outer_func(), включая переменную x. Когда inner_func() вызывается, она увеличивает значение переменной x на 5 и выводит его на экран. Затем outer_func() также выводит значение переменной x на экран.

Важной особенностью вложенной функции является то, что она имеет доступ к переменным внешней функции даже после того, как была выполнена. Кроме того, объявление переменной x как нелокальной с помощью ключевого слова nonlocal позволяет изменять ее значение внутри вложенной функции. Это может быть полезным, если вы хотите изменить значение переменной внутри вложенной функции и сохранить это изменение после того, как вложенная функция завершится.

Вложенные функции могут иметь доступ к переменным внешней функции. Для этого необходимо использовать ключевое слово nonlocal.

На рис. 33 представлен пример, который демонстрирует доступ к внешним переменным, то есть переменным, которые определены вне функции, но доступны внутри нее. Программа имеет две переменные с именем x. Первая переменная объявляется вне функций и имеет начальное значение 5. Вторая переменная также называется x и объявляется внутри функции outer_func() с начальным значением 10. Затем вложенная функция inner_func() изменяет значение второй переменной x на 5 с помощью ключевого слова nonlocal и выводит ее значение на экран. Затем outer_func() также выводит значение второй переменной x на экран. За пределами функций первая переменная x сохраняет свое значение и выводится на экран.

```
x = 5

def outer_func():
    x = 10

    def inner_func():
        nonlocal x
        x += 5
        print(f"Значение x внутри вложенной функции: {x}")

    inner_func()
    print(f"Значение x внутри внешней функции: {x}")

outer_func()
print(f"Значение x вне функций: {x}")
```

Рис. 33. Доступ к внешним переменным

Доступ к внешним переменным в Python работает следующим образом:

– если переменная используется только внутри функции, Python ищет ее значение в локальной области видимости функции;
– если переменная не найдена в локальной области видимости функции, Python ищет ее значение в глобальной области видимости;
– если переменная не найдена в глобальной области видимости, Python ищет ее значение во встроенных именах;
– если переменная определена внутри другой функции, Python ищет ее значение в области видимости этой функции;
– если переменная не найдена в области видимости внутренней функции, Python ищет ее значение в области видимости внешней функции;
– если переменная не найдена в области видимости внешней функции, Python продолжает искать ее значение в глобальной области видимости и во встроенных именах.

В примере программы первая переменная x находится в глобальной области видимости, а вторая переменная x находится в локальной области видимости функции outer_func(). Вложенная функция inner_func() имеет доступ к переменной x, объявленной внутри внешней функции outer_func(), потому что в Python переменные внутренних функций имеют доступ к переменным внеш-

них функций. С помощью ключевого слова nonlocal вложенная функция может изменять значение переменной x, объявленной внутри внешней функции.

Доступ к внешним переменным в Python позволяет функциям иметь доступ к переменным, объявленным во внешних областях видимости, и изменять их значения при необходимости.

Вложенные функции могут использоваться для создания замыканий. **Замыкание** – функция, которая запоминает значения из окружающей среды даже после того, как эта среда была изменена. Замыкание – функция, которая ссылается на переменные в своей внешней области видимости, которые уже были удалены из стека вызовов. В Python замыкание создается тогда, когда внутренняя функция ссылается на переменные внешней функции. Внутренняя функция сохраняет ссылки на эти переменные в замыкании, что позволяет ей использовать их значения в будущем.

На рис. 34 представлен фрагмент программы, которая демонстрирует замыкание, то есть возможность вложенной функции сохранять значения внешних переменных, которые были переданы ей в качестве аргументов.

```
def outer_func(x):
    def inner_func(y):
        return x + y
    return inner_func

add_five = outer_func(5)

result = add_five(10)

print(result)
```

Рис. 34. Описание замыкания

В этой программе функция outer_func() возвращает вложенную функцию inner_func(), которая принимает аргумент y и возвращает сумму x и y. Затем outer_func() вызывается с аргументом 5, и возвращаемая функция сохраняется в переменной add_five. После этого add_five() вызывается с аргументом 10, и результат 15 сохраняется в переменной result и выводится на экран. Когда outer_func() вы-

зывается с аргументом 5, Python создает замыкание, которое содержит ссылку на переменную x. Затем outer_func() возвращает функцию inner_func(), которая сохраняет ссылку на x в замыкании. Когда add_five() вызывается с аргументом 10, inner_func() использует сохраненную ссылку на x в замыкании и возвращает сумму x и y, то есть 5 + 10 = 15.

Программа демонстрирует, как замыкания позволяют функциям в Python использовать переменные, объявленные внутри других функций, и сохранять их значения между вызовами. В этом примере замыкание создается при вызове outer_func() и сохраняет ссылку на переменную x внутри возвращаемой функции inner_func(). Когда inner_func() вызывается через add_five(), она использует сохраненную ссылку на переменную x для вычисления суммы.

Код на рис. 35 демонстрирует еще один пример использования замыкания.

```
def make_multiplier(x):
    def multiplier(n):
        return x * n
    return multiplier

times_three = make_multiplier(3)
times_five = make_multiplier(5)

print(times_three(10)) # Выводит 30
print(times_five(10)) # Выводит 50
```

Рис. 35. Пример описания замыкания

В этом примере функция make_multiplier() возвращает вложенную функцию multiplier(), которая принимает аргумент n и возвращает произведение x и n. Затем make_multiplier() вызывается дважды, и возвращаемые функции сохраняются в переменных times_three и times_five. После этого times_three() вызывается с аргументом 10, и результат 30 сохраняется в переменной result. Затем times_five() вызывается с аргументом 10, и результат 50 сохраняется в переменной result, и результаты выводятся на экран. Когда make_multiplier() вызывается с аргументом x, Python создает замыкание, которое содержит ссылку на переменную x. Затем make_multiplier() возвращает функцию multiplier(), которая сохраняет ссылку на x в замыкании.

Когда times_three() вызывается с аргументом 10, функция multiplier() использует сохраненную ссылку на x в замыкании и возвращает произведение 3 * 10 = 30. Аналогично, когда times_five() вызывается с аргументом 10, функция multiplier() использует сохраненную ссылку на x в замыкании и возвращает произведение 5 * 10 = 50.

Этот пример демонстрирует, как замыкания могут использоваться для создания функций, которые зависят от параметров, переданных при их создании. В этом примере замыкание создается при вызове make_multiplier() и сохраняет ссылку на переменную x внутри возвращаемой функции multiplier(). Когда multiplier() вызывается через times_three() или times_five(), она использует сохраненную ссылку на переменную x в замыкании и возвращает произведение x и n.

Модули в Python

Это файлы, содержащие код, который можно использовать в других программах. Модули упрощают написание кода, позволяют избежать дублирования кода и повторного использования кода. В Python модули могут быть импортированы в другие программы с помощью ключевого слова import.

Для создания модуля нужно определить функции, классы и переменные, которые будут использоваться в других программах, в отдельном файле с расширением «.py». В модуле можно определить любое количество функций, классов и переменных.

На рис. 36 представлен пример программы, демонстрирующей создание модуля, то есть файла, содержащего определения функций, классов и переменных, которые могут быть использованы в других программах.

```
# Определение функции say_hello()
def say_hello(name):
    print(f"Hello, {name}!")

# Определение функции add_numbers()
def add_numbers(x, y):
    return x + y
```

Рис. 36. Создание модуля

В этом примере мы создаем модуль my_module, который содержит две функции: say_hello() и add_numbers(). Функция say_hello() принимает аргумент name и выводит на экран приветствие с именем, переданным в качестве аргумента. Функция add_numbers() принимает два аргумента x и y и возвращает их сумму.

Чтобы использовать этот модуль в других программах, нам нужно импортировать его (рис. 37). В этой программе мы импортируем модуль my_module с помощью инструкции import. Затем вызываем функцию say_hello() из модуля, передавая ей аргумент «John». После этого вызываем функцию add_numbers() из модуля, передавая ей аргументы 5 и 10, и сохраняем результат в переменной result и выводим результат на экран.

```
# Импортирование модуля my_module
import my_module

# Вызов функции say_hello()
my_module.say_hello("John")

# Вызов функции add_numbers()
result = my_module.add_numbers(5, 10)

# Вывод результата на экран
print(result)
```

Рис. 37. Использование модуля

Когда импортируем модуль my_module, Python выполняет код в этом файле и создает объект модуля. Функции say_hello() и add_numbers() становятся доступными через этот объект модуля. Мы вызываем функции, используя синтаксис <имя_модуля>.<имя_функции>().

Для импортирования модуля в другую программу можно использовать ключевое слово import. После импорта модуля его функции, классы и переменные становятся доступными в программе.

Фрагмент кода программы (рис. 38) показывает, как можно импортировать модуль mymodule с помощью команды import. Затем программа вызывает функцию add_numbers() из модуля mymodule с аргументами 5 и 7 и выводит результат на экран с помощью функции print(). Далее программа создает объект person класса Person

из модуля mymodule, передавая ему аргументы «John» и 30, и вызывает метод say_hello() для этого объекта. Программа выводит значение переменной message из модуля mymodule на экран.

```
# Модуль mymodule

def add_numbers(a, b):
    return a + b
message = "Hello from mymodule!"
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def say_hello(self):
        print("Hello, my name is", self.name, "and I am", self.age, "years old.")


import mymodule

print(mymodule.add_numbers(5, 7))
person = mymodule.Person("John", 30)
person.say_hello()
print(mymodule.message)
```

Рис. 38. Импортирование модуля

Импортирование модуля позволяет использовать определения функций, классов и переменных из другого модуля в текущей программе. При импортировании модуля все его определения становятся доступными в текущей программе через имя модуля. Например, для вызова функции add_numbers() из модуля mymodule используем имя модуля и имя функции, разделенные точкой (mymodule.add_numbers()). Аналогично для создания объекта класса Person используем имя модуля и имя класса (mymodule.Person(«John», 30)).

В языке программирования Python модуль представляет собой файл с расширением «.py», содержащий определения функций, классов, переменных и других объектов, которые могут быть использованы в других программах. Иногда может возникнуть необходимость импортировать только некоторые объекты из модуля, а не все.

Для импортирования конкретных объектов из модуля в Python используется ключевое слово from, за которым следует имя модуля, затем ключевое слово import и список импортируемых объектов через запятую. Для импортирования конкретных функций, классов

или переменных из модуля можно использовать ключевое слово from, как показано на рис. 39. Программа в примере импортирует только функцию add_numbers и класс Person из модуля mymodule с помощью команды from ... import. Затем программа вызывает функцию add_numbers() из модуля mymodule с аргументами 5 и 7 и выводит результат на экран с помощью функции print(). Далее программа создает объект person класса Person из модуля mymodule, передавая ему аргументы «John» и 30, и вызывает метод say_hello() для этого объекта.

```
from mymodule import add_numbers, Person

print(add_numbers(5, 7))
person = Person("John", 30)
person.say_hello()
```

Рис. 39. Импортирование конкретного объекта

Импортирование конкретных объектов из модуля позволяет использовать только нужные объекты из модуля в текущей программе, что может упростить код и ускорить его выполнение. При импортировании конкретных объектов из модуля не нужно указывать имя модуля при вызове этих объектов, так как они становятся доступными в текущей программе под своими именами (add_numbers() и Person() в данном примере).

При импортировании конкретных объектов из модуля остальные объекты в модуле не будут доступны в текущей программе, и если они понадобятся, их нужно будет импортировать отдельно или импортировать весь модуль целиком.

Для удобства чтения кода можно использовать псевдонимы для имен модулей. Для этого используется ключевое слово as.

На рис. 40 показан пример программы, демонстрирующей создание псевдонимов модулей.

В этом примере мы импортируем модуль numpy с помощью инструкции import и задаем ему псевдоним np с помощью ключевого слова as. Модуль numpy содержит функции для работы с массивами, векторами и матрицами. Когда импортируем этот модуль

с псевдонимом, можно использовать этот псевдоним вместо имени модуля при вызове его функций.

```
# Импортирование модуля numpy с псевдонимом np
import numpy as np

# Создание массива чисел с помощью функции arange()
my_array = np.arange(10)

# Вывод массива на экран
print("Массив чисел:", my_array)

# Вычисление суммы элементов массива
array_sum = np.sum(my_array)

# Вывод суммы на экран
print("Сумма элементов массива:", array_sum)
```

Рис. 40. Создание псевдонима модуля

Использование псевдонимов модулей может быть полезным, когда у нас есть модуль с длинным именем, которое неудобно использовать в коде. Псевдонимы могут также помочь избежать конфликтов имен, если импортируется несколько модулей с функциями, имеющими одинаковые имена. Создание псевдонимов модулей позволяет использовать модуль под другим именем в текущей программе. Это может быть полезно, если имя модуля слишком длинное или неудобное для использования в коде. При задании псевдонима для модуля все его определения становятся доступными в текущей программе через новое имя модуля.

Модули для работы с файлами в Python, которые предоставляют дополнительные функции

В Python существует несколько модулей для работы с файлами, которые предоставляют дополнительные функции и возможности по сравнению с базовыми функциями работы с файлами.

Модуль os предоставляет функции для работы с операционной системой, в том числе для работы с файлами и директориями. Базо-

вые функции, которые можно использовать для работы с файлами, представлены на рис. 41 в примере программы. Дадим пошаговое объяснение, что делает каждая строка кода в этой программе:

– импортирует модуль os для работы с функциями операционной системы;
– используя функцию getcwd(), программа получает текущую рабочую директорию и сохраняет ее в переменную current_directory;
– выводит текущую рабочую директорию на экран;
– создает новую переменную new_directory и присваивает ей значение «новая_директория»;
– используя функцию mkdir(), программа создает новую директорию с именем, указанным в переменной new_directory;
– выводит сообщение о том, что новая директория была успешно создана;
– используя функцию chdir(), программа переходит в новую директорию, указанную в переменной new_directory;
– выводит сообщение о том, что программа перешла в новую директорию;
– используя функцию getcwd(), программа получает текущую рабочую директорию в новой директории и сохраняет ее в переменную current_directory;
– выводит текущую рабочую директорию в новой директории на экран;
– используя функцию chdir(), программа возвращает текущую директорию на один уровень выше;
– выводит сообщение о том, что программа вернулась в исходную директорию;
– используя функцию rmdir(), программа удаляет новую директорию, указанную в переменной new_directory;
– выводит сообщение о том, что новая директория была успешно удалена.

```
import os

# Получаем текущую рабочую директорию
current_directory = os.getcwd()
print("Текущая директория:", current_directory)

# Создаем новую директорию
new_directory = "новая_директория"
os.mkdir(new_directory)
print("Новая директория создана")

# Переходим в новую директорию
os.chdir(new_directory)
print("Переход в новую директорию")

# Получаем текущую рабочую директорию в новой директории
current_directory = os.getcwd()
print("Текущая директория:", current_directory)

# Возвращаемся в исходную директорию
os.chdir("..")
print("Возвращение в исходную директорию")

# Удаляем новую директорию
os.rmdir(new_directory)
print("Новая директория удалена")
```

Рис. 41. Базовые функции модуля os

Модуль shutil предоставляет функции для работы с файлами и директориями, такие как копирование файлов, перемещение файлов, удаление директорий и т. д.

На рис. 42 представлен пример использования функций модуля. Эта программа использует модуль shutil для копирования и перемещения файлов и директорий, а также модуль os для создания временной директории и файлов в ней.

```
import shutil
import os

# Создаем временную директорию
temp_dir = "temp"
os.mkdir(temp_dir)
print("Временная директория создана")

# Создаем файлы во временной директории
file1 = "temp/file1.txt"
file2 = "temp/file2.txt"
with open(file1, "w") as f1:
    f1.write("Это файл 1")
with open(file2, "w") as f2:
    f2.write("Это файл 2")
print("Файлы созданы во временной директории")

# Копируем файлы в новую директорию
new_dir = "новая_директория"
shutil.copy(file1, new_dir)
shutil.copy(file2, new_dir)
print("Файлы скопированы в новую директорию")

# Перемещаем файлы в новую директорию
shutil.move(file1, new_dir)
shutil.move(file2, new_dir)
print("Файлы перемещены в новую директорию")

# Удаляем временную директорию
shutil.rmtree(temp_dir)
print("Временная директория удалена")
```

Рис. 42. Базовые функции модуля shutil

Представим пошаговое объяснение, что делает каждая строка кода в этой программе:

– импортирует модули shutil и os для работы с функциями файловой системы;
– создает переменную temp_dir и присваивает ей значение temp;
– используя функцию mkdir(), программа создает новую временную директорию с именем, указанным в переменной temp_dir;
– выводит сообщение о том, что временная директория была успешно создана;

– создает переменные file1 и file2 и присваивает им значения «temp/file1.txt» и «temp/file2.txt» соответственно;
– используя оператор with, программа открывает файлы file1 и file2 для записи;
– записывает строки в файлы file1 и file2 соответственно;
– выводит сообщение о том, что файлы были успешно созданы во временной директории;
– создает переменную new_dir и присваивает ей значение «новая_директория»;
– используя функцию copy(), программа копирует файлы file1 и file2 в новую директорию new_dir.
– выводит сообщение о том, что файлы были успешно скопированы в новую директорию;
– используя функцию move(), программа перемещает файлы file1 и file2 в новую директорию new_dir;
– выводит сообщение о том, что файлы были успешно перемещены в новую директорию;
– используя функцию rmtree(), программа удаляет временную директорию temp_dir и все ее содержимое;
– выводит сообщение о том, что временная директория была успешно удалена.

Программа демонстрирует базовые функции модуля shutil для копирования и перемещения файлов и директорий. Она создает временную директорию, создает в ней файлы, копирует и перемещает их в новую директорию, а затем удаляет временную директорию.

Модуль glob в Python предоставляет возможность поиска файлов и директорий по шаблону. Шаблон – это строка, которая может содержать символы «*» (заменяет любое количество символов), «?» (заменяет один символ) и «[]» (символьный класс для указания допустимых символов). Модуль glob позволяет поискать файлы в директории и ее поддиректориях, используя указанный шаблон.

На рис. 43 представлен пример использования функций модуля. Программа ищет все файлы в текущей директории с расширением .txt и выводит их имена, а также находит все директории в текущей директории, кроме тех, которые имеют расширение .txt, и выводит их имена.

```
import glob

# Находим все файлы в текущей директории с расширением .txt
txt_files = glob.glob("*.txt")

# Выводим найденные файлы
print("Найдены файлы:")
for file in txt_files:
    print(file)

# Находим все директории в текущей директории
dirs = glob.glob("*[!txt]")

# Выводим найденные директории
print("Найдены директории:")
for dir in dirs:
    print(dir)
```

Рис. 43. Базовые функции модуля glob

Представим пошаговое объяснение, что делает каждая строка кода в этой программе:

– импортирует модуль glob для работы с шаблонами поиска файлов и директорий;
– используя функцию glob(), программа находит все файлы в текущей директории, имена которых заканчиваются на .txt, и сохраняет их имена в переменную txt_files;
– выводит сообщение о том, что были найдены файлы;
– используя оператор for, программа перебирает каждый файл, найденный в предыдущем шаге;
– выводит имя каждого файла, найденного в предыдущем шаге;
– используя функцию glob(), программа находит все директории в текущей директории, имена которых не заканчиваются на .txt, и сохраняет их имена в переменную dirs;
– выводит сообщение о том, что были найдены директории;
– используя оператор for, программа перебирает каждую директорию, найденную в предыдущем шаге;
– выводит имя каждой директории, найденной в предыдущем шаге.

Программа демонстрирует функциональность модуля glob для поиска файлов и директорий в директории с помощью шаблонов.

Она находит все файлы в текущей директории, имена которых заканчиваются на .txt, и выводит их имена, а также находит все директории в текущей директории, кроме тех, которые имеют расширение .txt, и выводит их имена. В реальном приложении вы можете использовать этот модуль для поиска файлов и директорий с определенными шаблонами, а также для выполнения других операций с найденными файлами и директориями.

Исключения в Python

Исключения – это ошибки, которые возникают во время выполнения программы. Когда возникает исключение, программа останавливается и выводит сообщение об ошибке. В Python исключения могут быть обработаны с помощью конструкции try/except.

Исключения могут быть вызваны явно с помощью ключевого слова raise. Когда исключение вызывается, программа останавливается и выводит сообщение об ошибке.

На рис. 44 представлен пример программы, демонстрирующей вызов исключений.

```
def divide_numbers(a, b): # Определяем функцию divide_numbers(), которая принимает два аргумента и возвращает их частное
    if b == 0:
        raise ZeroDivisionError("Деление на ноль невозможно")  # Если второй аргумент равен нулю, вызываем исключение ZeroDivisionError
    else:
        return a / b
try:
    print(divide_numbers(10, 2)) # Вызываем функцию divide_numbers() с разными аргументами
    print(divide_numbers(5, 0))
except ZeroDivisionError as error:
        print("Произошла ошибка:", error)  # Обрабатываем исключение ZeroDivisionError, которое может быть вызвано в функции divide_numbers()
```

Рис. 44. Вызов исключений

Программа определяет функцию divide_numbers(), которая выполняет деление двух чисел и возвращает результат. Однако если второй аргумент равен нулю, функция вызывает исключение ZeroDivisionError с помощью ключевого слова raise. Затем программа вызывает функцию divide_numbers() с разными аргументами

и обрабатывает возможное исключение ZeroDivisionError с помощью блока except. Ключевое слово raise используется для вызова исключения в явном виде. В данном примере мы вызываем исключение ZeroDivisionError с сообщением «Деление на ноль невозможно», если второй аргумент функции равен нулю. Это позволяет остановить выполнение функции и передать управление в блок except в том месте, где функция была вызвана. В блоке try вызываем функцию divide_numbers() с аргументами 10 и 2, а затем с аргументами 5 и 0. При попытке выполнения деления на ноль во втором вызове функции возникает исключение ZeroDivisionError, которое обрабатывается в блоке except. В данном примере мы просто выводим сообщение об ошибке с помощью функции print() и передаем объект исключения в качестве аргумента функции print() с помощью переменной error.

Вызов исключений позволяет явно указать на возникновение ошибки в программе и передать управление в блок except, где она может быть обработана. В блоке try вызываем функцию, которая может вызвать исключение, а в блоке except обрабатываем это исключение и выводим сообщение об ошибке. В результате программа может корректно работать даже в случае возникновения ошибок.

Исключения могут быть обработаны с помощью конструкции try/except. Код, который может вызвать исключение, помещается в блок try (рис. 45). Если исключение возникает, программа переходит в блок except, где исключение может быть обработано.

```
try:
    # блок кода, который может вызвать исключение
    pass
except:
    # блок кода, который обрабатывает исключение
    pass
```

Рис. 45. Описание конструкции для исключений

Пример программы, демонстрирующей обработку исключений, представлен на рис. 46. Эта программа запрашивает у пользователя два числа, выполняет деление и выводит результат на экран.

```
try:
    # Запрашиваем у пользователя два числа
    num1 = int(input("Введите первое число: "))
    num2 = int(input("Введите второе число: "))

    # Выполняем деление и выводим результат на экран
    result = num1 / num2
    print("Результат деления:", result)

# Обрабатываем исключение, которое может возникнуть при неправильном вводе данных или делении на ноль
except (ValueError, ZeroDivisionError) as error:
    print("Произошла ошибка:", error)
```

Рис. 46. Пример исключений с использованием конструкции

Однако она также предусматривает обработку возможных исключений, которые могут возникнуть при неправильном вводе данных или делении на ноль. Ключевое слово try обозначает начало блока кода, в котором может произойти исключение. Если исключение происходит внутри блока try, то программа переходит к блоку except, где обрабатывается исключение.

В данном примере мы используем несколько исключений, объединенных в кортеж с помощью скобок: (ValueError, ZeroDivisionError). Если любое из этих исключений возникает в блоке try, то программа переходит к блоку except, где обрабатывается исключение. В нашем случае выводим сообщение об ошибке с помощью функции print() и передаем объект исключения в качестве аргумента функции print() с помощью переменной error.

Обработка исключений позволяет предвидеть возможные ошибки в программе и обрабатывать их, чтобы программа не прерывалась и не выдавала ошибки пользователю. В блоке try размещаем код, который может вызвать исключение, а в блоке except обрабатываем это исключение и выводим сообщение об ошибке. В результате программа может корректно работать даже в случае возникновения ошибок.

Тема 3. Объектно ориентированное программирование на Python

Основные понятия ООП

Объектно ориентированное программирование (ООП) – это парадигма программирования, основанная на концепции объектов, которые объединяют данные и функциональность в единую сущность. В ООП объекты являются экземплярами классов, которые определяют свойства и методы объектов.

Основные понятия в ООП

Класс – шаблон или описание объекта, который определяет свойства и методы объектов класса.

Объект – экземпляр класса, который содержит данные и функциональность, определенные в классе.

Атрибут – переменная, которая хранит данные объекта.

Метод – функция, которая определена в классе и может изменять данные объекта.

Наследование – механизм, который позволяет создавать новые классы на основе существующих классов, наследуя их свойства и методы.

Полиморфизм – возможность объектов разных классов использовать одинаковое имя метода, но с различной реализацией.

Инкапсуляция – концепция, которая обеспечивает скрытие реализации объекта от пользователей объекта.

Специфика ООП в языке Python:

– Python поддерживает все основные концепции ООП, включая классы, объекты, наследование, полиморфизм и инкапсуляцию;
– в Python нет модификаторов доступа к атрибутам и методам, но существует соглашение о том, что атрибуты и методы, начинающиеся с символа подчеркивания, считаются защищенными и не должны использоваться вне класса;
– в Python конструктор класса __init__ и метод self используются для инициализации атрибутов объекта;
– в Python методы могут быть определены как обычные методы, статические методы или методы класса с помощью декораторов @staticmethod, @classmethod и @property соответственно;

– Python поддерживает множественное наследование, то есть класс может наследовать свойства и методы сразу от нескольких классов.

На рис. 47 представлен пример программы на Python, которая демонстрирует использование ООП.

В этом примере мы определяем два класса: Fruit и Apple. Класс Apple наследуется от класса Fruit. Класс Fruit имеет конструктор __init__, который инициализирует свойства name и color. Класс Fruit также имеет метод describe, который выводит название и цвет фрукта. Класс Apple также имеет конструктор __init__, который вызывает конструктор родительского класса с помощью функции super(). Класс Apple также имеет свойство variety и метод describe, который вызывает метод describe родительского класса и добавляет информацию о сорте яблока. Затем создаем объекты классов Fruit и Apple. Вызываем метод describe для каждого объекта, чтобы вывести информацию о фрукте или яблоке.

```
# Определение класса "Фрукт"
class Fruit:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def describe(self):
        print(f"Этот фрукт называется {self.name}, его цвет - {self.color}")

# Определение класса "Яблоко", который наследуется от класса "Фрукт"
class Apple(Fruit):
    def __init__(self, name, color, variety):
        super().__init__(name, color)
        self.variety = variety

    def describe(self):
        super().describe()
        print(f"Это яблоко сорта {self.variety}")

# Создание объектов классов "Фрукт" и "Яблоко"
fruit = Fruit("Апельсин", "оранжевый")
apple = Apple("Яблоко", "красный", "Голден")

# Вызов методов объектов классов
fruit.describe()
apple.describe()
```

Рис. 47. Использование ООП

Использование классов и наследования позволяет создавать объекты с общими свойствами и методами. Классы позволяют нам абстрагироваться от конкретных реализаций и создавать универсальные решения. Наследование позволяет создавать новые классы на основе существующих и использовать их свойства и методы.

Класс

Одним из основных понятий ООП является класс.

Классы в Python – это шаблоны для создания объектов. Класс определяет состояние (атрибуты) и поведение (методы) объектов. Классы позволяют создавать объекты с общими свойствами и методами, что делает код более структурированным и удобным для использования.

Классы в Python – основа объектно ориентированного программирования. Они позволяют определять новые типы данных и выступают в роли «шаблонов» для создания объектов. Классы могут содержать методы (функции), атрибуты (переменные) и конструкторы (специальные методы, которые вызываются при создании нового объекта).

На рис. 48 представлен пример программы, демонстрирующей создание класса Person, который имеет конструктор __init__, который инициализирует свойства name и age. Класс Person также имеет метод describe, который выводит информацию о человеке.

```
# Определение класса "Человек"
class Person:
    # Конструктор класса
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # Метод класса для вывода информации о человеке
    def describe(self):
        print(f"{self.name} - {self.age} лет")

# Создание объекта класса "Человек" и вызов методов
person = Person("Иван", 25)
person.describe()
```

Рис. 48. Создание класса

Использование классов позволяет создавать объекты с общими свойствами и методами. Классы позволяют абстрагироваться от конкретных реализаций и создавать универсальные решения.

Создание классов позволяет определять новые типы объектов в Python и задавать им свойства и методы. Классы могут содержать атрибуты, методы и конструкторы, которые могут быть использованы для создания и манипуляции объектами класса. Создание объектов класса позволяет работать с экземплярами класса и использовать их методы и свойства в программе.

Наследование

Это механизм, который позволяет классам наследовать свойства и методы других классов. Класс, который наследует свойства и методы, называется дочерним классом, а класс, от которого наследуются свойства и методы, называется родительским классом.

На рис. 49 представлен пример программы на Python, которая демонстрирует наследование классов. Поясним, как работает программа.

• Определяем класс Shape, который имеет конструктор __ init__, инициализирующий свойство color и метод info, который выводит информацию о фигуре.

• Определяем класс Square, который наследуется от класса Shape и имеет конструктор __init__, который вызывает конструктор родительского класса с помощью функции super(). Класс Square также имеет свойство side_length и переопределяет метод info, чтобы вывести информацию о квадрате и длине его стороны.

• Создаем объекты классов Shape и Square.

• Вызываем метод info для каждого объекта, чтобы вывести информацию о фигуре или квадрате.

Использование наследования классов позволяет создавать классы, которые наследуют свойства и методы от других классов, что позволяет нам использовать код многократно и экономить время. В данном примере класс Square наследует свойство color и метод info от класса Shape. Затем класс Square добавляет свойство side_length и переопределяет метод info, чтобы вывести информацию о квадрате.

```python
# Определение родительского класса "Фигура"
class Shape:
    # Конструктор класса
    def __init__(self, color):
        self.color = color

    # Метод класса, выводящий информацию о фигуре
    def info(self):
        print(f"Это {self.color} фигура")

# Определение дочернего класса "Квадрат", который наследуется от класса "Фигура"
class Square(Shape):
    # Конструктор класса
    def __init__(self, color, side_length):
        super().__init__(color)
        self.side_length = side_length

    # Метод класса, выводящий информацию о квадрате
    def info(self):
        super().info()
        print(f"Сторона квадрата: {self.side_length}")

# Создание объектов классов "Фигура" и "Квадрат"
shape = Shape("зеленая")
square = Square("красный", 5)

# Вызов методов объектов классов
shape.info()
square.info()
```

Рис. 49. Наследование классов

Наследование позволяет создавать новые классы на основе существующих, повторно используя код и избегая дублирования. Класс-наследник может использовать все свойства и методы родительского класса, а также добавлять свои собственные свойства и методы. В результате код становится более читабельным, легко поддерживаемым и масштабируемым.

Специальные методы

Поведение объектов классов в ответ на стандартные операторы и функции Python позволяют определять специальные методы. Они могут быть переопределены для определенных классов, чтобы обеспечить правильное выполнение операций, которые могут быть не очевидными для стандартных операторов и функций Python. Они используются для переопределения поведения стандартных операторов и встроенных функций Python для объектов класса.

Специальные методы начинаются и заканчиваются двойным подчеркиванием, например, __str__().

На рис. 50 представлен пример программы, демонстрирующей специальные методы классов. В примере мы определяем класс Book, который имеет конструктор __init__, который инициализирует свойства title, author и pages. Класс Book также имеет специальные методы:

__str__: переопределяет метод str(), который вызывается, когда мы хотим преобразовать объект в строку. Метод __str__ возвращает строку, которая содержит название книги и имя автора;

__len__: переопределяет метод len(), который вызывается, когда хотим узнать длину объекта. Метод __len__ возвращает количество страниц в книге.

```
# Определение класса "Книга"
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def __str__(self):
        return f"{self.title} - {self.author}"

    def __len__(self):
        return self.pages

# Создание объекта класса "Книга"
book = Book("Мастер и Маргарита", "Михаил Булгаков", 512)

# Вызов специальных методов объекта класса "Книга"
print(book)
print(len(book))
```

Рис. 50. Специальные методы класса

Специальные методы классов в Python позволяют переопределять стандартное поведение объектов. Можно определять специальные методы для того, чтобы наш класс мог работать с операторами Python, такими как +, -, *, /, %, ==, != и многими другими. Также можем определять специальные методы для того, чтобы наш класс мог работать с встроенными функциями Python, такими как str(), len(), max(), min() и другими.

Пример программы, демонстрирующей класс с использованием наследования, представлен на рис. 51.

В примере определяем два класса: Person и Student. Класс Student наследуется от класса Person. Класс Person имеет конструктор __init__, который инициализирует свойства name и age. Класс Person также имеет метод info, который выводит информацию о персоне. Класс Student также имеет конструктор __init__, который вызывает конструктор родительского класса с помощью функции super(). Класс Student также имеет свойство grade и переопределяет метод info, чтобы вывести информацию о студенте и его классе. Затем создаем объекты классов Person и Student. Вызываем метод info для каждого объекта, чтобы вывести информацию о персоне или студенте.

```
# Определение класса "Персона"
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"Имя: {self.name}, Возраст: {self.age}")

# Определение класса "Студент", который наследуется от класса "Персона"
class Student(Person):
    def __init__(self, name, age, grade):
        super().__init__(name, age)
        self.grade = grade

    def info(self):
        super().info()
        print(f"Класс: {self.grade}")

# Создание объектов классов "Персона" и "Студент"
person = Person("Иван", 25)
student = Student("Петр", 19, 11)

# Вызов методов объектов классов
person.info()
student.info()
```

Рис. 51. Класс с использованием наследования

Использование наследования классов позволяет нам создавать новые классы, которые наследуют свойства и методы от родительских классов. Это позволяет нам избежать дублирования кода и создавать более читаемые и легко поддерживаемые программы.

Объект

Еще одним основным понятием объектно ориентированного подхода к программированию являются объекты.

Объекты в Python представляют собой экземпляры классов. Каждый объект имеет свои уникальные атрибуты, которые могут быть установлены и получены извне. Атрибуты объекта могут быть переменными (числовыми, строковыми, логическими и т. д.) или функциями (методами).

Атрибуты объектов представляют собой переменные, связанные с конкретным экземпляром класса. Они могут быть использованы для хранения информации о состоянии объекта и быть изменены в процессе выполнения программы. На рис. 52 представлен пример программы, демонстрирующей атрибуты объектов.

```
# Определение класса "Автомобиль"
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.mileage = 0

    def info(self):
        print(f"Марка: {self.make}, Модель: {self.model}, Год выпуска:
{self.year}, Пробег: {self.mileage}")

    def drive(self, miles):
        self.mileage += miles

# Создание объектов класса "Автомобиль"
car1 = Car("Toyota", "Camry", 2023)
car2 = Car("Honda", "Accord", 2024)

# Вывод информации о каждом объекте
car1.info()
car2.info()

# Изменение атрибута объекта
car1.drive(100)

# Вывод измененной информации о первом объекте
car1.info()
```

Рис. 52. Атрибуты объектов

В этом примере мы определяем класс Car, который имеет конструктор __init__, который инициализирует атрибуты объекта make, model, year и mileage. Класс Car также имеет метод info, который выводит информацию об автомобиле, и метод drive, который увеличивает пробег автомобиля на заданное количество миль. Затем создаем два объекта класса Car, car1 и car2, с разными значениями атрибутов. Вызываем метод info для каждого объекта, чтобы вывести информацию об автомобиле. Затем вызываем метод drive для объекта car1, чтобы увеличить его пробег на 100 миль, и затем вызываем метод info для этого объекта, чтобы вывести измененную информацию о нем.

Использование объектов и атрибутов классов позволяет создавать и манипулировать данными в объектах. Атрибуты объектов могут быть изменены в любое время, что делает объекты гибкими и адаптивными к различным сценариям использования.

Декораторы в Python

Это функции, которые принимают другую функцию в качестве аргумента и возвращают новую функцию, изменяющую поведение исходной функции без ее изменения. Декораторы позволяют добавлять дополнительную функциональность к функциям и методам, не изменяя их код, что делает код более гибким и модульным. С помощью декораторов можно реализовать различные задачи, например, измерение времени выполнения функции, кэширование результатов функции, логирование вызовов функции и т. д.

На рис. 53 представлен пример программы на Python, которая демонстрирует использование декоратора.

Суть работы программы заключается в следующем.

• Определяем декоратор timer, который принимает функцию в качестве аргумента и возвращает новую функцию wrapper. Функция wrapper засекает время выполнения функции, вызывает переданную функцию и выводит время выполнения.

• Определяем функцию factorial, которая вычисляет факториал числа n.

• Используем декоратор @timer, чтобы применить декоратор timer к функции factorial.

• Вызываем функцию factorial с аргументом 5.

• Декоратор timer засекает время выполнения функции factorial, вызывает ее и выводит время выполнения.

```
Определение декоратора, который выводит время выполнения функции
port time

f timer(func):
def wrapper(*args, **kwargs):
   start_time = time.time()
   result = func(*args, **kwargs)
   end_time = time.time()
   print(f"Время выполнения функции {func.__name__}: {end_time - start_time} секунд")
   return result
return wrapper
Использование декоратора для функции "factorial"
timer
f factorial(n):
if n <= 1:
   return 1
else:
   return n * factorial(n-1)
```

Рис. 53. Использование декоратора

Использование декораторов позволяет добавлять дополнительную функциональность к функциям и методам без изменения их кода. Декораторы определяются как функции, которые принимают другую функцию в качестве аргумента и возвращают новую функцию, которая обычно вызывает переданную функцию и выполняет дополнительные действия.

Декораторы – мощный инструмент в Python, который позволяет добавлять новые функциональные возможности к существующему коду без его изменения. Изучение декораторов может помочь улучшить качество и читаемость кода, а также повысить эффективность программирования в Python.

Рассмотрим еще один пример программы на Python, которая демонстрирует использование нескольких встроенных декораторов (рис. 54).

```
# Определение декоратора, который замеряет время выполнения функции
import time

def timer(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"Время выполнения функции {func.__name__}: {end_time -
start_time:.5f} секунд")
        return result
    return wrapper

# Определение класса "User"
class User:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Декоратор property, который превращает метод "is_adult" в свойство объекта
    @property
    def is_adult(self):
        return self.age >= 18

# Определение функции, которая принимает объект класса "User"
@timer
def print_user_info(user):
    print(f"Имя: {user.name}")
    print(f"Возраст: {user.age}")
    print(f"Совершеннолетие: {'да' if user.is_adult else 'нет'}")

# Создание объекта класса "User"
user = User("Иван", 25)

# Вызов функции "print_user_info" с объектом "user"
print user info(user)
```

Рис. 54. Использование нескольких встроенных декораторов

Суть работы программы заключается в следующем.

• Определяем декоратор timer, который принимает функцию в качестве аргумента и замеряет время ее выполнения. Декоратор выводит время выполнения функции в консоль и возвращает результат выполнения функции.

• Определяем класс User, который имеет два атрибута: name и age.

• Используем декоратор @property, чтобы превратить метод is_adult в свойство объекта. Свойство is_adult вычисляет, является ли пользователь совершеннолетним, основываясь на его возрасте.

• Определяем функцию print_user_info, которая принимает объект класса User и выводит информацию о пользователе в консоль. Функция использует декоратор @timer, чтобы замерить время выполнения.

• Создаем объект класса User с именем Иван и возрастом 25.

• Вызываем функцию print_user_info с объектом user.

Использование встроенных декораторов позволяет применять готовые функции для преобразования методов и свойств объектов, а также для добавления дополнительной функциональности к функциям.

Множественное наследование

Возможность класса наследовать свойства и методы сразу от нескольких родительских классов – множественное наследование. Это означает, что класс может наследовать свойства и методы не только от одного родительского класса, но и от нескольких.

В Python множественное наследование реализуется путем указания нескольких родительских классов в определении дочернего класса. При этом если несколько родительских классов имеют методы с одинаковыми именами, то при вызове такого метода у дочернего класса будет использоваться первый подходящий метод из всех родительских классов в порядке, определенном в списке наследования.

Рассмотрим пример программы на Python, демонстрирующей множественное наследование (рис. 55). В программе определены три класса: A, B и C. Класс A имеет свойство a и метод method_a. Класс B имеет свойство b и метод method_b. Класс C наследует свойства и методы от A и B и имеет свойство c и метод method_c.

При создании объекта класса C все его свойства и методы становятся доступными для использования. В данном случае мы создали объект c и вызвали его методы method_a, method_b и method_c. Когда создается объект c, Python вызывает конструктор класса C. Конструктор класса C вызывает конструкторы классов A и B, передавая им нужные аргументы. Затем конструктор класса C инициализирует свойство c. При вызове метода method_a Python ищет его в классе C. Так как метод method_a отсутствует в классе C, Python

продолжает поиск в родительском классе A. Метод method_a найден в классе A, и поэтому он вызывается. Аналогично при вызове метода method_b Python ищет его в классе C, но не находит, и затем ищет его в родительском классе B. Метод method_b найден в классе B, и он вызывается. При вызове метода method_c, Python находит его в классе C и вызывает его. Таким образом, при вызове c.method_a() будет выведено «Method A called», при вызове c.method_b() будет выведено «Method B called», а при вызове c.method_c() будет выведено «Method C called».

```
class A:
    def __init__(self, a):
        self.a = a

    def method_a(self):
        print("Method A called")

class B:
    def __init__(self, b):
        self.b = b

    def method_b(self):
        print("Method B called")

class C(A, B):
    def __init__(self, a, b, c):
        A.__init__(self, a)
        B.__init__(self, b)
        self.c = c

    def method_c(self):
        print("Method C called")

c = C("a", "b", "c")
c.method_a()
c.method_b()
c.method_c()
```

Рис. 55. Множественное наследование

Множественное наследование может быть полезным, когда необходимо создать класс, который объединяет функциональность нескольких классов.

Контрольные вопросы

1. Для решения каких задач применяется язык программирования Python?
2. Что такое переменная? От чего зависит результат при использовании операций с переменными в Python?
3. Какие циклические структуры используются в Python? Приведите примеры.
4. Какие форматы условных операторов используются в Python? Приведите примеры.
5. Какие типы коллекций используются в Python? Дайте им характеристику.
6. Что такое множество? Какие операторы используются для работы со множествами?
7. Что такое функция? Каким образом происходит объявление функции?
8. Какие функции называются вложенными? Что используется для создания замыкания?
9. Что такое модуль в Python? Что нужно определить для создания модуля?
10. Что представляют собой классы в Python? Как определить класс?
11. Для чего нужны специальные методы? Как записать специальные методы?
12. Что представляет собой наследование в Python? Какой класс называется дочерним классом, а какой – родительским классом?
13. Что представляют собой объекты в Python? Чем могут быть определены атрибуты объекта?
14. Что такое декораторы в Python? Для чего они нужны?
15. Что такое множественное наследование? Как реализуется в Python множественное наследование?

Тесты для самоконтроля

(ответы см. в приложении)

1. Как называется упорядоченная последовательность символов, где каждый символ имеет свой индекс? (*один вариант ответа*)

а) строка
б) запись
в) кортеж
г) словарь

2. Какой метод используется для замены одной подстроки на другую? (*один вариант ответа*)

а) format()
б) upper()
в) lower()
г) replace()

3. Какие условные операторы используются в Python для управления выполнения программы в зависимости от выполнения определенных условий? (*несколько вариантов ответа*)

а) if
б) else
в) elif
г) ifel

4. Как обрабатывать ошибки при вводе данных пользователем с помощью функции input() в Python? (*один вариант ответа*)

а) использовать функцию read()
б) использовать функцию get()
в) использовать конструкцию try-except
г) не обрабатывать ошибки

5. Какие символы могут содержать строки в Python? (*один вариант ответа*)

а) только буквы
б) только цифры
в) любые символы, включая буквы, цифры и специальные символы
г) только специальные символы

6. Для чего используется цикл for в Python? (*один вариант ответа*)

а) для повторения блока кода, пока определенное условие истинно

б) для перебора элементов в последовательности, такой как строка, список или кортеж

в) для создания последовательностей в Python

г) для управления циклами

7. Как проверяется условие в цикле while в Python? (*один вариант ответа*)

а) условие проверяется после каждой итерации цикла

б) условие проверяется перед каждой итерацией цикла

в) условие проверяется только один раз перед началом цикла

г) условие не проверяется

8. Какие основные типы условных операторов существуют в Python? (*один вариант ответа*)

а) if и for

б) if и else

в) while и else

г) for и else

9. Какой тип условного оператора необходимо использовать, если нужно проверить несколько условий? (*один вариант ответа*)

а) if

б) else

в) elif

г) while

10. Что такое кортежи в Python? (*один вариант ответа*)

а) неупорядоченные коллекции объектов

б) упорядоченные коллекции объектов

в) коллекции, которые могут содержать объекты только одного типа

г) коллекции, которые могут быть изменены, добавлены или удалены

11. Какой оператор используется для доступа к элементам кортежа по индексу? (*один вариант ответа*)
а) ()
б) []
в) {}
г) <>

12. Какой метод используется для доступа к значению по ключу в словаре в Python? (*один вариант ответа*)
а) index()
б) append()
в) remove()
г) get()

13. Какой оператор используется для создания множества в Python? (*один вариант ответа*)
а) {}
б) []
в) ()
г) set()

14. Какой режим открытия файла используется по умолчанию в Python? (*один вариант ответа*)
а) ‘r’
б) ‘w’
в) ‘x’
г) ‘a’

15. Какой метод используется для записи данных в файл в Python? (*один вариант ответа*)
а) read()
б) write()
в) close()
г) print()

16. Какая функция модуля os используется для получения текущей рабочей директории? (*один вариант ответа*)
а) mkdir()
б) chdir()

в) getcwd()
г) rmdir()

17. Какая функция модуля os используется для перехода в другую директорию? (*один вариант ответа*)
а) mkdir()
б) chdir()
в) getcwd()
г) rmdir()

18. Какая функция модуля shutil используется для удаления директории? (*один вариант ответа*)
а) mkdir()
б) chdir()
в) getcwd()
г) rmtree()

19. Какие элементы входят в определение функции в Python? (*несколько вариантов ответа*)
а) имя функции
б) список аргументов в скобках
в) блок кода
г) пароль

20. Что такое необязательные аргументы в функции Python? (*один вариант ответа*)
а) аргументы, которые не могут быть переданы функции
б) аргументы, которые должны быть переданы функции
в) аргументы, значения которых не заданы, но могут быть переданы функции
г) аргументы, значения которых всегда заданы

21. Какой тип переменной x находится в глобальной области видимости в программе, демонстрирующей работу вложенных функций? (*один вариант ответа*)
а) локальная
б) глобальная
в) нелокальная
г) константа

22. Как определяются вложенные функции в Python? (*один вариант ответа*)

а) с помощью ключевого слова nested

б) с помощью ключевого слова inner

в) с помощью ключевого слова def, внутри другой функции

г) с помощью ключевого слова func, внутри другой функции

23. Что представляет собой модуль в языке программирования Python? (*один вариант ответа*)

а) файл с расширением .py, содержащий определения функций, классов, переменных и других объектов

б) файл с расширением .exe, содержащий исполняемый код программы

в) файл с расширением .txt, содержащий комментарии и описания кода

г) файл с расширением .html, содержащий веб-страницу

24. Что такое исключения в Python? (*один вариант ответа*)

а) ошибки, возникающие во время выполнения программы

б) уведомления об успешном выполнении программы

в) директивы для компилятора

г) ошибки, возникающие при компиляции программы

25. Какое ключевое слово используется для вызова исключения в Python? (*один вариант ответа*)

а) catch

б) raise

в) throw

г) handle

26. Для чего могут быть использованы атрибуты объектов в Python? (*один вариант ответа*)

а) для хранения информации о состоянии объекта

б) для изменения кода класса

в) для создания новых типов данных

г) для изменения встроенных функций Python

27. Что такое декораторы в Python? (*один вариант ответа*)

а) это функции, которые принимают другую функцию в качестве аргумента и возвращают новую функцию, изменяющую поведение исходной функции без ее изменения

б) это функции, которые изменяют поведение исходной функции, изменяя ее код

в) это функции, которые добавляют новые возможности к языку Python

г) это функции, которые удаляют ненужный код из существующих функций

28. Какие ключевые слова используются для определения наследования классов в языке Python? (*один вариант ответа*)

а) extends, implements

б) extends, inherits

в) inherits, implements

г) None of the above

29. Что такое метод класса в языке Python? (*несколько вариантов ответа*)

а) функция, определенная внутри класса

б) функция, определенная вне класса

в) метод может иметь доступ к атрибутам класса

г) метод может иметь доступ к атрибутам экземпляра класса

30. Какие методы вызываются автоматически при создании экземпляра класса? (*несколько вариантов ответа*)

а) init

б) str

в) new

г) del

Практические задания по темам модуля 1

Тема «Базовые структуры языка Python и их обработка»

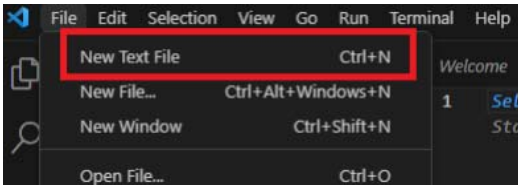
Задание 1. Создайте файл с расширением «.py» и напишите программу, которая выводит сообщение «Привет, Python!».

Ход работы

1. Изучите методические рекомендации.

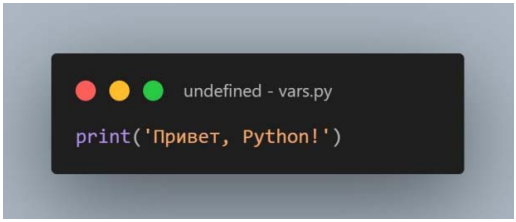
2. Выполните установку интерпретатора Python и напишите программу, выводящую на консоль текст «Привет, Python!». Для этого используйте один из способов взаимодействия.

3. Создайте файл с расширением «.py». Для этого в Visual Studio Code нажмите в меню File → New Text File.



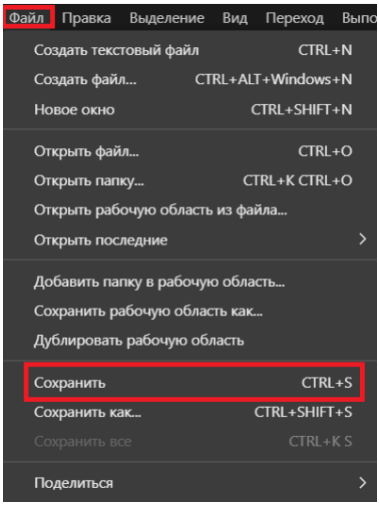
4. В появившемся окне с редактором нажмите ссылку Select a language и в поисковой строке напишите Python, выберите Python.

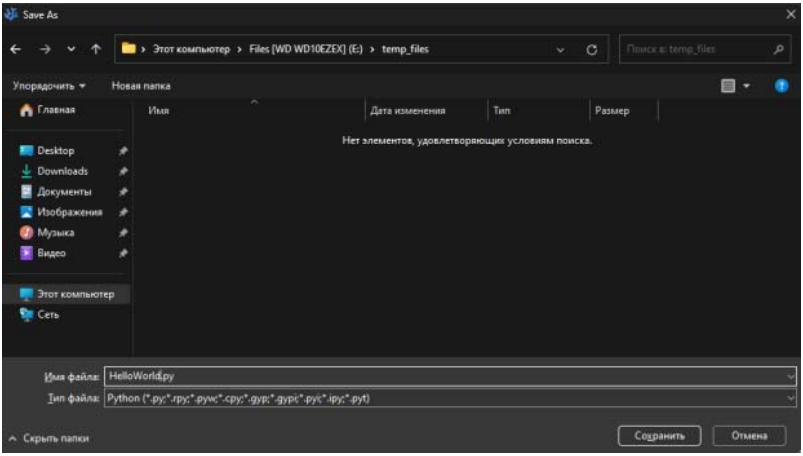
5. В Python-файле, используя текстовый редактор, наберите следующую программу:



Функция print() используется для вывода информации на экран. В данном случае она выводит строку «Hello, World».

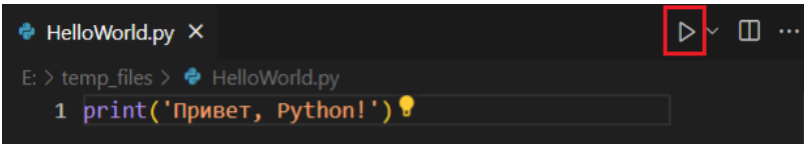
6. Сохраните файл под именем HelloWorld.py, выбрав пункт меню «Файл» – > «Сохранить». Лучше всего создать для файла отдельную папку в удобном для вас расположении.





7. Запустите сохраненный файл при помощи командной строки командой **py HelloWorld.py** или **python HelloWorld.py**

Это можно сделать также из-под редактора Visual Studio Code, нажав на кнопку запуска в правом верхнем углу.

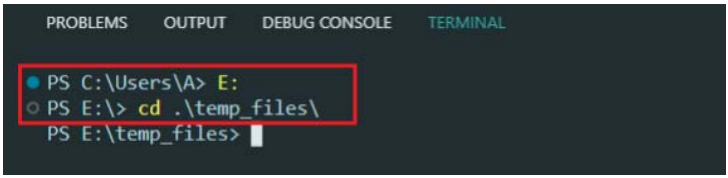


Так вы увидите результат работы программы во встроенном терминале.

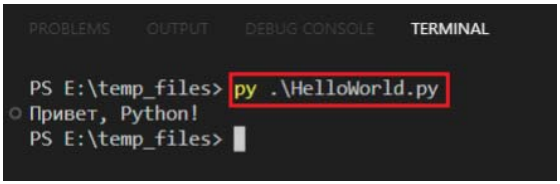
Или нажмите в левом нижнем углу на иконки вывода ошибок и предупреждений.

8. В открывшемся окне выберите вкладку «ТЕРМИНАЛ».

9. Перейдите в папку с расположением файла при помощи команды: **cd <путь до папки с файлом>**. Например, **cd E:\temp_files**



10. Запустите Python-файл при помощи команды: **py <название_файла>.py** или **python <название_файла>.py**



В результате запуска вы получите строку «Hello, World», прописанную в содержимом файла HelloWorld.py.

Методические указания

В Python операция вывода на экран тесно связана с функцией print(). Эта функция является одним из наиболее часто используемых средств для вывода данных в стандартный вывод (обычно это консоль).

Функция print() выводит заданные объекты в стандартный поток вывода (консоль) или в другой текстовый поток. Она преобразует объекты в строки (используя встроенную функцию str()) и выводит их.

Функция может принимать несколько параметров:

– *objects. Произвольное количество объектов, которые нужно вывести. Объекты будут преобразованы в строки и выведены;

– sep. Разделитель между объектами при выводе. Разделителем по умолчанию является пробел;

– end. Строка, которая будет добавлена после последнего объекта в выводе; по умолчанию символ новой строки (\n);

– file. Объект файла, в который должен быть направлен вывод. По умолчанию – стандартный поток вывода sys.stdout;

– flush. Логическое значение, указывающее, следует ли принудительно «сбрасывать» поток вывода. Если True, поток будет сброшен сразу после вывода.

Существуют два способа взаимодействия с языком Python:

– локальный способ. Предполагает установку Python на свой компьютер. Пользователь может создавать и запускать программы на Python, используя установленное ПО. Данный подход предоставляет полный контроль над окружением и доступ к различным библиотекам и инструментам;

– использование онлайн-сервисов. В качестве альтернативного способа предлагается использование онлайн-сервиса Replit.

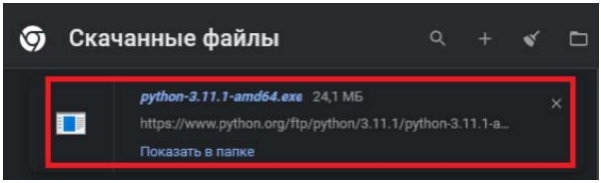
Рассмотрим каждый из этих способов.

Основной способ. Локальная установка

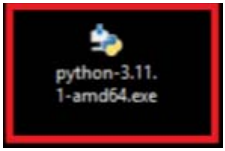
Локальный способ подразумевает установку интерпретатора Python, а также установку и настройку редактора кода. В качестве редактора кода предлагается использование Visual Studio Code.

Скачайте и установите Python 3 для операционной системы семейства Windows

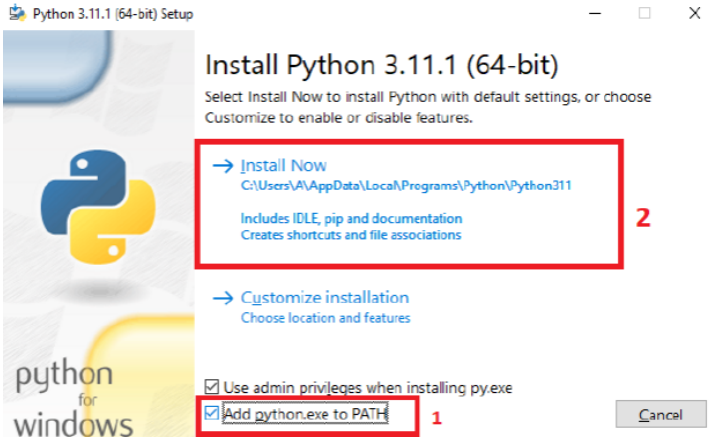
https://www.python.org/ftp/python/3.11.1/python-3.11.1-amd64.exe



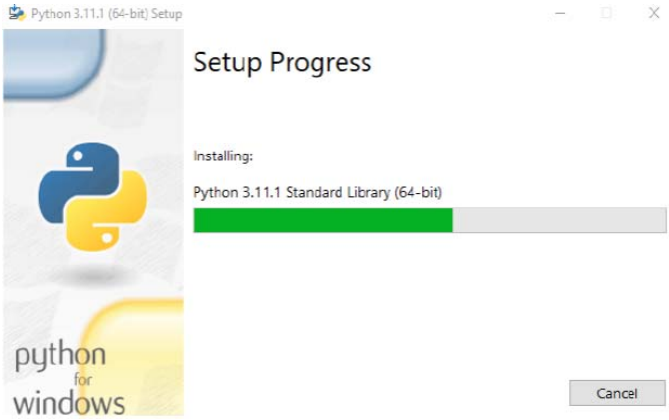
После того как был скачан установочный файл, запустите его двойным кликом мыши.



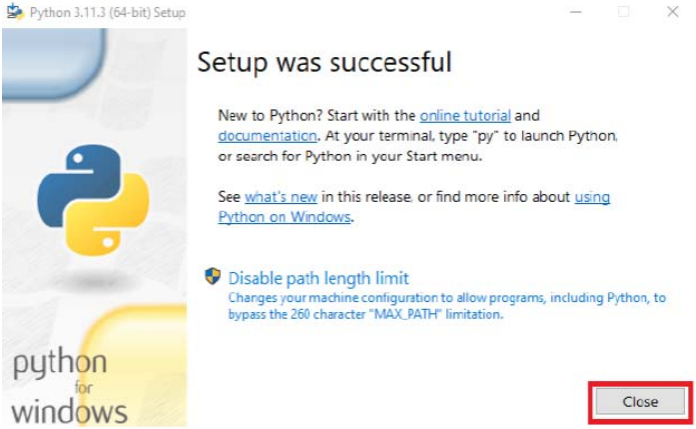
Отметьте галочкой пункт Add python.exe to PATH и нажмите Install Now.



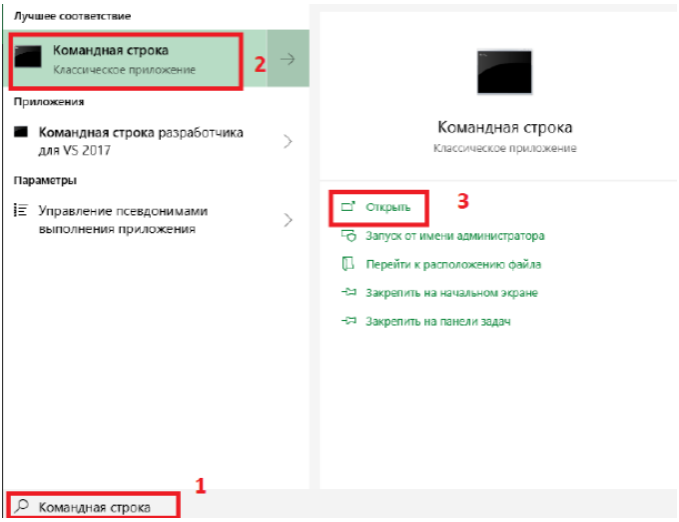
Дождитесь окончания установки.



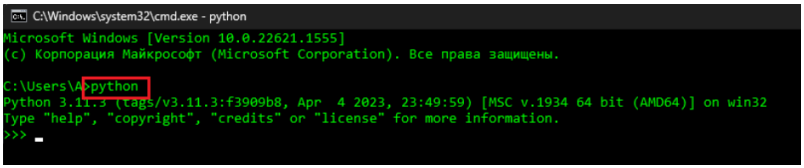
Как только Python установится, нажмите на кнопку Close, чтобы закрыть окно. Перед этим убедитесь, что в окне появилась надпись Setup was successful.



Проверьте запуск Python. Для этого откройте меню «Пуск», напишите в поиске «Командная строка» и откройте командную строку.

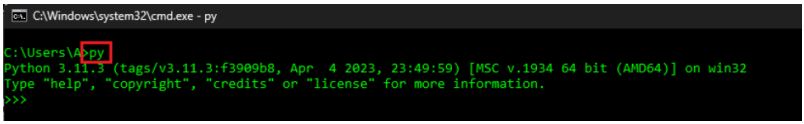


Напишите в командной строке команду **python**, чтобы убедиться, что Python нужной версии установлен.



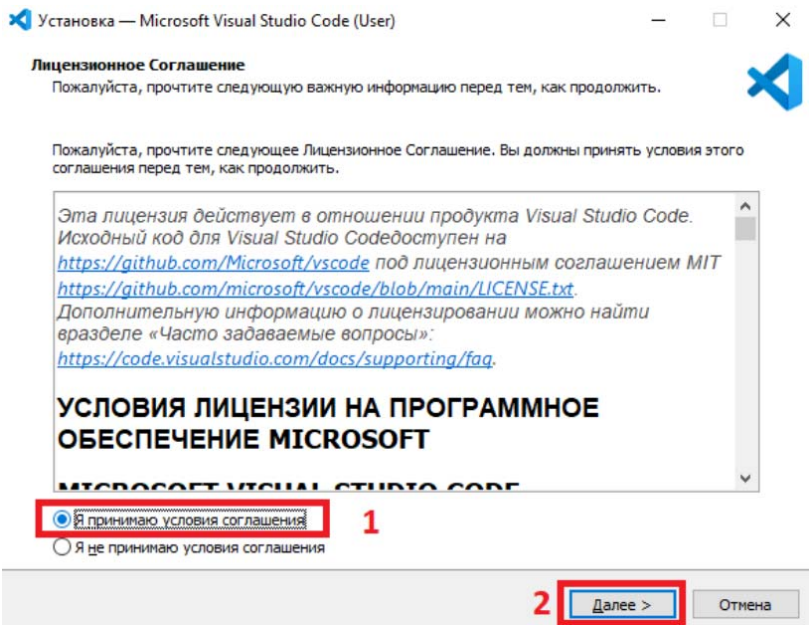
Командную строку можно закрыть. Python успешно установлен.

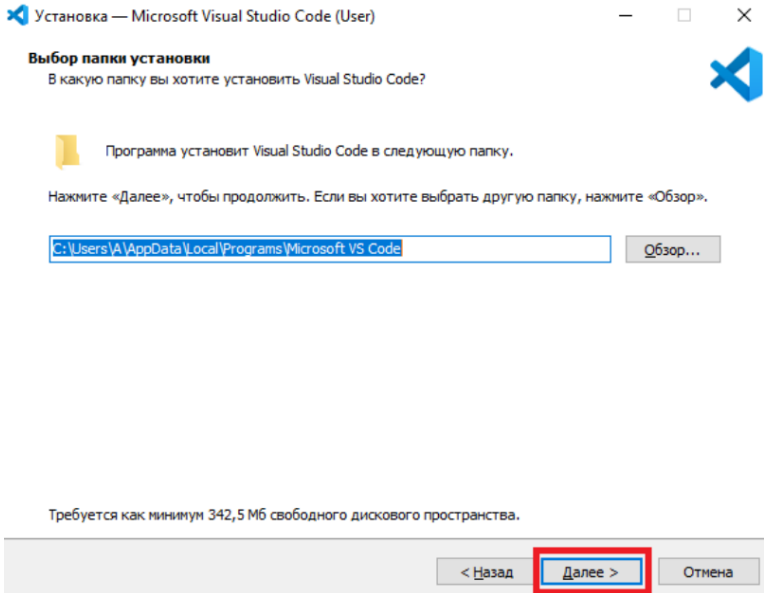
Альтернативным вариантом вызова интерпретатора является команда **py**.

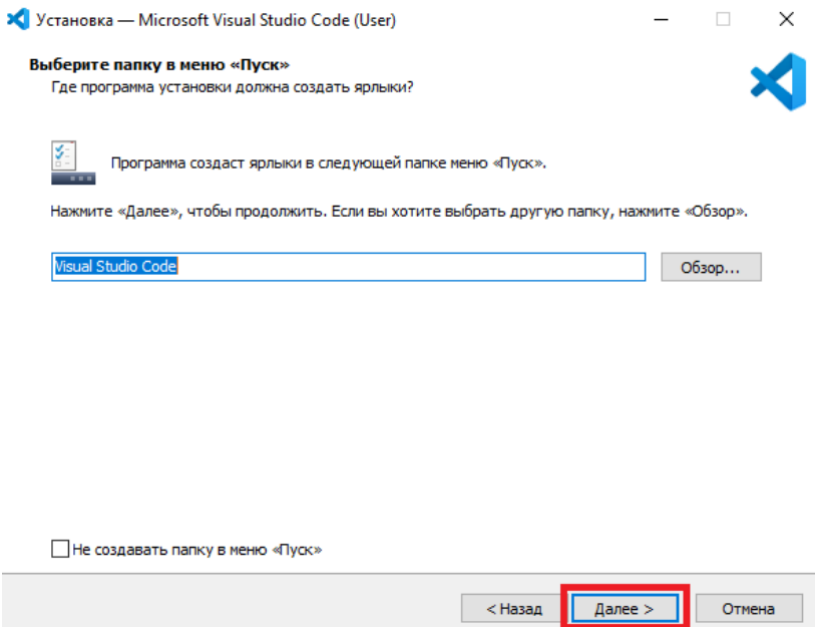


Скачайте и установите редактор кода Microsoft Visual Studio Code.

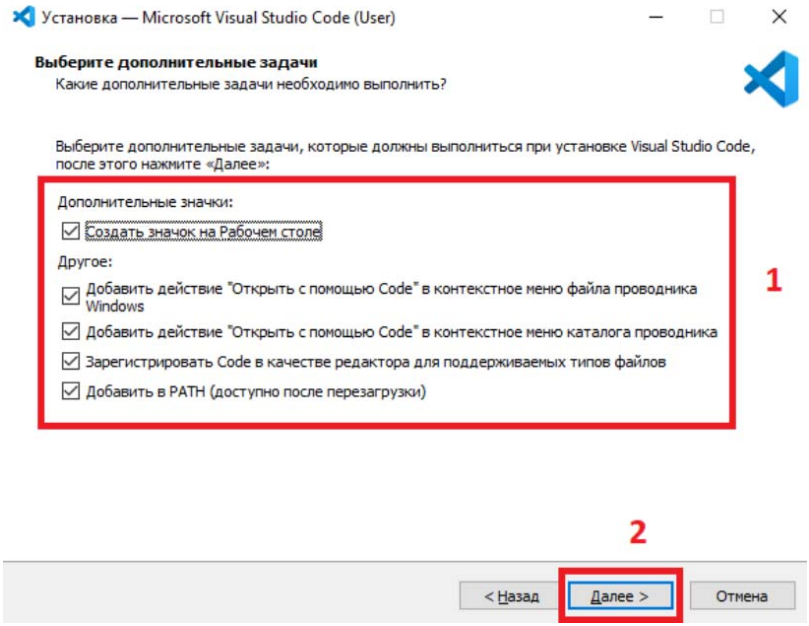
Для Windows: (https://code.visualstudio.com/sha/download?build=stable&os=win32-x64-user)



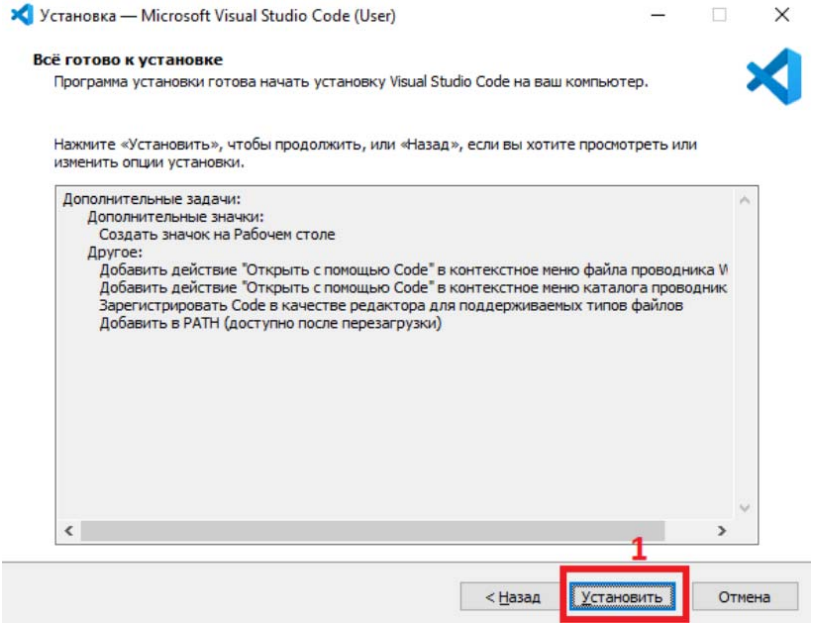




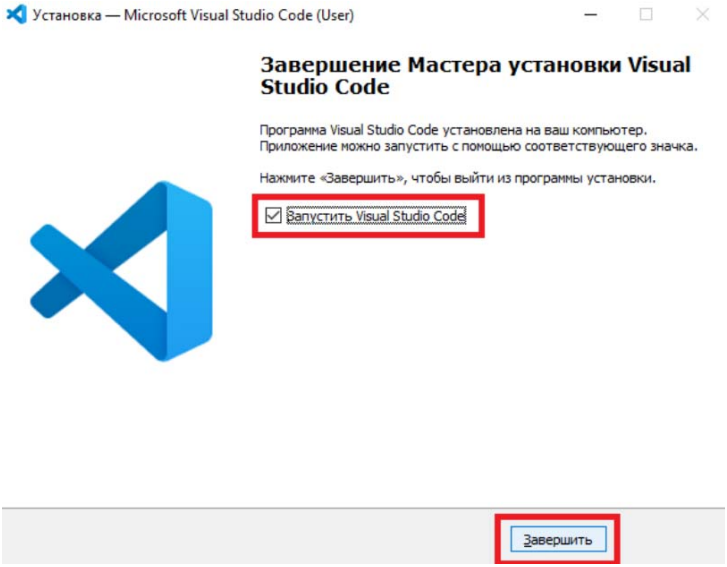
Отметьте галочками все пункты.



Нажмите на кнопку «Установить».

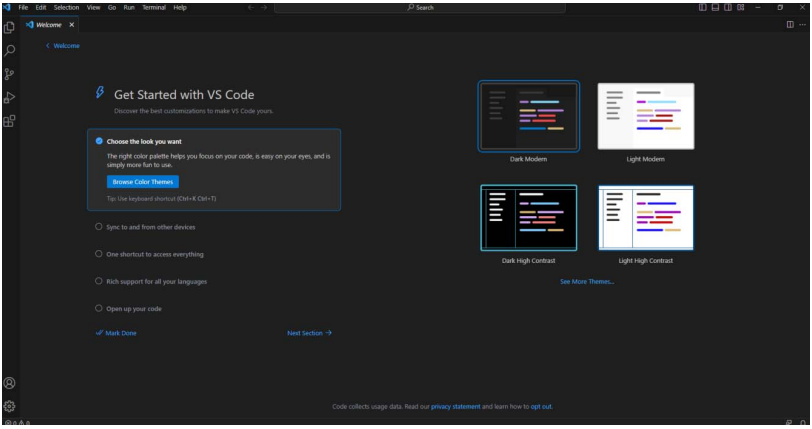


Запустите установленный редактор.



Для Ubuntu-based ОС. Скачайте установочный пакет по ссылке https://go.microsoft.com/fwlink/?LinkID=760868 и установите его командой sudo apt install ./<file>.deb. Вместо <file> вставьте название скачанного файла.

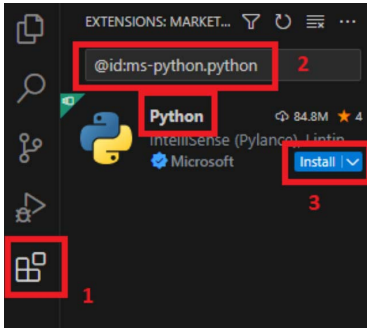
После установки откроется интерфейс Visual Studio Code.



Установите из магазина расширений расширение Python.

Для этого слева в меню выберите иконку с расширениями (1).

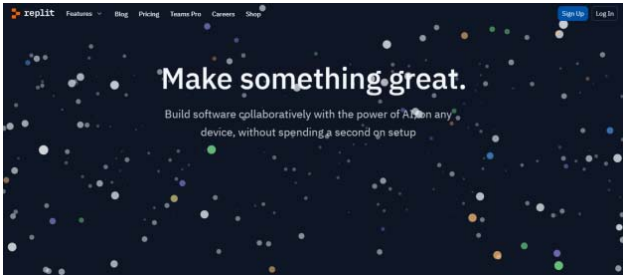
В строке поиска (2) напишите следующий запрос @id:ms-python.python.



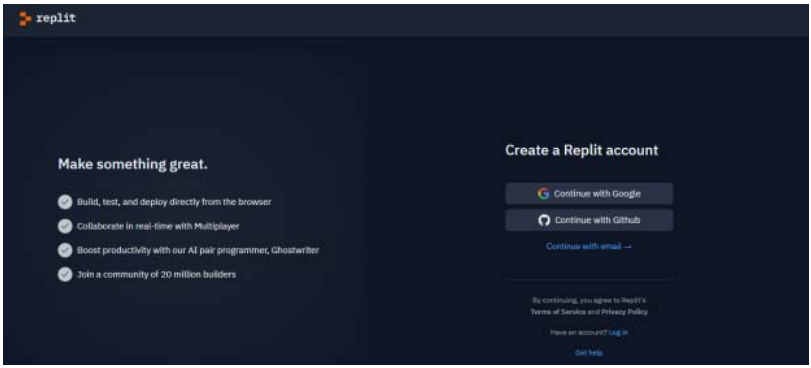
У необходимого расширения нажмите на кнопку Install (3).

Альтернативный способ. Использование online-сервиса

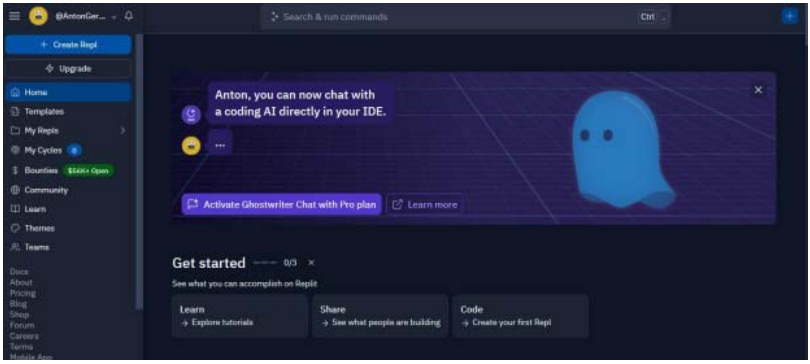
В качестве среды разработки используем сервис https://replit.com/



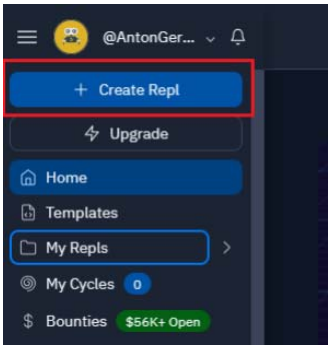
Необходимо зарегистрироваться на сайте (https://replit.com/signup).



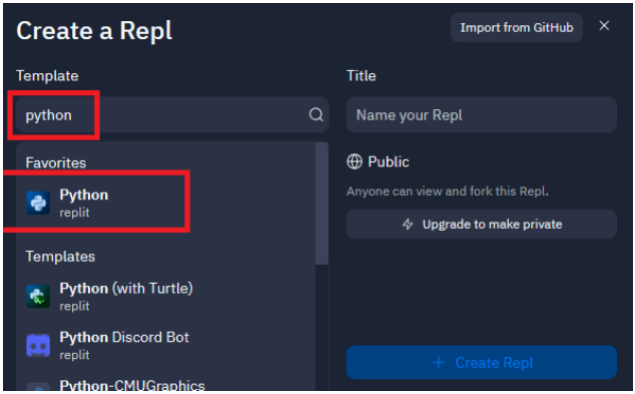
После регистрации появляется интерфейс сервиса.



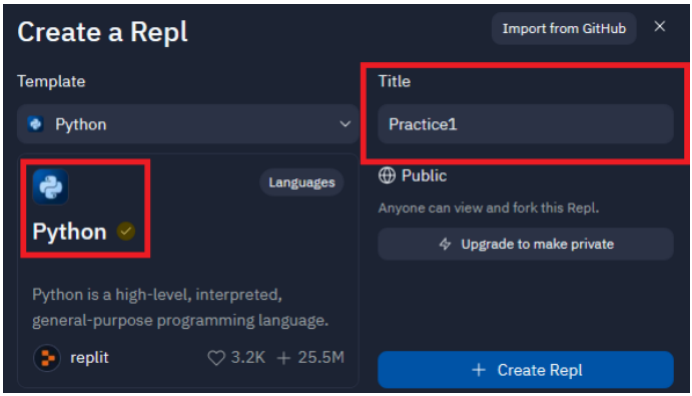
Необходимо нажать на кнопку Create Repl.



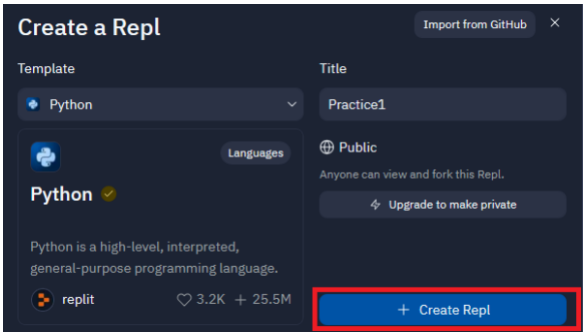
В открывшемся окне выберем Python.



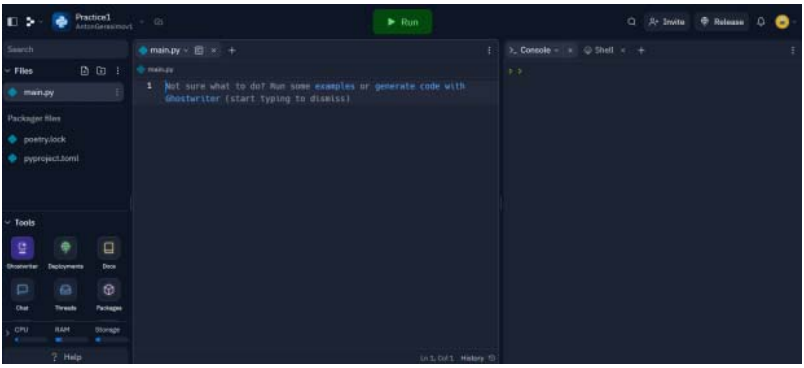
Затем в поле Title необходимо добавить название для создаваемого окружения.



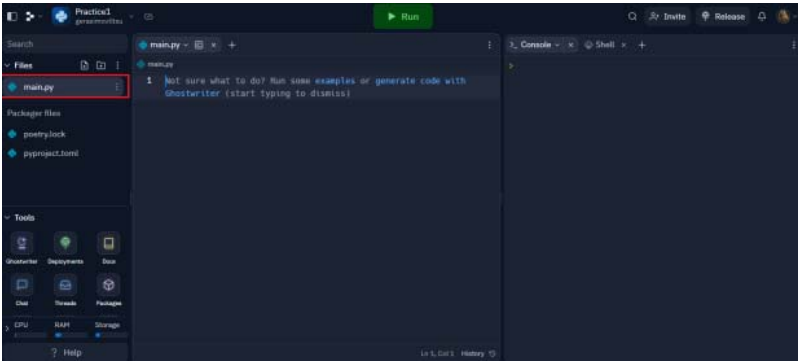
Теперь необходимо создать окружение, нажав на кнопку Create Repl.



Появляется редактор кода.



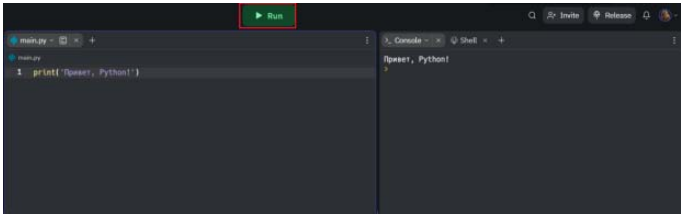
В редакторе кода должен открыться файл main.py. Если этого не произошло, выберите его самостоятельно в левой колонке редактора.



Напишите в файле main.py код для вывода текста «Hello, World!»:

```
print(«Привет, Python!»)
```

и запустите, нажав кнопку Run.



Появилась надпись «Привет, Python!». Это значит, что код работает правильно.

Задание 2. Напишите программу, которая запрашивает у пользователя два числа и выводит на экран их сумму, разность, произведение и частное.

Ход работы

1. Изучите методические рекомендации.

2. Создайте переменные, которые будут считывать с консоли два числа, а также операции.

```
# Запрашиваем у пользователя два числа
num1 = int(input("Введите первое число: "))
num2 = int(input("Введите второе число: "))
```

3. Выполните базовые математические операции с представленными числами и запишите результаты вычислений в отдельные переменные.

```
# Выполняем операции
summa = num1 + num2
diff = num1 - num2
prod = num1 * num2
quot = num1 / num2
```

4. Выведите результаты вычислений, записанные в соответствующие переменные.

```
# Выводим результаты
print("Сумма:", summa)
print("Разность:", diff)
print("Произведение:", prod)
print("Частное:", quot)
```

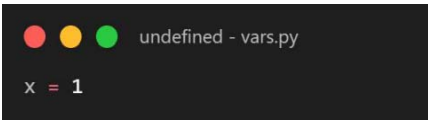
Методические указания

В программировании переменные играют важную роль, так как они предназначены для хранения данных. В языке Python, как и во многих других языках программирования, наименование переменной должно начинаться с буквы алфавита или символа подчеркивания, а далее могут следовать алфавитно-цифровые символы и знак подчеркивания. Важно помнить, что имя переменной не может совпадать с одним из ключевых слов языка Python. Несмотря на то, что ключевых слов в языке Python немного, их стоит запомнить, чтобы избежать ошибок при написании кода.

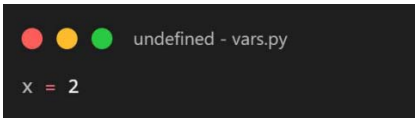
False	await	Else	Import	pass
None	Break	Except	In	Raise
True	Class	Finaly	Is	Return
And	Continue	For	Lambda	Try
As	Def	From	Nonlocal	While
Assert	Del	Global	Not	With
Async	Elif	If	Or	yield

Для того чтобы создать переменную в Python, нужно выбрать уникальное имя переменной, начинающееся с буквы алфавита или символа подчеркивания, и затем присвоить ей значение с помощью оператора присваивания «=».

Например, чтобы создать переменную x и присвоить ей значение 1, необходимо написать следующую строку кода:



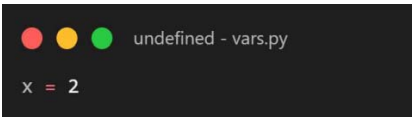
После этого значение 1 будет сохранено в переменной x и его можно будет использовать в дальнейшем в программе. Если же переменная уже была создана и нужно изменить ее значение, достаточно присвоить ей новое значение:



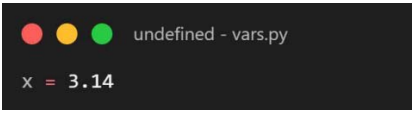
Теперь значение переменной x будет равно 2. Важно помнить, что в Python тип переменной определяется автоматически в зависимости от значения, которое ей было присвоено.

В Python есть несколько встроенных типов данных для переменных.

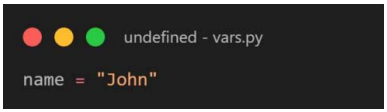
Целочисленный тип данных (int) – используется для хранения целых чисел, например:



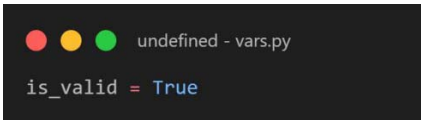
Вещественный тип данных (float) – используется для хранения дробных чисел, например:



Строковый тип данных (str) – используется для хранения текстовых значений, заключенных в кавычки, например:



Логический тип данных (bool) – используется для хранения логических значений True (истина) или False (ложь), например:



В Python можно выполнять различные операции с переменными, в зависимости от их типа данных. Рассмотрим основные операции.

Операции присваивания позволяют присваивать значения переменным.

Пример:

```
x = 5 # присваивание целочисленного значения переменной x
name = "John" # присваивание текстового значения переменной name
is_valid = True # присваивание логического значения переменной is_valid
```

Операции арифметических вычислений позволяют выполнять арифметические операции над числовыми переменными. Пример:

```
b = 3
c = a + b # Сложение переменных a и b. Результат: 8
d = a - b # Вычитание переменных a и b. Результат: 2
e = a / b # Перемножение переменных a и b. Результат: 15
f = a / b # Деление переменных a и b. Результат 1.(6)
g = a // b # Получение целой части деления. Результат: 1
h = a % b # Остаток от деления. Результат: 2
i = -a # Смена знака числа. Результат: -5
j = a ** b # Возведение в степень числа a на число b. Результат: 125
```

Операции сравнения позволяют сравнивать значения переменных. Результатом такой операции является булево значение True или False.

Примеры:

```
a = 5
b = 3
c = a == b # Сравнение переменных a и b на равенство. Результат: False
d = a > b # Сравнение переменных a и b на "больше". Результат: True
e = a < b # Сравнение переменных a и b на "меньше". Результат: False
f = a >= b # Сравнение переменных a и b на "больше или равно". Результат: True
h = a <= b # Сравнение переменных a и b на "меньше или равно". Результат: False
```

Операции конкатенации позволяют объединять значения строковых переменных. Пример:

```
name = "Имя"
surname = "Фамилия"
full_name = name + " " + surname # Объединение значений переменных name и surname
# Результат: "Имя Фамилия"
```

В Python для ввода и вывода данных используются стандартные функции input() и print().

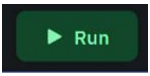
Функция input() позволяет получить данные от пользователя через консольное окно. Она принимает один необязательный аргумент – строку приглашения, которая выводится пользователю перед вводом данных.

Например:

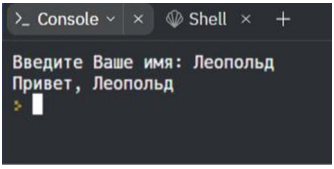
```
name = input("Введите Ваше имя: ")
```

В этом примере пользователю будет выведено сообщение «Введите ваше имя: », после чего он сможет ввести свое имя через консоль. Введенное пользователем значение будет сохранено в переменной name.

Запустим код, нажав на кнопку Run.



Теперь программа выводит текст «Привет, {имя}».

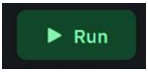


В этом примере функция print() выведет на экран строку «Привет» и значение переменной name, разделенные пробелом.

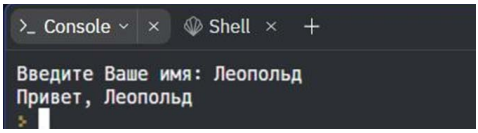
Кроме того, функция print() поддерживает различные способы форматирования вывода, например, использование специальных символов или метода форматирования строк f-strings. Например:

```
name = input("Введите Ваше имя: ")
print(f"Привет, {name}")
```

Запустим код, нажав на кнопку Run.



Программа выведет следующий текст:



В этом примере функция print() использует f-строку для вывода значения переменной name в дополнительной строке формата. Это позволяет более гибко управлять форматированием вывода.

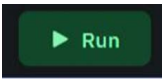
Для считывания числовых значений и логических значений (булевых) в Python используется функция input(). Однако при считывании таких значений необходимо учитывать их тип данных, чтобы правильно обработать их в дальнейшем.

Для считывания целочисленных значений в Python можно использовать функцию int(), которая преобразует введенную строку в целое число.

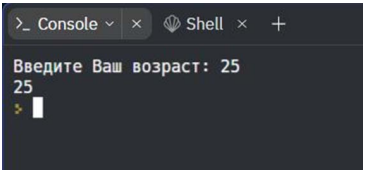
Например:

```
age = int(input("Введите Ваш возраст: "))
print(age)
```

Запустим код, нажав на кнопку Run.



Введите возраст в первую строку.



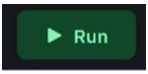
В этом примере функция input() считывает введенную пользователем строку, которая затем преобразуется в целое число с помощью функции int().

Результат сохраняется в переменную age.

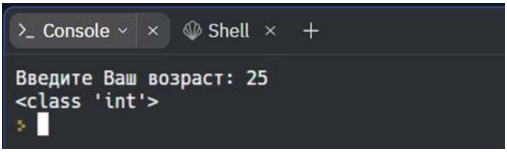
Для проверки типа значения изменим код, обернув переменную age в функцию type() в выводе print().

```
age = int(input("Введите Ваш возраст: "))
print(type(age))
```

Запустим код, нажав на кнопку Run.



В выводе увидим, что появилась строка <class ‘int’>.

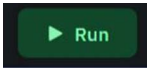


Это означает, что переменная age является типом int(), то есть содержит целочисленное значение.

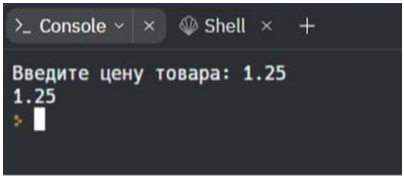
Для считывания вещественных значений в Python можно использовать функцию float(), которая преобразует введенную строку в число с плавающей точкой. Например:

```
value = float(input("Введите цену товара: "))
print(value)
```

Запустим код, нажав на кнопку Run.

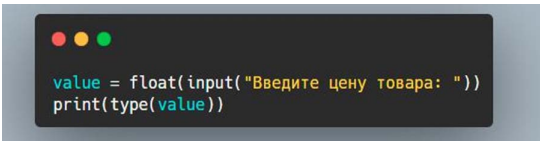


Вводить значения типа float необходимо через символ «.»



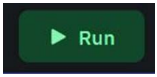
В этом примере функция input() считывает введенную пользователем строку, которая затем преобразуется в число с плавающей точкой с помощью функции float(). Результат сохраняется в переменную value.

Для проверки типа значения изменим код, обернув переменную value в функцию type() в выводе print().

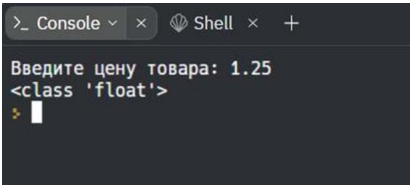


```
value = float(input("Введите цену товара: "))
print(type(value))
```

Запустим код, нажав на кнопку Run.



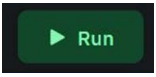
В выводе увидим, что появилась строка <class ‘float’>. Это означает, что переменная value стала типа float.



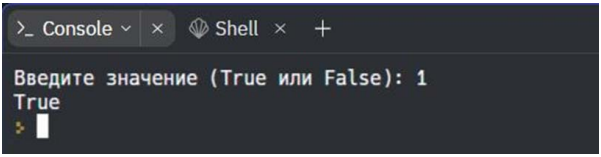
Для считывания логических значений (булевых) в Python можно использовать функцию bool(), которая преобразует введенную строку в логическое значение True или False. Например:

```
is_valid = bool(input("Введите значение (True или False или 1 или 0)"))
print(is_valid)
```

Запустим код, нажав на кнопку Run.



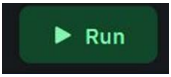
В данном случае может вводиться только True или False, а также 1 или 0.



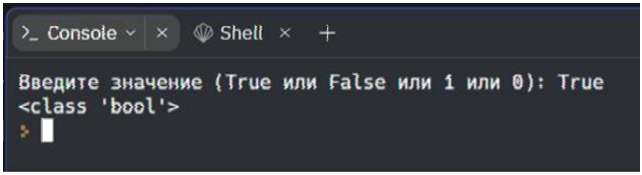
Для проверки типа значения изменим код, обернув переменную is_valid в функцию type() в выводе print().

```
is_valid = bool(input("Введите значение (True или False или 1 или 0)"))
print(type(is_valid))
```

Запустим код, нажав на кнопку Run.

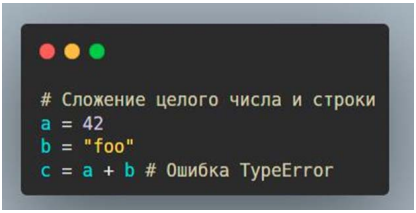


В выводе увидим, что появилась строка <class ‘bool’>.



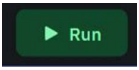
В Python нельзя складывать между собой переменные разных типов данных. Например, нельзя складывать строку и число.

Изменим код.



```
# Сложение целого числа и строки
a = 42
b = "foo"
c = a + b # Ошибка TypeError
```

Запустим, нажав на кнопку Run.

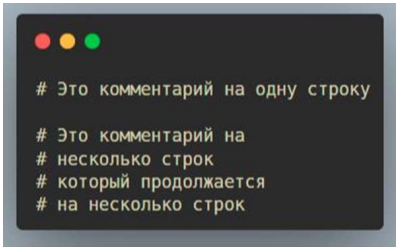


Если попытаться выполнить такую операцию, Python выдаст ошибку TypeError.

Стоит также упомянуть комментарии. В Python комментарий – это текст, который игнорируется интерпретатором Python при выполнении программы. Комментарии используются для описания кода, объяснения его работы, документирования функций, классов и других целей, которые помогают сделать код более понятным и читабельным.

Комментарии в Python начинаются со знака # и продолжаются до конца строки. Все символы после знака # до конца строки будут игнорироваться интерпретатором.

Например, вот как выглядят комментарии в Python.



```
# Это комментарий на одну строку

# Это комментарий на
# несколько строк
# который продолжается
# на несколько строк
```

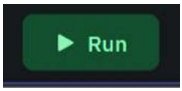
Многострочные комментарии в Python – это особый тип комментариев, который позволяет вставлять комментарии на несколько строк. Они используются для документирования кода и описания его функциональности.

В Python многострочные комментарии создаются с помощью тройных кавычек («» или ‘’’). Все, что находится между двумя наборами тройных кавычек, будет считаться комментарием и проигнорировано интерпретатором Python при выполнении программы.

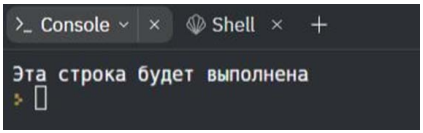
Ниже представлен пример использования многострочного комментария в Python:

```
"""
Это многострочный комментарий.
Он может содержать несколько строк текста,
которые будут проигнорированы интерпретатором Python.
"""

print("Эта строка будет выполнена")  # это комментарий на одну строку
```

Запустим файл, нажав кнопку Run.



В результате получим следующий вывод в консоли:



Строки, циклы и условия – это базовые концепции языка программирования Python, которые используются для обработки данных и управления логикой программы.

Строки

Строка (или строковый тип данных) – это последовательность символов, которая может содержать любые символы, включая буквы, цифры, знаки препинания и пробелы. В программировании строки широко используются для хранения и обработки текстовой информации.

В языке программирования Python строки являются неизменяемыми объектами, то есть после создания строки нельзя изменить ее содержимое. Строки в Python могут быть созданы с помощью одинарных (‘), двойных («) или тройных (‘’’ или «»») кавычек.

Например, следующий код создает строку, содержащую текст «Hello, World!», и записывает её в переменную s:

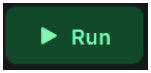
```
# Объявление строки
s = "Hello, World!"
```

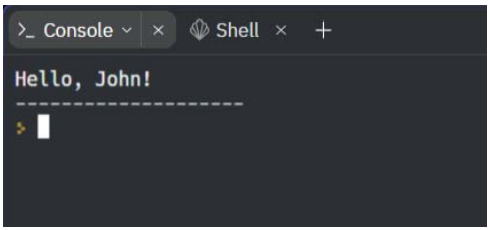
Рассмотрим теперь возможности работы с текстом. Допустим, что в текст нашей программы Hello, world нам нужно вставить одинарную кавычку.

Строки в Python поддерживают множество операций и методов, которые позволяют работать с ними. Например, можно объединять строки с помощью оператора «+» или умножать строку на целое число, чтобы создать новую строку, содержащую несколько копий исходной строки.

```
greeting = "Hello"
name = "John"
message = greeting + ", " + name + "!"
print(message) # "Hello, John!"

line = "-" * 20
print(line) # 20 раз выведет символ "-"
```

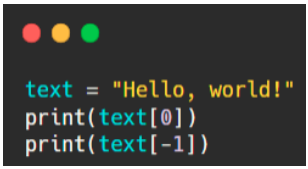
Запустим, нажав на кнопку Run.



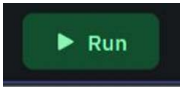


Строки также поддерживают индексацию, то есть можно получить доступ к отдельным символам строки по их индексу (начинающемуся с 0).

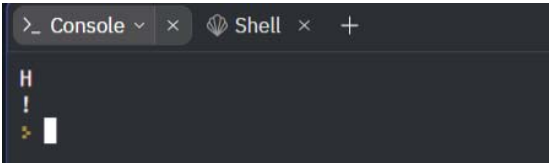
Кроме того, можно использовать срезы (slice), чтобы получить подстроку из строки.



Запустим, нажав на кнопку Run.



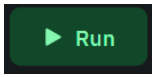
В результате получаем первый и последний символ строки.

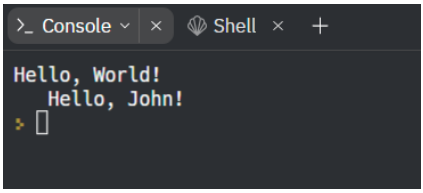


Строки в Python также имеют множество методов, которые позволяют выполнять различные операции с ними. Например, методы upper() и lower() позволяют преобразовывать строку в верхний или нижний регистр соответственно, метод strip() удаляет пробелы с начала и конца строки, метод replace() заменяет часть строки на другую строку и т. д.

```
text = "    Hello, world!   "
print(text.strip()) # Hello, World! (без пробелов)
print(text.replace("world", "John")) # Hello, John (с пробелами)
```

Запустим, нажав на кнопку Run.





Стоит заметить, что пробелы во второй строке остались.

Применение методов осуществляется через точку после строки или переменной.

```
text = "    Hello, world!   "
print(text.strip()) # Hello, World! (без пробелов)
print(text.replace("world", "John")) # Hello, John (с пробелами)
```

Выделяют следующие методы для работы со строками

Функция или метод	Назначение
S = ‘str’; S = “str”; S = ‘’’str’’’; S = «»»str»»»	Литералы строк
S = «s\np\ta\nbbb»	Экранированные последовательности
S = r»C:\temp\new»	Неформатированные строки (подавляют экранирование)
S = b»byte»	Строка байтов
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза

Функция или метод	Назначение
len(S)	Длина строки
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или –1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или –1
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace(шаблон, замена[, maxcount])	Замена шаблона на замену. maxcount ограничивает количество замен
S.split(символ)	Разбиение строки по разделителю
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.isspace()	Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы (‘\f’), «новая строка» (‘\n’), «перевод каретки» (‘\r’), «горизонтальная табуляция» (‘\t’) и «вертикальная табуляция» (‘\v’))
S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Начинается ли строка S с шаблона str

Функция или метод	Назначение
S.endswith(str)	Заканчивается ли строка S шаблоном str
S.join(список)	Сборка строки из списка с разделителем S
ord(символ)	Символ в его код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные в нижний
S.center(width, [fill])	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)
S.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
S.expandtabs([tabsize])	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам
S.lstrip([chars])	Удаление пробельных символов в начале строки
S.rstrip([chars])	Удаление пробельных символов в конце строки
S.strip([chars])	Удаление пробельных символов в начале и в конце строки
S.partition(шаблон)	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустые строки
S.rpartition(sep)	Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустые строки, а затем саму строку

Функция или метод	Назначение
S.swapcase()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
S.zfill(width)	Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями
S.ljust(width, fillchar=» «)	Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=» «)	Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar
S.format(*args, **kwargs)	Форматирование строки

Задание 3. Напишите программу, которая принимает на вход список целых чисел и выводит на экран только четные числа из списка.

Ход работы

1. Изучите методические рекомендации.

2. Создайте список my_list, который содержит последовательность целых чисел от 1 до 10.

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3. Инициируйте цикл for, который будет итерировать по каждому элементу в списке my_list, присваивая текущее значение элемента переменной num. В теле цикла используйте условный оператор if для проверки, является ли текущее число num чётным. Для этого используйте оператор деления по модулю %, который возвращает остаток от деления num на 2.

```
for num in my_list:
  if num % 2 == 0:
    print(num)
```

Методические указания

Циклы

В Python есть два основных вида циклов: цикл for и цикл while.

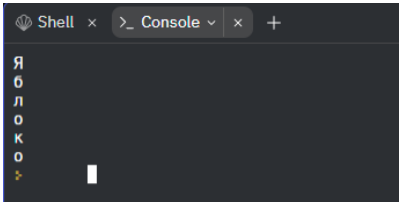
Цикл for используется для перебора элементов в итерируемом объекте, таком как список, кортеж или строка. Синтаксис цикла for в Python выглядит следующим образом:

```
for <переменная> in <итерируемый объект>:
    <тело цикла>
```

Например, следующий код выводит все элементы списка:

```
for char in "Яблоко":
    print(char)
```

В результате выполнения кода будут построчно выведены символы из строки «Яблоко».



Цикл while используется для выполнения блока кода до тех пор, пока указанное условие истинно. Синтаксис цикла while в Python выглядит следующим образом:

```
while <условие>:
    <тело цикла>
```

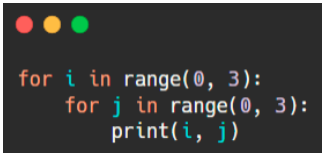
Например, следующий код выводит числа от 1 до 5 с помощью цикла while:

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

Вложенные циклы

В Python можно использовать вложенные циклы – циклы, которые находятся внутри других циклов. Вложенные циклы используются, когда требуется обработать данные, которые имеют многомерную структуру, например, или матрицы.

Синтаксис вложенных циклов очень простой – в теле одного цикла можно разместить другой цикл:



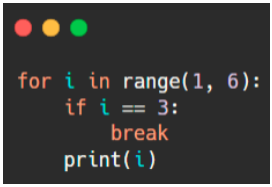
```
for i in range(0, 3):
    for j in range(0, 3):
        print(i, j)
```

В этом примере выводятся два столбца из 9 элементов, и для вывода всех его элементов используются вложенные циклы. Внешний цикл перебирает строки (i), а внутренний цикл – столбцы (j) массива.

Операторы break и continue

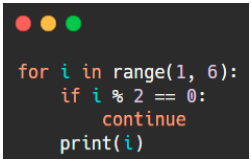
Внутри циклов можно использовать операторы break и continue.

Оператор break используется для прерывания выполнения цикла, если выполняется определенное условие. Например, следующий код выводит числа от 1 до 5, но прерывает цикл, если число 3:



```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

Оператор continue используется для пропуска текущей итерации цикла и перехода к следующей. Например, следующий код выводит только нечетные числа от 1 до 5:



```
for i in range(1, 6):
    if i % 2 == 0:
        continue
    print(i)
```

Условия

Условные конструкции в Python используются для выполнения различных действий в зависимости от выполнения определенных условий. В Python есть два основных оператора условия: if и else.

Оператор if

Оператор if позволяет выполнить определенный блок кода, если определенное условие истинно. Синтаксис оператора if выглядит следующим образом:

```
if <условие>:
    <блок кода>
```

Оператор if-else

Оператор if-else позволяет выполнить один блок кода, если условие истинно, и другой блок кода, если условие ложно. Синтаксис оператора if-else выглядит следующим образом:

```
if <условие>:
    <блок кода, если условие истинно>
else:
    <блок кода, если условие ложно>
```

Оператор if-elif-else

Оператор if-elif-else позволяет проверить несколько условий и выполнить соответствующий блок кода для первого истинного условия. Синтаксис оператора if-elif-else выглядит следующим образом:

```
if <условие 1>:
    <блок кода, если условие 1 истинно>
elif <условие 2>:
    <блок кода, если условие 2 истинно>
else:
    <блок кода, если все условия ложны>
```

Коллекции объектов в Python – это структуры данных, которые позволяют хранить и организовывать группы объектов. В Python

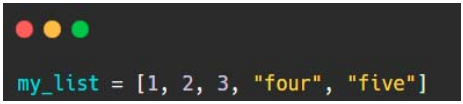
есть несколько типов коллекций объектов, включая списки, кортежи, словари и множества (sets). Каждый из них имеет свои особенности и подходит для определенных задач.

Списки (list) – это упорядоченные коллекции объектов, которые могут содержать элементы разных типов. Списки могут изменяться (mutable), то есть вы можете добавлять, удалять и изменять элементы.

В Python списки можно создавать двумя способами: с помощью квадратных скобок [] и с помощью функции list(). Оба способа используются для создания списка, но есть некоторые отличия.

Индексация начинается с 0.

Создание списка через квадратные скобки [] более компактное и простое. Например, для создания списка можно использовать следующий код:



```
my_list = [1, 2, 3, "four", "five"]
```

Доступ к элементам списка осуществляется через указание индекса элемента.

Например:

z =	[4,	1,	5,	4,	10,	4]
index	0	1	2	3	4	5

Создание списка через функцию list() требует передачи итерируемого объекта в качестве аргумента. Итерируемый объект может быть любым объектом, который можно перебирать поэлементно, например, строкой, кортежем или другим списком. Например, чтобы создать список из строки, можно использовать следующий код:

```
my_list = list("Строка") # ['С', 'т', 'р', 'о', 'к', 'а']
```

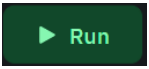
В качестве примера рассмотрим первый список.

```
my_list = [1, 2, 3, "four", "five"]
```

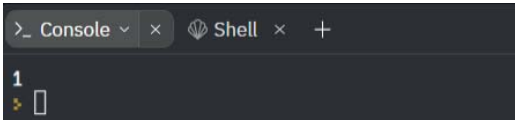
Обратимся к списку my_list по индексу 0. Получим в результате число 1.

```
print(my_list[0]) #1
```

Запустим, нажав на кнопку Run.



Получили первый элемент списка.



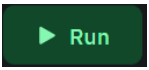
Кортежи (tuple) – это упорядоченные коллекции объектов, которые также могут содержать элементы разных типов. Кортежи неизменяемы (immutable), то есть вы не можете добавлять, удалять или изменять элементы после их создания. Кортежи создаются с помощью круглых скобок (), и элементы разделяются запятой.

```
my_tuple = (1, 2, 3, "four", "five")
```

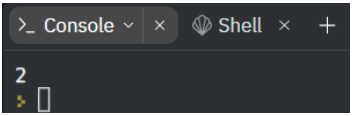
Доступ к элементам также осуществляется через указание индекса элемента. Обратимся к кортежу my_tuple по индексу 1. Получим в результате число 2.

```
my_tuple[1] # 2
```

Запустим, нажав на кнопку Run.



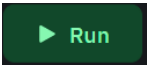
Получили второй элемент кортежа.

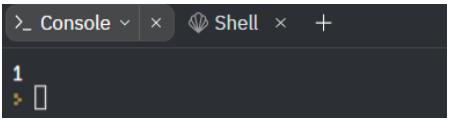


Словари (dictionary) – это неупорядоченные коллекции объектов, которые хранятся в виде пар «ключ-значение». Ключи должны быть уникальными и неизменяемыми (часто используются строки или числа), а значения могут быть любого типа. Словари могут изменяться (mutable), то есть вы можете добавлять, удалять и изменять элементы. Словари создаются с помощью фигурных скобок {} или при помощи встроенной функции dict(). Элементы записываются в формате «ключ : значение», разделенные запятой.

Доступ к элементам словаря осуществляется по ключу.

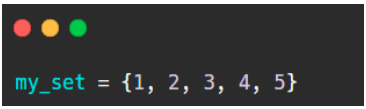
Запустим, нажав на кнопку Run.





Получили элемент словаря по ключу ‘key1’.

Множества (sets) – это неупорядоченные коллекции уникальных объектов, которые могут содержать элементы разных типов. Множества могут изменяться (mutable), то есть вы можете добавлять и удалять элементы. Множества создаются с помощью фигурных скобок {} или функции set(), и элементы разделяются запятой.



Создание множества через функцию set() требует передачи итерируемого объекта в качестве аргумента. Операция аналогична функции list(). Например, чтобы создать множество из строки, можно использовать следующий код:

```
my_set = set("Строка") # {'р', 'т', 'о', 'а', 'к', 'С'}
```

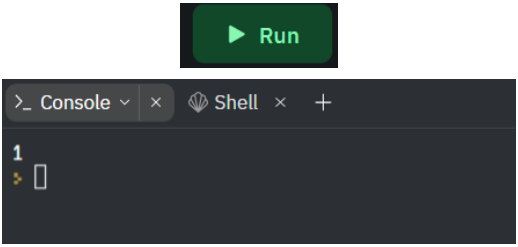
Стоит заметить, что при использовании функции list() каждый символ в строке добавляется в список в том порядке, в котором они встречаются. При этом повторяющиеся элементы добавляются в список несколько раз, так как список может содержать повторяющиеся элементы. В результате получается список, содержащий все символы исходной строки в том порядке, в котором они встречаются.

Однако порядок элементов в множестве не сохраняется, поскольку множество – это коллекция элементов, которые не упорядочены. Порядок элементов множества зависит от хэш-функций, которые используются для хранения и поиска элементов внутри множества, и может меняться от запуска к запуску.

Получить методы из множества можно путем преобразования в список через встроенную функцию list().

```
my_set = {1, 2, 3, 4, 5}
print(list(my_set)[0]) # 1
```

Запустим, нажав на кнопку Run.



Получили значение из множества по индексу 0.

В Python каждый тип коллекции объектов (списки, кортежи, словари, множества) имеет свой набор методов для работы с ними. Рассмотрим наиболее распространенные методы каждой коллекции.

Методы для списков

append(x) – добавляет элемент x в конец списка;

extend(iterable) – добавляет элементы из итерируемого объекта в конец списка;

insert(i, x) – добавляет элемент x на позицию i в списке;

remove(x) – удаляет первый найденный элемент x из списка;

pop([i]) – удаляет и возвращает элемент на позиции i (или последний элемент, если i не указан);

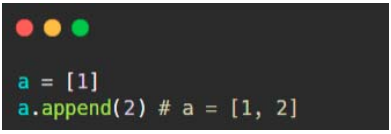
index(x) – возвращает индекс первого найденного элемента x в списке;

count(x) – возвращает количество вхождений элемента x в списке;

sort() – сортирует элементы списка по возрастанию;

reverse() – изменяет порядок элементов списка на обратный.

Пример метода append() для списка:

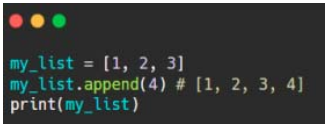


Методы для кортежей

Так как кортежи являются неизменяемыми объектами, то их методы ограничены базовыми операциями:

count(x) – возвращает количество вхождений элемента x в кортеже;

index(x) – возвращает индекс первого найденного элемента x в кортеже.



Методы для словарей

keys() – возвращает список всех ключей словаря;

values() – возвращает список всех значений словаря. items() – возвращает список всех ключей и значений словаря в виде кортежей (key, value);

get(key[, default]) – возвращает значение ключа key или значение по умолчанию default, если ключа не существует;

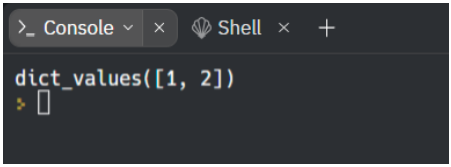
pop(key[, default]) – удаляет и возвращает значение ключа key или значение по умолчанию default, если ключа не существует;

update(other_dict) – обновляет словарь значениями из другого словаря other_dict;

clear() – удаляет все элементы словаря.

Например:

В данном примере получаем все значения словаря my_dict и выводим через print().



Методы для множеств (sets)

add(x) – добавляет элемент x в множество;

update(iterable) – добавляет элементы из итерируемого объекта в множество;

remove(x) – удаляет элемент x из множества. Если элемент не найден, возникает исключение;

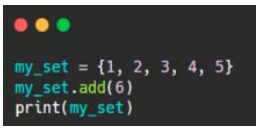
discard(x) – удаляет элемент x из множества. Если элемент не найден, ничего не происходит;

pop() – удаляет и возвращает произвольный элемент из множества. Если множество пустое, возникает исключение;

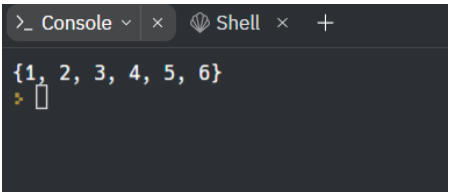
clear() – удаляет все элементы множества.

Все представленные методы для списков, множеств, словарей применяются по следующему принципу: *название_элемента*.*метод_элемента*.

Например:



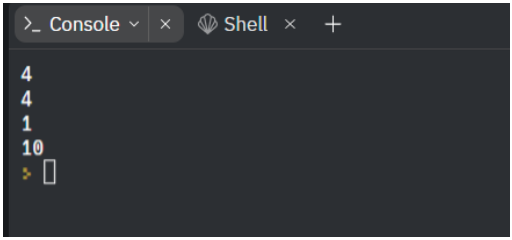
В данном примере добавляем целочисленный элемент «6» в конец множества my_set и выводим его через print().



Кроме того, для всех коллекций в Python есть встроенные функции, такие как len(), max(), min(), sum(), которые позволяют получить информацию о количестве элементов в коллекции, максимальном и минимальном значении элементов и сумме элементов соответственно.

```
my_list = [1, 2, 3]
my_list.append(4)
print(len(my_list))
print(max(my_list))
print(min(my_list))
print(sum(my_list))
```

В данном примере добавляем к списку my_list в конец значение 4 и выводим через print() длину списка, максимальное значение, минимальное значение и сумму элементов.



Задания для самостоятельного выполнения

1. Напишите программу, которая создает список из 10 случайных целых чисел и выводит только те числа, которые больше 5.

2. Создайте словарь для хранения информации о студентах (имя студента и его оценка). Заполните словарь данными, введенными пользователем, и выведите среднюю оценку всех студентов.

3. Напишите программу, которая принимает два списка чисел от пользователя, преобразует их в множества и выводит список уникальных элементов, которые встречаются в обоих списках.

4. Создайте кортеж из нескольких различных типов данных (строки, числа, списки), затем напишите функцию, которая принимает этот кортеж и печатает каждый элемент в новой строке.

5. Напишите программу, которая принимает строку от пользователя и выводит эту же строку, где каждое слово начинается с большой буквы, а остальные буквы маленькие.

6. Используя списковое включение, создайте список всех чисел от 1 до 100, которые делятся на 7.

7. Напишите программу, которая принимает строку от пользователя и использует словарь для подсчета количества каждой буквы в строке. Выведите результат в формате «буква : количество».

8. Создайте программу, которая принимает от пользователя несколько слов через запятую (в одной строке), затем разделяет строку по запятым в список слов. После этого программа должна вывести слова в обратном порядке, каждое на новой строке.

9. Напишите программу, которая создает список из 10 чисел. Затем программа должна найти и вывести на экран максимальное число, минимальное число и сумму всех чисел в списке.

10. Напишите программу, которая принимает два списка чисел от пользователя, преобразует их в множества и выводит на экран следующие результаты:

а) элементы, которые присутствуют в обоих множествах (пересечение);
б) элементы, которые присутствуют в первом множестве, но отсутствуют во втором (разность);
в) элементы, которые присутствуют во втором множестве, но отсутствуют в первом (разность);
г) все уникальные элементы из обоих списков (объединение).

Тема «Файлы, модули и функции в Python»

Задание 1. Напишите программу, которая открывает файл, считает количество строк в нем и выводит результат на экран.

Ход работы

1. Изучите методические рекомендации.

2. Создайте переменную filename и присвойте ей имя файла, например, ‘example.txt’, который необходимо открыть. Инициали-

зируйте переменную line_count значением 0, чтобы использовать её для подсчёта строк.

```
filename = 'example.txt'
line_count = 0
```

3. Откройте файл для чтения. Используйте контекстный менеджер with для открытия файла, на который указывает filename. Мод ‘r’ означает, что файл открыт для чтения. Подсчитайте количество строк. Организуйте цикл for, который будет проходиться по каждой строке открытого файла. Для каждой строки цикл увеличивает счётчик line_count на единицу.

```
with open(filename, 'r') as f:
  for line in f:
    line_count += 1
```

4. После завершения цикла используйте функцию print(), чтобы вывести на экран количество строк. Используйте f-строку для форматирования вывода, включив в неё значения переменных filename и line_count.

```
print(f'Количество строк в файле {filename} = {line_count}')
```

```
with open('example.txt', 'r') as file:
  content = file.read()
```

Этот код открывает файл ‘example.txt’ для чтения и автоматически закрывает его после того, как выполнение кода выйдет из блока with.

Методические указания

Для работы с файлами в Python используются встроенные функции open(), read(), write() и close().

Функция open() используется для открытия файла. Она принимает два аргумента: имя файла и режим открытия файла. Режим открытия файла может быть: «r» – для чтения, «w» – для записи, «a» – для добавления (дописывания) в конец файла, «x» – для создания нового файла и записи в него, «b» – для работы в бинарном режиме и «t» – для работы в текстовом режиме.

Пример открытия файла для чтения

Функция read() используется для чтения данных из файла. Она может быть вызвана после открытия файла. Метод read() читает файл и возвращает его содержимое в виде строки.

Функция write() используется для записи данных в файл. Она может быть вызвана после открытия файла с режимом «w», «a» или «x». Метод write() записывает данные в файл.

Функция close() используется для закрытия файла, когда работа с ним завершена.

Также для работы с файлами в Python можно использовать конструкцию with, которая автоматически закрывает файл после окончания работы с ним. With – это контекстный менеджер в Python, который позволяет автоматически управлять ресурсами в блоке кода.

Задание 2. Напишите программу, которая создает файл и записывает в него список чисел, разделенных запятыми.

Ход работы

1. Изучите методические рекомендации.

2. Создайте переменную numbers и присвойте ей список чисел, который вы хотите записать в файл. Создайте переменную filename и присвойте ей строку с названием файла, например, «numbers.txt».

```
numbers = [1, 2, 3, 4, 5]
filename = "numbers.txt"
```

3. Используйте контекстный менеджер with и функцию open() для открытия файла с именем, указанным в переменной filename, в режиме ‘w’, что указывает на запись в файл. Инициируйте цикл for, который будет проходить по каждому элементу списка numbers. Внутри цикла используйте метод write() для записи каждого числа в файл. Поскольку write() требует строкового значения, преобразуйте число в строку, используя функцию str(). После каждого числа добавьте запятую и пробел, чтобы числа были разделены в файле.

```
with open(filename, 'w') as f:
  for number in numbers:
    f.write(str(number) + ",")
```

4. После выхода из контекстного менеджера with (то есть после записи всех чисел) используйте print() для вывода подтверждения, что список чисел был записан в файл. Сообщение также форматируется с использованием f-строки для включения имени файла.

```
print(f"Список чисел записан в файл {filename}")
```

Работа с файлами в Python очень важна для обработки и хранения данных. Python предоставляет удобный и простой интерфейс для чтения и записи файлов на диске.

Методические указания

Функция read() используется для чтения данных из файла. Она может быть вызвана после открытия файла. Метод read() читает файл и возвращает его содержимое в виде строки.

Функция write() используется для записи данных в файл. Она может быть вызвана после открытия файла с режимом «w», «a» или «x». Метод write() записывает данные в файл.

Функция close() используется для закрытия файла, когда работа с ним завершена.

Также для работы с файлами в Python можно использовать конструкцию with, которая автоматически закрывает файл после окончания работы с ним. With – это контекстный менеджер в Python, который позволяет автоматически управлять ресурсами в блоке кода.

Задание 3. Напишите функцию с именем divide_numbers, которая принимает два аргумента – числа a и b. Функция должна делить число a на число b и возвращать результат деления. Однако функция должна быть защищена от возможных исключений, таких как деление на ноль или передача аргументов неправильного типа.

Ход работы

1. Изучите методические рекомендации.

2. Создайте функцию с названием divide_numbers, которая принимает два параметра: a и b. Используйте блок try, чтобы попытаться выполнить операцию деления. Внутри блока try преобразуйте аргументы a и b в тип float и выполните операцию деления a на b. Если деление проходит без ошибок, верните результат деления. В слу-

чае возникновения ошибки ZeroDivisionError (деление на ноль) верните строку «Ошибка». Для обработки ошибок неправильного типа аргументов (если a или b не могут быть преобразованы в float) или если передано значение, которое не поддерживается операцией деления (например, строка), используйте except для перехвата исключений ValueError и TypeError. В этом случае также верните строку «Ошибка».

```
def divide_numbers(a, b):
    try:
        result = float(a) / float(b)
        return result
    except ZeroDivisionError:
        return "Ошибка"
    except (ValueError, TypeError):
        return "Ошибка"
```

3. Вызовите функцию divide_numbers с различными аргументами для проверки корректности работы: с двумя числами, чтобы увидеть результат деления; с числом и нулем, чтобы проверить обработку ZeroDivisionError; со строкой и числом, чтобы проверить обработку ValueError и TypeError.

```
print(divide_numbers(10, 2))  # Вывод: 5.0
print(divide_numbers(8, 0))   # Вывод: Ошибка
print(divide_numbers("10", 2)) # Вывод: Ошибка
```

Методические указания

Функции – это именованные блоки кода, которые можно вызывать из других частей программы. Функции создаются с помощью ключевого слова def, за которым следует имя функции и в скобках – аргументы функции. Функция может возвращать значение с помощью ключевого слова return.

Синтаксис определения функции в Python выглядит следующим образом:

```
def имя_функции(аргументы):
    блок кода
    return значение
```

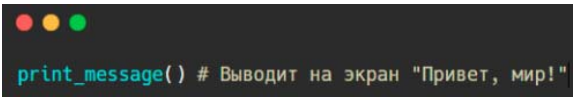
где имя_функции – это имя функции, аргументы – это список аргументов, которые принимает функция, блок кода – это код, который должен быть выполнен при вызове функции, и значение – это значение, которое должна вернуть функция.

```
def hello(name):
  return "Hello, " + name + "!"

print(hello("Alice"))
```

Функции могут также использоваться без возвращаемых значений. В этом случае функция выполняет необходимые действия, но не возвращает никаких значений. Например, следующая функция выводит на экран сообщение:

```
def print_message():
    print("Привет, мир!")
```

Функция может быть вызвана из другой части программы следующим образом:

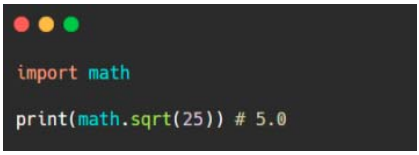


```
print_message() # Выводит на экран "Привет, мир!"
```

Модули – это файлы, содержащие определения функций и других объектов. Чтобы использовать определения из модуля, его нужно импортировать в текущий файл с помощью ключевого слова import.

Например, модуль math содержит функции для математических вычислений, модуль os содержит функции для работы с операционной системой, а модуль datetime содержит функции для работы с датами и временем.

Для использования модуля в программе нужно сначала импортировать его с помощью ключевого слова import. Например, чтобы использовать функцию sqrt из модуля math, нужно написать следующий код:



Можно также импортировать отдельные функции из модуля, чтобы не загружать в память весь модуль. Например, чтобы использовать только функцию sqrt из модуля math, можно написать такой код:

Обратите внимание, если мы используем ключевое слово from, то при использовании модуля библиотеки не нужно вызывать функцию через точку, например math.sqrt(), достаточно использовать импортированную функцию.

В Python также есть возможность создавать свои собственные модули. Для этого нужно создать файл с расширением «.py» и определить в нем нужные функции, классы и переменные. Затем этот файл можно импортировать в других программах.

Например, предположим, что у нас есть файл my_module.py со следующим содержимым:

Чтобы использовать эту функцию в другой программе (другом файле), нужно импортировать модуль my_module:

В Python можно обрабатывать исключения – это ошибки, возникающие во время выполнения программы. Обработка исключений позволяет избежать прерывания программы и предоставить пользователю более информативное сообщение об ошибке.

Когда возникает ошибка в программе, интерпретатор Python создает объект исключения, который содержит информацию об ошибке. Затем интерпретатор ищет блок try-except, который может обработать это исключение.

Блок try-except в Python предназначен для обработки исключений. Он состоит из двух частей: блока try, в котором содержится код, могущий вызвать исключение, и блока except, который содержит код для обработки исключения.

```
try:
  # Код, который может вызвать исключение
  x = int(input("Введите x: "))
  y = 1 / x
except ZeroDivisionError:
  # Код для обработки исключения
  print("Деление на 0 невозможно")
```

В этом примере используется блок try-except для обработки исключения ZeroDivisionError, которое может возникнуть, если пользователь введет ноль в качестве делителя. Если исключение возникает, программа выводит сообщение об ошибке.

Кроме того, в Python есть ключевое слово finally, которое позволяет выполнить код, независимо от того, произошло исключение или нет. Блок finally используется, например, для освобождения ресурсов, которые были заняты в блоке try.

```
try:
  # Код, который может вызвать исключение
  f = open("file.txt", "r")
  content = f.read()
except IOError:
  # Код для обработки исключения
  print("Не удалось открыть файл")
finally:
  # Код, который будет выполнен в любом случае
  f.close()
```

В этом примере используем блок try-except-finally для чтения файла. Если файл не удается открыть, программа выводит сообщение об ошибке. В блоке finally закрываем файл, чтобы освободить ресурсы.

Задания для самостоятельного выполнения

1. Напишите функцию, которая читает текстовый файл и выводит его содержимое на экран.

2. Создайте функцию, которая принимает строку от пользователя и записывает её в текстовый файл.

3. Напишите программу, которая читает файл и подсчитывает количество слов в каждой строке, затем записывает результат в новый файл в формате «Строка 1: 5 слов».

4. Создайте собственный модуль, который содержит функции для выполнения основных математических операций (сложение, вычитание, умножение, деление) и импортируйте этот модуль в другой Python скрипт для использования этих функций.

5. Напишите функцию say_hello, которая просто печатает «Привет, мир!» при её вызове.

6. Создайте скрипт, который запрашивает у пользователя его имя и сохраняет его в текстовый файл.

7. Напишите скрипт, который открывает текстовый файл и выводит все его строки на экран, добавляя номер строки перед текстом каждой строки.

8. Создайте текстовый файл с несколькими числами, каждое из которых находится на новой строке. Напишите функцию, которая читает файл и выводит сумму этих чисел.

9. Импортируйте модуль math и используйте функцию sqrt для вычисления квадратного корня числа, полученного от пользователя.

10. Напишите функцию convert_to_upper, которая принимает имя файла как аргумент. Функция должна открыть указанный файл, прочитать его содержимое, преобразовать все текстовые данные в верхний регистр и затем сохранить преобразованный текст в новом файле.

Тема «Объектно ориентированное программирование на Python»

Задание 1. Напишите программу для хранения информации о книгах в библиотеке. Создайте класс Book (книга), имеющий атрибуты title (название), author (автор) и year (год издания), а также метод display_book_info (отображение информации о книге).

Метод display_book_info должен выводить на экран информацию о книге в формате «Название: <название>, Автор: <автор>, Год издания: <год издания>».

Ход работы

1. Изучите методические рекомендации.

2. Создайте класс Book, который будет служить шаблоном для создания объектов-книг. Определите метод __init__, который инициализирует атрибуты класса: title (название книги), author (автор книги) и year (год издания книги).

```
class Book:
  def __init__(self, title, author, year):
    self.title = title
    self.author = author
    self.year = year
```

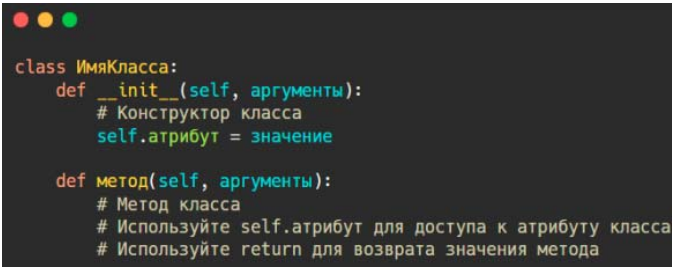
3. Добавьте метод display_book_info в класс Book. Этот метод будет выводить информацию о книге, используя атрибуты объекта. Внутри метода display_book_info используйте функцию print для вывода информации о книге в формате: «Название: <название>, Автор: <автор>, Год издания: <год издания>».

```
def display_book_info(self):
  print("Название:", self.title)
  print("Автор:", self.author)
  print("Год издания:", self.year)
```

Методические указания

Класс в Python – это некий «шаблон» для создания объектов. Класс определяет набор свойств (какие данные будут храниться в объекте) и методов (какие операции можно выполнить с объектом), которые можно использовать для создания конкретных экземпляров этого класса.

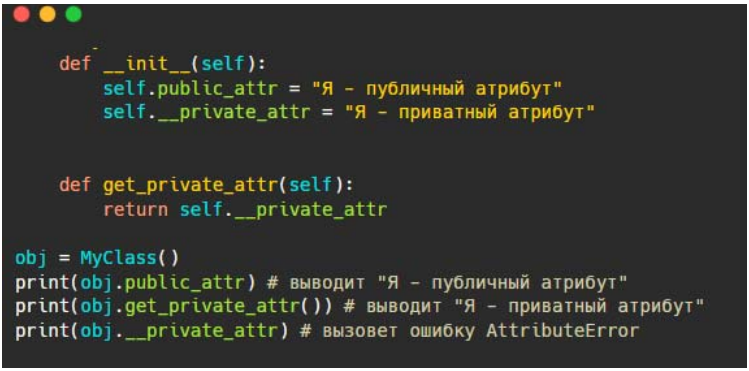
Создание класса в Python начинается с ключевого слова class, за которым следует имя класса и двоеточие. Затем в теле класса определяются атрибуты и методы класса.



```
class ИмяКласса:
    def __init__(self, аргументы):
        # Конструктор класса
        self.атрибут = значение

    def метод(self, аргументы):
        # Метод класса
        # Используйте self.атрибут для доступа к атрибуту класса
        # Используйте return для возврата значения метода
```

Конструктор класса определяется методом __init__, который вызывается при создании экземпляра класса. В конструкторе класса определяются атрибуты класса, которые могут быть использованы в других методах класса.

В Python нет строгой поддержки приватных и публичных полей, как в некоторых других языках программирования, таких как Java или C++. Однако в Python есть соглашение об использовании двойных подчеркиваний перед именами атрибутов класса, чтобы указать, что они являются приватными. Атрибут, который начинается с двух подчеркиваний, будет скрыт от внешнего доступа, и его имя будет изменено, чтобы начинаться с имени класса и одного подчеркивания. Например, атрибут с именем __foo в классе MyClass будет переименован в атрибут _MyClass__foo и будет доступен только внутри класса MyClass.



```
    def __init__(self):
        self.public_attr = "Я - публичный атрибут"
        self.__private_attr = "Я - приватный атрибут"


    def get_private_attr(self):
        return self.__private_attr

obj = MyClass()
print(obj.public_attr) # выводит "Я - публичный атрибут"
print(obj.get_private_attr()) # выводит "Я - приватный атрибут"
print(obj.__private_attr) # вызовет ошибку AttributeError
```

Методы класса определяются в теле класса и могут использовать атрибуты класса для выполнения задач. Методы могут возвращать значения, используя оператор return.

Например, можно создать класс «Человек», который будет иметь свойства «имя» и «возраст» и метод «приветствие», который будет выводить на экран приветствие с именем и возрастом человека.

Чтобы создать класс в Python, нужно использовать ключевое слово class, а затем указать имя класса и определить его свойства и методы. Например, вот как может выглядеть определение класса «Человек»:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def say_hello(self):
    print(f"Привет, меня зовут {self.name} и мне {self.age} лет")
```

Здесь метод __init__ – это специальный метод, который будет вызываться при создании объекта класса и инициализирует его свойства (в данном случае имя и возраст). А метод say_hello будет выводить приветствие на экран с использованием этих свойств.

Задание 2. Напишите программу для хранения информации о студентах в университете. Создайте класс Student (студент), имеющий атрибуты name (имя), stud_id (номер студенческого билета), а также метод display_student_info (отображение информации о студенте).

Метод display_student_info должен выводить на экран информацию о студенте в формате «Имя: <имя>, Номер студенческого билета: <номер>».

Ход работы

1. Изучите методические рекомендации.

2. Создайте класс Student, который будет представлять студента с атрибутами и методами для хранения и отображения его информации.

```
class Student:
  university = "ТГУ"
  faculty = "ИМФиИТ"
```

3. Определите конструктор класса (метод __init__()) с параметрами self, name (имя студента), stud_id (номер студенческого билета). Добавьте атрибут speciality (специальность) в определение класса, если он должен быть общим для всех студентов, или в метод __init__, если он должен быть уникальным для каждого студента.

```
class Student:
  university = "ТГУ"
  faculty = "ИМФиИТ"

  def __init__(self, name, stud_id):
    self.name = name
    self.stud_id = stud_id
```

4. Внутри класса Student определите метод display_student_info, который выводит информацию о студенте, используя атрибуты объекта.

```
def display_student_info(self):
  print("Имя:", self.name)
  print("Номер студенческого билета:", self.stud_id)
```

5. Создайте экземпляры класса Student, передав соответствующие аргументы в конструктор класса (name и stud_id). Вызовите метод display_student_info для каждого экземпляра класса Student, чтобы отобразить информацию о студенте.

```
student1 = Student("Иван Иванов", 123456)
student1.display_student_info()
print("Университет:", student1.university)
print("Факультет:", student1.faculty)

student2 = Student("Петров Петр", 123457)
student2.display_student_info()
print("Университет:", student2.university)
print("Факультет:", student2.faculty)
```

```
>_ Console  Shell
Имя: Иван Иванов
Номер студенческого билета: 123456
Университет: ТГУ
Факультет: ИМФиИТ
Имя: Петров Петр
Номер студенческого билета: 123457
Университет: ТГУ
Факультет: ИМФиИТ
```

Методические указания

Объект класса – компонент, который нами создается на основе шаблона, называемого классом. Класс описывает, каким должен

быть объект, какие данные он должен содержать и какие действия он может выполнить.

Создавая объект класса, мы создаем конкретный экземпляр этого шаблона, который может хранить конкретные значения данных и выполнить определенные действия, которые были определены в классе. Например, если класс – это «Человек», то объект класса «Человек» может содержать информацию о конкретном человеке, такую как имя, возраст и адрес, и может выполнить определенные действия, например, печатать свои данные.

Например, при создании объекта класса «Человек» с именем «Алексей» и возрастом «30» создается конкретный объект, который представляет собой одного человека.

Задание 3. Напишите программу для хранения информации о фрукте. Создайте класс Fruit (фрукт) с атрибутами класса category (категория) и color (цвет), а также атрибутами объекта name (название) и price (цена). Создайте два объекта этого класса – «fruit1» и «fruit2». Выведите на экран информацию о каждом фрукте.

Ход работы

1. Изучите методические рекомендации.

2. Определите класс с именем Fruit. В классе Fruit объявите атрибуты класса: category, который обозначает категорию и изначально установлен в «Фрукты», и color, который обозначает цвет фрукта и изначально имеет значение «не указан».

```
class Fruit:
  category = "Фрукты"
  color = "не указан"
```

3. В методе __init__ класса Fruit инициализируйте атрибуты объекта: name, который хранит название фрукта, и price, который хранит цену фрукта.

```
def __init__(self, name, price):
  self.name = name
  self.price = price
```

4. Создайте метод display_fruit_info, который выводит информацию о фрукте. Метод должен выводить название и цену фрукта, используя значения соответствующих атрибутов объекта.

```
def display_fruit_info(self):
  print("Название:", self.name)
  print("Цена:", self.price)
```

5. Создайте два объекта: fruit1 и fruit2 класса Fruit, передав в конструктор название и цену для каждого фрукта. Для каждого объекта вызовите метод display_fruit_info для отображения информации о фрукте. Отдельно выведите атрибуты класса category и color для каждого объекта.

```
fruit1 = Fruit("Яблоко", 50)
fruit1.display_fruit_info()
print("Категория:", fruit1.category)
print("Цвет:", fruit1.color)

fruit2 = Fruit("Апельсин", 30)
fruit2.color = "Оранжевый"
fruit2.display_fruit_info()
print("Категория:", fruit2.category)
print("Цвет:", fruit2.color)
```

Методические указания

Каждый объект класса имеет свои уникальные значения свойств (атрибутов), которые были инициализированы при создании объекта.

Например, объект «Алексей» имеет свойство «имя» со значением «Алексей» и свойство «возраст» со значением «30».

Чтобы создать объект класса «Человек», нужно вызвать конструктор класса, передав ему нужные параметры (имя и возраст). Например, вот как можно создать два объекта класса «Человек»:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def say_hello(self):
    print(f"Привет, меня зовут {self.name} и мне {self.age} лет")

person1 = Person("Алексей", 30)
person2 = Person("Наталья", 25)
```

Теперь можно вызывать метод say_hello для каждого объекта, чтобы вывести на экран приветствие с их именем и возрастом:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def say_hello(self):
    print(f"Привет, меня зовут {self.name} и мне {self.age} лет")

person1 = Person("Алексей", 30)
person2 = Person("Наталья", 25)

person1.say_hello() # "Привет, меня зовут Алексей и мне 30 лет"
person2.say_hello() # "Привет, меня зовут Наталья и мне 25 лет"
```

Кроме того, класс может иметь общие свойства (атрибуты), которые будут использоваться всеми объектами этого класса.

Атрибуты – это переменные, которые хранят значения для определенного объекта. В Python атрибуты могут быть определены как на уровне класса, так и на уровне объекта.

Рассмотрим несколько примеров.

Атрибуты на уровне класса

Например, класс «Человек» может иметь общее свойство «пол», которое будет одинаковым для всех созданных объектов этого класса.

```
class Person:
  gender = "" # общий атрибут для всех объектов класса

  def __init__(self, name, age):
    self.name = name
    self.age = age
```

В этом примере класс Person имеет общий атрибут gender, который будет одинаковым для всех объектов, созданных на основе этого класса. Атрибуты name и age определены на уровне объекта и будут уникальными для каждого созданного объекта.

Атрибуты на уровне объекта

Атрибуты объекта и общие атрибуты класса могут быть использованы в методах класса для выполнения различных операций. Например, метод «приветствие» класса «Человек» использует свойства объекта «имя» и «возраст» для вывода на экран персонализированного приветствия.

```python
class Person:
  gender = "" # общий атрибут для всех объектов класса

  def __init__(self, name, age):
    self.name = name
    self.age = age

person1 = Person("Алексей", 30)
person2 = Person("Наталья", 25)

print(person1.name) # "Алексей"
print(person2.age) # 25
```

Здесь класс Person имеет конструктор, который инициализирует атрибуты name, age на уровне объекта. Создаются два объекта – person1 и person2, каждый из которых имеет уникальные значения своих атрибутов. Мы можем обращаться к атрибутам объектов, используя имя объекта и имя атрибута, например, person1.name, получив значение поля name объекта person1 или person2.age, получив значение поля age объекта person2.

Методы класса в Python

Методы класса в Python – это функции, которые определены внутри класса и могут быть вызваны на объектах этого класса. Методы могут изменять состояние объекта, работать с его атрибутами и выполнять различные действия в зависимости от требований программы.

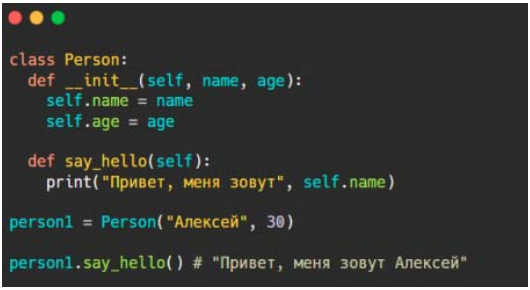
Как и обычные функции, методы могут принимать аргументы и возвращать значения. Однако в отличие от обычных функций методы могут иметь доступ к атрибутам объекта через ключевое слово self. Это позволяет методам работать с данными объекта и изменять их.

Пример класса Person с методом say_hello():

```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def say_hello(self):
    print("Привет, меня зовут", self.name)

person1 = Person("Алексей", 30)
person2 = Person("Наталья", 25)

print(person1.name) # "Алексей"
print(person2.age) # 25
```

Здесь класс Person содержит метод say_hello, который выводит на экран приветствие с именем объекта.

Чтобы вызвать метод объекта, нужно указать его имя после имени объекта и передать необходимые аргументы, если они есть. Например, чтобы вызвать метод say_hello для объекта person1, можно использовать следующий код:



```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def say_hello(self):
    print("Привет, меня зовут", self.name)

person1 = Person("Алексей", 30)

person1.say_hello() # "Привет, меня зовут Алексей"
```

Наследование является одним из ключевых концепций объектно ориентированного программирования (ООП). Оно позволяет создавать новые классы на основе уже существующих (родительских или базовых классов), наследуя их свойства и методы.

Новый класс, созданный на основе родительского класса, называется дочерним классом или производным классом. Он наследует все атрибуты и методы родительского класса, а также может добавлять свои собственные атрибуты и методы, переопределять методы родительского класса и расширять их функциональность.

В Python наследование реализуется через ключевое слово class и указание базового класса в скобках.

Дочерний класс наследует атрибуты и методы родительского класса, используя ключевое слово super().

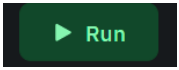
Например, если у вас есть класс «Фрукты», вы можете создать новый класс «Яблоки», который наследует свойства и методы класса «Фрукты». Таким образом, вы можете определить уникальные свойства и методы для класса «Яблоки», не повторяя код, который уже есть в классе «Фрукты».

```python
class Fruit:
  def __init__(self, name, color):
    self.name = name
    self.color = color

  def describe(self):
    print(f"Это фрукт {self.name} и он(а)(о) {self.color}")

class Apple(Fruit):
  def __init__(self, name, color, taste):
    super().__init__(name, color)
    self.taste = taste

  def describe(self):
    super().describe()
    print(self.taste)

apple = Apple("Яблоко", "красное", "сладкое")
apple.describe()
```

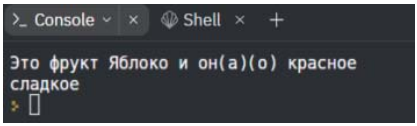
В этом примере нами был создан базовый класс Fruit, который имеет свойства name (название) и color (цвет), а также метод describe, который выводит информацию о фрукте. Затем создали класс Apple, который наследует свойства и методы класса Fruit. Класс Apple имеет дополнительное свойство taste (вкус) и переопределяет метод describe, чтобы добавить информацию о вкусе яблока.

Далее мы создали экземпляр класса Apple с названием Apple, красным цветом и сладким вкусом, а затем вызвали метод describe. Результатом выполнения кода будет вывод информации о яблоке, включая его название, цвет и вкус.

Запустим код через кнопку Run



и получим следующий результат:



Декораторы – это способ добавления дополнительной функциональности к уже существующему коду, без изменения самого кода. Так вы можете создать функцию, которая добавляет проверку наличия файла перед выполнением другой функции. Декораторы могут быть довольно сложными, но для понимания базовых принципов

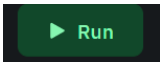
достаточно знать, что они позволяют добавлять «навороты» к уже существующему коду.

Теперь давайте создадим класс «Собаки», который наследует свойства и методы класса «Животные». Также добавим декоратор, который будет выводить сообщение о том, что собака начинает и заканчивает лай.

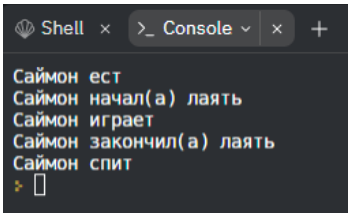
Возьмем за пример декоратор @staticmethod. Он используется для создания методов внутри класса, которые можно вызывать без создания объекта класса. Это значит, что статические методы не зависят от свойств объекта и могут быть использованы для выполнения простых действий, которые не связаны с конкретным объектом класса. Например, статический метод может использоваться для выполнения математических операций над числами. Декоратор @staticmethod устанавливает метод класса как статический, что позволяет вызывать его без создания объекта класса. Это упрощает код и ускоряет его выполнение.

```
class Animals:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print(f"{self.name} ест")

    def sleep(self):
        print(f"{self.name} спит")

class Dog(Animals):
    def __init__(self, name):
        super().__init__(name)

    @staticmethod
    def bark(func):
        def wrapper(self):
            print(f"{self.name} начал(а) лаять")
            func(self)
            print(f"{self.name} закончил(а) лаять")
        return wrapper

    @bark
    def play(self):
        print(f"{self.name} играет")

d = Dog("Саймон")
d.eat()
d.play()
d.sleep()
```

Запустим код через кнопку Run



и получим следующий результат:



В этом примере нами был создан класс «Собаки», который наследует свойства и методы класса «Животные». Также добавили методы eat, sleep в классе «Животные», которые выводят сообщение о том, что собака лает. Наконец, мы переопределили методы bark и play, добавив дополнительный функционал, который выводит сообщение о начале и конце лая, а также о том, что собака играет.

Применяем декоратор bark к методу play, используя символ @. Теперь в момент вызова метода play на экземпляре класса Dog декоратор bark будет выводить сообщения о начале и конце лая.

Далее создаем экземпляр класса Dog, и от этого экземпляра класса вызываем методы eat(), play(), sleep().

Задания для самостоятельного выполнения

1. Создайте класс Book, который будет иметь два атрибута: title и author. Добавьте метод, который печатает информацию о книге в формате Title by Author.

2. Определите класс Student с атрибутами name и age. Добавьте метод introduce, который выводит “Hello, my name is [name] and I am [age] years old”.

3. Создайте класс BankAccount с атрибутами account_number и balance. Добавьте методы deposit и withdraw для управления балансом счёта.

4. Определите базовый класс Pet с атрибутами name и age, а также метод speak, который печатает «I don’t know what to say». Создайте производный класс Dog, который переопределяет метод speak для вывода «Bark».

4. Создайте абстрактный класс Shape с абстрактным методом area. Определите производные классы, такие как Circle и Rectangle, которые реализуют метод area.

6. Определите класс Point с атрибутами x и y. Переопределите оператор «+» и метод __str__() так, чтобы они позволяли складывать два объекта класса Point и корректно выводили координаты точек.

7. Создайте класс Rectangle с закрытыми атрибутами width и height и методами доступа к этим атрибутам (геттеры и сеттеры), обеспечивая проверку, что ширина и высота больше нуля.

8. Определите класс Engine, который содержит метод start. Затем создайте класс Car, который содержит объект Engine как атрибут и метод drive, который запускает двигатель, когда машина начинает движение.

9. Определите класс FileWrapper, который открывает файл в конструкторе и закрывает его в деструкторе __del__(). Создайте экземпляр этого класса и убедитесь, что файл корректно закрывается при удалении объекта.

10. Создайте класс Product, представляющий товар в интернет-магазине. У товара должны быть атрибуты name (название), price (цена) и quantity (количество на складе). Реализуйте методы для установки цены и количества товара, а также метод для вычисления общей стоимости всех единиц товара на складе.

Модуль 2. ОСНОВЫ WEB-РАЗРАБОТКИ НА PYTHON С ИСПОЛЬЗОВАНИЕМ DJANGO

Тема 4. Создание веб-приложения на Python с использованием Django

Основные понятия в web-разработке

Web-разработка – процесс создания веб-сайтов и веб-приложений, которые доступны через Интернет.

Основными понятиями в web-разработке являются:

- HTML (HyperText Markup Language) – язык разметки, который используется для создания содержимого веб-страниц;
- CSS (Cascading Style Sheets) – язык стилей, который используется для оформления веб-страниц;
- JavaScript – язык программирования, который позволяет создавать интерактивные элементы на веб-страницах;
- базы данных – системы, которые используются для хранения и организации данных, которые могут быть доступны через веб-приложения;
- серверные языки программирования – языки программирования, которые используются для создания серверной части веб-приложений, такие как Python, PHP, Ruby и другие.

Python популярен в web-разработке благодаря своей простоте и удобству использования. Он может быть использован как для создания серверной части веб-приложений, так и для создания скриптов и утилит для автоматизации процессов веб-разработки.

Для разработки веб-приложений используются популярные фреймворки Python: Django, Flask, Pyramid и Bottle. Django предоставляет всестороннюю инфраструктуру для создания веб-приложений, включая ORM (Object-Relational Mapping) для работы с базами данных, аутентификацию и авторизацию пользователей, а также многое другое. Flask и Bottle являются более легковесными фреймворками, которые позволяют создавать простые веб-приложения с минимальным количеством настроек. Pyramid предоставляет более гибкую и настраиваемую архитектуру для разработки веб-приложений.

Основные принципы веб-разработки в Python включают использование фреймворков и библиотек для ускорения и упрощения процесса разработки, использование ORM для работы с базами данных, использование MVC (Model-View-Controller) архитектуры для разделения логики приложения, представлений и моделей, а также использование шаблонов для создания динамических веб-страниц.

Виртуальное окружение в Python

Это изолированная среда, в которой можно устанавливать и использовать различные версии пакетов и модулей Python без влияния на другие проекты.

Рассмотрим пошаговое руководство по работе с виртуальным окружением при создании программы на языке Python.

• Установите модуль virtualenv, выполнив команду **pip install virtualenv** в терминале.

• Создайте новое виртуальное окружение, выполнив команду **virtualenv venv**.

• Активируйте виртуальное окружение, выполнив команду **source venv/bin/activate**.

• После активации виртуального окружения в командной строке появится префикс (venv), указывающий на то, что вы находитесь внутри виртуального окружения.

• Установите необходимые пакеты в виртуальное окружение, выполнив команду **pip install package_name**.

• Создайте файл requirements.txt для хранения списка всех зависимостей вашего проекта. Вы можете создать этот файл самостоятельно либо автоматически с помощью команды **pip freeze > requirements.txt**.

• При работе над проектом в будущем вы можете активировать виртуальное окружение, выполнив команду, после которой вы можете запустить свой код в активированном виртуальном окружении, чтобы использовать установленные пакеты.

• Если вы переносите свой проект на другую систему, вам нужно будет создать новое виртуальное окружение и установить все необходимые зависимости из файла requirements.txt. Для этого введите фрагмент кода

```
source venv/bin/activate
pip install -r requirements.txt
```

• По завершении работы с виртуальным окружением вы можете его деактивировать deactivate. Это вернет вас в глобальное пространство Python.

Таким образом, создание и использование виртуального окружения в Python позволяет изолировать проекты друг от друга и управлять версиями пакетов и зависимостей. Это особенно полезно для проектов, которые требуют разных версий одного и того же пакета или зависимости, которые могут конфликтовать между проектами.

Библиотеки Python

Библиотеки – это наборы модулей и пакетов, предназначенные для решения определенных задач. Библиотеки могут содержать функции, классы, методы, константы и другие объекты, которые могут быть использованы в приложениях Python. Могут быть установлены с помощью менеджера пакетов pip или других инструментов.

Рассмотрим наиболее популярные библиотеки Python и их функциональность.

• NumPy – это библиотека для работы с массивами данных. Она предоставляет быстрые и эффективные операции с массивами, такие как матричные операции, операции с линейной алгеброй, Фурье-преобразования и многое другое.

• Pandas – библиотека для работы с данными, которая предоставляет высокоуровневые структуры данных и инструменты для анализа и манипулирования данными. Она позволяет работать с таблицами данных (так называемыми DataFrame) и выполнять различные операции с ними, такие как фильтрация, сортировка, группировка и агрегирование.

• Matplotlib – библиотека для визуализации данных. Она позволяет создавать графики, диаграммы и другие виды визуализации данных. Она также предоставляет множество опций для настройки внешнего вида графиков.

• Scikit-learn – библиотека для машинного обучения. Она предоставляет инструменты для классификации, регрессии, класте-

ризации и других задач машинного обучения. Она также включает инструменты для предобработки данных и выбора признаков.

• Django – библиотека для создания веб-приложений. Она предоставляет мощный набор инструментов для создания веб-сайтов и веб-приложений. Она также включает в себя встроенную административную панель, которая упрощает управление базой данных.

• Flask – библиотека для создания микровеб-приложений. Она предоставляет минимальный набор инструментов для создания простых веб-приложений. Она также легко расширяема и имеет большое количество плагинов и расширений.

• Кроме того, в Python существует множество других библиотек, которые можно использовать для различных задач, таких как обработка изображений, работа с базами данных, парсинг веб-страниц и многое другое.

Для того чтобы начать работать с библиотекой, ее нужно установить. **Установка библиотек в Python** – процесс установки пакетов, которые предоставляют дополнительные функции и возможности для языка программирования Python. В Python для установки библиотек используется менеджер пакетов pip, который входит в стандартную поставку Python.

Рассмотрим пошаговое руководство по установке библиотек в Python с помощью менеджера пакетов pip.

✓ Откройте командную строку или терминал на своем компьютере.

✓ Установите необходимую библиотеку с помощью команды **pip install library_name**, где library_name – имя библиотеки, которую вы хотите установить.

✓ Если вы устанавливаете библиотеку для определенного проекта, рекомендуется установить ее в виртуальное окружение (virtual environment), чтобы изолировать зависимости проекта от других проектов и упростить управление зависимостями.

✓ Дождитесь завершения установки. pip автоматически загрузит и установит все зависимости библиотеки.

✓ После установки библиотеки вы можете импортировать ее в свой код Python с помощью оператора import.

✓ Если вы хотите обновить уже установленную библиотеку, выполните команду **pip install --upgrade library_name**, где library_name – имя библиотеки, которую нужно обновить.

✓ Если вы хотите удалить библиотеку, выполните команду **pip uninstall library_name**, где library_name – имя библиотеки, которую нужно удалить.

✓ Вот некоторые дополнительные параметры, которые вы можете использовать при установке библиотек с помощью pip:

– user – устанавливает пакеты в домашний каталог пользователя, а не в системный каталог. Это может быть полезно, если у вас нет прав администратора на компьютере;

– proxy – позволяет установить прокси-сервер для загрузки пакетов. Это может быть полезно, если вы находитесь за корпоративной сетью;

– no-cache-dir – запрещает использование кэша pip, что может помочь решить проблемы с зависимостями.

Django

Рассмотрим более подробно библиотеку, предназначенную для создания веб-приложений.

Django – бесплатная и открытая библиотека для Python, предназначенная для создания веб-приложений.

Django предоставляет разработчикам мощный набор инструментов для создания веб-приложений, включая следующие:

– ORM (Object-Relational Mapping) – инструмент для работы с базами данных, который позволяет разработчикам работать с данными в виде объектов Python, а не SQL-запросов. Django ORM автоматически создает таблицы в базе данных и обрабатывает связи между таблицами;

– административная панель – встроенный инструмент в Django, который позволяет управлять данными в базе данных через веб-интерфейс. Разработчики могут настроить административную панель для своих моделей данных, чтобы пользователи могли легко добавлять, удалять и редактировать данные;

– URL-маршрутизация – позволяет разработчикам определять, какие URL-адреса должны обрабатываться, какие функции.

Django автоматически обрабатывает URL-адреса и вызывает соответствующие функции;

– шаблонизация – позволяет разработчикам создавать динамические HTML-страницы, используя шаблоны. Шаблоны Django позволяют разработчикам создавать повторно используемый код для отображения данных на веб-страницах;

– Middleware – инструмент, который позволяет разработчикам добавлять дополнительную функциональность к запросам и ответам, обрабатываемым Django;

– фреймворк для тестирования – позволяет разработчикам создавать и запускать автоматические тесты для своего приложения. Django предоставляет множество инструментов для тестирования, включая тестирование моделей, представлений и URL-маршрутизации;

– безопасность – Django включает в себя множество инструментов для обеспечения безопасности приложений, таких как защита от CSRF-атак, защита от SQL-инъекций и многое другое.

В целом Django предоставляет разработчикам все необходимые инструменты для создания мощных веб-приложений на Python. Она обладает множеством преимуществ, таких как быстрое развертывание, масштабируемость, безопасность и многие другие, что делает ее популярным выбором для веб-разработки.

Создание проекта в Django

Процесс создания проекта – это первый шаг в создании веб-приложения с использованием библиотеки Django.

Рассмотрим пошаговое руководство, которое поможет вам создать новый проект в Django.

• Установите Django на свой компьютер, если она еще не установлена. Для этого выполните команду **pip install django** в терминале или командной строке.

• Откройте командную строку или терминал на своем компьютере.

• Создайте новую папку для проекта, в которой будут храниться все файлы проекта. Для этого выполните команду **mkdir project_name**, где project_name – имя вашего проекта.

• Перейдите в созданную папку, используя команду **cd project_name**.

• Создайте новый проект Django, используя команду **django-admin startproject project_name**. Эта команда создаст новый проект Django в текущей папке с именем project_name.

После выполнения команды структура проекта будет создана в вашей папке. Она будет выглядеть примерно так, как показано на рис. 56, где:

– manage.py – файл, который используется для управления проектом. Он позволяет выполнять различные задачи, такие как запуск сервера, создание миграций и многое другое;
– project_name/ – пакет, который содержит все файлы, связанные с проектом Django;
– __init__.py – пустой файл, который указывает Python, что это пакет;
– settings.py – файл, который содержит настройки проекта Django, такие как базы данных, настройки безопасности и многое другое;
– urls.py – файл, который содержит маршруты (URL-адреса) вашего приложения;
– asgi.py и wsgi.py – файлы, используемые для обслуживания вашего приложения с помощью ASGI (Asynchronous Server Gateway Interface) и WSGI (Web Server Gateway Interface) соответственно.

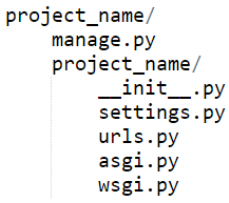


Рис. 56. Структура проекта

После создания проекта вы можете создать новые приложения внутри проекта, используя команду **python manage.py startapp app_name**. Эта команда создаст новое приложение в папке проекта с именем app_name и необходимыми файлами для работы приложения.

Создание веб-страницы на языке Python – это процесс создания страницы, которая может отображаться в веб-браузере.

Рассмотрим последовательность шагов для создания веб-страницы в Django.

Шаг 1. Создание проекта Django. Для создания проекта нужно выполнить в командной строке команду **django-admin startproject project_name**, где project_name – название вашего проекта.

Шаг 2. Создание приложения Django. Для его создания нужно выполнить в командной строке команду **python manage.py startapp app_name**, где app_name – название вашего приложения.

Шаг 3. Определение моделей данных. Для определения моделей данных нужно создать файл models.py в приложении и определить модели данных, используя классы моделей Django. Например, как представлено на рис. 57.

```
from django.db import models

class MyModel(models.Model):
    field1 = models.CharField(max_length=50)
    field2 = models.IntegerField()
```

Рис. 57. Определение класса модели

Шаг 4. Создание миграций, то есть модуля, созданного на основе определенной модели и предназначенного для формирования в базе данных всех требуемых этой моделью структур.

Для создания миграций нужно выполнить в командной строке команду **python manage.py makemigrations**, которая создаст файлы миграций на основе определенных моделей данных.

Шаг 5. Применение миграций. Для применения миграций нужно выполнить в командной строке команду **python manage.py migrate**, которая создаст таблицы в базе данных на основе определенных моделей данных.

Шаг 6. Создание представлений. Для создания представлений нужно создать файл views.py в приложении и определить функции-представления, которые будут обрабатывать запросы и возвращать ответы. Например, так, как представлено на рис. 58.

```
from django.shortcuts import render
from .models import MyModel

def my_view(request):
    my_objects = MyModel.objects.all()
    return render(request, 'my_template.html', {'my_objects': my_objects})
```

Рис. 58. Создание представлений

Шаг 7. Создание шаблонов. Для создания шаблонов нужно создать директорию templates в приложении и создать файлы шаблонов. Шаблоны – это файлы HTML, которые определяют внешний вид веб-страницы. Например, файл my_template.html может выглядеть так, как представлено на рис. 59.

```
{% extends 'base.html' %}

{% block content %}
    <h1>My Objects</h1>
    <ul>
        {% for obj in my_objects %}
            <li>{{ obj.field1 }} - {{ obj.field2 }}</li>
        {% endfor %}
    </ul>
{% endblock %}
```

Рис. 59. Файл HTML

Шаг 8. Создание маршрутов. Для создания маршрутов нужно создать файл urls.py в приложении и определить маршруты, которые указывают, какой URL должен обрабатываться каким представлением. Например, файл urls.py может выглядеть так, как представлено на рис. 60.

```
from django.urls import path
from .views import my_view

urlpatterns = [
    path('my-url/', my_view, name='my_view'),
]
```

Рис. 60. Создание маршрута

Шаг 9. Подключение приложения к проекту. Для подключения приложения к проекту нужно отредактировать файл settings.py в проекте и добавить приложение в список INSTALLED_APPS. Например, как представлено на рис. 61.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'my_app',
]
```

Рис. 61. Код подключения

Шаг 10. Запуск сервера Django. Для запуска сервера Django нужно выполнить в командной строке команду **python manage.py runserver, python manage.py runserver**, которая запустит сервер на локальном хосте. Например: **python manage.py runserver**.

После выполнения всех этих шагов вы сможете открыть веб-страницу, указанную в маршрутах, и увидеть результат.

Шаблоны в Django

Это файлы, которые содержат HTML-разметку и дополнительные теги шаблонов Django, позволяющие создавать динамические веб-страницы. Например, вы можете использовать теги шаблонов Django для вставки переменных Python в HTML-код или для создания циклов для отображения списков данных.

Шаблон определяет, как будет выглядеть ваша веб-страница, включая HTML-разметку, CSS-стили и JavaScript-скрипты.

1. Чтобы создать шаблон, создайте новый файл home.html в папке templates в вашем приложении. Например, чтобы создать простой шаблон для отображения текста, вы можете добавить следующий код в файл home.html (рис. 62).

```
<!DOCTYPE html>
<html>
<head>
    <title>My Home Page</title>
</head>
<body>
    <h1>Welcome to my home page</h1>
    <p>This is the content of my home page.</p>
</body>
</html>
```

Рис. 62. Простой шаблон

2. Подключение шаблона к функции представления. Чтобы функция представления знала, какой шаблон использовать для отображения страницы, необходимо добавить имя шаблона в вызов функции render. Например, чтобы использовать шаблон home.html для отображения главной страницы, вы можете изменить функцию home в файле views.py так, как представлено на рис. 63.

```
def home(request):
    return render(request, 'home.html')
```

Рис. 63. Изменение шаблона

3. Запуск сервера. Чтобы увидеть свою веб-страницу, запустите локальный сервер Django с помощью команды **python manage.py runserver**. Затем откройте браузер и перейдите на адрес http://localhost:8000 (или другой порт, если вы указали другой порт при запуске сервера).

Создание веб-страницы в Django – это процесс создания интерфейса пользователя, который отображается в веб-браузере. В Django это делается с помощью шаблонов, которые содержат HTML-разметку, CSS-стили и JavaScript-скрипты.

Для создания веб-страницы в Django нужно сделать следующее.

• Создать новый файл шаблона в папке templates вашего приложения. Например, вы можете создать файл home.html, который будет содержать HTML-разметку для вашей главной страницы.

• Создать функцию представления в файле views.py вашего приложения. Функция представления определяет, какой контент будет отображаться на странице. Например, вы можете создать функцию представления home, которая будет отображать вашу главную страницу.

• Подключить шаблон к функции представления. Чтобы функция представления знала, какой шаблон использовать для отображения страницы, необходимо добавить имя шаблона в вызов функции render (рис. 64).

```
from django.shortcuts import render
def home(request): return render(request, 'home.html')
```

Рис. 64. Подключение шаблона к функции представления

• Создать маршрут в файле urls.py вашего приложения. Маршрут определяет, какой URL будет отображаться, какая функция представления будет вызвана. Например, чтобы создать маршрут для отображения главной страницы вашего сайта, вы можете добавить следующий код (рис. 65) в файл urls.py.

```
from django.urls import path
from . import views
 urlpatterns = [
   path('', views.home, name='home'),
]
```

Рис. 65. Создание маршрута

Здесь path (‘’, views.home, name=’home’) указывает на вызов функции home в файле views.py при переходе на главную страницу вашего сайта. name=’home’ – это имя маршрута, которое вы можете использовать для генерации URL-адресов в шаблонах.

• Запустить сервер, чтобы увидеть свою веб-страницу. Чтобы увидеть свою веб-страницу, запустите локальный сервер Django с помощью команды **python manage.py runserver**. Затем откройте браузер и перейдите на адрес http://localhost:8000 (или другой порт, если вы указали другой порт при запуске сервера).

Создание веб-страницы в Django включает в себя не только создание шаблона и функции представления, но и другие задачи, такие как настройка статических файлов (например, CSS-стилей и JavaScript-скриптов), использование форм и обработка данных, валидация данных и многое другое. Однако эти шаги могут быть выполнены по мере необходимости в соответствии с вашими потребностями в разработке веб-страницы.

Шаблоны веб-страниц в Python – повторно используемые блоки HTML-кода, которые позволяют создавать динамические веб-страницы. Шаблоны – это файлы, которые определяют, как информация должна быть представлена на веб-странице в Django. Они содержат HTML-разметку и специальные теги и фильтры Django, которые позволяют вставлять динамические данные в HTML-шаблон. Использование шаблонов позволяет разделять логику приложения и представление данных на веб-странице, упрощая разработку и поддержку кода.

Шаблоны Django состоят из HTML-кода и специальных тегов шаблонного языка, которые позволяют включать в шаблон динамические данные и другие элементы.

Рассмотрим несколько наиболее распространенных тегов шаблонного языка Django и их описание.

Тег {% extends %} (рис. 66) используется для указания шаблона, который будет применяться в качестве основы для текущего шаблона. Он позволяет наследовать код из других шаблонов и переопределять отдельные блоки кода.

```
{% extends "base.html" %}
{% block content %}
  <h1>{{ title }}</h1>
  <p>{{ content }}</p>
{% endblock %}
```

Рис. 66. Тег {% extends %}

Тег {% block %} (рис. 67) используется для определения блока кода, который может быть переопределен в наследуемом шаблоне. Он позволяет создавать универсальные шаблоны, которые могут быть адаптированы к различным страницам.

```
{% block content %}
  <h1>{{ title }}</h1>
  <p>{{ content }}</p>
{% endblock %}
```

Рис. 67. Тег {% block %}

Тег {{ переменная }} (рис. 68) используется для включения значения переменной в шаблон. Значение переменной может быть получено из представления.

```
<h1>{{ title }}</h1>
```

Рис. 68. Тег {{ переменная }}

Тег {% for %} (рис. 69) используется для создания цикла, который повторяет определенный блок кода для каждого элемента в списке.

Тег {% if %} (рис. 70) используется для создания условия, которое определяет, будет ли определенный блок кода включен в шаблон.

```
<ul>
{% for item in items %}
  <li>{{ item }}</li>
{% endfor %}
</ul>
```

Рис. 69. Тег {% for %}

```
{% if user.is_authenticated %}
  <p>Welcome back, {{ user.username }}!</p>
{% else %}
  <p>Please log in to continue.</p>
{% endif %}
```

Рис. 70. Тег {% if %}

При использовании шаблонов важно следить за тем, чтобы не усложнять код ненужными деталями и убедиться, что код остается легко читаемым и поддерживаемым.

Рассмотрим пример программного кода, демонстрирующего использование шаблонов Django. Есть модель BlogPost, которая представляет статью блога, и нужно отобразить список статей блога на странице с помощью шаблона. Для этого можно использовать следующий код файла views.py, представленный в табл. 2.

Таблица 2

Использование шаблонов

views.py	**blog_post_list.html**
from django.shortcuts import render from .models import BlogPost def blog_post_list(request): posts = BlogPost.objects.all() context = {'posts': posts} return render(request, 'blog_post_list.html', context)	{% extends 'base.html' %} {% block content %} Blog Posts {% for post in posts %} {{ post.title }} {% endfor %} {% endblock %}

Эта функция представления получает все объекты BlogPost из базы данных и передает их в шаблон. Затем она возвращает ответ

с использованием render(), который отображает шаблон blog_post_list.html с переданным контекстом.

Этот шаблон наследует базовый шаблон base.html и определяет блок content, который содержит заголовок страницы и список статей блога. Тег {% for %} используется для итерации по каждому объекту BlogPost в переданном контексте и создания ссылки на каждую статью блога. В результате при обращении к странице, связанной с функцией представления blog_post_list, будет отображаться список статей блога, полученных из базы данных, в соответствующем шаблоне.

Рассмотрим пример создания страницы «О нас» с использованием шаблона в Django. Нужно создать страницу «О нас», на которой хотим отобразить информацию о нашей компании. Для этого можно использовать следующий код, представленный в табл. 3 для файла views.py.

Таблица 3

Шаблон для создания страницы «О нас»

views.py	**about.html**
from django.shortcuts import render def about(request): company_name = 'My Company' company_description = 'We are a company that does things.' context = {'company_name': company_name, 'company_description': company_description} return render(request, 'about.html', context)	{% extends 'base.html' %} {% block content %} About {{ company_name }} {{ company_description }} {% endblock %}

Эта функция представления определяет переменные company_name и company_description, которые содержат информацию о нашей компании, и передает их в шаблон. Затем она возвращает ответ с использованием render(), который отображает шаблон about.html с переданным контекстом.

В данном примере были использованы шаблоны Django для создания статической веб-страницы, отображающей информацию о нашей компании. Мы определили функцию представления, которая передает информацию о компании в шаблон, где использу-

ем теги шаблонного языка Django для отображения этой информации. В итоге получаем статическую страницу, которая отображает предопределенную информацию.

Рассмотрим пример создания списка пользователей с использованием шаблона в Django. Есть модель User, которая представляет пользователей нашего веб-приложения, и мы хотим отображать список пользователей на странице с помощью шаблона. Для этого можно использовать следующий код, представленный в табл. 4 для файла views.py.

Таблица 4

Создание списка пользователей с использованием шаблона

views.py	**user_list.html**

from django.shortcuts import render
from django.contrib.auth.models import User
 	
def user_list(request):
  users = User.objects.all()
  context = {'users': users}
  return render(request, 'user_list.html', context)
	{% extends 'base.html' %}
	
	{% block content %}
	 Users
	
	{% for user in users %}
	  {{ user.username }}
	{% endfor %}
	
	{% endblock %}

Функция представления получает все объекты User из базы данных и передает их в шаблон. Затем она возвращает ответ с использованием render(), который отображает шаблон user_list.html с переданным контекстом.

Шаблон наследует базовый шаблон base.html и определяет блок content, который содержит заголовок страницы и список пользователей. Тег {% for %} используется для итерации по каждому объекту User в переданном контексте и создания списка пользователей. В результате при обращении к странице, связанной с функцией представления user_list, будет отображаться список пользователей, полученных из базы данных, в соответствующем шаблоне.

В данном примере нами были использованы шаблоны Django для создания динамической веб-страницы, которая отображает список пользователей нашего веб-приложения. Мы определили

функцию представления, которая получает список пользователей и передает его в шаблон, где используем теги шаблонного языка Django для создания списка пользователей. В итоге получаем динамическую страницу, которая отображает актуальную информацию из базы данных.

Тема 5. Базы данных и ORM

База данных в Python

Python – высокоуровневый язык программирования, который позволяет работать с базами данных (БД) различных типов. Для внесения изменений в БД в Python необходимо использовать библиотеки, которые предоставляют соответствующие инструменты для работы с БД.

База данных в Python – это организованная коллекция данных, которую можно хранить, обрабатывать и извлекать в программе на языке Python.

Базы данных используются для хранения, управления и обработки информации. Базы данных используются для хранения информации в виде таблиц, которые содержат различные типы данных, такие как числа, строки, даты и другие.

Python поддерживает несколько типов баз данных, таких как реляционные БД (например, MySQL, PostgreSQL, SQLite), NoSQL БД (например, MongoDB, Cassandra, Redis) и другие. Каждый тип базы данных имеет свои особенности и преимущества, и выбор зависит от требований конкретного проекта.

База данных в Python – мощный инструмент, который позволяет хранить и управлять большими объемами структурированных данных, а также обеспечивает быстрый доступ к информации, что делает ее необходимой для многих типов приложений и проектов.

В Python существует множество библиотек и фреймворков для работы с базами данных, от простых SQLite и MySQL до мощных PostgreSQL и MongoDB. Библиотеки для работы с базами данных в Python предоставляют удобный API для выполнения запросов к базам данных и обработки результатов.

В Python для работы с базами данных используются различные библиотеки, наиболее популярные из которых – SQLAlchemy, Django ORM, psycopg2, PyMySQL, Peewee, Pony ORM, sqlite3 и другие. Они позволяют устанавливать соединение с базой данных, создавать, изменять и удалять таблицы, выполнять запросы и получать данные из базы данных. Поддержка Python ORM-библиотек позволяет работать с базами данных на более высоком уровне абстракции, представляя объекты программы в виде записей в базе данных и наоборот.

Для работы с БД в Python необходимо использовать специальные библиотеки, которые предоставляют соответствующие инструменты для работы с БД.

Общий подход к работе с БД в Python включает выполнение нескольких взаимосвязанных шагов.

Первый шаг – подключение к БД. Для этого нужно использовать соответствующий драйвер для работы с конкретной БД и создать соединение с БД, используя учетные данные для доступа к БД (рис. 71).

```
import sqlite3

conn = sqlite3.connect('example.db')
```

Рис. 71. Создание соединения с SQLite-БД, используя модуль sqlite3

Второй шаг – создание таблиц и полей в БД. Для этого нужно выполнить SQL-запросы для создания таблиц и полей в БД. В примере (рис. 72) мы используем метод cursor() для создания курсора, который позволяет выполнить SQL-запрос в БД, и метод execute() для выполнения SQL-запроса для создания таблицы users в БД.

```
cur = conn.cursor()
cur.execute(
    """
    CREATE TABLE users (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        email TEXT NOT NULL
    )
    """
)
```

Рис. 72. Создание таблицы

Третий шаг – внесение изменений в таблицы БД. Для этого нужно выполнить SQL-запросы для добавления, изменения или удаления данных в таблицах БД. В примере (рис. 73) мы используем метод execute() для выполнения SQL-запроса для добавления нового пользователя в таблицу users в БД.

```
cur.execute(
    """
    INSERT INTO users (name, email)
    VALUES (?, ?)
    """,
    ("John Doe", "john.doe@example.com")
)
```

Рис. 73. Внесение изменений в таблицу

Четвертый шаг – получение данных из таблиц БД. Для этого нужно выполнить SQL-запросы для выборки данных из таблиц БД. В примере (рис. 74) мы используем метод execute() для выполнения SQL-запроса для выборки всех данных из таблицы users в БД, и метод fetchall() для получения всех строк, соответствующих запросу.

```
cur.execute(
    """
    SELECT * FROM users
    """
)

rows = cur.fetchall()
for row in rows:
    print(row)
```

Рис. 74. Получение данных

Пятый шаг – закрытие соединения с БД. Для этого нужно использовать метод close() для закрытия соединения с БД.

Таким образом, работа с БД в Python предполагает использование соответствующих библиотек для работы с БД, создание соединения с БД, выполнение SQL-запросов для создания таблиц и полей, внесения изменений в таблицы БД, получения данных из таблиц БД и закрытия соединения с БД. Важно следовать правильной практике и проводить тестирование перед работой с БД, чтобы избежать проблем в работе приложения.

Модели в Django

Модели – это объектно-реляционное отображение (ORM), которое позволяет определить структуру базы данных приложения. Модели могут содержать поля, методы и другие свойства, которые определяют, как данные будут храниться и взаимодействовать с базой данных.

Модели в Django – это классы Python, которые определяют структуру таблиц в базе данных. Они используются для хранения данных в приложениях Django. Каждый атрибут класса модели представляет собой поле таблицы в базе данных, а каждый экземпляр класса модели представляет отдельную запись в этой таблице.

Модели в Django включают в себя поля (поля), которые определяют атрибуты данных. Например, целое поле CharField представляет собой состав текста, а поле IntegerField представляет собой число. Также используются модели методов, которые позволяют работать с данными базы данных.

Чтобы создать модель в Django, нужно создать класс, наследующийся от класса models.Model. Каждый класс модели должен иметь атрибуты и методы, которые соответствуют структуре данных объекта.

Пример создания модели представлен на рис. 75.

```
class User(models.Model):
    id = models.IntegerField(primary_key=True)  # уникальный идентификатор пользователя
    username = models.CharField(max_length=50)  # имя пользователя
    password = models.CharField(max_length=32)  # пароль
    email = models.EmailField(unique=True)     # электронная почта

    class Meta:
        verbose_name = 'Пользователь'  # название модели
        verbose_name_plural = 'Пользователи'    # название модели во множественном числе
```

Рис. 75. Пример создания модели

В данном примере мы создали модель User, которая имеет четыре поля: id, username, password и email. Также модель имеет метаданные, которые используются для настройки модели. Метаданные определяют название модели и ее название во множественном числе.

Созданную модель можно использовать в приложении Django. Например, можно создать объект модели User и сохранить его в базе данных (рис. 76). При сохранении объекта в базу данных Django автоматически создает уникальный идентификатор (id) для объекта.

```
# создание объекта модели User
user = User(username='admin', password='password', email='admin@example.com')

# сохранение объекта в базе данных
user.save()
```

Рис. 76. Создание и сохранение объекта

Модели в Django используются в приложениях для работы с базой данных. Они позволяют определить структуру данных, которые хранятся в базе данных, а также методы для работы с шестью данными. Модели используются для создания, чтения, обновления и удаления данных в базе данных.

При определении модели в Django необходимо определить ее поля связи и метаданные, такие как имя таблицы в базе данных, настройки для сортировки, фильтрации и т. д. На рис. 77 представлен пример определения модели в Django.

```
from django.db import models
class Book(models.Model):
    title = models.CharField(max_length=255)
    author = models.CharField(max_length=255)
    published_date = models.DateField()
    is_published = models.BooleanField(default=False)
    def __str__(self):
        return self.title
```

Рис. 77. Пример определения модели в Django

Здесь видна модель книги с полями: название, автор, дата издания и когда опубликовано. Поля title и author определены как CharField, publish_date как DateField и is_published как BooleanField. Также определен метод str (), который возвращает строковое представление объекта.

Модели в Django являются основой для работы с базой данных во многих приложениях. Они позволяют определить структуру данных и методы для работы с ними, что позволяет развивать приложения и улучшать производительность при работе с базой данных.

Модели Django обычно встречаются в файле models.py, который находится внутри приложений Django. Каждая модель представляет собой отдельную таблицу в базе данных и содержит поля, которые определяют структуру структуры. За счет использования моделей Django можно не только определить структуру данных, но и выде-

лить ограничения на данные, хранящиеся в ней, задавать отношения между таблицами, а также создавать методы для работы с данными.

Кроме того, модели Django используют интеграцию с ORM (Object-Relational Mapping), что позволяет работать с данными в базе данных, используя объекты Python, а не SQL-запросы. ORM позволяет упростить работу с базовыми данными, сделать более читаемым и легко обнаруживаемым.

Рассмотрим пример программы на языке Python, работающей с шаблонами в Django. Программа представляет собой отдельный блог, в котором пользователи создают посты и размещают комментарии к ним. Для этого создадим две модели: «Пост» и «Комментарий».

Для модели «Пост» определим поля:
– титул (заголовок поста);
– содержание (текст поста);
– автор (автор поста);
– created_at (дата создания поста).

Для модели «Комментарий» определим поля:
– post (ссылка на пост, к этой оставленный комментарий);
– автор (автор комментария);
– контент (текст комментария);
– created_at (дата создания комментария).

На рис. 78 представлен программный код, демонстрирующий описание создаваемых моделей.

Здесь представлены две модели: «Пост» и «Комментарий». Каждая модель наследуется от класса models.Model. Поля моделей зависят от классов CharField, TextFieldи DateTimeField. Также мы определили отношение «один-ко-многим» между шаблонами Post and Comment, используя поле ForeignKey. Это означает, что каждый пост может иметь несколько связанных комментариев, но каждый комментарий относится только к одному посту. В методе __str__ определили, что будет показано при выводе объектов модели в консоль. Для модели «Пост» будет выводиться ее заголовок, а для модели «Комментарий» будет выводиться имя автора и заголовок поста, к которой оставлен комментарий.

```python
# импортируем необходимые модули
from django.db import models
from django.contrib.auth.models import User

# модель поста
class Post(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

# модель комментария
class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.author.username}\'s comment on {self.post.title}'
```

Рис. 78. Описание моделей

ORM Django

Для работы с данными в базе данных используется ORM Django, который предоставляет множество методов, упрощающих написание и понимание кода. Кроме того, ORM автоматически генерирует SQL-запросы для операций с базой данных, что снижает вероятность ошибок и ускоряет процесс разработки.

ORM (Object-Relational Mapping) в Django – высокоуровневый интерфейс для работы с базой данных, который позволяет работать с данными в объектно ориентированном стиле. ORM Django позволяет избежать написания SQL-запросов на языке баз данных и вместо этого предоставляет набор методов для работы с данными.

ORM (Object-Relational Mapping) – это библиотека, которая позволяет абстрагироваться от физической реализации базы данных и работать с объектами в памяти, а не с таблицами в базе данных.

В Python есть несколько ORM-библиотек, которые можно использовать для работы с базами данных. Рассмотрим некоторые из наиболее популярных ORM-библиотек в Python:

– SQLAlchemy – одна из самых популярных ORM-библиотек для Python. Она позволяет работать с различными типами баз данных, такими как MySQL, PostgreSQL, Oracle и другими.

SQLAlchemy предоставляет мощные функции для работы с запросами, представлениями и транзакциями.

– Django – это ORM-библиотека, входящая в состав фреймворка Django для веб-разработки на Python, который также поддерживает ORM. Она позволяет работать с базой данных, используя модели Django, которые являются классами Python, связанными с таблицами базы данных.

– Pyramid – еще один фреймворк для разработки веб-приложений на Python. Pyramid также поддерживает ORM и позволяет создавать приложения с использованием модульной архитектуры.

– Flask – микрофреймворк для создания веб-приложений на Python, который также имеет поддержку ORM. Flask позволяет создавать быстрые и масштабируемые приложения.

– Peewee – минималистичная ORM-библиотека для Python, которая поддерживает реляционные базы данных, такие как SQLite, MySQL и PostgreSQL.

Выбор конкретной ORM-библиотеки зависит от потребностей и требований проекта. Каждая из них имеет свои преимущества и недостатки, и выбор зависит от того, какие функции и возможности необходимы для конкретного проекта.

Для работы с базой данных, связанной с моделью данных, необходимо создать менеджер объекта для этой модели. Методы менеджера объектов модели в Django представляют собой функции, которые могут быть вызваны на менеджере объектов модели, чтобы выполнить определенные действия или получить информацию о выбранных объектах модели.

Django предоставляет различные методы для работы с данными, такие как:

– метод all(). Получение всех объектов модели. В коде (рис. 79) мы импортируем модель MyModel из модуля models приложения myapp. Затем вызываем метод all() на менеджере объектов MyModel.objects, чтобы получить все объекты модели MyModel, которые хранятся в базе данных. Результат вызова метода all() – это QuerySet-объект, который представляет собой запрос к базе данных, выбирающий все объекты модели MyModel из базы данных. QuerySet-объект может быть использован для выполнения допол-

нительных операций, таких как фильтрация, сортировка или ограничение количества выбранных записей;

```
from myapp.models import MyModel
all_objects = MyModel.objects.all()
```

Рис. 79. Метод all()

– метод filter(). Получение объектов модели, удовлетворяющих определенному условию. В коде (рис. 80) мы импортируем модель MyModel из модуля models приложения myapp. Затем вызываем метод filter() на менеджере объектов MyModel.objects, чтобы получить объекты модели MyModel, у которых значение поля name равно ‘John’. Результат вызова метода filter() – это QuerySet-объект, который представляет собой запрос к базе данных, выбирающий объекты модели MyModel, у которых значение поля name равно ‘John’. Этот QuerySet-объект может быть использован для выполнения дополнительных операций, таких как сортировка, ограничение количества выбранных записей или выполнение других фильтров;

```
from myapp.models import MyModel
filtered_objects = MyModel.objects.filter(name='John')
```

Рис. 80. Метод filter()

– метод get(). Получение единственного объекта модели, удовлетворяющего определенному условию. В коде (рис. 81) мы импортируем модель MyModel из модуля models приложения myapp. Затем вызываем метод get() на менеджере объектов MyModel.objects, чтобы получить единственный объект модели MyModel, у которого значение поля name равно ‘John’. Результат вызова метода get() – это единственный объект модели MyModel, который удовлетворяет определенному условию. Если объект не найден или найдено больше одного объекта, которые удовлетворяют условию, то будет выброшено исключение;

```
from myapp.models import MyModel
specific_object = MyModel.objects.get(name='John')
```

Рис. 81. Метод get()

– метод update(). Обновление значений полей у объектов модели, удовлетворяющих определенному условию. В коде (рис. 82) мы импортируем модель MyModel из модуля models приложения myapp. Затем вызываем метод filter() на менеджере объектов MyModel.objects, чтобы выбрать объекты модели MyModel, у которых значение поля name равно ‘John’. Затем вызываем метод update() на QuerySet-объекте, чтобы обновить значение поля age у всех выбранных объектов на 30. Результат вызова метода update() – это количество обновленных записей в базе данных. Если не было выбрано ни одного объекта, удовлетворяющего условию, то никакие записи не будут обновлены;

```
from myapp.models import MyModel
MyModel.objects.filter(name='John').update(age=30)
```

Рис. 82. Метод update()

– метод delete(). Удаление объектов модели, удовлетворяющих определенному условию. В коде (рис. 83) мы импортируем модель MyModel из модуля models приложения myapp. Затем вызываем метод filter() на менеджере объектов MyModel.objects, чтобы выбрать объекты модели MyModel, у которых значение поля name равно ‘John’. Затем вызываем метод delete() на QuerySet-объекте, чтобы удалить все выбранные объекты из базы данных. Результат вызова метода delete() – это количество удаленных записей в базе данных. Если не было выбрано ни одного объекта, удовлетворяющего условию, то никакие записи не будут удалены.

```
from myapp.models import MyModel
MyModel.objects.filter(name='John').delete()
```

Рис. 83. Метод delete()

Методы менеджера объектов модели в Django предоставляют различные способы выборки и обработки объектов модели в базе данных, что делает работу с моделями очень гибкой и удобной.

Рассмотрим пример (рис. 84) программы на Python для создания модели «Сотрудник» в Django и применим методы save() и all().

```python
# импортируем модуль Django и создаем класс модели "Сотрудник"
from django.db import models

class Employee(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    job_title = models.CharField(max_length=50)
    salary = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return f"{self.first_name} {self.last_name} - {self.job_title} ({self.salary})"

    def save(self, *args, **kwargs):
        if self.salary < 0:
            raise ValueError("Salary cannot be negative")
        super().save(*args, **kwargs)

# используем методы save() и all()
from myapp.models import Employee

# Создание и сохранение объекта "Сотрудник"
employee = Employee(first_name="Иван", last_name="Иванов", job_title="Менеджер", salary=50000)
employee.save()

# Получение всех объектов "Сотрудник"
all_employees = Employee.objects.all()
for employee in all_employees: print(employee)
```

Рис. 84. Пример создания модели и применения методов

Для начала необходимо импортировать модуль Django и создать класс модели «Сотрудник» с помощью наследования от базового класса модели Django django.db.models.Model. Затем нужно определить поля модели и их типы данных.

В этом примере мы создали модель «Сотрудник» с полями: first_name (имя), last_name (фамилия), job_title (должность) и salary (зарплата). Поле salary имеет тип DecimalField с максимальным числом цифр 10 и 2 знаками после запятой. Метод __str__ определен, чтобы возвращать строковое представление объекта при его выводе в консоль или в административном интерфейсе Django. Метод save переопределен, чтобы проверять, что зарплата не отрицательная, перед сохранением объекта в базе данных. Для использования методов save() и all() нужно создать объекты модели в Django-приложении.

В результате выполнения этой программы вы получите список всех объектов «Сотрудник», которые были сохранены в базе данных, и их строковое представление, определенное в методе __str__.

Миграции в Django

Это механизм, который позволяет автоматически создавать и обновлять схему базы данных на основе изменений, внесенных в модели Django. Миграции в Django используются для того, чтобы сохранить изменения в структуре базы данных и обеспечить совместимость между моделями и базой данных. Миграции позволяют создавать, изменять и удалять таблицы и поля в базе данных, а также управлять индексами, ограничениями и другими аспектами базы данных. Миграции создаются на основе изменений в моделях Django. Каждый раз, когда нами вносятся изменения в модель, Django автоматически создает новую миграцию, которую можно применить к базе данных. Миграции хранятся в папке migrations внутри каждого приложения Django.

Каждая миграция представляет собой изменение базы данных в виде Python-кода, который Django может автоматически применить к базе данных. Код миграции описывает, как изменить структуру базы данных, добавляя, изменяя или удаляя таблицы, поля или индексы. Эти миграции создаются с помощью команды makemigrations из утилиты управления Django.

Миграции в Django позволяют разработчикам легко изменять структуру базы данных, не переживая о том, что они случайно потеряют данные. Миграции также позволяют разработчикам работать с базой данных в команде, обмениваясь миграциями, чтобы каждый мог применить изменения к своей локальной копии базы данных.

Миграции в Django – это система управления схемой базы данных, которая позволяет автоматически создавать, изменять и удалять таблицы и поля в базе данных в соответствии с изменениями в моделях Django. Это делает процесс разработки приложений более гибким и удобным, позволяя сохранять изменения в базе данных вместе с изменениями в коде приложения.

Процесс создания миграций в Django включает несколько шагов.

• Определение моделей Django. Создайте модели Django, которые будут представлять таблицы в базе данных. Модели определяются в файле models.py внутри Django-приложения и могут со-

держать поля с различными типами данных, такими как CharField, IntegerField, BooleanField и т. д.

• Создание миграции. Создайте миграцию с помощью команды **python manage.py makemigrations** в консоли. Эта команда создаст файл миграции в папке migrations внутри Django-приложения. В этом файле определены изменения в базе данных, которые нужно выполнить, чтобы привести ее в соответствие с определенными моделями.

• Применение миграции. Примените миграцию с помощью команды **python manage.py migrate** в консоли. Эта команда применяет изменения, определенные в файле миграции, к базе данных. После выполнения этой команды таблицы и поля, определенные в моделях, будут созданы в базе данных.

• Проверка базы данных. Проверьте базу данных, чтобы убедиться, что таблицы и поля были созданы правильно. Можно использовать административный интерфейс Django или инструменты для работы с базой данных, такие как SQL-клиенты или команды для работы с базой данных.

• Изменение моделей и создание новых миграций. Если нами внесены изменения в модели Django, которые требуют изменения базы данных, нужно создать новую миграцию и применить ее, используя те же команды, рассмотренные ранее. Django автоматически определит изменения в моделях и создаст миграцию, которая изменит базу данных в соответствии с ними.

Для более подробного объяснения миграций в Django рассмотрим пример, работу которого опишем пошагово.

Первый шаг. Определение модели Django (рис. 85). Допустим, у нас есть Django-приложение blog, и нужно создать модель Post для хранения постов блога в базе данных. Для этого нужно определить модель в айле models.py внутри приложения.

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    published_date = models.DateTimeField(auto_now_add=True)
```

Рис. 85. Определение модели

Второй шаг. Создание миграции (рис. 86). Чтобы создать миграцию для этой модели, выполним команду **python manage.py makemigrations blog** в консоли. blog здесь – название вашего Django-приложения. Эта команда создаст файл миграции в папке migrations внутри приложения.

```
from django.db import migrations, models

class Migration(migrations.Migration):

    dependencies = [
        ('blog', '0001_initial'),
    ]

    operations = [
        migrations.CreateModel(
            name='Post',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=200)),
                ('content', models.TextField()),
                ('published_date', models.DateTimeField(auto_now_add=True)),
            ],
        ),
    ]
```

Рис. 86. Создание миграции

Третий шаг. Применение миграции. Чтобы применить миграцию и создать таблицу Post в базе данных, выполним команду python manage.py migrate blog в консоли. Эта команда применит все отложенные миграции в приложении blog, включая новую миграцию для создания таблицы Post.

Четвертый шаг. Проверка базы данных. После выполнения миграции можно проверить базу данных и убедиться, что таблица ‘Post’ была создана. Для этого можно использовать административный интерфейс Django, чтобы просмотреть таблицы базы данных.

Пятый шаг. Изменение моделей и создание новых миграций. Если мы внесли изменения в модели Django, нужно создать новую миграцию и применить ее, используя те же команды. Например, если добавили новое поле author в модель Post, то можно создать новую миграцию с помощью команды **python manage.py makemigrations blog** и применить ее с помощью команды **python manage.py migrate blog**. Django автоматически определит изменения в моделях и создаст миграцию, которая изменит базу данных в соответствии с ними.

Таким образом, процесс создания миграций в Django состоит из определения моделей Django, создания миграции, применения миграции, проверки базы данных и создания новых миграций при изменении моделей.

Тема 6. Django ORM: работа с данными и формами

Получение доступа к данным

Получение доступа к данным в проекте с использованием Django ORM относится к способам работы с данными в проектах, созданных на основе фреймворка Django с использованием объектно-реляционной модели (ORM).

Для получения доступа к данным в проекте с использованием Django ORM разработчики могут использовать различные методы, такие как:

– моделирование базы данных: Django ORM позволяет создавать модели, которые представляют таблицы в базе данных. Эти модели определяют структуру и связи между таблицами. Модели могут быть созданы с помощью классов Python, что позволяет описать структуру таблицы и задать отношения между таблицами;

– использование менеджеров объектов: Django ORM предоставляет менеджеры объектов, которые позволяют получать доступ к данным в базе данных с помощью простых методов Python. Менеджеры объектов могут использоваться для выполнения запросов к базе данных, включая выборку данных, фильтрацию, сортировку и агрегацию;

– использование запросов: Django ORM позволяет создавать запросы к базе данных, которые можно использовать для получения данных из базы данных. Запросы могут быть созданы с помощью методов менеджеров объектов или напрямую с помощью моделей;

– использование ORM-инструментов: Django ORM предоставляет множество инструментов для работы с данными, таких как миграции базы данных, механизмы кэширования и многое другое. Эти инструменты позволяют разработчикам управлять базой данных и работать с данными более эффективно.

Django ORM предоставляет удобный и интуитивно понятный способ работы с данными в проектах Django, что позволяет разработчикам быстро и эффективно создавать приложения, которые используют базы данных.

Рассмотрим примеры, которые демонстрируют, как можно получить доступ к данным в проекте с использованием Django ORM.

1. Моделирование базы данных. Предположим, у нас есть проект, который отслеживает заказы в интернет-магазине. Для хранения информации о заказах можно создать модель, которая будет представлять таблицу «Заказы» в базе данных. На рис. 87 представлен фрагмент кода. В этом примере мы создали модель Order, которая наследуется от базовой модели models.Model. Модель содержит три поля: customer_name, order_date и total_amount. Каждое поле определено с помощью класса models.CharField или models.DateTimeField и задает тип данных, который будет храниться в соответствующем столбце таблицы.

```
from django.db import models

class Order(models.Model):
    customer_name = models.CharField(max_length=100)
    order_date = models.DateTimeField(auto_now_add=True)
    total_amount = models.DecimalField(max_digits=8, decimal_places=2)
```

Рис. 87. Моделирование базы данных

2. Использование менеджеров объектов. Предположим, нужно получить список всех заказов, которые были сделаны в последние 7 дней. Для этого можно использовать менеджер объектов objects для модели Order и метод filter, который позволит нам отфильтровать заказы по дате. Код представлен на рис. 88. В этом примере мы создали переменную last_week, которая содержит дату, которая была 7 дней назад. Затем использовали метод filter для модели Order, чтобы получить список заказов, где дата заказа больше или равна last_week.

```
from datetime import datetime, timedelta

last_week = datetime.now() - timedelta(days=7)
orders = Order.objects.filter(order_date__gte=last_week)
```

Рис. 88. Использование менеджеров объектов

3. Использование запросов. Можно использовать запросы для получения данных из базы данных, используя Django ORM. Например, можно получить список всех заказов и отсортировать их по дате заказа. Код представлен на рис. 89. В этом примере мы использовали метод order_by для модели Order, чтобы отсортировать заказы по убыванию даты заказа.

```
orders = Order.objects.order_by('-order_date')
```

Рис. 89. Использование запросов

4. Использование ORM-инструментов. Django ORM также предоставляет различные инструменты для управления базой данных и работой с данными. Например, можно использовать механизм миграций для обновления структуры базы данных. Код представлен на рис. 90. Эти команды создадут миграционные файлы и применят их к базе данных, чтобы обновить ее структуру.

```
python manage.py makemigrations
python manage.py migrate
```

Рис. 90. Использование ORM-инструментов

Django ORM предоставляет удобный и интуитивно понятный способ работы с данными в проектах Django.

Методы для выполнения запросов в базу данных

Django ORM предоставляет различные методы для выполнения запросов в базу данных, включая выборку данных, фильтрацию, сортировку и агрегацию.

Рассмотрим некоторые примеры запросов, которые могут использоваться для получения данных в проекте Django ORM.

1. Выборка всех объектов (рис. 91). Для выборки всех объектов из таблицы в базе данных можно использовать метод all. В этом примере мы импортировали модель MyModel из приложения mayapp и использовали метод all для получения всех объектов из соответствующей таблицы в базе данных.

```
from myapp.models import MyModel

all_objects = MyModel.objects.all()
```

Рис. 91. Выборка всех объектов

2. Фильтрация данных (рис. 92). Для фильтрации данных в базе данных можно использовать метод filter. Например, можно выбрать все объекты, у которых значение поля name равно ‘John’. В этом

примере мы использовали метод filter для модели MyModel и указали условие фильтрации, где поле name должно быть равно ‘John’.

```
from myapp.models import MyModel

john_objects = MyModel.objects.filter(name='John')
```

Рис. 92. Фильтрация данных

3. Сортировка данных (рис. 93). Для сортировки данных в базе данных можно использовать метод order_by. Например, можно отсортировать все объекты модели MyModel по возрастанию значения поля id. В этом примере мы использовали метод order_b для модели MyModel и указали поле ‘id’ как поле для сортировки.

```
from myapp.models import MyModel

sorted_objects = MyModel.objects.order_by('id')
```

Рис. 93. Сортировка данных

4. Агрегация данных (рис. 94). Для агрегации данных в базе данных можно использовать методы count, sum, avg и другие. Например, можно вычислить сумму всех значений поля ‘price’ в модели MyModel. В этом примере мы использовали метод aggregate для модели MyModel и указали функцию Sum, которая вычисляет сумму значений поля ‘price’. Результат вычислений сохраняется в словаре с ключом total_price.

```
from myapp.models import MyModel

total_price = MyModel.objects.aggregate(total_price=Sum('price'))
```

Рис. 94. Агрегация данных

Использование запросов для получения данных в проекте Django ORM является одним из основных способов работы с базой данных в приложениях Django. Разработчики могут использовать различные методы, такие как all, filter, order_by, aggregate и другие, чтобы получать данные из базы данных и выполнять различные операции с ними.

Инструменты по работе с данными, хранящимися в разных таблицах

Django ORM (Object-Relational Mapping) является мощным инструментом для работы с данными, хранящимися в различных таблицах базы данных. Он позволяет работать с данными, используя объектно ориентированный подход, и автоматически преобразовывает данные между объектами Python и записями в базе данных.

Для работы с данными, хранящимися в разных таблицах с использованием Django ORM, необходимо использовать связи между моделями. В Django ORM есть несколько типов связей:

– OneToOneField – связь «один-к-одному»;
– ForeignKey – связь «один-ко-многим»;
– ManyToManyField – связь «многие-ко-многим».

Для работы с данными в Django ORM необходимо выполнить следующие шаги:

– определить модели данных и их поля;
– определить связи между моделями;
– создать или обновить базу данных с помощью миграций;
– выполнить операции с данными с использованием Django ORM.

Работа с данными, хранящимися в разных таблицах с использованием Django ORM, является одной из основных задач при работе с базой данных в проектах Django. В проектах, особенно крупных, данные часто хранятся в нескольких таблицах, связанных между собой ключами. Django ORM предоставляет множество инструментов для работы с такими данными.

Рассмотрим некоторые из инструментов по работе с данными, хранящимися в разных таблицах.

1. Определение связей между моделями (рис. 95). Для работы с данными, хранящимися в разных таблицах, необходимо определить связи между моделями. Django ORM поддерживает несколько типов связей, таких как «один-к-одному», «один-ко-многим» и «многие-ко-многим». Например, если у нас есть модели Author и Book, можно определить связь «один-ко-многим», где каждый автор может иметь несколько книг. В этом примере мы определили модели Author и Book и связь «многие-ко-многим» между ними. У каждой книги есть поле author, которое является внешним ключом, связывающим ее с соответствующим автором.

```python
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
```

Рис. 95. Определение связей между моделями

2. Использование связей для доступа к данным (рис. 96). После определения связей между моделями можно использовать их для доступа к данным. Например, можно получить список всех книг, написанных определенным автором, используя связь между моделями Author и Book. В этом примере мы сначала получили объект автора с именем «John Smith» из базы данных, используя метод get для модели Author. Затем использовали связь book_set, которая автоматически создается Django ORM для связи «один-ко-многим» между моделями Author и Book, чтобы получить список всех книг, написанных этим автором.

```python
author = Author.objects.get(name='John Smith')
books = author.book_set.all()
```

Рис. 96. Использование связей для доступа к данным

3. Использование обратных связей (рис. 97). Для доступа к связанным данным в обратном направлении, то есть от модели, у которой есть внешний ключ, к модели, на которую он ссылается, можно использовать обратные связи. Например, можно получить список всех авторов, написавших книги с определенным заголовком. В этом примере мы сначала получили объект книги с заголовком «The Great Gatsby» из базы данных, используя метод get для модели Book. Затем использовали обратную связь author, которая автоматически создается Django ORM для связи «многие-ко-многим» между моделями Author и Book, чтобы получить список всех авторов, которые написали эту книгу.

```python
book = Book.objects.get(title='The Great Gatsby')
authors = book.author.all()
```

Рис. 97. Использование обратных связей

Работа с данными, хранящимися в разных таблицах с использованием Django ORM, требует определения связей между моделями и использования соответствующих инструментов для доступа к данным.

Рассмотрим конкретный пример работы с данными, хранящимися в разных таблицах, с использованием Django ORM (рис. 98).

```
from django.db import models # создание моделей

class Author(models.Model):
    name = models.CharField(max_length=100)
    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    def __str__(self):
        return self.title

author = Author(name='John Smith') # создание и сохранение нового автора и книги
author.save()
book = Book(title='The Great Gatsby', author=author)
book.save()

john_smith_books = Book.objects.filter(author__name='John Smith') # книги определенного автора

author = Author.objects.get(name='John Smith') # изменение автора
author.name = 'Jane Doe'
author.save()

book = Book.objects.get(title='The Great Gatsby') # удаление книги
book.delete()
```

Рис. 98. Пример кода для создания моделей и решения задачи

В этом примере мы сначала определили две модели – Author и Book с помощью класса models.Model из Django ORM. У каждой модели есть свои поля, которые соответствуют столбцам в базе данных. У модели Author есть поле name, которое является строковым полем с максимальной длиной 100 символов. У модели Book есть поле title, которое также является строковым полем с максимальной длиной 200 символов, и внешний ключ author, который связывает книгу с соответствующим автором. Метод str в каждой модели определяет строковое представление объекта. В нашем случае он возвращает имя автора для модели Author и заголовок книги для модели Book.

Мы создали нового автора с именем ‘John Smith’ и сохранили его в базе данных, используя метод save для модели Author. Затем создали новую книгу с заголовком ‘The Great Gatsby’ и связали

ее с автором ‘John Smith’, используя внешний ключ author. После этого сохранили книгу в базе данных, используя метод save для модели Book. Использовали метод filter для модели Book и указали условие фильтрации, где поле author__name должно быть равно имени автора. Для изменения автора сначала получили объект автора с именем ‘John Smith’ из базы данных, используя метод get для модели Author. Затем изменили имя автора на ‘Jane Doe’ и сохранили изменения в базе данных, используя метод save для модели Author. Для удаления книги сначала получили объект книги с заголовком ‘The Great Gatsby’ из базы данных, используя метод get для модели Book. Затем вызвали метод delete для объекта книги, чтобы удалить его из базы данных.

Связи и получение данных

Django ORM предоставляет удобный интерфейс для работы с данными, хранящимися в разных таблицах и связанными между собой. Одним из типов связей, которые можно использовать в Django ORM, является связь «многие-ко-многим» (many-to-many).

Связь «многие-ко-многим» используется, когда каждый объект в одной таблице может быть связан с несколькими объектами в другой таблице, и наоборот. Например, модель Book может иметь несколько авторов, а модель Author может иметь несколько книг. В этом случае мы создаем промежуточную таблицу, которая связывает объекты из обеих таблиц.

Для определения связи «многие-ко-многим» в Django ORM можно использовать поле ManyToManyField. Например, если мы хотим связать модели Book и Author, можно определить поле authors в модели Book так, как представлено на рис. 99.

```
# определение связи много-ко-многим
class Author(models.Model):
    name = models.CharField(max_length=100)

class Book(models.Model):
    title = models.CharField(max_length=200)
    authors = models.ManyToManyField(Author)
```

Рис. 99. Определение связи «многие-ко-многим»

Чтобы добавить связанный объект к модели, можно использовать метод add для поля ManyToManyField. Например, чтобы добавить автора к книге, можно использовать следующий код, показанный на рис. 100. Здесь мы сначала получили объект книги с помощью метода get для модели Book и объект автора с помощью метода get для модели Author. Затем вызвали метод add для поля authors объекта книги, чтобы добавить объект автора в список связанных объектов.

```
# добавление связанного объекта к модели
book = Book.objects.get(id=1)
author = Author.objects.get(id=1)
book.authors.add(author)
```

Рис. 100. Добавить связанный объект к модели

Для получения связанных объектов можно использовать обратную связь. Например, чтобы получить всех авторов, связанных с книгой, можно использовать код на рис. 101. Здесь мы сначала получили объект книги с помощью метода get для модели Book. Затем использовали обратную связь authors для получения всех связанных объектов из промежуточной таблицы с помощью метода all.

```
# использование обратной связи
book = Book.objects.get(id=1)
authors = book.authors.all()
```

Рис. 101. Обратная связь

Для удаления связи можно использовать метод remove для поля ManyToManyField. Например, чтобы удалить автора из списка связанных объектов, можем использовать следующий код, позволяющий получить объект книги с помощью метода get для модели Book и объект автора с помощью метода get для модели Author. Затем мы вызвали метод remove для поля authors объекта книги, чтобы удалить объект автора из списка связанных объектов. Пример представлен на рис. 102.

```
# удаление связи
book = Book.objects.get(id=1)
author = Author.objects.get(id=1)
book.authors.remove(author)
```

Рис. 102. Удаление связи

Получение записей из связанных таблиц является одной из ключевых операций при работе с реляционными базами данных. В Django ORM это можно сделать с помощью использования связей между таблицами и методов доступа к связанным объектам.

Еще одним из типов связей, которые можно использовать в Django ORM, является связь «один-ко-многим» (one-to-many). В этом случае один объект в одной таблице связан с несколькими объектами в другой таблице. Например, модель Author может иметь несколько книг. В этом случае мы определяем в модели Book внешний ключ, который указывает на модель Author (рис. 103).

```
# определение связи один-ко-многим

class Author(models.Model):
    name = models.CharField(max_length=100)
class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
```

Рис. 103. Связь «один-ко-многим»

Чтобы получить все книги, связанные с определенным автором, можно использовать метод filter для модели Book и указать условие фильтрации, где поле author должно быть равно объекту автора. В примере (рис. 104) мы сначала получили объект автора с помощью метода get для модели Author. Затем использовали метод filter для модели Book и указали условие фильтрации, где поле author должно быть равно объекту автора. Метод filter возвращает QuerySet, который содержит все книги, связанные с этим автором.

```
# использование метода filter для получения данных

author = Author.objects.get(id=1)
books = Book.objects.filter(author=author)
```

Рис. 104. Использование метода filter

Если нам нужно получить связанные объекты из обратной связи, можно использовать метод related_name. Например, если хотим получить все книги, связанные с автором, можно использовать следующий код, показанный на рис. 105. Здесь мы определили обратную связь book_set в модели Author, которая позволяет получить все

связанные объекты из модели Book. Метод all возвращает QuerySet, который содержит все книги, связанные с этим автором.

```
# получение связанных объектов из обратной связи

author = Author.objects.get(id=1)
books = author.book_set.all()
```

Рис. 105. Получить связанные объекты из обратной связи

Также можно использовать связь «многие-ко-многим» (many-to-many), позволяющая связывать объекты из двух таблиц, каждый из которых может быть связан с несколькими объектами из другой таблицы. Например, модель Book может иметь несколько категорий, а модель Category может иметь несколько книг (рис. 106).

```
# использование связи много-ко-многим

class Category(models.Model):
    name = models.CharField(max_length=100)
class Book(models.Model):
    title = models.CharField(max_length=200)
    categories = models.ManyToManyField(Category)
```

Рис. 106. Связать объекты из двух таблиц

Чтобы получить все книги, связанные с определенной категорией, можно использовать метод filter для модели Book и указать условие фильтрации, где поле categories должно содержать объект категории. Здесь мы сначала получили объект категории с помощью метода get для модели Category. Затем использовали метод filter для модели Book и указали условие фильтрации, где поле categories должно содержать объект категории. Метод filter возвращает QuerySet, который содержит все книги, связанные с этой категорией (рис. 107).

```
#  получение книг из категории

category = Category.objects.get(id=1)
books = Book.objects.filter(categories=category)
```

Рис. 107. Получение объектов из категории

Для получения связанных объектов можно использовать обратную связь. Например, чтобы получить все категории, связанные с книгой, можно использовать следующий код, показанный на рис. 108.

```
# получение связанных объектов

book = Book.objects.get(id=1)
categories = book.categories.all()
```

Рис. 108. Получение связанных объектов

Здесь мы сначала получили объект книги с помощью метода get для модели Book. Затем использовали обратную связь categories для получения всех связанных объектов из промежуточной таблицы с помощью метода all.

Получение записей из связанных таблиц в Django ORM может быть достигнуто с помощью определения связей между таблицами, использования методов доступа к связанным объектам и обратной связи.

Управление данными

Одной из ключевых задач в разработке веб-приложения является управление данными.

Управление данными в Django-приложении – это процесс создания, чтения, обновления и удаления данных в базе данных, используя модели Django и ORM (объектно-реляционное отображение). ORM в Django позволяет работать с базой данных, используя объекты Python вместо SQL-запросов, что делает процесс управления данными более удобным и интуитивно понятным.

Для управления данными в Django-приложении необходимо выполнить следующие действия.

1. Определение модели (рис. 109). Модель – это класс Python, который определяет структуру таблицы в базе данных. Он определяет название таблицы, ее поля и типы данных. Для определения модели в Django используется класс Model, который наследуется от базового класса models.Model.

```
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=200)
    body = models.TextField()
    pub_date = models.DateTimeField(auto_now_add=True)
```

Рис. 109. Определение модели для таблицы «Статьи»

2. Создание миграций. После определения модели необходимо создать миграции, которые применят изменения в базе данных. Миграции – это способ изменения структуры базы данных без потери данных. Django создает миграции автоматически при изменении модели.

Для создания миграций в Django используется команда **makemigrations: python manage.py makemigrations**.

3. Применение миграций. После создания миграций их необходимо применить к базе данных. Для этого используется команда **migrate: python manage.py migrate**.

4. Создание объектов модели. Для создания новых объектов в базе данных необходимо создать экземпляр модели и сохранить его, выполнив команду, представленную на рис. 110.

```
article = Article(title='Заголовок', body='Текст статьи')
article.save()
```

Рис. 110. Создание объектов модели

5. Получение объектов модели. Для получения объектов модели из базы данных используется метод objects.all(), который возвращает все объекты модели, с помощью команды на рис. 111. А для получения конкретного объекта модели используется метод objects.get().

```
articles = Article.objects.all()
article = Article.objects.get(id=1)
```

Рис. 111. Метод для возвращения объектов

6. Обновление объектов модели. Для обновления объектов модели необходимо получить объект из базы данных, изменить его атрибуты и сохранить его. Для этого выполняется код программы, представленный на рис. 112.

```
article = Article.objects.get(id=1)
article.title = 'Новый заголовок'
article.save()
```

Рис. 112. Обновление объектов модели

7. Удаление объектов модели. Для удаления объектов модели необходимо получить объект из базы данных и вызвать метод delete() (рис. 113).

```
article = Article.objects.get(id=1)
article.delete()
```

Рис. 113. Удаление объектов модели

Управление данными в Django-приложении – важная часть разработки веб-приложения, и правильное использование ORM и моделей может существенно упростить процесс работы с базой данных.

Инструменты управления данными

В Django для управления данными в приложении можно использовать несколько инструментов:

– ORM (Объектно-реляционное отображение) – инструмент, который позволяет работать с базой данных, используя объекты Python вместо SQL-запросов. ORM в Django позволяет создавать, обновлять и удалять данные в базе данных с помощью моделей;

– Django Admin – встроенный интерфейс администратора, который позволяет управлять данными в Django-приложении без написания кода. С помощью Django Admin можно легко создавать, изменять, удалять и просматривать объекты моделей в базе данных, а также настраивать пользовательские права доступа и группы;

– Django Forms – инструмент, который позволяет создавать HTML-формы для ввода и обработки данных. Django Forms позволяет создавать формы для создания, чтения, обновления и удаления данных в базе данных;

– Django Rest Framework – инструмент, который позволяет создавать API для взаимодействия с базой данных. Django Rest Framework позволяет создавать эндпойнты для создания, чтения, обновления и удаления данных в базе данных, а также предоставляет различные форматы данных для обмена информацией.

Сторонние инструменты. Существует множество сторонних инструментов для управления данными в Django-приложении, таких как Django Suit, Grappelli и Django Jet, которые предоставляют расширенный функционал для управления данными в приложении.

Рассмотрим инструмент Django Admin. Это встроенный интерфейс администратора, который позволяет управлять данными в Django-приложении без написания кода. С его помощью можно легко создавать, изменять, удалять и просматривать объекты моделей в базе данных, а также настраивать пользовательские права доступа и группы.

Для работы с ним нужно выполнить ряд действий:

– перед использованием Django Admin необходимо создать суперпользователя, который будет иметь полный доступ к интерфейсу администратора. Для этого необходимо выполнить команду **python manage.py createsuperuser**;

– для того чтобы модели были доступны в Django Admin, их необходимо зарегистрировать в файле admin.py внутри Django-приложения (рис. 114);

```
from django.contrib import admin
from .models import MyModel

admin.site.register(MyModel)
```

Рис. 114. Регистрация

– после регистрации моделей в Django Admin можно зайти в интерфейс администратора и через него управлять объектами моделей;

– для создания новых объектов моделей в Django Admin необходимо зайти в раздел «Добавить» соответствующей модели, заполнить поля формы и нажать «Сохранить»;

– для просмотра объектов моделей в Django Admin необходимо выбрать соответствующую модель в списке объектов, который доступен в разделе «Изменить». После выбора объекта можно просмотреть его атрибуты и связанные объекты;

– для обновления объектов моделей в Django Admin необходимо выбрать соответствующую модель в списке объектов, выбрать объект для редактирования и изменить его атрибуты. После изменения атрибутов необходимо нажать «Сохранить»;

– для удаления объектов моделей в Django Admin необходимо выбрать соответствующую модель в списке объектов, выбрать объект для удаления и нажать «Удалить». После подтверждения удаления объект будет удален из базы данных;

– Django Admin позволяет настраивать права доступа и группы пользователей. Для этого необходимо создать пользователей и группы и назначить соответствующие права доступа.

Django Admin позволяет настраивать пользовательские права доступа и группы, что делает его гибким и удобным инструментом для управления данными.

Django Admin – встроенная веб-административная панель, которая предоставляет интерфейс для управления данными в Django-приложении. Django Admin является одним из ключевых компонентов фреймворка Django и позволяет создавать пользовательские интерфейсы для управления базой данных, моделями и другими аспектами приложения.

Django Admin предоставляет множество функций, которые упрощают работу с приложением:

– автоматически создает интерфейс для управления моделями, которые вы определили в приложении. Это позволяет быстро создавать мощные административные панели без необходимости писать дополнительный код;

– позволяет настраивать представления данных, которые отображаются в административной панели. Вы можете определить, какие поля модели должны отображаться, какие действия пользователи могут выполнять, а также настраивать фильтры и поиск данных;

– является очень гибким и расширяемым. Вы можете создавать собственные представления данных, добавлять новые поля в интерфейс, создавать собственные действия и многое другое;

– предоставляет механизмы аутентификации и авторизации, которые позволяют контролировать доступ к административной панели вашего приложения;

– интегрируется с другими компонентами Django, такими как ORM, формы и шаблоны, что позволяет создавать более сложные административные панели.

Django Admin позволяет быстро создавать мощные административные панели для вашего приложения. Он предоставляет широкий набор инструментов для управления данными и позволяет настроить интерфейс в соответствии с вашими потребностями. Однако не всегда Django Admin является оптимальным решением для создания пользовательских интерфейсов, поэтому необходимо оценивать его преимущества и недостатки в каждом конкретном случае.

Для создания интерфейса управления моделью в Django Admin необходимо сделать следующие шаги.

1. Определите модель, для которой вы хотите создать интерфейс управления. Это может быть любая модель из приложения Django.

2. Создайте файл admin.py в каталоге, содержащем модель. Этот файл будет содержать код для настройки интерфейса управления.

3. Импортируйте модель, для которой вы хотите создать интерфейс управления, и класс ModelAdmin из модуля django.contrib.admin.

4. Создайте класс, унаследованный от ModelAdmin. В этом классе вы можете определить, какие поля модели будут отображаться в интерфейсе управления, какие поля будут доступны для редактирования, какие фильтры и поисковые поля будут использоваться и многое другое.

5. Зарегистрируйте модель и класс Admin в функции admin.site.register(), чтобы Django знал, как обрабатывать эту модель.

Например, есть модель Book с полями ‘title’, ‘author’ и ‘published_date’. Нужно создать интерфейс управления для этой модели. Это можно сделать так, как представлено на рис. 115.

```
# admin.py
from django.contrib import admin
from .models import Book

class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'published_date')
    list_filter = ('author', 'published_date')
    search_fields = ('title', 'author')

admin.site.register(Book, BookAdmin)
```

Рис. 115. Создать интерфейс управления для модели

Здесь нами создан класс BookAdmin, унаследованный от ModelAdmin. Определили, какие поля модели будут отображаться в списке записей, какие поля будут доступны для фильтрации и поиска. Затем зарегистрировали модель Book и класс BookAdmin в функции admin.site.register(). Теперь можно управлять записями в модели Book через интерфейс управления Django Admin.

В открытой странице администрирования становится доступным выполнение следующих функций.

• Добавление и редактирование записей. После открытия страницы администрирования вы увидите интерфейс для управления моделью Book. На этой странице можно добавлять, редактировать и удалять записи. В форме редактирования можно изменять значения полей и сохранять изменения.

• Фильтрация и поиск данных. Django Admin позволяет фильтровать и искать данные в базе данных.

• Настраиваемые представления данных. Django Admin позволяет настраивать представления данных, которые отображаются в административной панели. Например, вы можете определить, какие поля модели должны отображаться, какие действия пользователи могут выполнять, а также настраивать фильтры и поиск данных. Можно изменить отображение списка записей модели Book, чтобы отображать только название книги и автора, а также добавить фильтр по статусу доступности книги. Код представлен на рис. 116.

```
from django.contrib import admin
from .models import Book

class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'is_available')
    list_filter = ('is_available',)

admin.site.register(Book, BookAdmin)
```

Рис. 116. Изменить отображение списка записей модели

• Расширяемость. Вы можете создавать собственные представления данных, добавлять новые поля в интерфейс, создавать собственные действия и многое другое.

Например, можно добавить поле ‘description’ в форму редактирования записи модели Book и определить собственное действие для изменения статуса доступности книги, как показано на рис. 117.

```
from django.contrib import admin
from .models import Book

class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'author', 'is_available')
    list_filter = ('is_available',)
    fields = ('title', 'author', 'pub_date', 'is_available', 'description')
    actions = ['make_available', 'make_unavailable']

    def make_available(self, request, queryset):
        queryset.update(is_available=True)
    make_available.short_description = "Mark selected books as available"

    def make_unavailable(self, request, queryset):
        queryset.update(is_available=False)
    make_unavailable.short_description = "Mark selected books as unavailable"

admin.site.register(Book, BookAdmin)
```

Рис. 117. Определить собственное действие для изменения статуса

Здесь мы добавили поле ‘description’ в форму редактирования записи модели Book с помощью атрибута fields. Также определили собственные действия make_available и make_unavailable, которые изменяют статус доступности книги. Эти действия добавлены в меню Action («Действие») на странице управления моделью Book.

Формы в Django

Это удобный способ создания HTML-форм для взаимодействия с пользователем на веб-сайте. Формы в Django позволяют генерировать HTML-код, обрабатывать отправленные данные и проверять их на корректность.

Определение формы в Django происходит на основе класса, который наследуется от класса forms.Form. В этом классе мы определяем поля формы и их типы, а также правила валидации и обработки данных.

На рис. 118 представлен пример определения простой формы в Django.

В первой части кода был определен класс ContactForm, который наследуется от класса forms.Form. В этом классе определили три поля формы: ‘name’, ‘email’ и ‘message’. Поле ‘name’ является текстовым полем с максимальной длиной 100 символов, поле ‘email’ является полем для ввода email-адреса, а поле ‘message’ является текстовым полем с многострочным вводом.

```
from django import forms # 1 часть
class ContactForm(forms.Form):
    name = forms.CharField(label='Имя', max_length=100)
    email = forms.EmailField(label='Email')
    message = forms.CharField(label='Сообщение', widget=forms.Textarea)

<form method="post"> # 2 часть
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Отправить</button>
</form>

from django.shortcuts import render # 3 часть
from .forms import ContactForm
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Обработка данных формы
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            message = form.cleaned_data['message']
            # ...
    else:
        form = ContactForm()
    return render(request, 'contact.html', {'form': form})
```

Рис. 118. Определение простой формы

Чтобы отобразить форму на веб-странице, можно использовать метод as_p или as_table для объекта формы. Например, чтобы отобразить форму в виде HTML-кода, можно использовать код, представленный во второй части.

После отправки формы данные могут быть обработаны в представлении (view) с помощью метода ‘POST’ объекта request. Например, чтобы обработать отправленные данные формы, можно использовать код третьей части, в которой идет обработка запросов.

Формы в Django могут использоваться для получения данных от пользователя, редактирования данных и создания новых записей в базе данных. В Django формы создаются с помощью классов форм, которые определяют поля формы и их типы.

На рис. 119 представлен пример программного кода для создания формы для регистрации пользователя в Django.

```python
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class RegistrationForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')

    def save(self, commit=True):
        user = super(RegistrationForm, self).save(commit=False)
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
        return user
```

Рис. 119. Создание формы для регистрации пользователя в Django

Здесь мы определили класс RegistrationForm, который наследуется от класса UserCreationForm. Класс UserCreationForm предоставляет стандартные поля для регистрации пользователя в Django, такие как ‘username’ и ‘password’. Добавили поле ‘email’ с типом forms.EmailField, которое требуется для заполнения. Для хранения данных пользователя используем модель User из стандартной библиотеки Django. Также определили метод save, который сохраняет данные пользователя в базе данных. Метод save вызывает родительский метод save и добавляет поле ‘email’ к объекту User.

Чтобы отобразить форму на веб-странице, можем использовать код на рис. 120.

```python
from django.shortcuts import render
from .forms import RegistrationForm

def registration(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            # Обработка успешной регистрации
    else:
        form = RegistrationForm()
    return render(request, 'registration.html', {'form': form})
```

Рис. 120. Отображение формы на веб-странице

Здесь мы определили представление registration, которое обрабатывает запросы на адрес /registration. Если запрос имеет метод POST, создаем объект формы RegistrationForm с данными из запроса и проверяем его на корректность с помощью метода is_valid. Если форма проходит проверку на корректность, сохраняем данные пользователя с помощью метода save и выполняем необходимые действия.

Если запрос имеет метод GET, создаем объект формы RegistrationForm без данных и передаем его в шаблон registration.html, который может быть представлен как код html (рис. 121).

Здесь была создана HTML-форма с методом «post» и добавлен токен защиты от CSRF-атаки с помощью тега csrf_token. Затем использовали метод as_p для объекта формы, чтобы отобразить поля формы в виде HTML-кода.

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Зарегистрироваться</button>
</form>
```

Рис. 121. Html-код

Представленный код создает форму для регистрации пользователя в Django с использованием класса RegistrationForm. Форма содержит поля ‘username’, ‘email’, ‘password1’ и ‘password2’, а также правила валидации и обработки данных. Форма отображается на веб-странице с помощью метода as_p и может быть отправлена на сервер для обработки. Данные формы сохраняются в базе данных с помощью метода save.

Тема 7. Сборка и запуск приложения, его публикация в Интернете

Сборка и запуск приложения на Python

Сборка и запуск приложения – процесс разработки и подготовки программного кода на языке Python, который выполняется на компьютере или другом устройстве.

Для того чтобы собрать и запустить приложение на Python, необходимо сделать следующие шаги.

• Установка Python: если у вас еще нет Python на компьютере, то первым шагом будет его установка. Python можно загрузить с официального сайта python.org или воспользоваться установщиком из пакетного менеджера операционной системы.

• Написание кода: после установки Python необходимо написать программный код, который будет выполнять нужные функции. Можно использовать любой текстовый редактор или интегрированную среду разработки (IDE) для написания кода.

• Установка зависимостей: если приложение использует дополнительные библиотеки, необходимо установить их. Для этого можно воспользоваться менеджером пакетов pip, который входит в стандартную поставку Python. Например, для установки библиотеки requests можно выполнить команду pip install requests.

• Сборка приложения: для того, чтобы собрать приложение, необходимо создать исполняемый файл, который можно запустить на другом компьютере. Для этого можно воспользоваться утилитой pyinstaller, которая упаковывает код и все его зависимости в один файл. Например, для упаковки файла main.py в исполняемый файл можно выполнить команду pyinstaller main.py.

• Запуск приложения: после того как приложение было собрано, его можно запустить на любом компьютере, на котором установлен Python. Для этого нужно запустить исполняемый файл, который был создан на предыдущем шаге.

Важно отметить, что процесс сборки и запуска приложения на Python может отличаться в зависимости от используемых библиотек, операционной системы и других факторов. Однако описанные

общие шаги дают общее представление о том, как можно создать и запустить приложение на Python.

Рассмотрим более подробно процесс сборки веб-приложения на Python, включающий в себя несколько шагов, которые обычно выполняются в следующем порядке:

– настройка виртуального окружения: виртуальное окружение – это изолированное пространство, в котором устанавливаются зависимости вашего проекта, чтобы избежать конфликтов с другими проектами и обеспечить портативность вашего приложения. Вы можете настроить виртуальное окружение с помощью инструмента venv или virtualenv (в зависимости от версии Python);

– установка зависимостей: зависимости проекта – это библиотеки Python, которые необходимы для работы приложения. Обычно зависимости перечисляются в файле requirements.txt, который можно создать вручную или автоматически с помощью инструмента pip (например, pip freeze > requirements.txt);

– настройка базы данных: если приложение использует базу данных, то нужно настроить соединение с базой данных и создать таблицы и индексы (если это необходимо). Обычно настройки базы данных перечисляются в файле settings.py проекта, и можно использовать инструменты Django для создания таблиц и индексов (например, команды **migrate и makemigrations**);

– настройка статических файлов: если приложение использует статические файлы (например, CSS, JavaScript, изображения), то нужно настроить их обслуживание с помощью веб-сервера (например, Nginx или Apache) или с помощью инструментов разработки (например, команды **collectstatic в Django**);

– настройка конфигурации приложения: в зависимости от фреймворка, который используется, нам может потребоваться настроить дополнительные параметры приложения, такие как URL-маршруты, шаблоны, аутентификация и авторизация и т. д.;

– развертывание приложения на сервере: обычно это включает в себя настройку веб-сервера, настройку обслуживания статических файлов, настройку соединения с базой данных и запуск приложения в режиме продакшена.

Для сборки веб-приложения в Django может применяться команда **python manage.py collectstatic**.

Команда collectstatic – важный инструмент в веб-разработке на Django, который позволяет собирать все статические файлы проекта в одном месте, что упрощает их обслуживание и ускоряет работу веб-приложения.

Статические файлы в веб-разработке – файлы, которые не изменяются в зависимости от конкретного запроса к серверу. Это может включать в себя файлы CSS, JavaScript, изображения и другие файлы, которые используются в веб-приложении.

Когда вы разрабатываете приложение локально, Django автоматически обслуживает статические файлы из папки static вашего приложения. Однако при развертывании приложения на сервере нужно собрать все статические файлы и поместить их в одно место, чтобы они могли быть обслужены веб-сервером.

Здесь и приходит на помощь команда collectstatic. Она копирует все статические файлы из папок STATIC вашего проекта (и любых других папок, указанных в настройках) в папку STATIC_ROOT, которая обычно располагается вне каталога приложения.

Команда collectstatic часто используется в автоматических сценариях развертывания, таких как Continuous Integration и Continuous Deployment, чтобы автоматически собирать и обслуживать статические файлы при каждом развертывании приложения.

Кроме того, команда collectstatic имеет дополнительные параметры, которые могут быть полезны в различных сценариях. Например, вы можете использовать параметры --ignore и --exclude для игнорирования определенных файлов или папок при сборке статических файлов.

В целом команда collectstatic – важный инструмент в веб-разработке на Django, который упрощает обслуживание статических файлов вашего веб-приложения и позволяет быстро и эффективно развертывать ваше приложение на серверах.

Запуск веб-приложения на Python

После сборки приложения следует его запуск. Запуск веб-приложения на Python – процесс создания и запуска приложения, которое может принимать запросы от клиентов через Интернет. Веб-приложения на Python создаются с использованием специальных инструментов, называемых веб-фреймворками, которые облегчают разработку и запуск веб-приложений.

Конечная цель запуска веб-приложения на Python – создание приложения, которое может быть запущено на веб-сервере и которое пользователи могут использовать в своих браузерах. При разработке веб-приложения на Python необходимо учитывать следующие аспекты.

✓ Выбор веб-фреймворка: Python имеет множество веб-фреймворков, которые могут быть использованы для создания веб-приложений.

✓ Разработка приложения: создание веб-приложения на Python включает в себя разработку кода, который будет выполняться на сервере, а также создание шаблонов, которые будут отображаться на страницах в браузерах пользователей. Для разработки веб-приложения на Python часто используются стандартные языки, такие как HTML, CSS и JavaScript, а также библиотеки и фреймворки, такие как jQuery и Bootstrap.

✓ Работа с базами данных: в большинстве веб-приложений на Python используются базы данных для хранения информации. Для создания базы данных и ее настройки можно использовать специальные инструменты веб-фреймворков.

✓ Тестирование приложения: перед запуском веб-приложения на сервере его необходимо протестировать. Тестирование веб-приложения на Python включает в себя проверку его функциональности, безопасности и производительности. Для тестирования веб-приложения на Python можно использовать различные инструменты, такие как Selenium, pytest и Django Testing Framework.

✓ Развертывание приложения на сервере: после тестирования и отладки веб-приложения на Python его можно развернуть на сервере. Для этого необходимо настроить веб-сервер и установить все

зависимости, которые требуются для работы приложения. Для развертывания веб-приложения на Python можно использовать различные инструменты, такие как Apache, Nginx, Gunicorn и uWSGI.

Процессы регистрации и авторизации пользователей

Одними из важных функций любого веб-приложения являются процессы регистрации и авторизации пользователей, которые позволяют им создавать учетные записи, входить в свои учетные записи и работать с приложением в соответствии со своими правами доступа.

Регистрация пользователей – процесс создания учетной записи для нового пользователя в приложении. Для регистрации пользователей в Django можно использовать встроенный модуль django.contrib.auth.

Для регистрации пользователей выполняется следующая последовательность действий:

– сначала нужно создать форму для регистрации новых пользователей;
– затем должны создать представление, которое будет обрабатывать запросы на регистрацию новых пользователей.

Авторизация пользователей – процесс проверки подлинности учетных данных пользователя и предоставления ему доступа к функциям приложения. Для авторизации пользователей в Django можем использовать встроенный модуль django.contrib.auth.

Для авторизации пользователей выполняется следующая последовательность действий:

– сначала следует создать форму для авторизации зарегистрированных пользователей;
– затем должны создать представление, которое будет обрабатывать запросы на авторизацию пользователей.

После регистрации и авторизации пользователей мы можем защитить некоторые представления, чтобы они были доступны только зарегистрированным и аутентифицированным пользователям. Для защиты представлений в Django можем использовать декораторы @login_required.

Таким образом, регистрация и авторизация пользователей являются важными элементами веб-приложений. В Django можем использовать встроенные модули и декораторы для реализации этих функций.

Рассмотрим последовательность шагов для процесса регистрации пользователей веб-приложения с помощью Python и фреймворка Django.

Сначала нужно создать форму, которая будет использоваться для регистрации новых пользователей. В Django можем использовать класс формы, который наследуется от forms.ModelForm. В этом классе определяем поля формы, которые будут использоваться для ввода данных пользователей. В примере, представленном на рис. 122, мы создали класс UserRegistrationForm, который наследуется от forms.ModelForm. Добавили дополнительное поле ‘password’, которое будет использоваться для ввода пароля пользователей. Затем определили класс Meta, указывающий модель, связанную с этой формой, и поля, которые будут использоваться для ввода данных пользователей.

```
from django import forms
from django.contrib.auth.models import User

class UserRegistrationForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ['username', 'email', 'password']
```

Рис. 122. Добавление пароля

После создания формы для регистрации новых пользователей нужно создать представление, которое будет обрабатывать запросы на регистрацию. В примере, представленном на рис. 123, мы определили функцию register_view, которая проверяет, была ли отправлена форма методом POST. Если форма была отправлена, проверяем ее на валидность, и если она прошла проверку, создаем новую учетную запись пользователя и перенаправляем на домашнюю страницу. Если форма не была отправлена, отображаем пустую форму.

```
from django.shortcuts import render, redirect
from .forms import UserRegistrationForm

def register_view(request):
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('home')
    else:
        form = UserRegistrationForm()
    return render(request, 'registration/register.html', {'form': form})
```

Рис. 123. Создание представления

Последний шаг – создание шаблона, который будет использоваться для отображения формы регистрации на сайте (рис. 124).

```
{% extends 'base.html' %}

{% block content %}
  <h2>Register</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Register</button>
  </form>
{% endblock %}
```

Рис. 124. Создание шаблона

Здесь мы создали шаблон register.html, который наследуется от базового шаблона base.html. Определили блок content, который будет отображать форму регистрации. Использовали тег {% csrf_token %} для защиты формы от атак межсайтовой подделки запросов (CSRF). Также использовали метод form.as_p, чтобы отобразить поля формы в виде параграфов.

Но чтобы этот шаблон работал, нужно определить маршрут для представления register_view. Можно сделать это, добавив код, представленный на рис. 125, в файл urls.py.

```
from django.urls import path
from . import views

urlpatterns = [
    path('register/', views.register_view, name='register'),
]
```

Рис. 125. Добавление маршрута

Процесс авторизации пользователей веб-приложения с помощью Python и фреймворка Django включает несколько шагов.

Первый шаг – следует создать форму, которая будет использоваться для авторизации пользователей. В Django можно использовать класс формы, который наследуется от AuthenticationForm. В примере на рис. 126 мы создали класс UserLoginForm, который наследуется от AuthenticationForm. Определили поля формы, которые будут использоваться для ввода имени пользователя и пароля.

```
from django import forms
from django.contrib.auth.forms import AuthenticationForm

class UserLoginForm(AuthenticationForm):
    username = forms.CharField(label='Username', widget=forms.TextInput(attrs={'autofocus': True}))
    password = forms.CharField(label='Password', widget=forms.PasswordInput)
```

Рис. 126. Форма для авторизации

После создания формы для авторизации пользователей надо создать представление, которое будет обрабатывать запросы на авторизацию (рис. 127). Здесь мы определили функцию login_view, которая проверяет, была ли отправлена форма методом ‘POST’. Если форма была отправлена, проверяем ее на валидность и, если она прошла проверку, аутентифицируем пользователя и создаем сеанс входа в систему. Затем перенаправляем пользователя на домашнюю страницу. Если форма не была отправлена, отображаем пустую форму.

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login
from .forms import UserLoginForm

def login_view(request):
    if request.method == 'POST':
        form = UserLoginForm(data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(request, username=username, password=password)
            if user is not None:
                login(request, user)
                return redirect('home')
    else:
        form = UserLoginForm()
    return render(request, 'registration/login.html', {'form': form})
```

Рис. 127. Создание представления

Последний шаг – создание шаблона, который будет использоваться для отображения формы авторизации на сайте. В примере на рис. 128 мы создали шаблон login.html, который наследуется

от базового шаблона base.html. Определили блок content, который будет отображать форму авторизации. Использовали тег {% csrf_token %} для защиты формы от атак межсайтовой подделки запросов (CSRF). Также использовали метод form.as_p, чтобы отобразить поля формы в виде параграфов.

```
{% extends 'base.html' %}

{% block content %}
  <h2>Login</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Login</button>
  </form>
{% endblock %}
```

Рис. 128. Создание шаблона

Но чтобы этот шаблон работал, нужно определить маршрут для представления login_view. Это можно сделать, добавив код, представленный на рис. 129, в файл urls.py.

```
from django.urls import path
from . import views

urlpatterns = [
    path('login/', views.login_view, name='login'),
]
```

Рис. 129. Определение маршрута

Защита представлений веб-приложения с помощью Python и фреймворка Django – процесс обеспечения безопасности доступа к страницам веб-приложения. Django предоставляет несколько методов для защиты представлений, таких как аутентификация пользователей, авторизация доступа к страницам и защита от атак межсайтовой подделки запросов (CSRF).

Аутентификация пользователей – процесс проверки подлинности пользователя с использованием имени пользователя и пароля. Для этого в Django используется функция authenticate(), которая принимает объект запроса и учетные данные пользователей (имя пользователя и пароль) и возвращает объект пользователя, если аутентификация прошла успешно.

Авторизация – процесс проверки, имеет ли пользователь право доступа к странице. В Django можем использовать декоратор @ login_required для защиты доступа к страницам только для аутентифицированных пользователей. Этот декоратор перенаправляет не аутентифицированных пользователей на страницу входа в систему. Например, как представлено на рис. 130.

```
from django.contrib.auth.decorators import login_required
from django.shortcuts import render

@login_required
def my_view(request):
    # здесь ваш код
    return render(request, 'my_template.html', {})
```

Рис. 130. Создание декоратора

Django также предоставляет встроенную защиту от атак межсайтовой подделки запросов (CSRF) с помощью механизма CSRF-токенов. Когда пользователь отправляет форму, Django автоматически генерирует уникальный CSRF-токен и включает его в форму в виде скрытого поля. При отправке формы этот токен проверяется на сервере, чтобы убедиться, что запрос отправлен именно тем же пользователем, который запрашивает страницу.

Отладка веб-приложения

Процесс выявления и устранения ошибок в коде – это отладка веб-приложения на Python и Django.

Первый шаг в отладке веб-приложения на Django – включение режима отладки. Режим отладки – режим, в котором Django выводит дополнительную информацию об ошибках и исключениях в вашем коде. Вы можете включить режим отладки, установив параметр DEBUG в файле проекта: settings.py (DEBUG = True).

Второй шаг в отладке веб-приложения на Django – логирование ошибок. Логирование – это процесс записи важной информации об ошибках и исключениях, которые могут возникнуть в вашем коде. Django предоставляет встроенный модуль логирования, который можно настроить в файле settings.py вашего проекта так, как представлено на рис. 131. Здесь мы настроили логирование в файл

django.log на уровне ‘DEBUG’. Логирование включено для модуля django, и все сообщения будут записываться в файл.

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': '/path/to/django.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}
```

Рис. 131. Логирование

Третий шаг в отладке веб-приложения на Django – использование отладчика. Отладчик – это инструмент, который позволяет выполнять код пошагово и проверять значения переменных в реальном времени. Django предоставляет встроенный отладчик, который можно использовать с помощью модуля pdb (import pdb; pdb.set_trace()). При выполнении кода программа остановится на этой строке.

Четвертый шаг в отладке веб-приложения на Django – использование тестов. Тесты – это наборы инструкций, которые проверяют правильность работы кода. Django предоставляет встроенный модуль тестирования, который позволяет создавать и запускать тесты. Пример представлен на рис. 132.

```
from django.test import TestCase
from myapp.models import Book

class BookTestCase(TestCase):
    def setUp(self):
        Book.objects.create(title="Book Title", author="Book Author")

    def test_book_title(self):
        book = Book.objects.get(title="Book Title")
        self.assertEqual(book.title, "Book Title")
```

Рис. 132. Запуск теста

Здесь мы создали тестовый класс BookTestCase, проверяющий правильность названия созданной книги. Запустить тесты можно с помощью команды python manage.py test.

Пятый шаг в отладке веб-приложения на Django – использование инструментов профилирования. Инструменты профилирования позволяют определить, какие участки вашего кода занимают больше всего времени. Django предоставляет встроенный инструмент профилирования (- django-debug-toolbar). Это расширение предоставляет информацию о времени выполнения запросов, использовании кэша, производительности и другую полезную информацию.

Развертывание веб-приложения

Следующим шагом после создания приложения с помощью Django является его развертывание. Развертывание веб-приложения на Python с использованием Django – это процесс подготовки и установки приложения, созданного с помощью Django фреймворка, на веб-сервере, чтобы пользователи могли получить доступ к нему через Интернет.

Процесс развертывания включает в себя несколько этапов.

• Выбор хостинга и настройка сервера. Для развертывания веб-приложения необходимо выбрать хостинг-провайдера и настроить сервер для работы с Python и Django.

• Установка и настройка веб-сервера. Веб-сервер отвечает за обработку запросов к приложению и отправку ответов. Наиболее распространенными веб-серверами для Python являются Apache и Nginx.

• Установка и настройка базы данных. Django поддерживает несколько типов баз данных, включая PostgreSQL, MySQL и SQLite. Необходимо установить выбранную базу данных на сервере и настроить ее для работы с Django.

• Установка и настройка Python и Django. На сервере необходимо установить нужную версию Python и Django, а также их зависимости.

• Конфигурация приложения. Необходимо настроить приложение для работы с выбранным веб-сервером и базой данных, а также для обработки статических файлов и медиафайлов.

• Тестирование и запуск приложения. После того, как приложение настроено, необходимо протестировать его работу и запустить на сервере.

Развертывание веб-приложения на Python с использованием Django может быть сложным и трудоемким процессом, но он является необходимым для того, чтобы пользователи могли получить доступ к вашему приложению через Интернет.

Для развертывания веб-приложения на Python могут быть использованы различные инструменты – программные средства и библиотеки, которые позволяют упростить процесс развертывания Django-приложений на веб-серверах.

Рассмотрим наиболее распространенные инструменты развертывания веб-приложений на Python с использованием Django.

• Gunicorn – это WSGI-сервер (Web Server Gateway Interface), который предназначен для запуска веб-приложений на Python. Gunicorn поддерживает несколько рабочих процессов, что позволяет обрабатывать большое количество запросов одновременно.

• uWSGI – еще один WSGI-сервер, который предназначен для запуска веб-приложений на Python. uWSGI также поддерживает несколько рабочих процессов и может работать с различными веб-серверами.

• Apache – один из самых популярных веб-серверов, который может использоваться для развертывания Django-приложений. Для работы с Django необходимо установить модуль mod_wsgi, который позволяет Apache запускать Python-приложения.

• Nginx – еще один популярный веб-сервер, который может использоваться для развертывания Django-приложений. Для работы с Django необходимо установить модуль uWSGI, который позволяет Nginx запускать Python-приложения.

• Docker – платформа для упаковки и запуска приложений в контейнерах.

• Ansible – инструмент для автоматизации развертывания приложений.

• Fabric – еще один инструмент для автоматизации развертывания приложений. Fabric предоставляет набор команд для выпол-

нения задач на удаленных серверах, таких как установка пакетов, настройка конфигурации и запуск приложений.

• Heroku – облачная платформа для развертывания веб-приложений. Heroku позволяет развернуть Django-приложение на серверах в облаке без необходимости настройки сервера и инфраструктуры.

Публикация веб-приложения

Последним шагом в разработке веб-приложения является его публикация. Подготовка к публикации веб-приложения на Python включает в себя несколько шагов.

• Подготовка кода приложения. Перед публикацией необходимо убедиться, что код приложения находится в стабильном состоянии и соответствует лучшим практикам программирования. Код должен быть хорошо организован, читаемый и поддерживаемый. Также необходимо убедиться, что приложение написано в соответствии с принципами безопасности.

• Установка зависимостей. Перед публикацией необходимо убедиться, что все зависимости приложения установлены и настроены правильно.

• Запуск тестов. Перед публикацией необходимо убедиться, что тесты приложения работают правильно и покрывают все функциональные возможности приложения.

• Настройка сервера. Перед публикацией необходимо настроить сервер, на котором будет работать приложение.

• Настройка базы данных. Если приложение использует базу данных, необходимо настроить ее перед публикацией. Создайте базу данных и пользователя для приложения и убедитесь, что конфигурация базы данных правильная.

• Настройка системы логирования. Необходимо настроить систему логирования, чтобы можно было отслеживать ошибки и проблемы, которые могут возникнуть в продакшн-среде. Включите логирование уровня ошибок и предупреждений, а также логирование запросов, чтобы можно было отслеживать производительность приложения.

• Защита приложения. Необходимо защитить приложение от атак, таких как SQL-инъекции, подделка запросов межсайто-

вого скриптинга (XSS) и подделка межсайтовой подписи запроса (CSRF). Используйте соответствующие инструменты и библиотеки для обеспечения безопасности приложения.

Подготовка к публикации веб-приложения на Python с использованием Django – это процесс подготовки приложения к развертыванию на веб-сервере после того, как приложение было протестировано и отлажено.

Рассмотрим наиболее важные шаги, необходимые для подготовки Django-приложения к публикации.

✓ Перед публикацией приложения необходимо создать backup-копии кода, базы данных и других важных файлов, чтобы в случае возникновения проблем можно было быстро восстановить работу приложения.

✓ Перед публикацией необходимо настроить окружение на веб-сервере. Необходимо установить все необходимые зависимости и библиотеки, а также настроить параметры окружения, такие как переменные окружения и параметры конфигурации.

✓ Перед публикацией необходимо настроить безопасность приложения. Необходимо убедиться, что приложение защищено от взлома и несанкционированного доступа.

✓ Настройка логирования. Логирование – это процесс записи действий приложения в журнал. В журнале должна быть информация о действиях пользователей, ошибках и других событиях, которые могут повлиять на работу приложения.

✓ Для обеспечения стабильной работы приложения необходимо настроить мониторинг, который позволяет отслеживать работу приложения, обнаруживать проблемы и предотвращать их возникновение.

✓ Если приложение должно обрабатывать большое количество запросов, необходимо настроить балансировку нагрузки. Балансировщик нагрузки позволяет распределять запросы между несколькими серверами, что обеспечивает более быстрый и стабильный ответ на запросы.

✓ После подготовки приложения необходимо протестировать его на веб-сервере. Тестирование должно включать проверку рабо-

тоспособности, производительности, безопасности и других аспектов приложения.

✓ После публикации необходимо документировать процесс развертывания и настройки приложения. Документация должна содержать описание каталогов, файлов, зависимостей и параметров конфигурации, а также инструкции по развертыванию и обслуживанию приложения.

Подготовка к публикации отлаженного веб-приложения является важным этапом, который позволяет обеспечить стабильную и безопасную работу приложения на веб-сервере. Каждый шаг в этом процессе играет важную роль, и необходимо уделить достаточное внимание каждому из них.

Контрольные вопросы

1. Почему Python популярен в веб-разработке?
2. Какие популярные фреймворки Python используются для разработки веб-приложений?
3. Что такое Django и какую функциональность предоставляет?
4. Какие принципы веб-разработки в Python вы знаете?
5. Что такое виртуальное окружение в Python? Как создать виртуальное окружение в Python? Как активировать виртуальное окружение? Как установить пакеты в виртуальное окружение?
6. Зачем нужен файл requirements.txt при работе с виртуальным окружением? Как активировать виртуальное окружение на другой системе и установить зависимости из файла requirements.txt? Как деактивировать виртуальное окружение?
7. Какие преимущества использования виртуального окружения в Python?
8. Что такое библиотеки Python? Какие популярные библиотеки Python для работы с массивами данных вы знаете?
9. Что делает библиотека NumPy? Что такое библиотека Pandas и какие возможности она предоставляет? Для чего используется библиотека Matplotlib?
10. Что такое библиотека Scikit-learn и какие задачи машинного обучения она решает?

11. Какие инструменты предоставляет библиотека Django?
12. Что такое библиотека Flask и какую функциональность она предоставляет?
13. Какие еще библиотеки Python вы знаете и для каких задач они используются?
14. Какие основные компоненты веб-приложения на Python с использованием Django?
15. Как организовать аутентификацию и авторизацию пользователей в веб-приложении на Django?
16. Что такое ORM и зачем оно используется в веб-разработке?
17. Что такое MVC-архитектура и как она применяется в веб-разработке на Python?
18. Что такое шаблоны веб-страниц и как они используются в веб-разработке?
19. Какие инструменты предоставляет Django для работы с базами данных?
20. Какие методы HTTP-запросов вы знаете?
21. Какие преимущества и недостатки у фреймворка Flask по сравнению с Django? Как создать маршруты (URL-адреса) в веб-приложении на Flask?
22. Как создать форму в веб-приложении на Django?
23. Как обрабатывать и валидировать данные, отправленные через форму в Django?
24. Как развернуть веб-приложение на Python на сервере?

Тесты для самоконтроля

(ответы см. в приложении)

1. Какие языки программирования используются для создания серверной части веб-приложений? (*один вариант ответа*)

а) HTML, CSS

б) JavaScript

в) Python, PHP, Ruby

г) SQL

2. Какой фреймворк для разработки веб-приложений предоставляет всестороннюю инфраструктуру, включая ORM для работы с базами данных, аутентификацию и авторизацию пользователей? (*один вариант ответа*)

а) Django
б) Flask
в) Pyramid
г) Bottle

3. Какие команды нужно выполнить, чтобы создать новое виртуальное окружение и установить все необходимые зависимости из файла requirements.txt при переносе проекта на другую систему? (*один вариант ответа*)

а) создать новое окружение и установить зависимости вручную
б) активировать старое окружение и установить зависимости вручную
в) создать новое окружение и установить зависимости из файла requirements.txt
г) активировать старое окружение и установить зависимости из файла requirements.txt

4. Что такое библиотека Django? (*один вариант ответа*)

а) библиотека для работы с данными
б) библиотека для визуализации данных
в) библиотека для машинного обучения
г) библиотека для создания веб-приложений

5. Что такое ORM в Django? (*один вариант ответа*)

а) инструмент для создания динамических HTML-страниц
б) инструмент для работы с базами данных
в) инструмент для тестирования приложений
г) инструмент для маршрутизации запросов

6. Что необходимо сделать для создания нового приложения внутри проекта Django? (*несколько вариантов ответа*)

а) создать новую папку для приложения
б) выполнить команду python manage.py startapp app_name
в) создать файлы для работы приложения
г) добавить маршруты (URL-адреса) в файл urls.py

7. Какие файлы содержатся в пакете project_name/ в структуре проекта Django? (*несколько вариантов ответа*)

а) init.py

б) settings.py

в) urls.py

г) manage.py

8. Для создания приложения в Django нужно выполнить команду (*один вариант ответа*)

а) python manage.py startproject app_name

б) python manage.py newapp app_name

в) python manage.py createapp app_name

г) python manage.py makeapp app_name

9. Для создания миграций в Django нужно выполнить команду, которая (*несколько вариантов ответа*)

а) создаст файлы миграций на основе определенных моделей данных

б) создаст таблицы в базе данных на основе определенных моделей данных

в) создаст файлы миграций и таблицы в базе данных на основе определенных моделей данных

г) создаст только файлы миграций без таблиц в базе данных

10. Какой файл нужно создать, чтобы создать шаблон в Django? (*один вариант ответа*)

а) models.py

б) views.py

в) settings.py

г) home.html

11. Что определяет шаблон в Django? (*несколько вариантов ответа*)

а) HTML-разметку

б) CSS-стили

в) JavaScript-скрипты

г) теги шаблонов Django

12. Какие шаги необходимо выполнить для создания веб-страницы в Django? (*несколько вариантов ответа*)

а) создать новый файл шаблона в папке templates вашего приложения

б) создать функцию представления в файле views.py вашего приложения

в) подключить шаблон к функции представления

г) создать маршрут в файле models.py вашего приложения

13. Какие элементы включаются в шаблоны Django? (*один вариант ответа*)

а) HTML-код и специальные теги шаблонного языка

б) CSS-стили и JavaScript-скрипты

в) файлы моделей и функции представлений

г) маршруты и настройки приложения

14. Какой тег используется для включения значения переменной в шаблон? (*один вариант ответа*)

а) {% extends %}

б) {% block %}

в) {{ переменная }}

г) {% for %}

15. Какие преимущества имеет использование шаблонов в Djan-go? (*несколько вариантов ответа*)

а) упрощение разработки кода

б) упрощение поддержки кода

в) создание динамических веб-страниц

г) упрощение настройки сервера

16. Какие типы баз данных поддерживаются в Python? (*один вариант ответа*)

а) только реляционные БД

б) только NoSQL БД

в) реляционные и NoSQL БД

г) только MongoDB

17. Какие операции можно выполнять с помощью библиотек для работы с базами данных в Python? (*несколько вариантов ответа*)

а) создание соединения с базой данных

б) создание, изменение и удаление таблиц

в) выполнение запросов и получение данных из базы данных

г) работа с файлами в операционной системе

18. Что такое модели в Django? (*один вариант ответа*)

а) классы Python, определяющие структуру таблиц в базе данных

б) библиотека для работы с базами данных

в) модуль для отображения данных на веб-страницах

г) язык программирования для создания веб-приложений

19. Что делает метод filter() в ORM Django? (*один вариант ответа*)

а) возвращает все объекты модели

б) возвращает единственный объект модели, удовлетворяющий определенному условию

в) возвращает объекты модели, удовлетворяющие определенному условию

г) обновляет значения полей у объектов модели, удовлетворяющих определенному условию

20. Что такое миграции в Django? (*один вариант ответа*)

а) механизм, который позволяет автоматически создавать и обновлять схему базы данных на основе изменений, внесенных в модели Django

б) способ создания визуальных отчетов на основе данных в базе данных

в) механизм, который позволяет автоматически создавать и обновлять схему базы данных на основе изменений, внесенных в контроллеры Django

г) инструмент для создания интерфейса базы данных для конечных пользователей

21. Какие инструменты предоставляет Django ORM для работы с данными? (*несколько вариантов ответа*)

а) миграции базы данных

б) механизмы кэширования

в) ORM-инструменты

г) инструменты для создания интерфейса базы данных для конечных пользователей

22. Какой метод Django ORM можно использовать для выборки всех объектов из таблицы в базе данных? (*один вариант ответа*)

а) «all»

б) «filter»

в) «order_by»

г) «aggregate»

23. Какое поле используется для определения связи «многие-ко-многим» в Django ORM? (*один вариант ответа*)

а) ForeignKey

б) OneToOneField

в) ManyToManyField

г) CharField

24. Какой метод используется для удаления связи в Django ORM? (*один вариант ответа*)

а) delete

б) remove

в) detach

г) все варианты ответа верны

25. Какой метод используется для получения всех объектов модели из базы данных? (*один вариант ответа*)

а) objects.all()

б) objects.get()

в) objects.create()

г) objects.update()

26. Что такое Django Forms? (*один вариант ответа*)

а) инструмент для работы с базой данных, используя объекты Python вместо SQL-запросов

б) инструмент для создания HTML-форм

в) встроенный интерфейс администратора

г) инструмент для создания API для взаимодействия с базой данных

27. Какие инструменты в Django позволяют создавать, обновлять и удалять данные в базе данных? (*несколько вариантов ответа*)

а) ORM

б) Django Admin

в) Django Forms

г) Django Rest Framework

28. Какие действия могут быть выполнены с помощью форм в Django? (*несколько вариантов ответа*)

а) получение данных от пользователя

б) редактирование данных

в) создание новых записей в базе данных

г) удаление записей из базы данных

29. Какой инструмент можно использовать для установки зависимостей проекта в Python? (*один вариант ответа*)

а) pip

б) venv

в) virtualenv

г) requirements.txt

30. Какие шаги включает процесс сборки веб-приложения на Python? (*несколько вариантов ответа*)

а) настройка виртуального окружения

б) установка зависимостей

в) настройка базы данных

г) настройка статических файлов

Практические задания по темам модуля 2

Тема «Создание веб-приложения на Python с использованием Django»

Задание 1. Создайте виртуальное окружение для разработки веб-приложения с использованием Django.

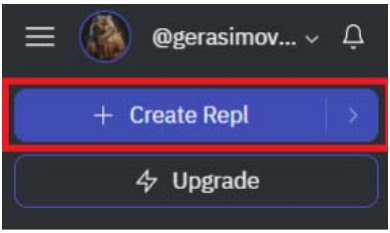
Ход работы

1. Изучите методические рекомендации.

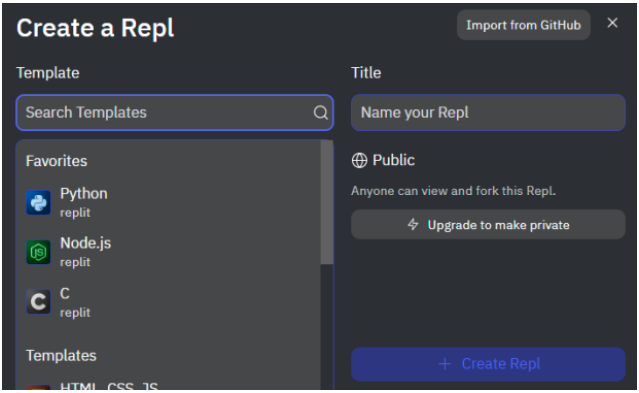
2. Создайте виртуальное окружение, которое будет содержать установленную библиотеку Django.

Рассмотрим выполнение задания при помощи сервиса replit

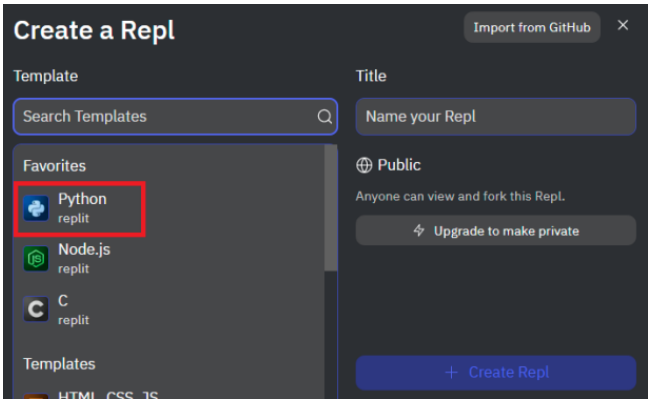
Для создания на сервисе replit виртуального окружения нажимаем Create Repl.



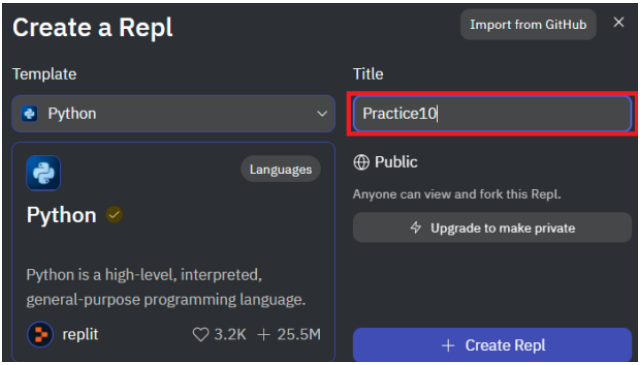
Появляется окно с выбором шаблона проекта.



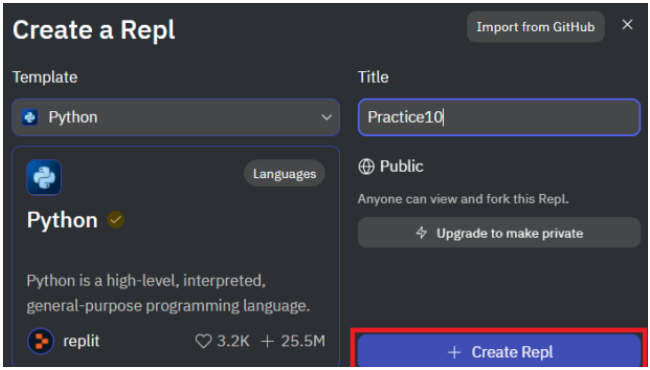
Выбираем Python.



Задаем название.

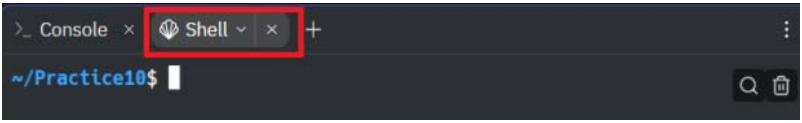


Нажимаем Create Repl.

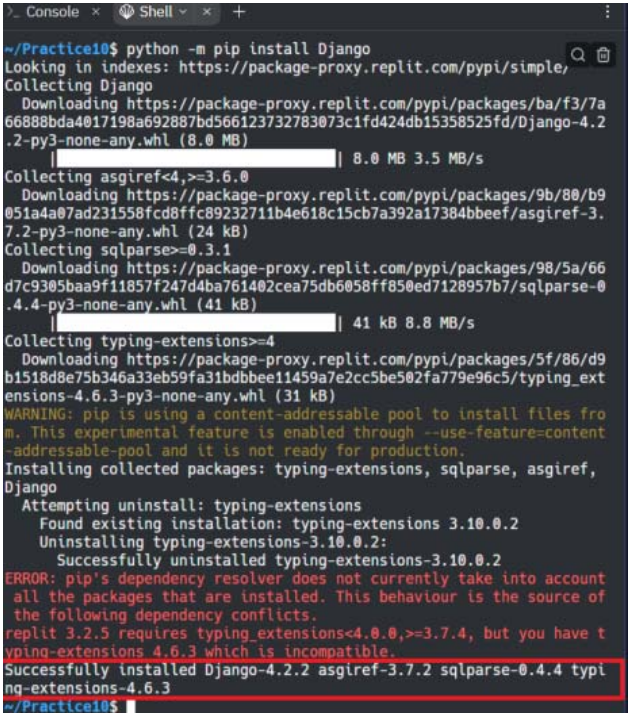


Создалось виртуальное окружение.

Справа в окне находим вкладку Shell и переходим в неё.



Установим фреймворк Django командой **python -m pip install Django**, вписав её в консоль:



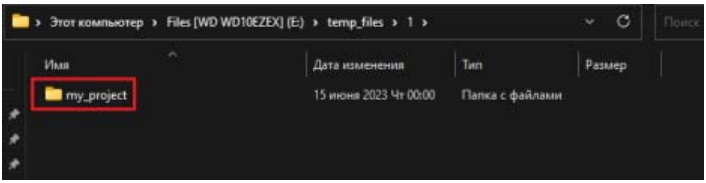
Как только в консоли появится надпись «Successfully installed Django...», значит, пакет установлен в виртуальное окружение.

Рассмотрим задачу для примера на локальной машине

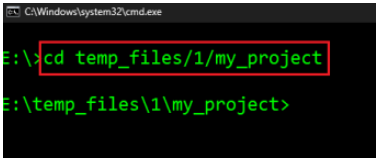
Необходимо написать программу, которая просит пользователя ввести два числа и выводит на экран их сумму, разность, произведение и частное.

Для выполнения задания необходимо сделать следующие шаги.

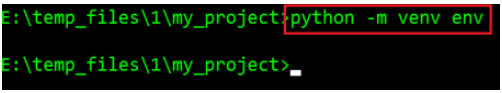
Создайте новую папку с названием my_project на рабочем столе или в любой другой удобной для вас директории.



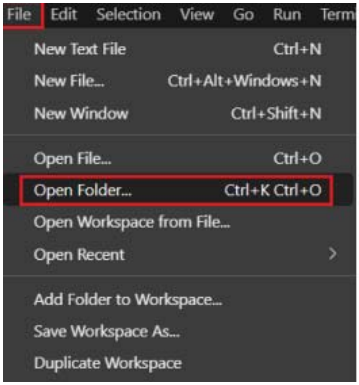
Откройте командную строку и перейдите в созданную папку, используя команду **cd /path/to/my_project**.



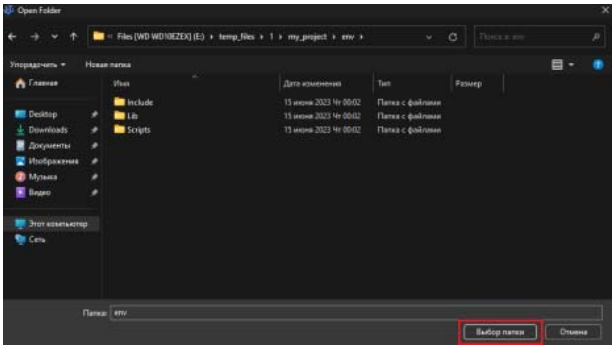
Создайте виртуальное окружение, используя команду **python -m venv env**.



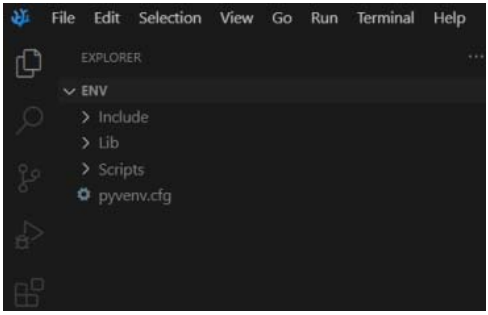
Откройте виртуальное окружение через Visual Studio Code, выбрав пункт File – Open Folder.



Перейдите в созданную папку и нажмите на кнопку «Выбор папки».

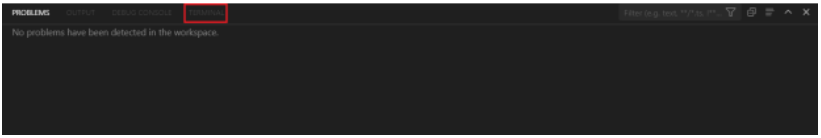


В левой части окна появятся файлы проекта.



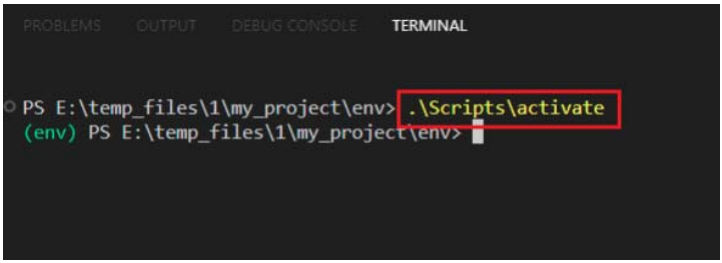
Откройте терминал, нажав на иконку восклицательного знака в левом нижнем углу Visual Studio Code.

В открывшемся окне перейдите во вкладку TERMINAL.

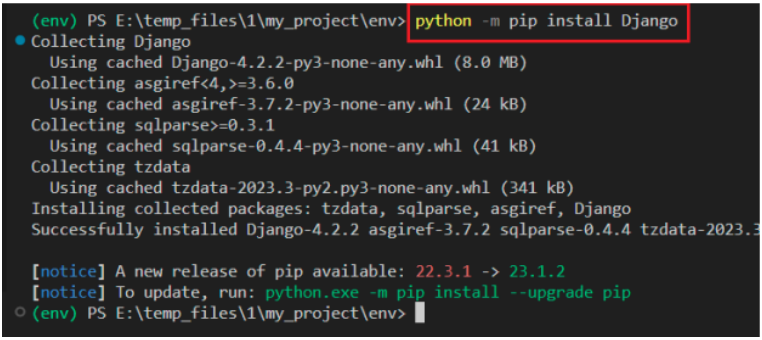


У вас откроется терминал.

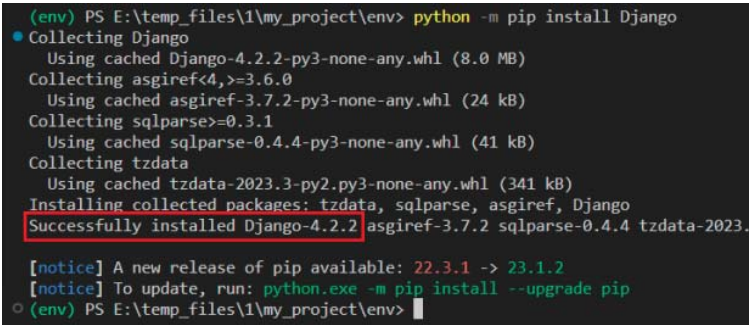
Активируйте виртуальное окружение, используя команду **source /bin/activate (Linux/MacOS)** или **\Scripts\activate (Windows)**.



Установите фреймворк Django, используя команду **python -m pip install Django**.



При успешной установке появится сообщение «Successfully installed Django...».



Методические указания

Виртуальное окружение Python – это инструмент, который позволяет создавать отдельное пространство для работы с Python на вашем компьютере. Это полезно, если вы хотите установить различные версии Python или различные пакеты и библиотеки с разными версиями.

Чтобы создать виртуальное окружение, нужно выполнить несколько простых шагов:

– установите модуль virtualenv, введя в командную строку pip install virtualenv;

– создайте новую папку для вашего проекта и перейдите в нее через командную строку;

– введите команду **virtualenv env**, где env – это название вашего виртуального окружения;

– активируйте виртуальное окружение, используя команду **source env/bin/activate (Linux/MacOS) или env\Scripts\activate.bat (Windows)**;

– теперь вы можете устанавливать библиотеки в ваше виртуальное окружение. Для установки библиотеки нужно выполнить команду **pip install library_name**, где library_name – это название библиотеки, которую вы хотите установить;

– если вы хотите сохранить список всех установленных библиотек, то можно выполнить команду **pip freeze > requirements.txt**. Эта команда создаст файл requirements.txt, содержащий список всех установленных библиотек. Для установки всех библиотек из файла requirements.txt нужно выполнить команду **pip install -r requirements.txt**. Чтобы выйти из виртуального окружения, нужно выполнить команду **deactivate**.

Задание 2. Создайте и настройте Django-проект для дальнейшей разработки веб-приложения.

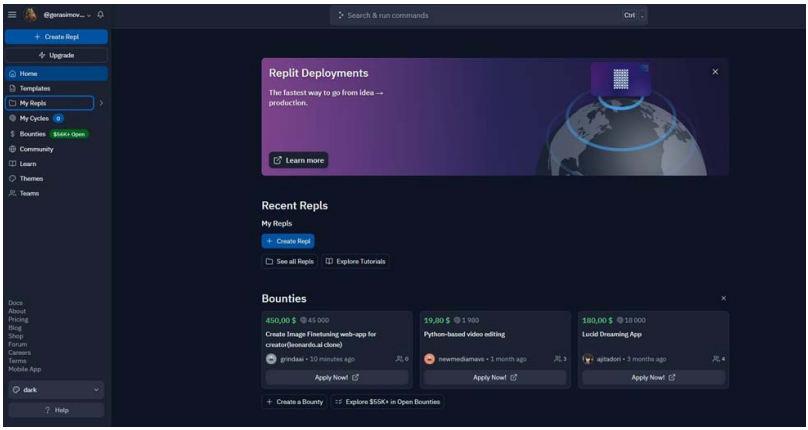
Ход работы

1. Изучите методические рекомендации.

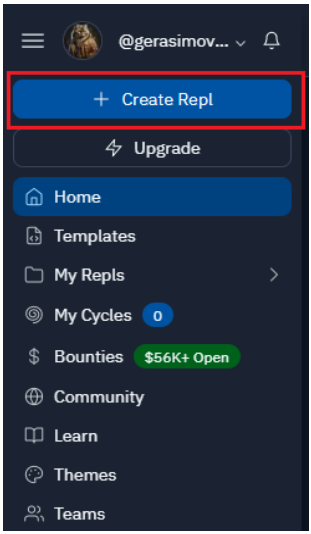
2. Создайте и настройте Django-проект для дальнейшей разработки веб-приложения.

Использование онлайн-сервиса Replit

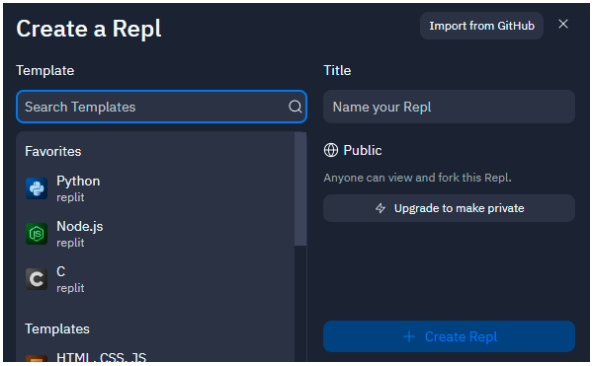
Перейдем на сайт Replit (https://replit.com).



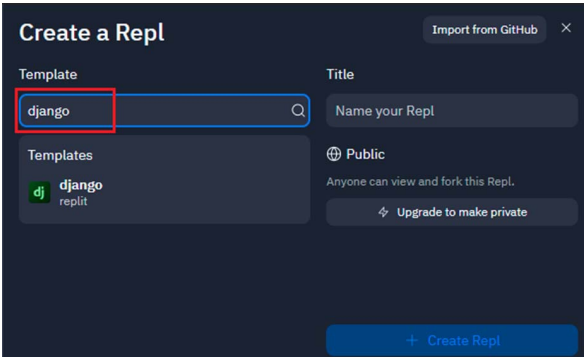
Нажмем кнопку Create Repl.



Откроется окно выбора инструмента/языка программирования.

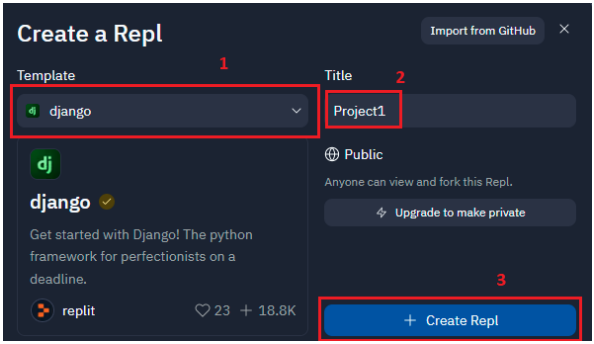


В поле поиска Search Templates впишем Django.



Выберем Django-шаблон и в поле Title впишем название проекта.

Далее нажмем Create Repl.

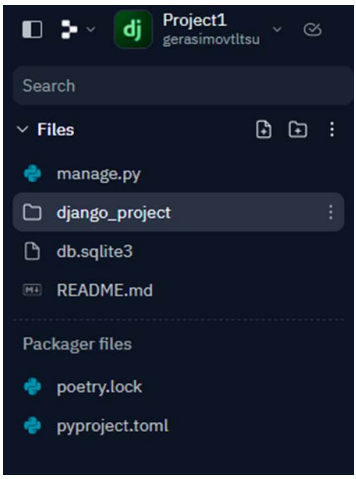


Создастся проект, где:

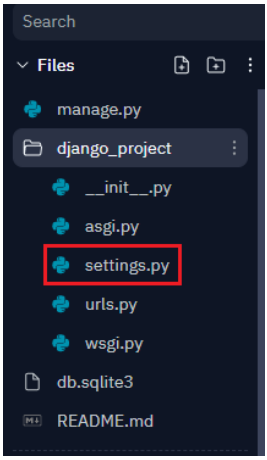
– django_project – главный каталог проекта;

– db.sqlite3 – файл базы данных приложения;

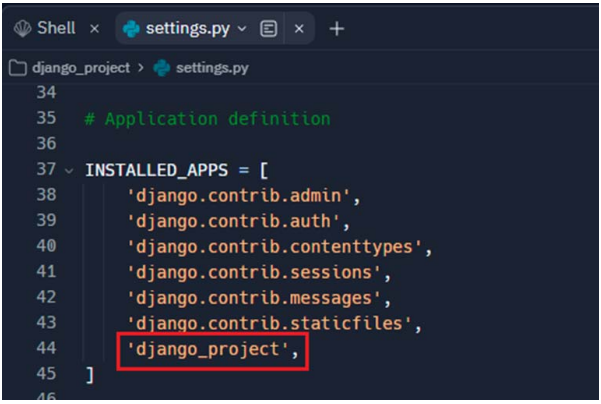
– manage.py – файл управления проектом Django.



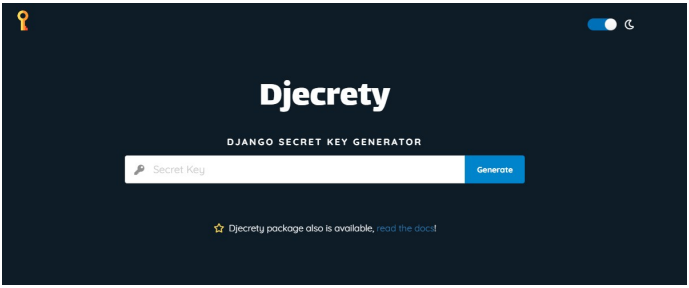
Откроем файл settings.py проекта django_project (папка django_project).



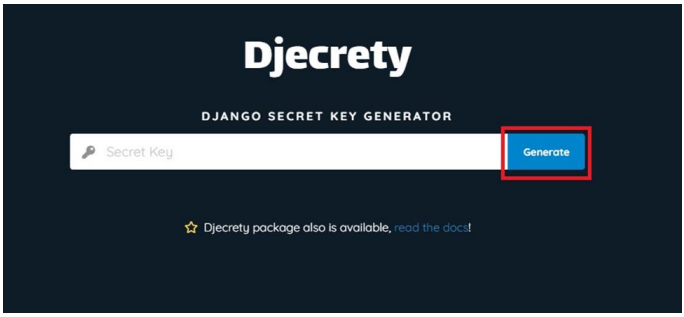
В параметр INSTALLED_APPS впишем название приложения ‘django_project’.



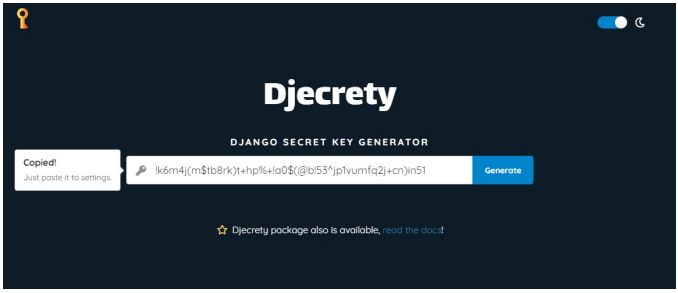
Теперь необходимо сгенерировать секретный ключ. Для этого переходим на сайт https://djecrety.ir/



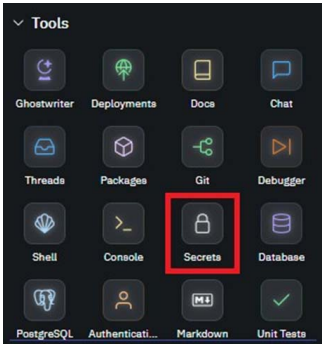
Нажимаем кнопку Generate.



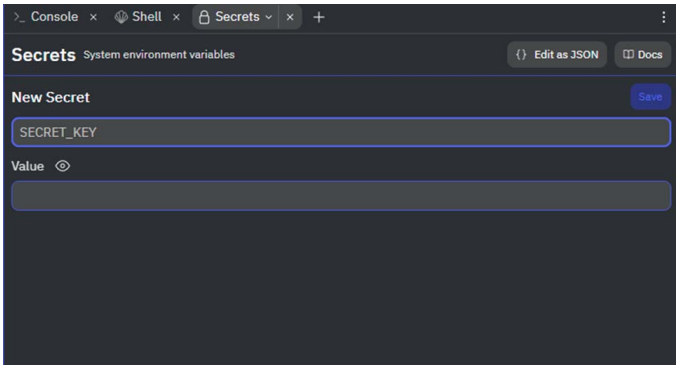
Копируем сгенерированный ключ, нажав на поле с ключом или выделив ключ и скопировав через контекстное меню вашей операционной системы.



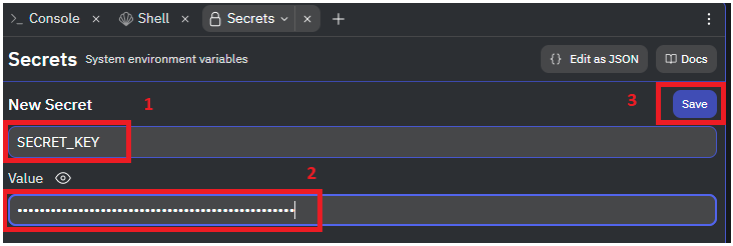
Теперь необходимо сохранить секретный ключ в переменные среды окружения. Для этого в правом нижнем углу в меню Tools выбираем пункт Secrets.



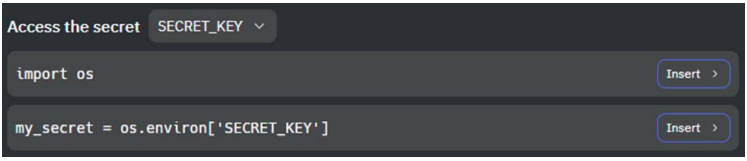
Справа появляется меню с системными переменными окружения.



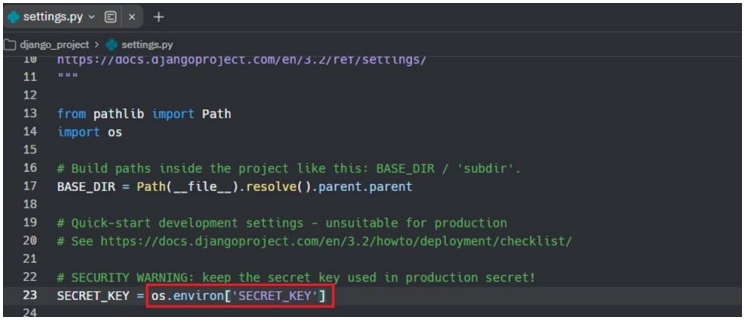
В поле New Secret (1) вписываем значение SECRET_KEY, а в поле Value (2) вставляем сгенерированный ключ. Затем нажимаем кнопку Save (3).



В том же окне появится инструкция для использования секретного ключа.



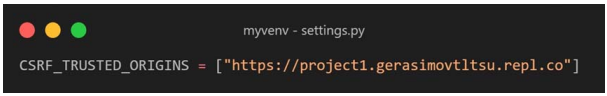
Перейдем в файл settings.py проекта и заменим значение SECRET_KEY.



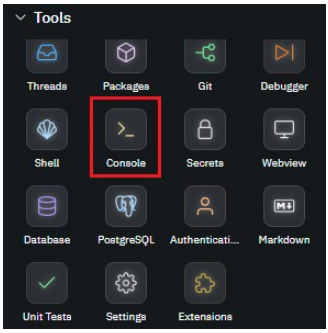
В этом же файле settings.py находим параметры LANGUAGE_CODE и TIME_ZONE и заменяем их.

В файл settings.py также необходимо добавить параметр CSRF_TRUSTED_ORIGINS.

CSRF_TRUSTED_ORIGINS = [“https://*название_проекта*.*логин_replit*.repl.co”]

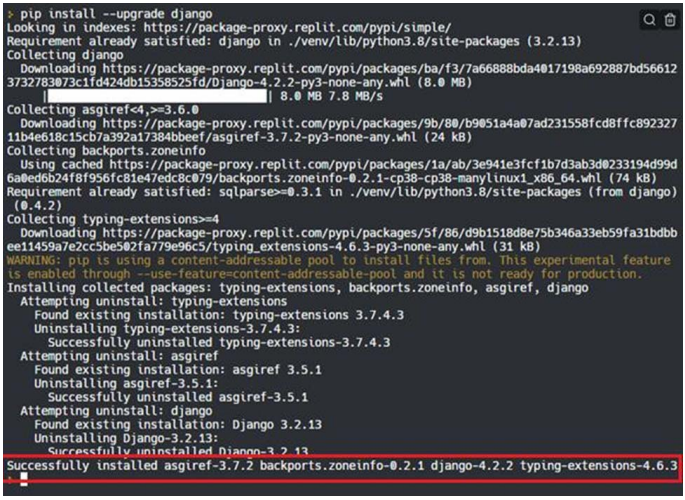


Теперь необходимо обновить Django до последней версии. Для этого из меню Tools открываем инструмент Console.

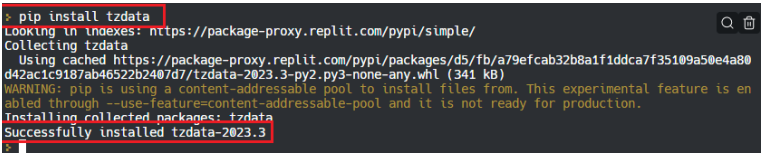


В открывшейся консоли вводим команду **pip install --upgrade django**.

Необходимо дождаться, когда в консоли появится надпись «Successfully installed...».



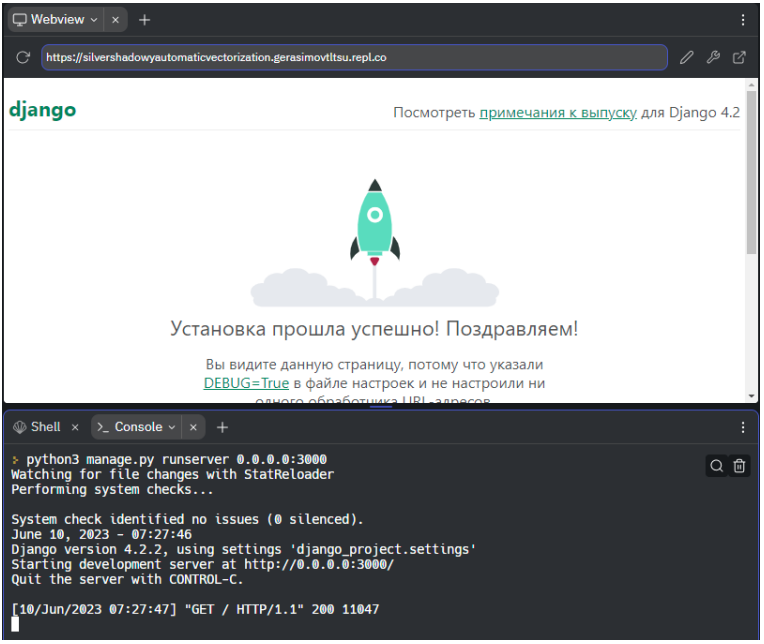
Также в проекте отсутствует пакет для часовых поясов. Установим и его командой **pip install tzdata**



Убедитесь, что в результате установки появилось сообщение «Successfully installed...».

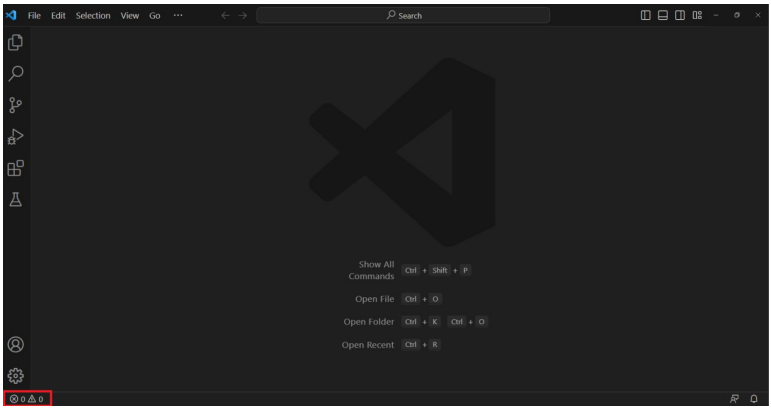
Запустите проект, нажав на кнопку Run.

Проект запущен, в окне Webview появится приветственное окно Django.



Использование локальных инструментов разработки

Для создания проекта запустим редактор кода Visual Studio Code и откроем консоль (в левом нижнем углу).



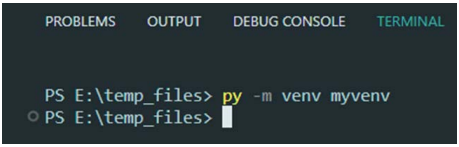
Перейдем во вкладку TERMINAL.

Теперь необходимо создать виртуальное окружение.

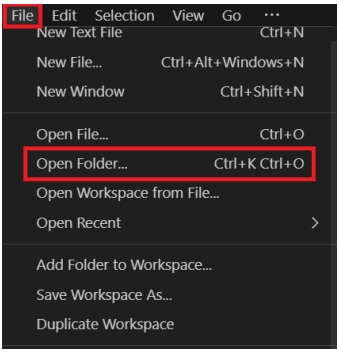
Создается командой **py -m venv myvenv** или **python -m venv myvenv**

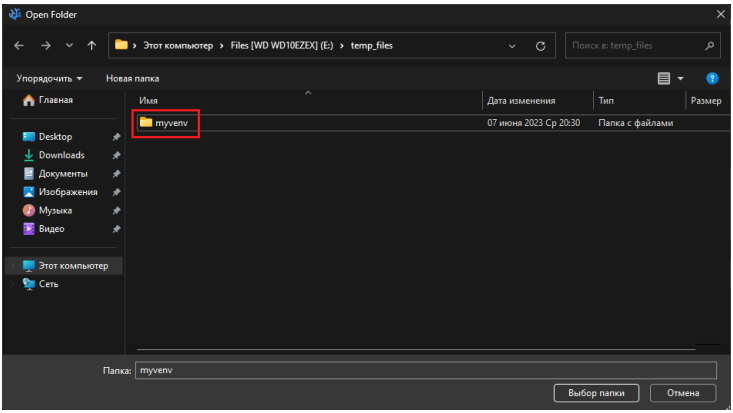
Вместо myvenv нужно указать название вашего виртуального окружения.

После того как виртуальное окружение было создано (не появилось никаких ошибок при создании, появилась пустая строка в терминале, ожидающая команду),



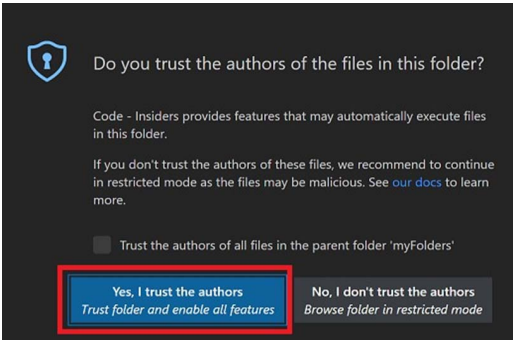
необходимо открыть его в редакторе. Для этого в меню программы VS Code выбираем File → Open Folder и выбираем созданную папку с виртуальным окружением.



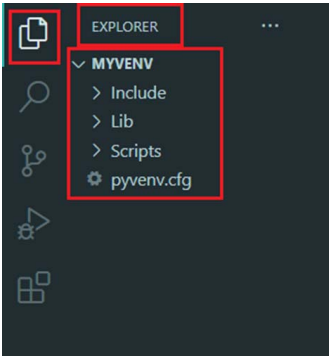


В редакторе после открытия отобразятся файлы проекта.

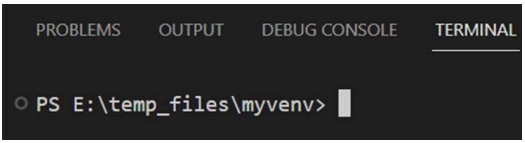
Если при открытии появится следующее диалоговое окно, то необходимо нажать на кнопку Yes, I trust the authors.



Проект открыт, в колонке Explorer отобразятся файлы проекта.

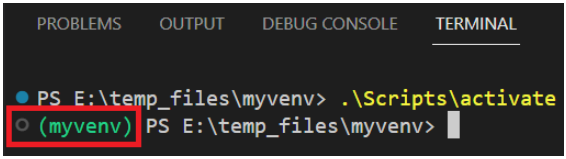


Снова откроем TERMINAL.

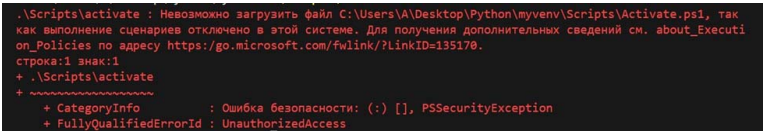


Активируем виртуальное окружение. В терминале необходимо написать команду **.\Scripts\activate**.

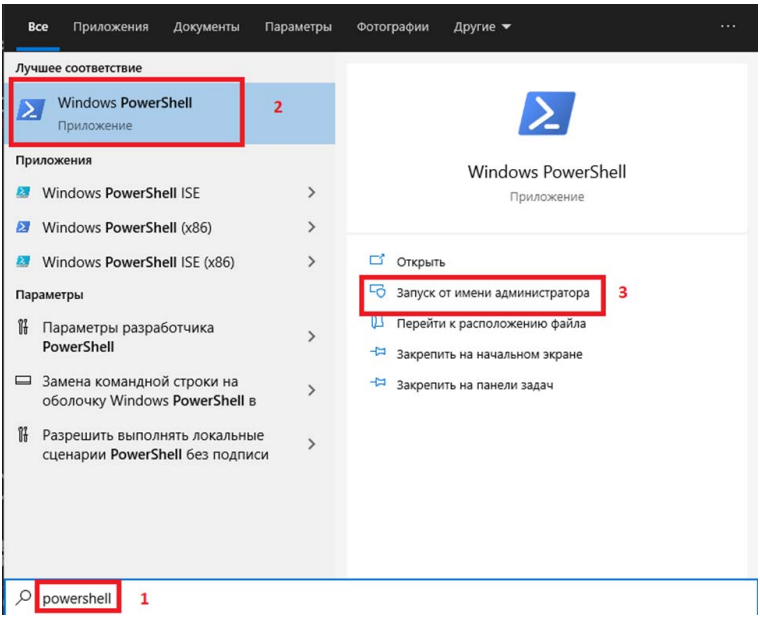
Если виртуальное окружение активировано, то около пути папки появится название виртуального окружения.



В случае если возникает ошибка активации виртуального окружения,



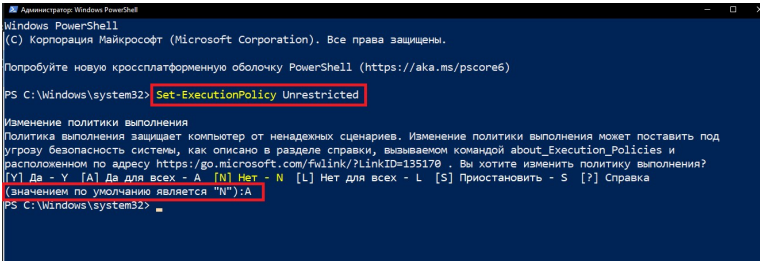
откройте Powershell через меню «Пуск» от имени администратора



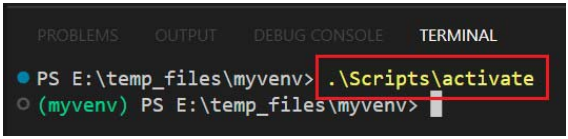
и введите следующую команду:

Set-ExecutionPolicy Unrestricted

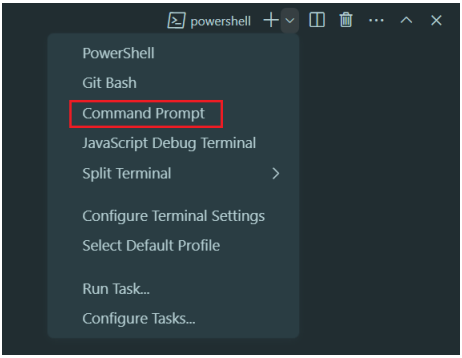
При ответе на вопрос об изменении политики необходимо написать в PowerShell «A» и нажать Enter.



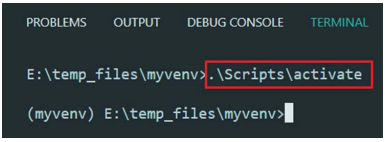
Затем снова активируйте виртуальное окружение через Powershell командой **.\Scripts\activate**



Альтернативный вариант: выберите в терминале Visual Studio Code Command prompt.

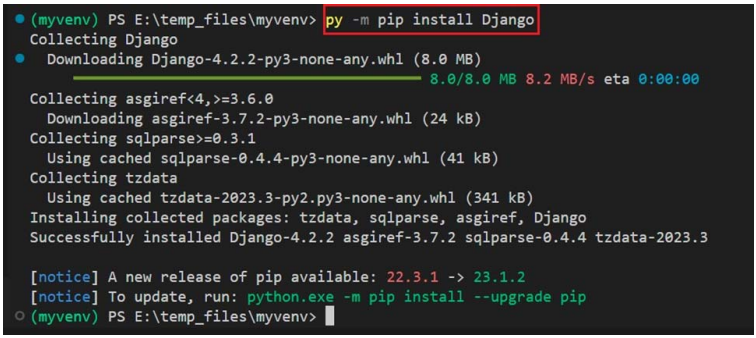


И снова введите команду **.\Scripts\activate**

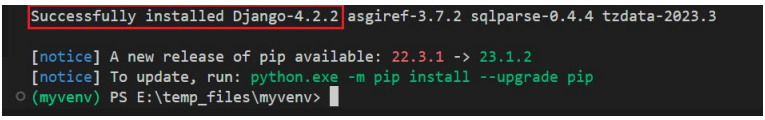


Если виртуальное окружение активировано, то в терминале появится название окружения.

Теперь, когда виртуальное окружение активировано, необходимо установить фреймворк Django. Для этого напишем следующую команду в терминале: **py –m pip install Django**

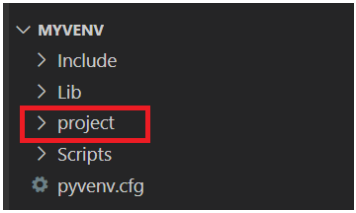


Если установка пройдет успешно, то в результате будет видно следующее сообщение: «Successfully installed Django-*версия* ...».

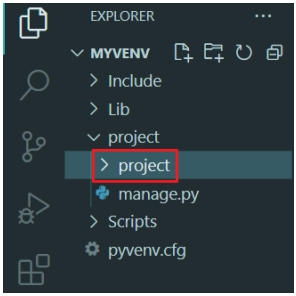


Теперь необходимо создать Django-проект. Для этого напишем следующую команду: **django-admin startproject project**

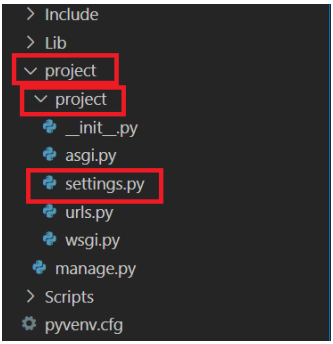
Вместо project можно указать название вашего проекта. В списке файлов проекта появится созданный проект.



И в нем есть приложение с таким же названием project.

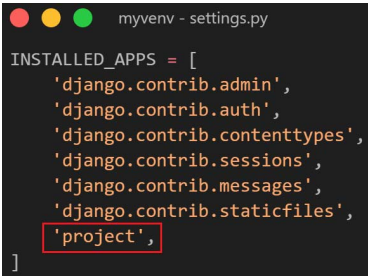


Откройте созданный проект (вторую папку project) и перейдите в файл настроек (settings.py).



В этом файле будем настраивать проект.

Найдем параметр INSTALLED_APPS и внесем в него наше приложение project.



В этом же файле (settings.py) найдем и изменим следующие параметры:

– LANGUAGE_CODE;

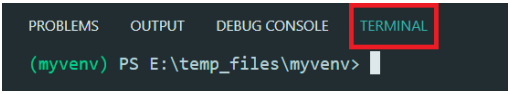
– TIME_ZONE.

Изменим параметры следующим образом.

Теперь необходимо создать таблицы в базе данных.

Для этого снова открываем терминал, нажав на иконки в левом нижнем углу Visual Studio Code.

И затем, выбрав вкладку TERMINAL,



переходим в папку проекта командой **cd .\project** и выполняем команду **py .\manage.py migrate**

```
(myvenv) PS E:\temp_files\myvenv> cd .\project\
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
```

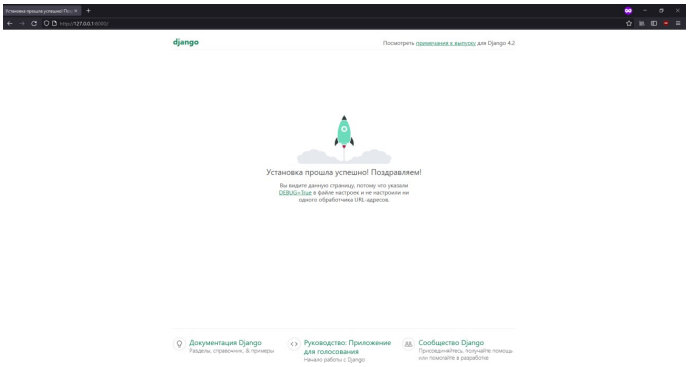
Теперь можно запустить сервер командой **py .\manage.py runserver** и посмотреть на результат.

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 07, 2023 - 21:14:25
Django version 4.2.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Фреймворк выведет сообщение о запуске сервера по адресу: http://127.0.0.1:8000/

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 07, 2023 - 21:14:25
Django version 4.2.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Перейдем в браузер и откроем страницу http://127.0.0.1:8000/ или http://localhost:8000/



В случае успеха будет видно приветственное окно, в котором написано, что установка прошла успешно.

Методические указания

Виртуальное окружение – это изолированная среда для Python-проектов. Это позволяет каждому проекту иметь свои собственные зависимости, независимо от того, какие библиотеки установлены в других проектах. Такой подход предотвращает конфликты между библиотеками и упрощает управление зависимостями проекта.

Для создания виртуального окружения в Python используется модуль venv. Этот модуль включен в стандартную библиотеку Python и позволяет создавать изолированную среду для Python-проектов. Команда для создания виртуального окружения выглядит следующим образом: **python -m venv имя_окружения**. После выполнения команды в указанной директории создаётся структура каталогов, содержащая Python-интерпретатор и место для установки пакетов.

Для использования созданного виртуального окружения необходимо его активировать. Это делается с помощью скрипта активации, который находится в каталоге Scripts внутри каталога виртуального окружения. На Windows активация производится с помощью команды **.\Scripts\activate**, после чего в командной строке будет отображаться имя активированного окружения.

Django – высокоуровневый Python-фреймворк для разработки веб-приложений. Он позволяет быстро создавать безопасные

и поддерживаемые веб-сайты. Установка Django внутри активированного виртуального окружения производится с помощью системы управления пакетами pip. Команда установки выглядит так: **pip install django**. Эта команда загрузит и установит последнюю версию Django вместе с необходимыми зависимостями.

Задание 3. Создайте простую веб-страницу с помощью фреймворка Django.

Ход работы

1. Изучите методические рекомендации.

2. Создайте простую веб-страницу с использованием фреймворка Django и с помощью html-шаблонов.

Для создания страниц в Django существуют представления (views).

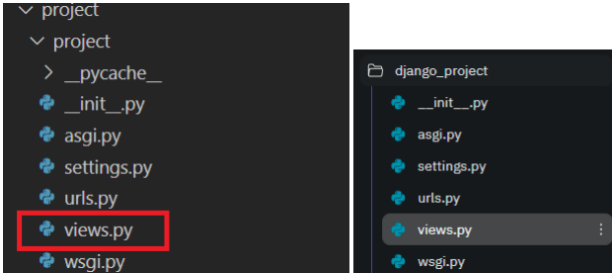
Представление – это компонент Django-приложения, который содержит логику работы и отвечает за отображение объектов в браузере. Обычно выполняет определенную функцию и имеет определенный шаблон.

Например, в приложении для блога у вас могут быть следующие представления:

– домашняя страница;

– страница записей и так далее.

Для того чтобы добавить представление, необходимо в папке приложения project или django_project (если вы выполняете работу в сервисе replit) создать файл views.py. Для этого щелкните правой кнопкой мыши по второй папке project (или по папке django_project для replit) и выберите пункт New File... (Add file для replit). Задайте ему соответствующее название.



Открываем файл views.py и вводим в него следующий код:

myvenv - views.py

```
from django.http import HttpResponse

def MainView(request):
    return HttpResponse("Первая страница приложения")
```

Теперь перейдем в файл urls.py, импортируем в него созданное view и добавим адрес до представления.

Файл необходимо изменить следующим образом.

myvenv - urls.py

```
from django.contrib import admin
from django.urls import path

from .views import MainView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', MainView),
]
```

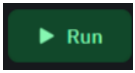
Теперь необходимо запустить сервер.

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 07, 2023 - 21:14:25
Django version 4.2.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Откроем браузер и перейдем по адресу http://127.0.0.1:8000/ или http://localhost:8000/.

Созданное приложение работает, и на экране должен появиться введенный текст.

В случае использования сервиса Replit необходимо нажать на кнопку Run.

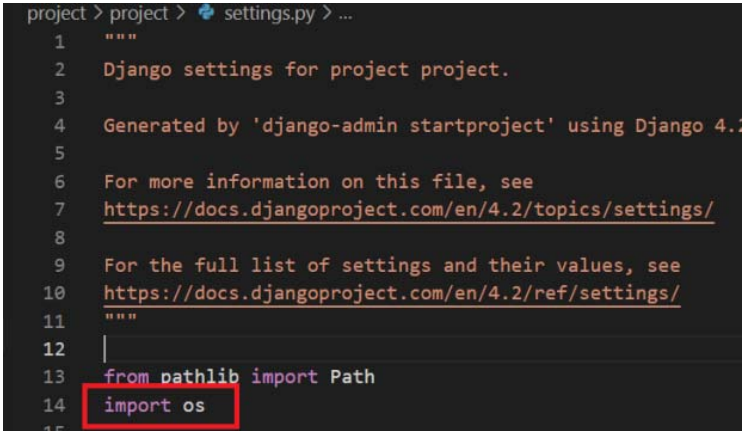


В окне webview откроется созданная страница.

Шаблоны (templates) отвечают за формирование внешнего вида приложения. Они предоставляют специальный синтаксис, который позволяет внедрять данные в код HTML.

Для того чтобы шаблоны работали, в проекте необходимо открыть файл settings.py.

В начале файла settings.py импортируйте библиотеку os, если она не импортирована,

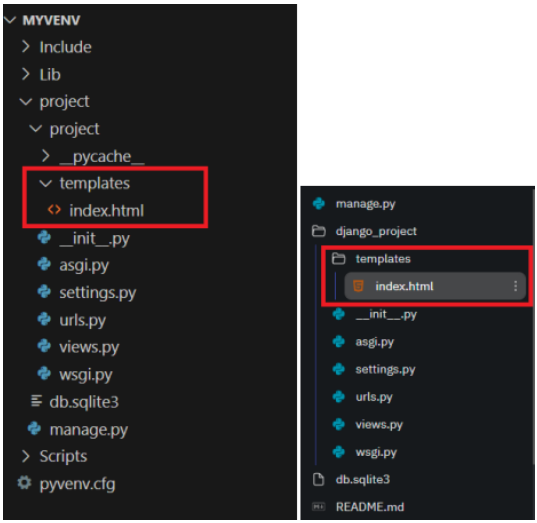


и измените параметр DIRS у параметра TEMPLATES (в этом же файле).

myvenv - settings.py

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, "templates")],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Добавим в папку приложения каталог TEMPLATES. А в нем определим файл index.html:



myvenv - index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Тестовая страница</title>
    <meta charset="utf-8" />
</head>
<body>
    <h2>Тестовая страница</h2>
</body>
</html>
```

По сути, это обычная веб-страница, которая содержит код html. Теперь используем эту страницу для отправки ответа пользователю. И для этого перейдем в приложение файлу views.py, который определяет функции для обработки запроса. Изменим этот файл следующим образом:

myvenv - views.py

```
from django.shortcuts import render

def MainView(request):
    return render(request, "index.html")
```

Теперь созданное представление вызывает передачу файла шаблона при открытии страницы.

Чтобы увидеть результат вывода, перейдем в urls.py и добавим вызов представления по переходу в корневой каталог проекта из браузера.

myvenv - urls.py

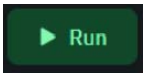
```
from django.contrib import admin
from django.urls import path, include

from project.views import MainView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', MainView),
]
```

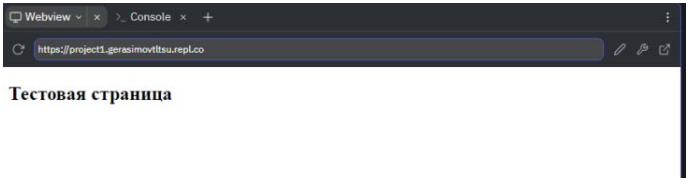
Запустим приложение. Приложение разместилось по адресу: http://127.0.0.1:8000/ или http://localhost:8000/

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 01, 2023 - 16:20:36
Django version 4.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

и откроем страницу http://127.0.0.1:8000/ или http://localhost:8000/.

При использовании сервиса Replit необходимо нажать на кнопку Run.





Появилась надпись «Тестовая страница», а это значит, что созданное представление работает.

Методические указания

Представления в Django отвечают за обработку запросов и формирование ответов. Для создания представления содержимое представлений располагается в файле views.py в папке Django-приложения.

Необходимо определить функцию представления, которая принимает объект запроса и возвращает объект ответа, используя функцию HttpResponse.

Для того чтобы Django мог перенаправить запросы на ваше представление, необходимо в файле urls.py в папке проекта импортировать представление из views.py. Далее в переменную urlpatterns добавить маршрут, указывая путь и связанное с ним представление.

Шаблоны управляют внешним видом приложения. Для работы с шаблонами необходимо выполнить несколько шагов:

– убедитесь, что в настройках Django (settings.py) указан каталог для шаблонов в параметре DIRS списка TEMPLATES;
– создайте каталог templates в папке вашего приложения и добавьте в него файл index.html;
– в представлении используйте функцию render для отправки HTML-шаблона как ответа, передавая ей объект запроса и путь к шаблону.

Задания для самостоятельного выполнения

1. Создайте базовый шаблон base.html, который будет содержать общую структуру вашего сайта, такую как шапка, навигационное меню и подвал. Включите в него блоки для контента, которые будут заполняться в других шаблонах.

2. Создайте несколько дочерних шаблонов, которые наследуют базовый шаблон. Каждый из них должен содержать уникальный контент для разных страниц вашего сайта, например, страницу «О нас», «Контакты» и «Услуги».

3. Создайте шаблон, который будет содержать ссылки на другие страницы вашего сайта. Используйте тег {% url 'name' %} для генерации URL-адресов на основе именованных URL-адресов в вашем проекте.

4. Создайте шаблон, который будет содержать приветственное сообщение на главной странице вашего сайта.

5. Создайте несколько различных макетов страниц (например, для страницы с контактами и страницы с информацией о компании) и используйте их для улучшения внешнего вида вашего сайта.

6. Создайте страницу, на которой будет отображаться список товаров. Для этого используйте список словарей с информацией о каждом товаре, такой как название, описание и цена. Отобразите информацию о каждом товаре на странице.

7. Создайте страницу, на которой будет отображаться список задач. Используйте список словарей с информацией о каждой задаче, такой как название и описание. Отобразите список задач на странице и добавьте возможность отметить задачи как выполненные.

8. Создайте страницу, на которой будет отображаться список пользователей. Используйте список словарей с информацией о каждом пользователе, такой как имя, фамилия и возраст. Отобразите на странице информацию о каждом пользователе.

9. Создайте страницу, на которой будет отображаться список книг. Используйте список словарей с информацией о каждой книге, такой как название, автор и год выпуска. Отобразите список книг на странице и добавьте возможность сортировки книг по различным критериям, например, по автору или году выпуска. Хранение данных о книгах возможно в словаре.

10. Создайте страницу, на которой будет отображаться расписание занятий. Используйте список словарей с информацией о каждом занятии, такой как название предмета, время начала и время окончания. Отобразите расписание занятий в виде таблицы на странице.

Тема «Базы данных и ORM»

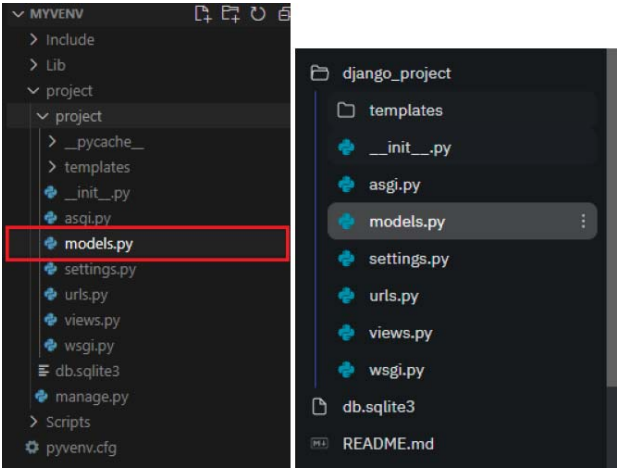
Задание 1. Создайте модель данных в Django для представления информации о людях. Модель должна содержать класс Person с полями name (имя типа CharField) и age (возраст типа IntegerField).

Ход работы

1. Изучите методические рекомендации.

2. Создайте модель Person, которая содержит поля name (имя) и age (возраст).

Чтобы начать работать с моделью, создайте файл models.py в проекте.



Теперь необходимо определить поля таблицы. Для создания модели используется следующий синтаксис.

Сначала импортируется компонент моделей из пакета django.db.

class ModelName(models.Model) – эта строка определяет нашу модель (объект).

- class – специальное ключевое слово для определения объектов.
- ModelName – имя нашей модели, можно поменять его при желании (специальные знаки и пробелы использовать нельзя). Всегда начинайте имена классов с прописной буквы.
- models.Model означает, что объект ModelName является моделью Django, так Django поймет, что он должен сохранить его в базу данных.

Дальше задаем поле модели: field_name. Чтобы это сделать, нам нужно определиться с типом полей: текст, число, дата, ссылка на другой объект.

Вместо FieldType необходимо написать нужный тип поля.

Теперь перейдем к созданию модели для проекта в файле models.py.

myvenv - models.py

```
1  from django.db import models
2
3  class Person(models.Model):
4      name = models.CharField(max_length=100)
5      age = models.IntegerField()
6
7      def __str__(self):
8          return self.name
```

В данной модели определим класс Person с полями name, age.

Метод «__str__» позволяет преобразовать объект класса и вернуть его в виде строки.

Задание 2. Создайте миграции для модели Person.

Ход работы

1. Изучите методические рекомендации.

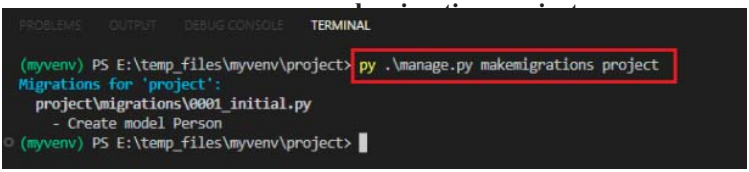
2. Создайте миграции для модели Person, которая содержит поля name (имя) и age (возраст).

Чтобы создать структуру модели во временный файл, которую в дальнейшем можно будет внести в базу данных, необходимо опре-

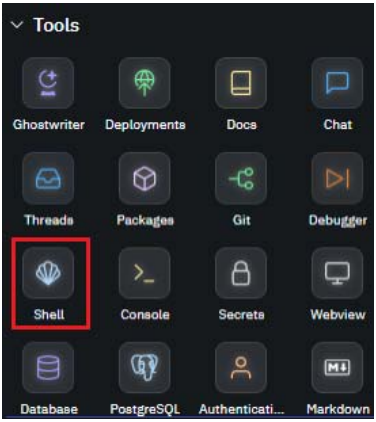
делить объявленные поля. Для этого переходим в терминал и вводим следующие команды:

python manage.py makemigrations project

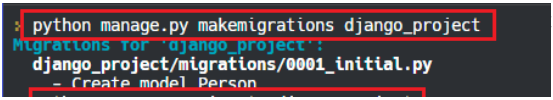
или



В случае использования сервиса Replit необходимо открыть инструмент Shell.



Затем необходимо вписать команду



В результате использования команд нами были созданы временные файлы структуры базы данных, которую описали в файле models.py.

Методические указания

Модель является единственным источником информации о ваших данных. Она содержит основные поля и поведение данных, которые вы храните. Как правило, каждая модель отображается в одну таблицу базы данных.

Выделяют следующие типы полей:

– BinaryField(): хранит бинарные данные;
– BooleanField(): хранит значение True или False (0 или 1);
– NullBooleanField(): хранит значение True или False или Null;
– DateField(): хранит дату;
– TimeField(): хранит время;
– DateTimeField(): хранит дату и время;
– DurationField(): хранит период времени;
– AutoField(): хранит целочисленное значение, которое автоматически инкрементируется, обычно применяется для первичных ключей;
– BigAutoField(): хранит 64-битное целочисленное значение, но в отличие от AutoField гарантирует, что число входит в диапазон от 1 до 9223372036854775807;
– SmallAutoField(): хранит 16-битное целочисленное значение в диапазоне от 1 до 32767;
– BigIntegerField(): представляет число – значение типа Number, которое укладывается в диапазон от 9223372036854775808 до 9223372036854775807. В зависимости от выбранной СУБД диапазон может немного отличаться;
– DecimalField(decimal_places=X, max_digits=Y): представляет значение типа Number, которое имеет максимум X разрядов и Y знаков после запятой;
– FloatField(): хранит значение типа Number, которое представляет число с плавающей точкой;
– IntegerField(): хранит значение типа Number, которое представляет целочисленное значение;
– PositiveIntegerField(): хранит значение типа Number, которое представляет положительное целочисленное значение (от 0 до 2147483647);
– PositiveBigIntegerField(): хранит значение типа Number, которое представляет положительное 64-битное целочисленное значение (от 0 до 9223372036854775807);
– PositiveSmallIntegerField(): хранит значение типа Number, которое представляет небольшое положительное целочисленное значение (от 0 до 32767);

– SmallIntegerField(): хранит значение типа Number, которое представляет небольшое целочисленное значение (от –32768 до 32767);
– CharField(max_length=N): хранит строку длиной не более N символов;
– TextField(): хранит строку неопределенной длины;
– EmailField(): хранит строку, которая представляет email-адрес. Значение автоматически валидируется встроенным валидатором EmailValidator;
– FileField(): хранит строку, которая представляет имя файла;
– FilePathField(): хранит строку, которая представляет путь к файлу длиной в 100 символов;
– ImageField(): хранит строку, которая представляет данные об изображении;
– GenericIPAddressField(): хранит строку, которая представляет IP-адрес в формате IP4v или IP6v;
– SlugField(): хранит строку, которая может содержать только буквы в нижнем регистре, цифры, дефис и знак подчеркивания;
– URLField(): хранит строку, которая представляет валидный URL-адрес;
– UUIDField(): хранит строку, которая представляет UUID-идентификатор;
– JSONField(): хранит данные в формате JSON, который представляет UUID-идентификатор.

Подробнее про типы полей можно узнать тут: https://docs.djangoproject.com/en/4.2/ref/models/fields/#field–types

Задание 3. Примените миграции для модели Person.

Ход работы

1. Изучите методические рекомендации.
2. Примените миграции для модели Person, созданные в задании 2.

После создания миграций необходимо применить их, чтобы обновить структуру базы данных. Для этого выполните команду

python manage.py migrate [название проекта]

или

py manage.py migrate [название проекта].

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py migrate project
Operations to perform:
  Apply all migrations: project
Running migrations:
  Applying project.0001_initial... OK
(myvenv) PS E:\temp_files\myvenv\project>
```

Для сервиса Replit необходимо использовать команду **python manage.py migrate [название папки проекта]**.

```
$ python manage.py makemigrations django_project
Migrations for 'django_project':
  django_project/migrations/0001_initial.py
    - Create model Person
$ python manage.py migrate django_project
Operations to perform:
  Apply all migrations: django_project
Running migrations:
  Applying django_project.0001_initial... OK
$
```

Команда migrate применяет все миграции, в том числе миграцию для модели Person, и обновляет базу данных, создавая таблицы и поля согласно определению модели.

Задания для самостоятельного выполнения

1. Создайте модель для хранения информации о пользователях (имя, фамилия, email) и примените миграции.

2. Создайте модель для хранения категорий товаров и примените миграции.

3. Создайте модель для хранения товаров (название, описание, цена, категория) и примените миграции.

4. Создайте модель User для хранения информации о пользователях. Включите поля, такие как имя, фамилия, электронная почта и дата регистрации. Создайте соответствующую таблицу в базе данных и напишите код для создания нового пользователя и получения списка всех пользователей.

5. Создайте модель Product для хранения информации о продуктах. Включите поля, такие как название, описание, цена и дата добавления. Создайте соответствующую таблицу в базе данных и напишите код для добавления нового продукта, изменения информации о продукте и удаления продукта.

5. Создайте модель Order для хранения информации о заказах. Включите поля, такие как связь с пользователем, список продуктов,

статус заказа и дата создания. Создайте соответствующую таблицу в базе данных и напишите код для создания нового заказа, обновления статуса заказа и получения списка всех заказов для определенного пользователя.

7. Создайте модель Category для хранения информации о категориях продуктов. Включите поле для названия категории. Создайте соответствующую таблицу в базе данных и напишите код для добавления новой категории, получения списка всех категорий и получения списка продуктов для определенной категории.

8. Создайте модель Review для хранения информации об отзывах пользователей о продуктах. Включите поля, такие как связь с пользователем, связь с продуктом, текст отзыва и рейтинг. Создайте соответствующую таблицу в базе данных и напишите код для добавления нового отзыва, получения списка отзывов для определенного продукта и получения среднего рейтинга продукта.

9. Создайте модель Address для хранения информации об адресах пользователей. Включите поля, такие как связь с пользователем, улица, город и почтовый индекс. Создайте соответствующую таблицу в базе данных и напишите код для добавления нового адреса для пользователя и получения списка адресов для определенного пользователя.

10. Создайте модель Event для хранения информации о событиях. Включите поля, такие как название, описание, дата и время начала, дата и время окончания. Создайте соответствующую таблицу в базе данных и напишите код для создания нового события, получения списка всех событий и получения списка событий в определенном временном диапазоне.

Тема «Django ORM: работа с данными и формами»

Задание 1. Используйте Django ORM для взаимодействия с базой данных. В предыдущей работе создана модель Person с полями name и age. Введите данные в базу через Django shell, проверьте их наличие, а затем отобразите их на главной странице веб-приложения. Используйте файлы views.py и urls.py для настройки отображения данных и index.html для визуализации. Запустите сервер, чтобы убедиться в корректности отображения данных.

Ход работы

1. Изучите методические рекомендации.

2. На основе созданной модели внесите данные в базу данных.

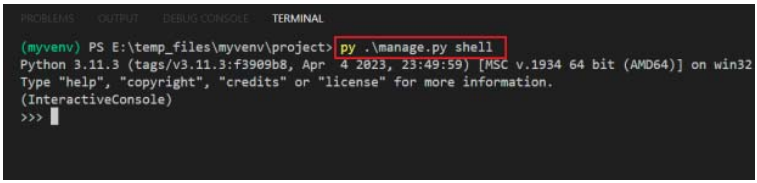
По результатам выполнения предыдущих заданий в базе данных создалась таблица по модели с необходимыми полями.

Попробуем внести данные в базу.

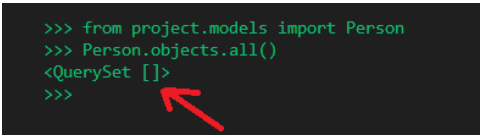
Открываем Django shell командой

python manage.py shell

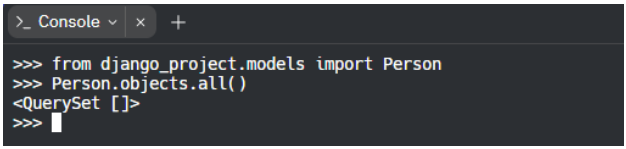
или



Убеждаемся, что модель пустая, то есть таблица не содержит данных (в квадратных скобках пусто).



Для сервиса replit команды будут выглядеть следующим образом.



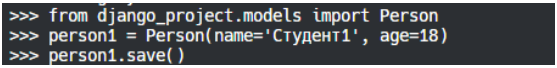
Если в терминале будет выведена строка <QuerySet []> – таблица не содержит записей.

Для тестирования модели внесем данные и сохраним изменения.

Для этого в том же Django Shell впишем следующие команды:

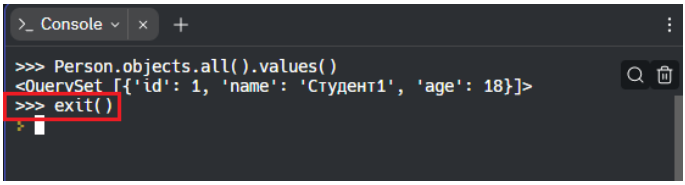
person1 = Person(name='Студент1', age=18)

person1.save()

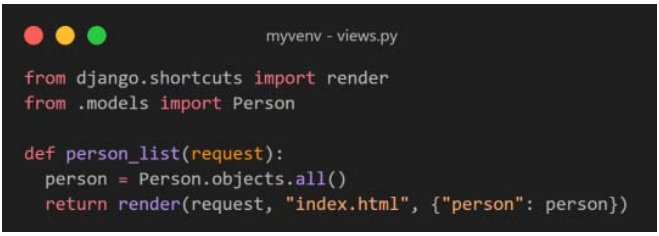


Данные внесены, так как QuerySet содержит данные в квадратных скобках.

Теперь выйдем из Django shell, прописав в терминале команду **exit()**



Для получения данных из модели перейдем в файл views.py и исправим код, дав логичное название функции и получив все данные из таблицы Person.



Теперь, чтобы отразить данные, добавим в шаблон index.html следующий код:

Подсказка: для быстрого формирования структуры html файла можно в редакторе Visual Studio Code написать «!» и нажать Enter.

Сформируется шаблон.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

Теперь изменим файл urls.py

myvenv - urls.py

```
from django.contrib import admin
from django.urls import path

from .views import person_list

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', person_list),
]
```

Запустим сервер командой

python manage.py runserver

или

py manage.py runserver

и посмотрим на результат.

ABC

Теперь внесенные данные отображаются на главной странице.

Методические указания

Django Shell представляет собой интерактивную Python-оболочку, интегрированную с вашим Django-проектом. Она позволяет напрямую взаимодействовать с моделями данных, что полезно для тестирования и манипуляции данными без необходимости создавать веб-интерфейсы. Запуск Django Shell осуществляется командой python manage.py shell, что дает прямой доступ к API Django ORM для работы с базой данных.

В Django каждая модель связана с таблицей в базе данных. Формирование новых записей в таблице модели осуществляется путем

создания экземпляра модели и вызова метода save(). Например, person1 = Person(name='Студент1', age=18) создает новый объект, а person1.save() сохраняет его в базе данных. Это добавляет запись в таблицу, соответствующую модели Person.

Для просмотра данных в модели используется метод objects.all(), который возвращает QuerySet всех объектов данной модели. Этот QuerySet может быть использован для дальнейших операций фильтрации, оценки или преобразования данных.

Для отображения данных из модели в веб-интерфейсе необходимо изменить файл views.py. В этом файле создается функция представления, которая извлекает данные из базы данных и передает их в шаблон. Например, функция может извлекать все объекты Person и передавать их в HTML-шаблон для отображения.

HTML-шаблоны в Django используются для определения структуры веб-страницы. Данные, передаваемые из представления, могут быть вставлены в шаблон с помощью Django Template Language. Это позволяет динамически генерировать HTML на основе данных, полученных из базы данных.

Файл urls.py в Django-проекте управляет маршрутизацией веб-запросов. Он определяет URL-адреса, которые связаны с конкретными представлениями. Изменения в urls.py позволяют направлять запросы на новые или измененные представления, что важно для правильного отображения данных на соответствующих веб-страницах.

После внесения изменений в код сервер можно перезапустить с помощью команды **python manage.py runserver**. Это запустит локальный сервер разработки, и можно будет проверить, как данные отображаются на созданной веб-странице, перейдя по соответствующему URL в браузере.

Задание 2. Разработайте и интегрируйте в ваш Django-проект модель StudentGroup, которая будет содержать информацию о группе студентов и связана с ранее созданной моделью Person. После создания моделей проведите миграцию данных, используя Django ORM. Введите данные через Django shell для Person и StudentGroup

и убедитесь в их наличии через запросы. Создайте представления и шаблоны для отображения информации о студентах и их группах на веб-странице и настройте соответствующие маршруты в файле urls.py. Запустите сервер и проверьте результаты в браузере.

Ход работы

Теперь попробуем создать модель StudentGroup, которая будет содержать данные о группе студента, и свяжем её с предыдущей моделью.

Создайте модели для таблиц, которые будут хранить ваши данные в том же файле models.py.

На рисунке показан конечный вариант файла models.py.

myvenv - models.py

```
from django.db import models

class StudentGroup(models.Model):
    group = models.CharField(max_length=10)
    entry_year = models.IntegerField()
    department = models.CharField(max_length=100)

    def __str__(self):
        return self.group


class Person(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    info = models.ForeignKey(StudentGroup(), on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

Теперь необходимо удалить из папки проекта следующие файлы и каталоги, если они есть в проекте:

1) файл db.sqlite3;
2) папку migrations;
3) папку __pycache__.

Запустите миграции для создания таблиц в базе данных с помощью команд:

python manage.py makemigrations project
python manage.py migrate project
python manage.py migrate

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py makemigrations project
Migrations for 'project':
  project\migrations\0001_initial.py
    - Create model StudentGroup
    - Create model Person
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py migrate project
Operations to perform:
  Apply all migrations: project
Running migrations:
  Applying project.0001_initial... OK
(myvenv) PS E:\temp_files\myvenv\project>
```

```
> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, django_project, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
```

В случае использования сервиса replit необходимо перейти в Shell и использовать следующие команды:

python manage.py makemigrations django_project

python manage.py migrate django_project

python manage.py migrate

```
> python manage.py makemigrations django_project
Migrations for 'django_project':
  django_project/migrations/0001_initial.py
    - Create model Person
    - Create model StudentGroup
> python manage.py migrate django_project
Operations to perform:
  Apply all migrations: django_project
Running migrations:
  Applying django_project.0001_initial... OK
>
```

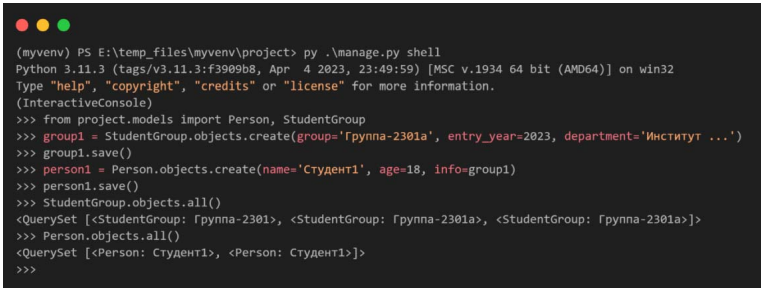
```
> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, django_project, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
```

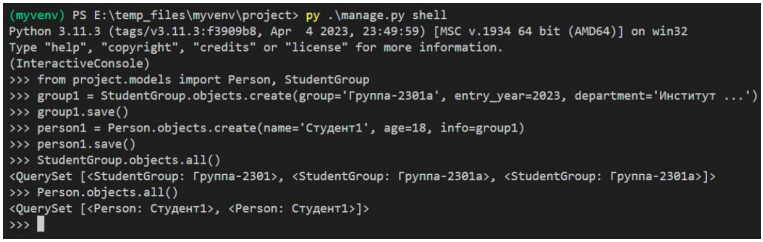
Миграции прошли успешно.

Теперь создайте данные с помощью Django shell.

Для открытия Django shell впишите команду

python manage.py shell





from project.models import Person, StudentGroup

group1 = StudentGroup.objects.create(group='Группа–2203a', entry_year=2023,

department='Институт ...')

group1.save()

person1 = Person.objects.create(name='ФИО студента', age=18, info=group1)

person1.save()

StudentGroup.objects.all()

Person.objects.all()

Для сервиса replit необходимо писать так:

from django_project.models import Person, StudentGroup

group1 = StudentGroup.objects.create(group='Группа–2203a', entry_year=2023,

department='Институт ...')

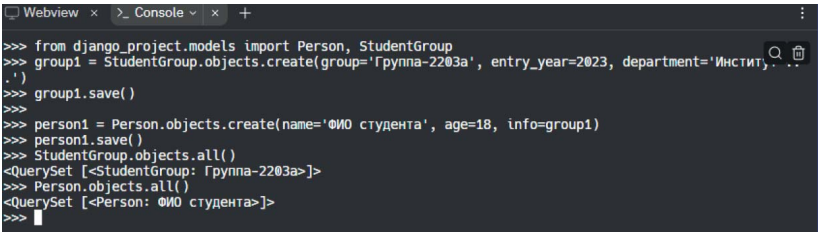
group1.save()

person1 = Person.objects.create(name='ФИО студента', age=18, info=group1)

person1.save()

StudentGroup.objects.all()

Person.objects.all()



В данном примере мы создаем группу и студента, который привязан к этой группе.

Теперь, чтобы выйти из Django Shell, напишите команду **exit()**

```
>>> exit()
```

Далее необходимо получить и обработать данные в существующем представлении views.py. В шаблон передаем данные person, которые получаем в строке 6 из таблицы.

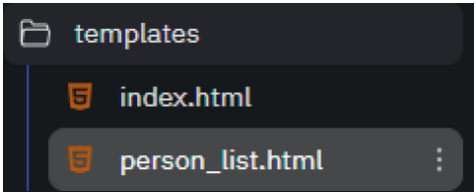
Переопределим передачу данных в шаблон person_list.html.

myvenv - views.py

```
1  from django.shortcuts import redirect, render
2
3  from project.models import Person
4
5  def person_list(request):
6      person = Person.objects.all()
7      return render(request, 'person_list.html', {'person': person})
8
```

Здесь получаем все данные из таблицы Person.

Теперь необходимо создать шаблон person_list.html в папке templates.



myvenv - person_list.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Список студентов</title>
</head>

<body>
    {% for per in person %}
    {{ per.name }} {{ per.age }} {{ per.info }} <br>
    {% endfor %}
</body>

</html>
```

Здесь выводим в виде «списка» имена, возраст и информацию о студентах.

Добавим в файл urls.py адрес до представления, который будет вызывать представление views.py.

myvenv - urls.py

```
1  from django.contrib import admin
2  from django.urls import path
3  from project.views import person_list
4
5  urlpatterns = [
6      path('admin/', admin.site.urls),
7      path('students', person_list),
8  ]
```

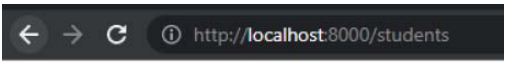
Запустим приложение через терминал py manage.py runserver.

```
(myvenv) PS C:\Users\A\Desktop\myvenv\project> py .\manage.py runserver
```

Запустится сервер по адресу http://127.0.0.1:8000/

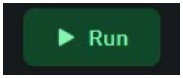
```
(myvenv) PS C:\Users\A\Desktop\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 03, 2023 - 10:20:33
Django version 4.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Теперь откроем браузер, перейдем по адресу 127.0.0.1:8000/students или localhost:8000/students и увидим результат работы. В браузере выводятся данные о студенте: его Ф. И. О., возраст и группа.



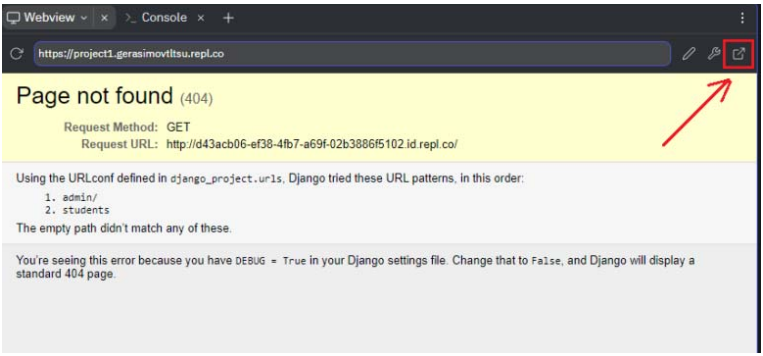
Студент1 18 Группа-2301а

В случае использования сервиса replit запустим проект кнопкой Run.

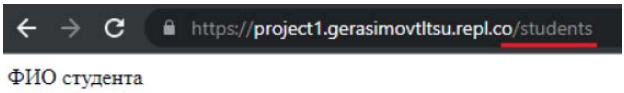


В webview откроется проект, в котором будет ошибка, – Page not found.

Откроем созданное приложение в новой вкладке.



И в адресной строке добавим открытие страницы students.



Методические указания

В Django модель представляет собой класс Python, который определяется для каждой таблицы базы данных. Модели включают поля и поведение данных. Создание модели StudentGroup, связанной с моделью Person, позволяет структурировать информацию о студенческих группах и отношениях между студентами и их группами. Модель StudentGroup может содержать поля, такие как название группы, год поступления и кафедра.

Миграции в Django используются для применения и отката изменений, внесенных в модели. После создания или изменения моделей необходимо выполнить миграции для обновления схемы базы данных. Это включает в себя команды makemigrations, которая генерирует скрипты миграции, и migrate, которая применяет их к базе данных.

Django Shell является мощным инструментом для работы с объектами модели напрямую. Он позволяет создавать, изменять, удалять и запрашивать данные в интерактивном режиме. Это полезно для тестирования моделей и быстрого внесения изменений в данные.

Представления в Django обрабатывают бизнес-логику и взаимодействуют с моделями для передачи данных в шаблоны. Шаблоны отвечают за формирование внешнего вида веб-страницы и могут динамически отображать данные, полученные из представлений. Создание эффективных представлений и соответствующих шаблонов позволяет представлять данные о студентах и их группах на веб-странице.

Маршрутизация в Django управляет тем, как запросы к веб-приложению обрабатываются представлениями. Она определяется в файле urls.py, где URL-адреса связываются с конкретными представлениями. Правильная настройка маршрутов позволяет пользователям переходить к нужным страницам для просмотра информации о студентах и их группах.

После разработки моделей, представлений, шаблонов и настройки маршрутизации важно тестировать приложение, запуская его на локальном сервере. Это позволяет проверить, как приложение работает в браузере, и убедиться, что все компоненты взаимодействуют корректно.

Задание 3. Разработайте форму для добавления студентов в ваш Django-проект, используя форму, созданную из модели. Создайте файл forms.py для определения формы и интегрируйте её в ваши представления в файле views.py. Измените файл index.html для отображения формы и настройте маршруты в файле urls.py, чтобы форма была доступна на главной странице сайта. Запустите сервер и проверьте работоспособность формы, убедитесь, что данные формы корректно добавляются в базу данных через интерфейс веб-страницы.

Ход работы

Создадим форму для добавления студентов.

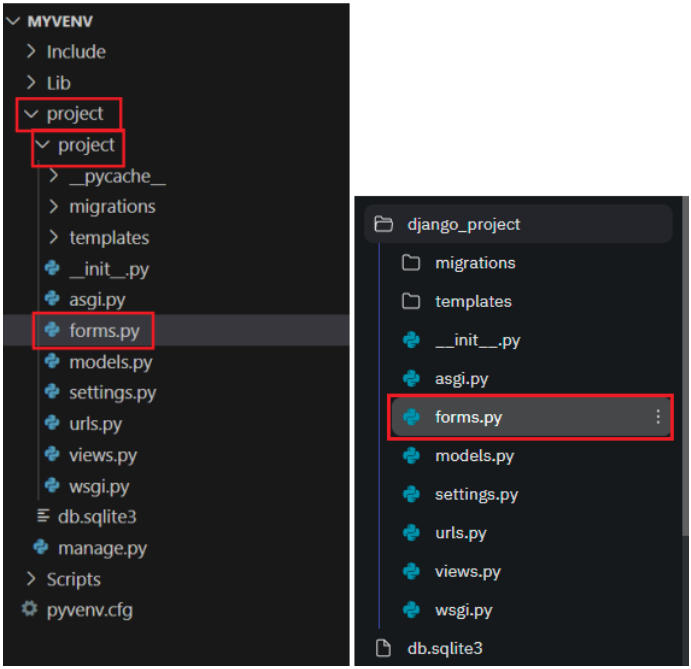
Формы в Django бывают двух видов:

1) форма, определяемая «вручную»;
2) форма, создаваемая из модели (models.py).

Оба типа формы определяются в файле forms.py.

Определим форму на основе модели, созданной в предыдущих практических работах.

Создадим файл forms.py



и внесем в него следующий код:

myvenv - forms.py

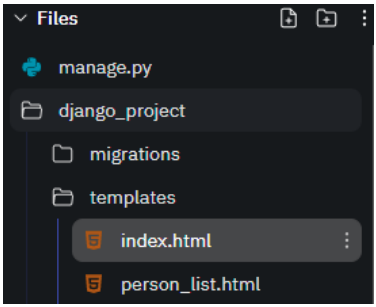
```
1  from django import forms
2
3  from .models import StudentGroup, Person
4
5  class PersonForm(forms.ModelForm):
6      class Meta:
7          model = Person
8          fields = ('__all__')
```

Теперь перейдем в файл views.py, чтобы вызвать нашу форму на странице.

myvenv - views.py

```python
from django.shortcuts import redirect, render
from django.views.decorators.csrf import csrf_exempt

from .models import Person
from .forms import PersonForm

@csrf_exempt
def create_student(request, *args, **kwargs):
    f = PersonForm(request.POST or None)
    if request.method == 'POST' and f.is_valid():
        stud_name = f.cleaned_data['name']
        stud_age = f.cleaned_data['age']
        stud_info = f.cleaned_data['info']
        f.save()
        return redirect('create_student')
    else:
        f = PersonForm()
    return render(request, 'index.html', {'form': f})

def person_list(request):
    person = Person.objects.all()
    return render(request, "person_list.html", {'person': person})
```

Переименуйте существующий файл ‘index.html’ в ‘’person_list.html’’ и создайте новый index.html в каталоге templates.



Затем необходимо задать, чтобы форма открывалась на главной странице. Для этого в файле urls.py добавим строку, чтобы он принял следующий вид:

myvenv - urls.py

```python
from django.contrib import admin
from django.urls import path

from .views import create_student, person_list

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', create_student, name='create_student'),
    path('students', person_list),
]
```

Теперь откроем файл index.html и передадим в него созданную форму.

myvenv - index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Тестовая страница</title>
</head>
<body>
    <form method="post" action="{% url 'create_student' %}">
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="Отправить">
    </form>
</body>
</html>
```

Запустим сервер командой

py manage.py runserver

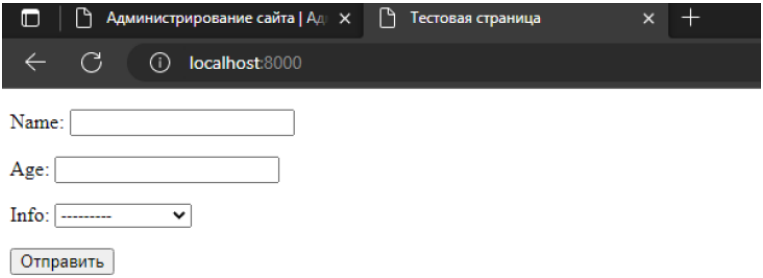
или

python manage.py runserver

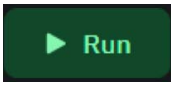
```
(myvenv) PS C:\Users\A\Desktop\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 03, 2023 - 11:27:33
Django version 4.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Откроем сайт по адресу http://127.0.0.1:8000/ или http://localhost:8000/

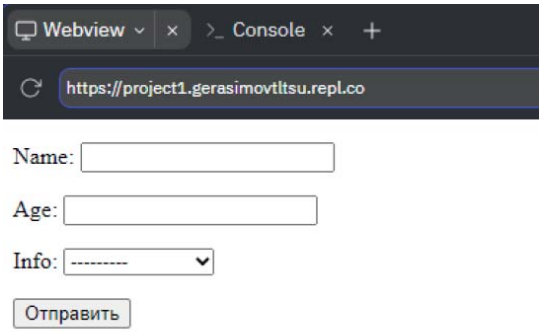
На экране появится созданная форма.



В случае использования сервиса Replit нажмите на кнопку Run.



Приложение будет доступно по адресу https://project1.{ваш_логин}.repl.co



Теперь при отправке формы данные будут добавляться в базу данных.

Методические указания

Формы в Django служат для сбора и обработки данных, вводимых пользователем через веб-интерфейс. Django предлагает два основных способа создания форм:

1) форма, определяемая «вручную»: позволяет разработчикам явно определять поля формы, их типы и валидацию;

2) форма, создаваемая из модели: использует информацию о полях модели для автоматического создания соответствующих полей формы, что ускоряет разработку и минимизирует дублирование кода.

Файл forms.py предназначен для определения форм Django. Здесь формы описываются через классы, наследующие от django.forms.ModelForm или django.forms.Form. Определение формы на основе модели включает автоматическое создание полей формы, соответствующих полям модели, что упрощает процесс разработки.

Для использования формы в приложении она должна быть интегрирована в представления (views.py). Представление загружает форму при GET-запросе и обрабатывает данные формы при POST-запросе. Обработка обычно включает проверку валидности данных и их сохранение или другие действия в зависимости от результатов валидации.

Маршрутизация в Django управляется файлом urls.py, который определяет URL-адреса и связывает их с конкретными представлениями. Добавление маршрута для формы позволяет пользователям переходить по определенному URL для доступа к форме.

Шаблоны (index.html) используются для отображения формы пользователю. В шаблоне можно определить, как форма будет отображаться, включая расположение и стилизацию полей и кнопок отправки.

Задания для самостоятельного выполнения

1. Напишите запрос, чтобы получить все объекты Product с рейтингом не ниже 4.

2. Отфильтруйте объекты User по имени и дате регистрации.

3. Отсортируйте объекты Event по дате и времени окончания.

4. Объедините объекты из моделей Product и Category с помощью запроса JOIN.

5. Аннотируйте объекты Product средним рейтингом отзывов.

6. Создайте форму набора для сбора нескольких адресов для одного пользователя.

7. Добавьте поля валидации к форме набора для обеспечения допустимых форматов адресов.

8. Обработайте и сохраните данные, отправленные через форму набора, в модель Address.

9. Создайте модельную форму для редактирования информации о пользователе.

10. Настройте представление, которое использует как форму набора, так и модельную форму.

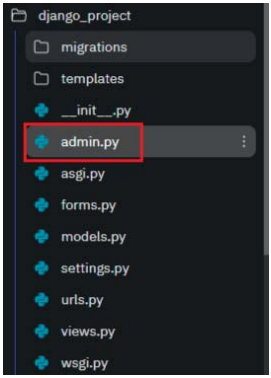
Тема «Сборка и запуск приложения на Python»

Задание 1. Интегрируйте ранее созданные модели данных в административную панель Django Admin вашего проекта.

Ход работы

1. Изучите методические рекомендации.
2. Зарегистрируйте созданные модели в файле admin.py.

Создадим файл в проекте.



myvenv - admin.py

```
from django.contrib import admin
from .models import Person, StudentGroup

admin.site.register(Person)
admin.site.register(StudentGroup)
```

Теперь необходимо создать администратора. Для этого в терминале введите команду **python manage.py createsuperuser**

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py createsuperuser
Имя пользователя (leave blank to use 'a'):
```

Введите имя пользователя, адрес электронной почты и пароль от учетной записи администратора.

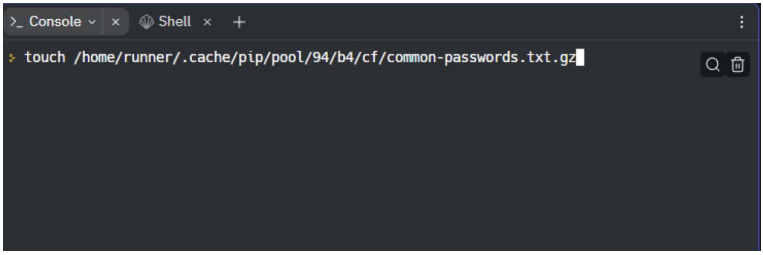
Если пароль не будет удовлетворять требованиям, то его все равно можно применить, подтвердив действие.

```
(myvenv) PS E:\temp_files\myvenv\project> py .\manage.py createsuperuser
Имя пользователя (leave blank to use 'a'): pmi
Адрес электронной почты: mail@pmi.tltsu
Password:
Password (again):
Введённый пароль слишком короткий. Он должен содержать как минимум 8 символов.
Введённый пароль слишком широко распространён.
Введённый пароль состоит только из цифр.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(myvenv) PS E:\temp_files\myvenv\project>
```

В случае использования сервиса replit при создании суперпользователя появится ошибка.

```
Traceback (most recent call last):
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/password_validation.py", line 236, in __init__
    with gzip.open(password_list_path, "rt", encoding="utf-8") as f:
  File "/nix/store/2vm88xw7513h9pyjyafw32cps51b0ia1-python3-3.8.12/lib/python3.8/gzip.py", line 58, in open
    binary_file = GzipFile(filename, gz_mode, compresslevel)
  File "/nix/store/2vm88xw7513h9pyjyafw32cps51b0ia1-python3-3.8.12/lib/python3.8/gzip.py", line 173, in __init__
    fileob) = self.myfileob) = builtins.open(filename, mode or 'rb')
FileNotFoundError: [Errno 2] No such file or directory: '/home/runner/.cache/pip/pool/94/b4/cf/common-passwords.txt.gz'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "manage.py", line 22, in <module>
    main()
  File "manage.py", line 18, in main
    execute_from_command_line(sys.argv)
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/core/management/__init__.py", line 442, in execute_from_command_line
    utility.execute()
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/core/management/__init__.py", line 436, in execute
    self.fetch_command(subcommand).run_from_argv(self.argv)
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/core/management/base.py", line 412, in run_from_argv
    self.execute(*args, **cmd_options)
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/management/commands/createsuperuser.py", line 80, in execute
    return super().execute(*args, **options)
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/core/management/base.py", line 458, in execute
    output = self.handle(*args, **options)
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/management/commands/createsuperuser.py", line 183, in handle
    validate_password(password2, self.UserModel(**fake_user_data))
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/password_validation.py", line 50, in validate_password
    password_validators = get_default_password_validators()
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/password_validation.py", line 22, in get_default_password_validators
    return get_password_validators(settings.AUTH_PASSWORD_VALIDATORS)
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/password_validation.py", line 36, in get_password_validators
    validators.append(klass(**validator.get("OPTIONS", {})))
  File "/home/runner/Project1/venv/lib/python3.8/site-packages/django/contrib/auth/password_validation.py", line 239, in __init__
    with open(password_list_path) as f:
FileNotFoundError: [Errno 2] No such file or directory: '/home/runner/.cache/pip/pool/94/b4/cf/common-passwords.txt.gz'
```

Для её исправления откройте Shell и введите следующую команду: **touch /home/runner/.cache/pip/pool/94/b4/cf/common-pas-swords.txt.gz**



Затем повторите действия по созданию суперпользователя.

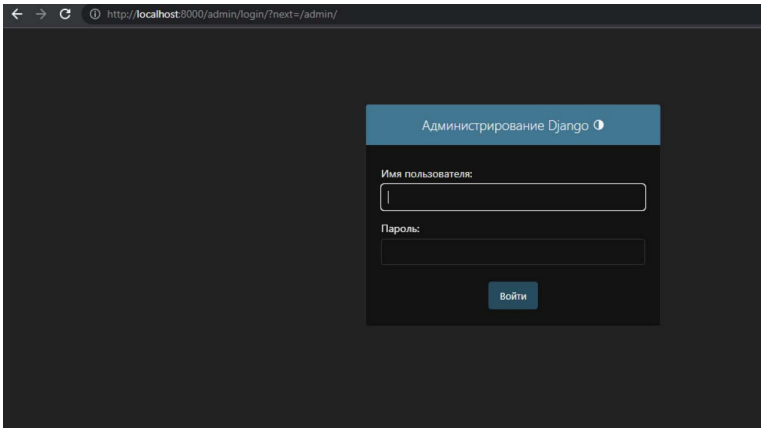
```
$ python manage.py createsuperuser
Имя пользователя (leave blank to use 'runner'): pmi
Адрес электронной почты: mail@pmi.tltsu
Password:
Password (again):
Введённый пароль слишком короткий. Он должен содержать как минимум 8 символов.
Введённый пароль состоит только из цифр.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

После создания суперпользователя запустим сервер командой **python manage.py runserver**

```
>_ Console
$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 10, 2023 - 09:28:52
Django version 4.2.2, using settings 'django_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

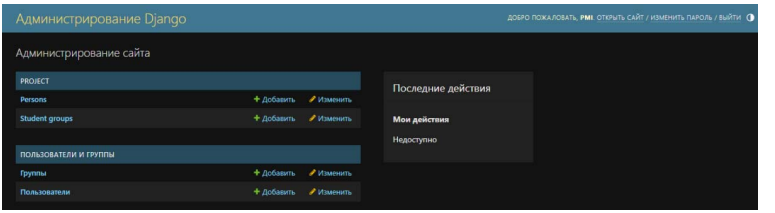
В случае использования replit нажмите на кнопку Run.

Перейдем в браузере по адресу http://127.0.0.1:8000/admin или http://localhost:8000/admin, или https://project1.{логин_replit}.repl.co/admin.



Введите логин и пароль в форму и нажмите «Войти».

Вы попали в панель администратора, и в ней представлены созданные модели.



На главной странице можно видеть модели Person и Student group, которые создавали.

Методические указания

Django Admin – мощный инструмент для управления данными приложения, который позволяет администраторам визуально работать с моделями данных. Для того чтобы модели данных были доступны в административной панели, необходимо их зарегистрировать. Это делается путём добавления специального кода в файл admin.py, который находится в соответствующем приложении проекта. Код регистрации обычно выглядит как admin.site.register(ModelName), где ModelName – это имя модели, которую вы хотите зарегистрировать.

Для доступа к административной панели Django необходим суперпользователь – пользователь с высокими привилегиями, который может управлять всеми аспектами Django-проекта. Создание суперпользователя происходит через командную строку с использованием команды **python manage.py createsuperuser**. В процессе создания будут запрошены имя пользователя, адрес электронной почты и пароль. Система может предупредить о слабом пароле, но позволит его использовать после подтверждения.

После создания суперпользователя необходимо запустить сервер разработки, используя команду **python manage.py runserver**. Затем можно перейти к административной панели, обычно доступной по адресу http://127.0.0.1:8000/admin. Вход в систему осуществляется с использованием ранее созданных учётных данных суперпользователя.

После входа в административную панель доступны все зарегистрированные модели. Интерфейс администратора позволяет добавлять, изменять и удалять записи в этих моделях, что обеспечивает удобное управление данными проекта.

Задание 2. Настройте систему регистрации и авторизации пользователей в вашем Django-проекте, используя встроенные возможности Django для управления аутентификацией.

Ход работы

Изучите методические рекомендации.

По умолчанию Django поставляется с представлением LoginView для страницы входа. Нам нужно только настроить несколько параметров в нашем проекте:

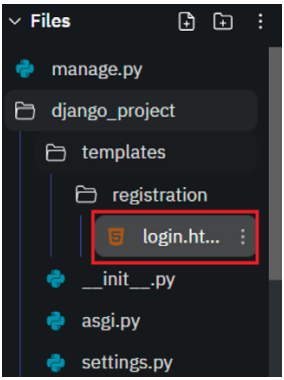
– URL-маршруты для системы аутентификации;
– создать HTML-шаблон для страницы входа;
– обновить файл settings.py.

Для начала добавим в файл urls.py страницы для входа и выхода на URL-префиксе accounts.

myvenv - urls.py

```
from django.contrib import admin
from django.urls import path, include

from .views import create_student, person_list

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', create_student, name='create_student'),
    path('students', person_list),
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Далее нужно создать новую директорию под названием registration в папке templates, а внутри нее создать необходимый HTML-файл шаблона.



Теперь откроем файл login.html и добавим в него следующий код:

myvenv - login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Страница входа</title>
</head>
<body>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Войти</button>
    </form>
</body>
</html>
```

На финальном этапе нам нужно уточнить, куда именно перенаправить пользователя после успешного входа. Для этого настроим переменную LOGIN_REDIRECT_URL. В файле settings.py добавляем следующую строку:

myvenv - settings.py

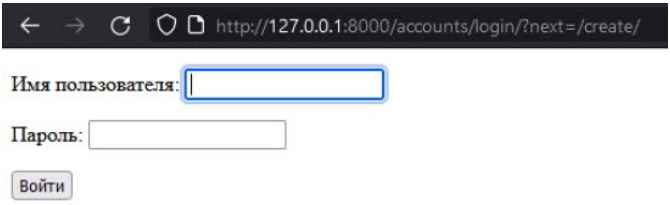
```
LOGIN_REDIRECT_URL = 'create_student'
```

Теперь пользователь будет направлен на URL-маршрут, который имеет название ‘create_student’ и является домашней страницей.

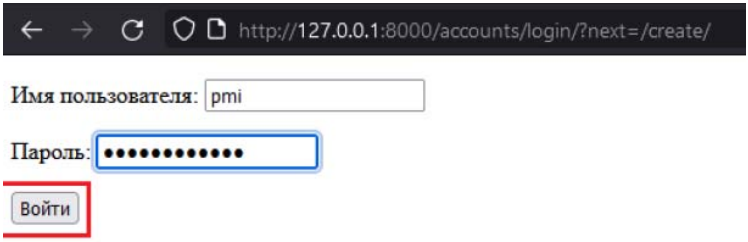
Запустим сервер командой **py manage.py runserver** или **python manage.py runserver**

```
(myvenv) PS E:\Files\ProgProjects\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 08, 2023 - 12:27:17
Django version 4.2.1, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Проверим страницу http://127.0.0.1:8000/accounts/login/ или http://localhost:8000/accounts/login/, или https://project1.{логин_replit}.repl.co/accounts/login/. Мы должны оказаться на созданной странице авторизации.



На созданной странице появилась форма авторизации. Впишем в нее ранее созданные логин и пароль от учетной записи администратора.

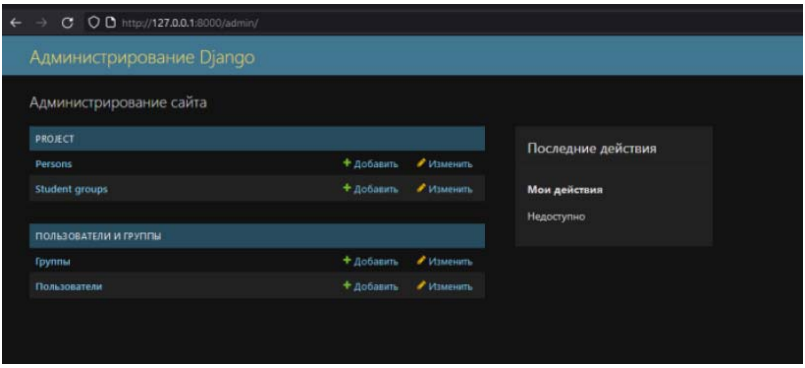


Нажмем кнопку «Войти».

Нас перевело на страницу с формой создания студента.

Перейдем так же на страницу http://127.0.0.1/admin или http://localhost/admin или https://project1.{логин_replit}.repl.co/admin.

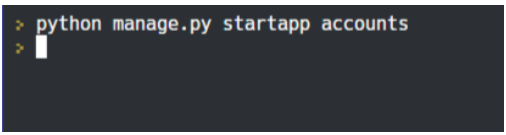
Как видим, в панели администратора мы также авторизовались.



Это значит, что теперь можем входить в приложение как администратор/пользователь через отдельно созданную страницу.

Для регистрации новых пользователей также понадобится представление (views). В Django для этого предусмотрен класс формы UserCreationForm, который сильно упрощает работу. По умолчанию в нем присутствуют три поля: username, password и password2.

Для страницы регистрации создадим новое приложение accounts командой **python manage.py startapp accounts**



Добавим приложение в файл settings.py (в переменную INSTALLED_APPS) основного проекта (django_project/settings.py или project/settings.py).

myvenv - settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'project',
    'accounts',
]
```

При использовании replit переменная INSTALLED_APPS будет выглядеть следующим образом:

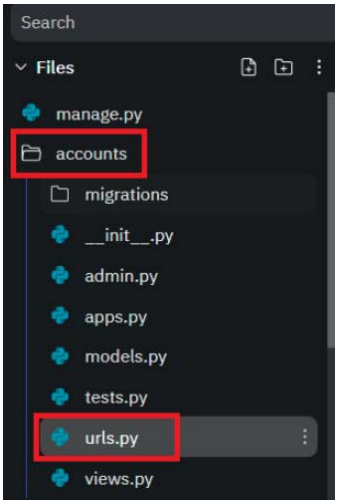
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django_project',
    'accounts',
]
```

Далее добавляем новый URL-маршрут в project/urls.py или django_project/urls.py, указывая на новое приложение.

myvenv - urls.py

```
from django.contrib import admin
from django.urls import path, include

from .views import create_student, person_list

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', create_student, name='create_student'),
    path('students', person_list),
    path('accounts/', include('django.contrib.auth.urls')),
    path('accounts/', include('accounts.urls')),
]
```

Теперь в приложении accounts необходимо создать файл urls.py.



Добавим в него следующий код:

myvenv - urls.py

```
from django.urls import path

from .views import SignUpView

urlpatterns = [
    path('signup/', SignUpView.as_view(), name='signup'),
]
```

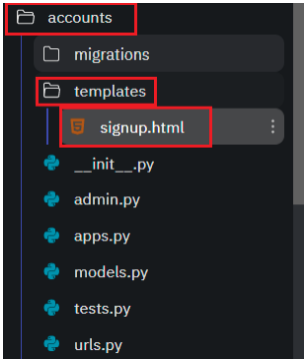
Теперь перейдем к представлению (views.py) в приложении accounts и используем встроенный класс UserCreationForm и общий класс представления CreateView.

myvenv - views.py

```
# accounts/views.py
from django.contrib.auth.forms import UserCreationForm
from django.urls import reverse_lazy
from django.views import generic


class SignUpView(generic.CreateView):
    form_class = UserCreationForm
    success_url = reverse_lazy('login')
    template_name = 'signup.html'
```

Теперь необходимо создать шаблон signup.html, который указали в представлении.

Создадим в приложении accounts папку templates и в папке templates создадим файл signup.html.



Теперь добавим в файл signup.html следующий код.

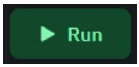
myvenv - signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Регистрация</title>
</head>
<body>
    <h2>Регистрация</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Зарегистрироваться</button>
    </form>
</body>
</html>
```

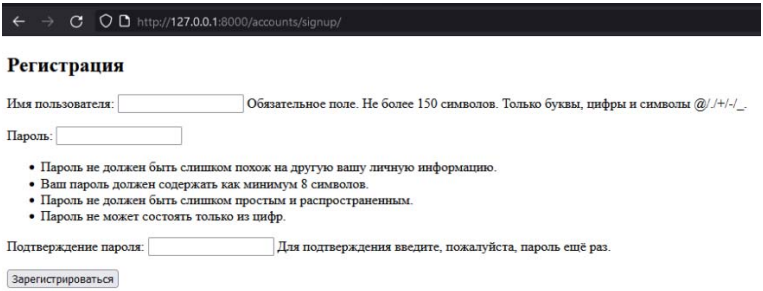
Запускаем локальный веб-сервер через команду **py manage.py runserver**

```
(myvenv) PS E:\Files\ProgProjects\myvenv\project> py .\manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 08, 2023 - 13:02:41
Django version 4.2.1, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[08/May/2023 13:02:48] "GET /accounts/signup/ HTTP/1.1" 200 1962
```

В случае использования replit нажимаем на кнопку Run.

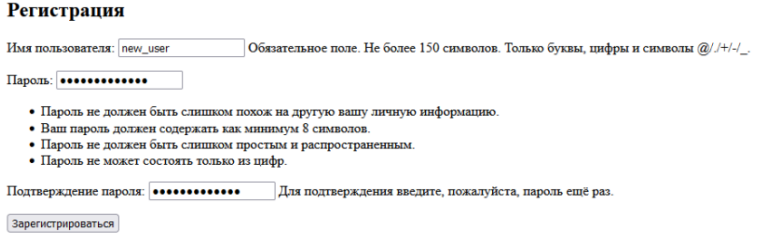


Перейдите в браузере по адресу http://127.0.0.1:8000/accounts/signup/ или https://project1.{ваш_логин}.repl.co/accounts/signup/.

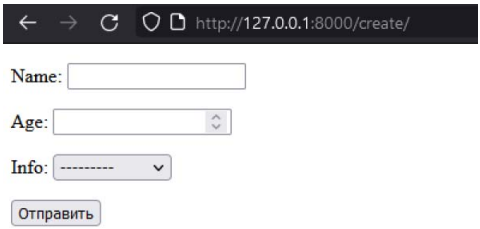


Перед нами появилась страница регистрации.

Введите имя пользователя и пароль для нового пользователя.

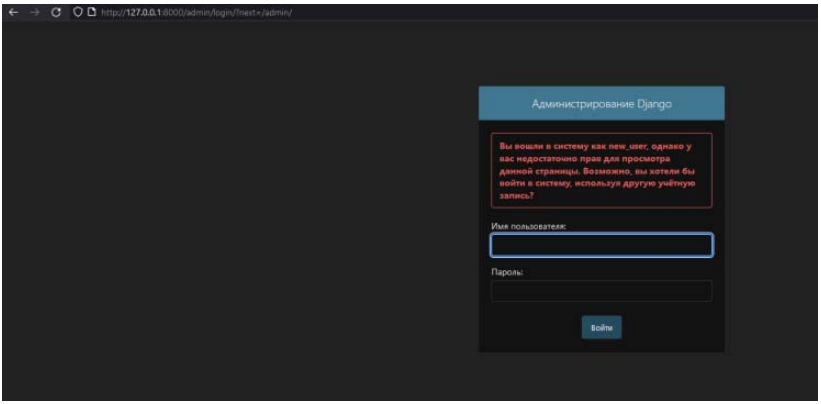


И нажмите кнопку «Зарегистрироваться».

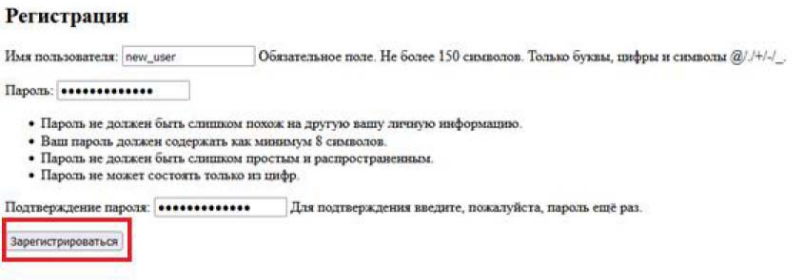


После регистрации мы окажемся на странице http:// 127.0.0.1: 8000/accounts/login/.

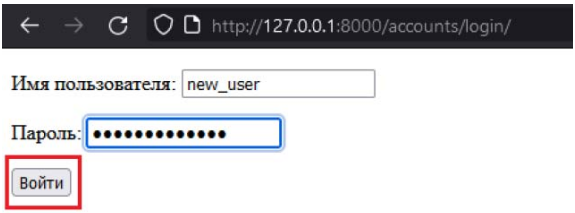
Введем данные от нового пользователя и нажмем кнопку «Войти».



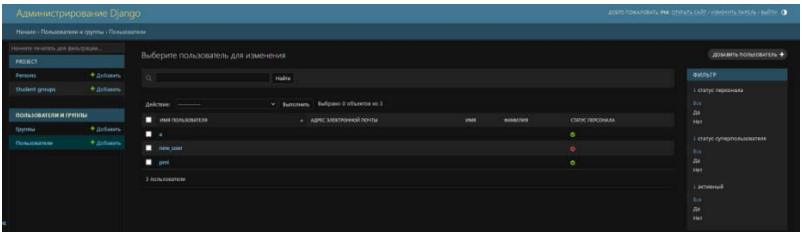
Нас отправит на страницу создания студента http://127.0.0.1:8000/create/ или https://project1.{Ваш логин Replit}.repl.co/create/.



Попробуем перейти по адресу http://127.0.0.1:8000/admin или https://project1.{Ваш логин Replit}.repl.co/admin/ и увидим сообщение о том, что создан простой пользователь и у него нет доступа к панели администратора.



Если подобное сообщение появилось, значит, все действия выполнены верно. Войдем в панель администратора под логином и паролем администратора, перейдем по адресу http://127.0.0.1:8000/admin/auth/user/ или https://project1.{Ваш логин Replit}.repl.co/admin/auth/user/ и увидим, что у нового пользователя статус персонала заблокирован. Это означает, что новый пользователь зарегистрирован как обычный пользователь приложения и не является администратором.



Методические указания

Django предоставляет обширные встроенные средства для управления аутентификацией пользователей, включая страницы входа и выхода, а также систему управления сессиями. Эти функции помогают разработчикам быстро и безопасно реализовать необходимые механизмы авторизации.

Настройка маршрутов в Django выполняется путём добавления специальных путей в файл urls.py проекта. Для системы аутентификации обычно создаются маршруты для страниц входа и выхода, а также для регистрации новых пользователей. Эти маршруты обеспечивают переход к соответствующим представлениям, которые обрабатывают формы входа и регистрации.

HTML-шаблоны для аутентификации создаются в директории registration в папке templates. Шаблон login.html обычно содержит форму входа, которая позволяет пользователям аутентифицироваться в системе. В шаблоне могут быть использованы стандартные поля Django для ввода логина и пароля, а также кнопки для отправки формы.

В файле settings.py проекта необходимо указать дополнительные параметры конфигурации, такие как LOGIN_REDIRECT_URL, который определяет URL, на который будет перенаправлен пользователь после успешного входа. Этот параметр обеспечивает более гладкий пользовательский опыт, направляя пользователей на нужную страницу сразу после аутентификации.

Для регистрации новых пользователей берется класс формы UserCreationForm, который включает поля для имени пользователя и пароля. Эта форма обычно размещается на странице регистрации и позволяет быстро и безопасно добавлять новых пользователей в систему. После заполнения и отправки формы данные проверяются на валидность, и при успешной регистрации пользователь может использовать свои учетные данные для входа в систему.

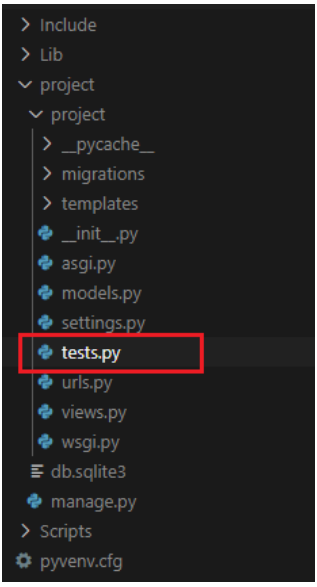
После настройки маршрутов, создания шаблонов и конфигурации параметров необходимо провести тестирование системы. Запуск локального сервера и переход по соответствующим URL позволят проверить работоспособность страниц входа и регистрации, а также корректность перенаправлений и обработки форм. Тестирование должно также включать проверку безопасности системы аутентификации.

Задание 3. Настройте и проведите тестирование моделей вашего веб-приложения на Django.

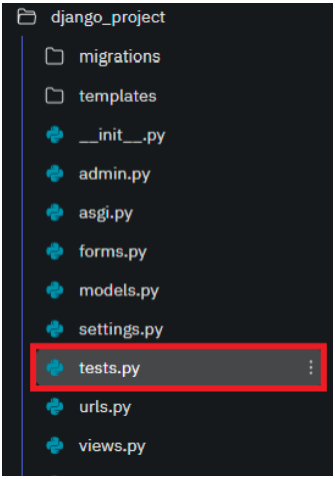
Методические указания

Для отладки веб-приложения необходимо произвести его тестирование. Тестирование может распространяться на все компоненты: модели, представления и так далее.

Для тестирования модели в приложении необходимо создать файл tests.py в папке project.



В случае использования replit создайте файл tests.py в папке django_project.



Напишите следующий тест-кейс в файл tests.py

```
myvenv - tests.py

from django.test import TestCase
from .models import Person, StudentGroup

class PersonTest(TestCase):
  def setUp(self):
    self.group = StudentGroup.objects.create(group='Группа1', entry_year=2023, department =
    self.person = Person.objects.create(name='Студент1', age=18, info=self.group)

  def test_get_items(self):
    person2 = Person.objects.get(name='Студент1', age=18)
```

Запустите тест командой **python manage.py test**

```
>_ Console  ×  +

$ python manage.py test
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.004s
```

Тесты успешно пройдены.

Задание 4. Разверните ваше Django-веб-приложение, используя сервис ngrok для организации доступа к нему из Интернета.

Ход работы

После окончания разработки веб-приложение должно быть размещено в Интернете, чтобы общественность могла получить доступ к нему из любого места.

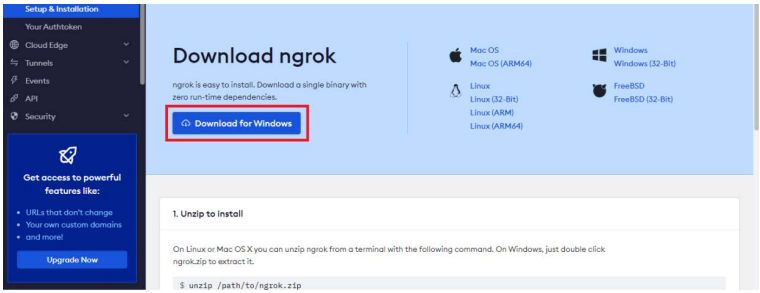
Для развертывания приложения понадобится отдельный сервер.

Перейдем к публикации. Для публикации будем использовать ngrok.

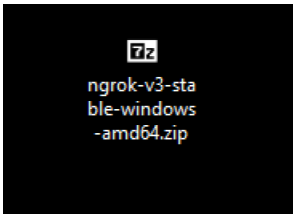
Ngrok – это сервис, который позволяет организовать удалённый доступ на веб-сервер или какой-то другой сервис, запущенный на вашем ПК.

Зарегистрируйтесь на сайте https://dashboard.ngrok.com/signup.

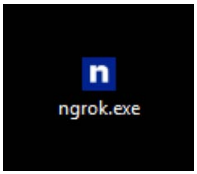
На главной странице скачайте ngrok.



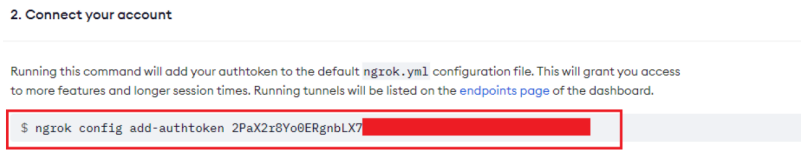
Скачается архив с ngrok.



Разархивируйте его.



Перейдите обратно в браузер, пролистайте страницу ниже, скопируйте команду для добавления токена приложения

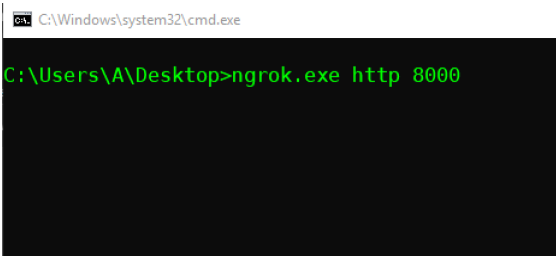


и из командной строки вызовите следующую команду:

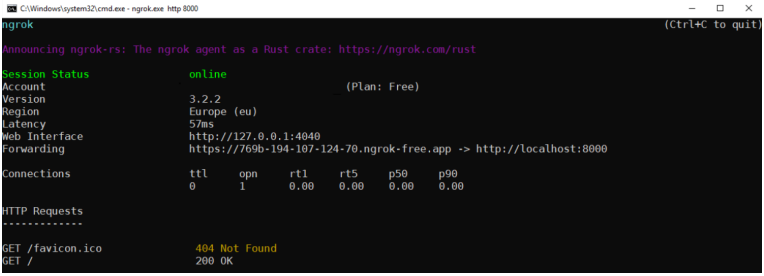
ngrok.exe config add–authtoken *ваш_токен*

Токен для авторизации сохранится в конфигурационном файле ngrok.yml.

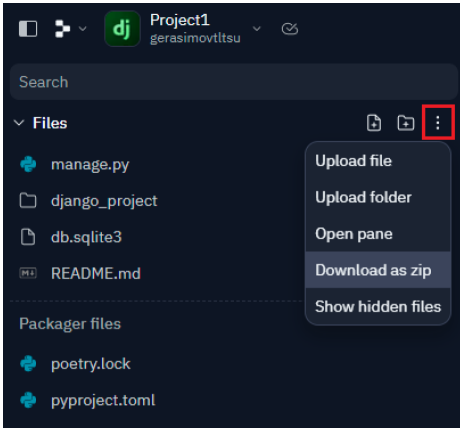
Запустите сервер ngrok через командную строку командой **ngrok.exe http 8000**, где 8000 – порт, на котором будет запущен проект Django.



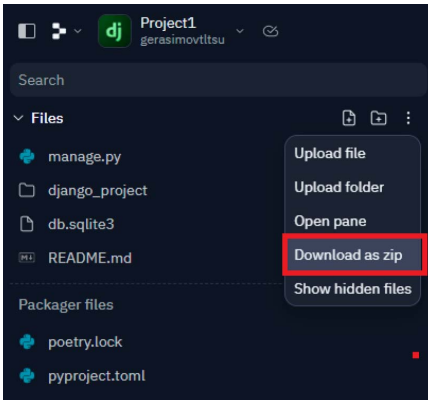
В результате вы получите следующее окно:



Если вы использовали сайт repl.it, то скачайте с него ваш проект, нажав три точки в списке файлов и выбрав пункт Download as zip.

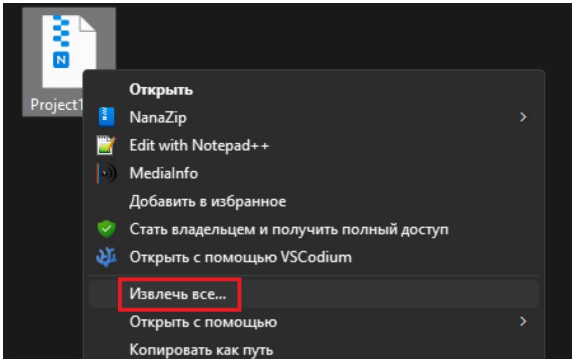


Выберите пункт Download as zip.

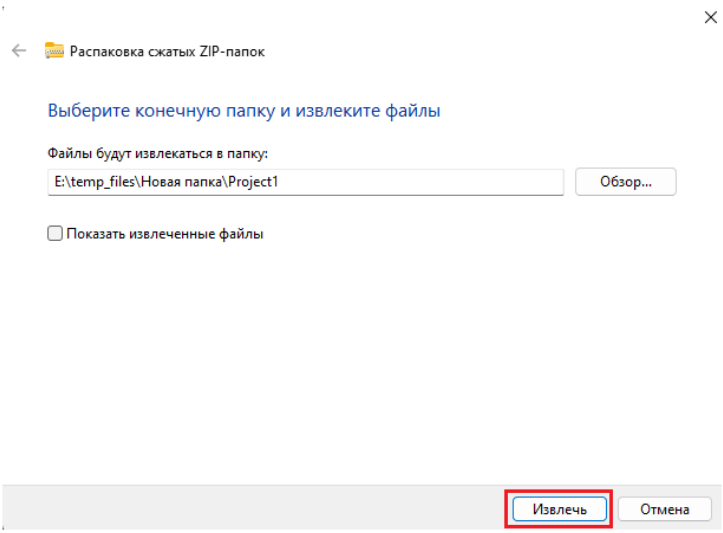


Скачается архив с проектом; его необходимо разархивировать.

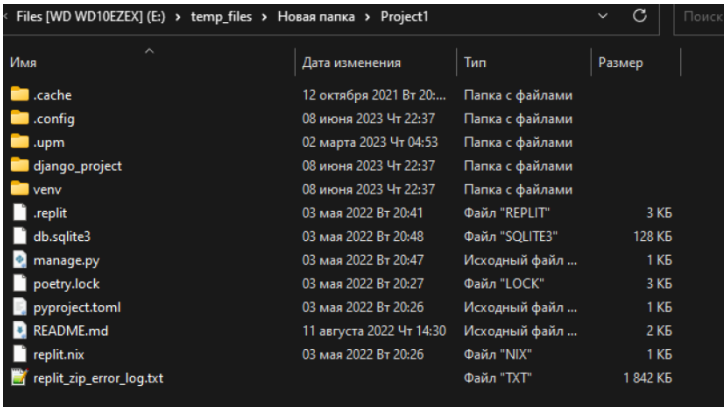
Для этого нажмем по архиву правой кнопкой мыши и выберем пункт «Извлечь все».



Нажимаем кнопку «Извлечь».



В папке Project1 появились извлеченные файлы.



Перейдите в папку django_project, откройте файл settings.py проекта Django и добавьте/измените параметры:

– DEBUG установите значение False;
– ALLOWED_HOSTS определите как «['*']» для возможности принятия запросов (без кавычек);
– CSRF_TRUSTED_ORIGINS (необходимо добавить после параметра ALLOWED_HOSTS), который представляет список

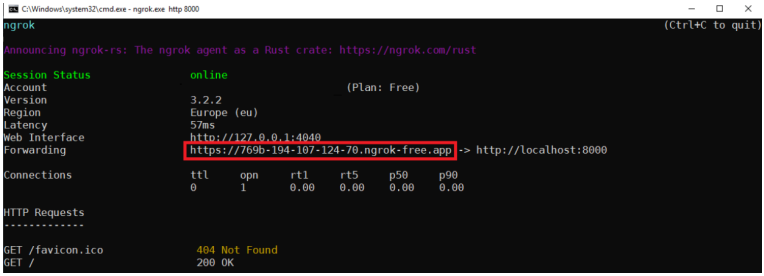
доверенных адресов для выполнения запросов со значением [‘https://*.ngrok-free.app/’].

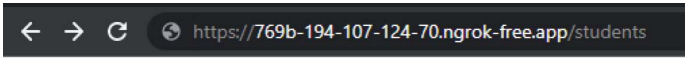
```
myvenv - settings.py

1  DEBUG = False
2
3  ALLOWED_HOSTS = ['*']
4  CSRF_TRUSTED_ORIGINS = ['https://*.ngrok-free.app/']
```

Запустите сервер Django командой **py manage.py runserver**

```
(myvenv) PS C:\Users\A\Desktop\myvenv\project> py .\manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
May 10, 2023 - 09:25:08
Django version 4.2, using settings 'project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Перейдите в ngrok. URL-адрес, который выдает ngrok в строке Forwarding, скопируйте-вставьте/перепишите в адресную строку браузера.





а 23 Группа-2301а

Ваш Django-проект успешно размещен в Интернете, и теперь он будет работать каждый раз, когда вы запускаете ngrok и сервер Django.

Методические указания

Django предоставляет мощные средства для создания и обработки форм. Мы можем создавать HTML-формы с помощью классов Form и ModelForm. Кроме того, Django предоставляет встроенную валидацию данных, введенных в формы.

После того как пользователь отправит форму, данные будут проверены на валидность. Если данные не прошли валидацию, Django вернет ошибки, которые можем отобразить пользователю для исправления.

Далее можно приступить к развертыванию приложения на сервере. Для этого нам необходимо настроить базу данных, статические файлы и другие параметры в файле настроек проекта.

Для развертывания приложения Django на сервере нам нужно установить Django и все его зависимости на сервере, скопировать файлы проекта на сервер, настроить базу данных и другие параметры в файле настроек и запустить сервер Django. В качестве сервера можно использовать ngrok.

Во время развертывания приложения на сервере необходимо настроить обработку статических файлов, таких как изображения, CSS и JavaScript. Django предоставляет возможности для хранения и обслуживания статических файлов.

Один из ключевых элементов Django – его административная панель. Административная панель позволяет управлять данными вашего приложения, такими как пользователи, группы, записи блога и другие, без необходимости писать дополнительный код.

Для того чтобы начать использовать административную панель Django, следует зарегистрировать модели данных в административной панели с помощью классов ModelAdmin. После этого сможем добавлять, удалять и изменять данные в административной панели.

Задания для самостоятельного выполнения

1. Создайте административный интерфейс для модели User, который позволит добавлять новых пользователей и просматривать список всех пользователей.

2. Реализуйте административный интерфейс для модели Category, который позволит добавлять новые категории товаров и просматривать список всех категорий.

3. Создайте административный интерфейс для модели Product, который позволит добавлять новые товары, просматривать и изменять информацию о них, а также удалять товары.

4. Реализуйте возможность создания новых заказов через административный интерфейс для модели Order. Добавьте функциональность для обновления статуса заказа и просмотра списка всех заказов для каждого пользователя.

5. Создайте административный интерфейс для модели Review, который позволит добавлять новые отзывы о продуктах, просматривать отзывы для каждого продукта и получать средний рейтинг для каждого продукта.

6. Реализуйте функционал добавления новых адресов пользователей через административный интерфейс для модели Address. Добавьте возможность просмотра списка адресов для каждого пользователя.

7. Создайте административный интерфейс для модели Event, который позволит добавлять новые события, просматривать список всех событий и получать список событий в определенном временном диапазоне.

8. Реализуйте функционал поиска пользователей по имени или фамилии через административный интерфейс для модели User. Добавьте возможность отображения списка пользователей, удовлетворяющих заданному критерию поиска.

9. Создайте административный интерфейс для модели Category, который позволит изменять информацию о категориях товаров. Добавьте функциональность для обновления названия каждой категории.

10. Реализуйте возможность фильтрации продуктов по цене в административном интерфейсе для модели Product. Добавьте функциональность для отображения списка продуктов, удовлетворяющих заданному диапазону цены.

Модуль 3. ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И МАШИННОЕ ОБУЧЕНИЕ. ПРИМЕРЫ НА PYTHON

Тема 8. Введение в искусственный интеллект и основные методы машинного обучения

Основные задачи систем искусственного интеллекта

Искусственный интеллект (ИИ) – область компьютерных наук, которая занимается созданием систем, способных имитировать интеллектуальное поведение человека.

Системы искусственного интеллекта (ИИ) представляют собой программные или аппаратные системы, способные выполнять задачи, которые требуют интеллектуальных способностей, обычно связанных с человеческим разумом. ИИ-системы используют различные методы и алгоритмы для анализа данных, обучения, принятия решений и выполнения задач, которые раньше требовали присутствия человека.

К системам искусственного интеллекта относятся:

– экспертные системы: системы, которые используют базы знаний, созданные экспертами в определенной области, чтобы принимать решения и решать проблемы, которые обычно требуют экспертных знаний;

– системы обработки естественного языка (Natural Language Processing, NLP): системы позволяют компьютерам взаимодействовать с людьми на естественном языке. Они способны анализировать, понимать и генерировать текст, распознавать речь и выполнять задачи, связанные с языком;

– системы машинного обучения: используют алгоритмы и статистические модели для обучения на основе данных и прогнозирования или принятия решений без явного программирования. Примеры включают нейронные сети, решающие деревья, методы кластеризации и многое другое;

– системы компьютерного зрения: позволяют компьютерам «видеть» и анализировать изображения и видео. Эти системы ис-

пользуют алгоритмы для распознавания образов, классификации объектов, обнаружения движения и других задач, связанных с обработкой визуальной информации;

– робототехника: сочетание ИИ и робототехники, где физические роботы обладают способностью воспринимать окружающую среду, принимать решения и взаимодействовать с ней. Роботы могут выполнять различные задачи – от промышленных операций до помощи в домашних делах и медицинских процедурах;

– системы автоматического планирования и принятия решений: используют алгоритмы и модели для принятия оптимальных решений и планирования действий. Они могут использоваться в таких областях, как логистика, автономные транспортные средства и управление ресурсами. Системы ИИ стремятся выполнять задачи, требующие понимания, обучения, решения проблем и адаптации на основе предоставленных данных.

Системы ИИ позволяют компьютерам обрабатывать и анализировать большие объемы данных, распознавать образы, принимать решения на основе логики и опыта, а также обучаться на основе предыдущих данных для улучшения своей производительности. Системы искусственного интеллекта (ИИ) имеют широкий спектр задач, которые они могут решать. Некоторые из основных задач, которые часто встречаются в системах искусственного интеллекта, включают классификацию, кластеризацию и регрессию.

Классификация в ИИ – задача, в которой модель обучается присваивать новым данным определенные метки классов на основе предварительно известных классов. В Python для классификации широко используется библиотека scikit-learn.

На рис. 133 представлен фрагмент кода, описывающий задачу классификации цифровых изображений рукописных чисел. Есть набор изображений рукописных цифр (0–9), каждое изображение сопровождается соответствующей меткой с указанием правильной цифры. На основе этого набора данных модель может быть обучена распознавать и классифицировать новые изображения рукописных чисел.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Загрузка набора данных рукописных цифр
digits = datasets.load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
    test_size=0.2)

# Создание и обучение модели классификатора K ближайших соседей
classifier = KNeighborsClassifier()
classifier.fit(X_train, y_train)

# Классификация нового изображения
new_image = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
    , 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
    39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
    58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
    77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95,
    96, 97, 98, 99]
predicted_class = classifier.predict([new_image])
print(predicted_class)
```

Рис. 133. Пример классификации

Кластеризация в ИИ – задача, в которой модель группирует данные на основе их сходства без предварительно известных классов. В примере (рис. 134) также используется библиотека scikit-learn для выполнения кластеризации.

```
from sklearn.cluster import KMeans
import numpy as np

# Набор данных о покупках в интернет-магазине
purchases = np.array([[10, 15], [20, 25], [12, 18], [22, 30], [8, 10], [30, 35]])

# Создание и обучение модели кластеризации K-средних
kmeans = KMeans(n_clusters=2)
kmeans.fit(purchases)

# Кластеризация новых данных
new_purchase = [[15, 20]]
predicted_cluster = kmeans.predict(new_purchase)
print(predicted_cluster)
```

Рис. 134. Пример кластеризации

Рассмотрим задачу кластеризации набора данных о покупках в интернет-магазине. У нас есть информация о покупках различных пользователей – сумма покупки, количество товаров и время

покупки. Модель может группировать пользователей на основе их сходства в покупках для выявления сегментов или групп покупателей с похожими покупательскими предпочтениями.

Регрессия в ИИ – задача, в которой модель предсказывает непрерывные значения на основе предоставленных данных.

Например, рассмотрим задачу предсказания цены домов на основе их характеристик, таких как площадь, количество комнат, расположение и т. д. Модель регрессии (рис. 135) может быть обучена на основе известных данных о домах и их ценах для предсказания цены нового дома на основе его характеристик.

```
from sklearn.linear_model import LinearRegression

# Данные о домах
house_features = [[120, 3], [180, 4], [150, 3], [200, 4], [100, 2]]
house_prices = [250000, 350000, 300000, 400000, 200000]

# Создание и обучение модели линейной регрессии
regression = LinearRegression()
regression.fit(house_features, house_prices)

# Предсказание цены нового дома
new_house = [[160, 3]]
predicted_price = regression.predict(new_house)
print(predicted_price)
```

Рис. 135. Пример регрессии

В рассмотренных примерах представлены базовые задачи ИИ – классификация, кластеризация и регрессия. Эти задачи позволяют системам ИИ обрабатывать данные, обнаруживать паттерны и структуры в информации, а также принимать решения на основе этих анализов.

Машинное обучение является подмножеством ИИ и представляет собой метод, который позволяет компьютерной системе автоматически извлекать знания из данных и использовать их для принятия решений или предсказания результатов на новых данных.

Машинное обучение позволяет системам ИИ обучаться на больших объемах данных и автоматически находить скрытые паттерны и зависимости. Это позволяет им обрабатывать сложные и неструктурированные данные, такие как изображения, аудио и тексты.

Модели машинного обучения могут быть обучены классифицировать объекты на основе их признаков или предсказывать значения на основе имеющихся данных.

Выделяют следующие типы машинного обучения.

Обучение с учителем (Supervised Learning). В обучении с учителем модель обучается на основе помеченных данных, где каждый образец данных имеет соответствующую выходную метку или правильный ответ. Задача модели состоит в том, чтобы научиться предсказывать правильные ответы для новых данных, которые ещё не были встречены во время обучения. В Python для реализации обучения с учителем можно использовать библиотеки scikit-learn или TensorFlow.

Например, есть набор данных о ценах домов, включающий информацию о площади, количестве комнат и цене продажи. Задача состоит в том, чтобы построить модель, которая может предсказывать цену дома на основе его характеристик. Можно использовать обучение с учителем (рис. 136), где входные данные (площадь, количество комнат) являются признаками, а цена – выходной меткой или целевой переменной.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Загрузка набора данных о домах
dataset = datasets.load_boston()
X = dataset.data
y = dataset.target

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Создание модели линейной регрессии
model = LinearRegression()

# Обучение модели на обучающем наборе данных
model.fit(X_train, y_train)

# Предсказание цены дома на тестовом наборе данных
predictions = model.predict(X_test)
```

Рис. 136. Обучение с учителем

Обучение без учителя (Unsupervised Learning). В обучении без учителя модель обучается на непомеченных данных, то есть данных без

явных выходных меток. Задача модели в этом случае заключается в выявлении скрытых структур, паттернов или группировке данных на основе их сходства. В Python для реализации обучения без учителя можно использовать библиотеки scikit-learn или TensorFlow.

Например, есть набор данных о покупках в интернет-магазине, но без информации о категориях или классах покупок. Задача состоит в том, чтобы исследовать данные и выявить группы или кластеры покупок на основе их сходства. Можно использовать обучение без учителя (например, алгоритм кластеризации), чтобы автоматически группировать покупки в различные сегменты (рис. 137).

```
from sklearn import datasets
from sklearn.cluster import KMeans

# Загрузка набора данных о цветках ириса
dataset = datasets.load_iris()
X = dataset.data

# Создание модели кластеризации KMeans
model = KMeans(n_clusters=3)

# Обучение модели на данных
model.fit(X)

# Получение меток кластеров для каждого образца данных
labels = model.labels_
```

Рис. 137. Обучение без учителя

Обучение с частичным привлечением учителя (Semi-Supervised Learning). В обучении с частичным привлечением учителя модель обучается на наборе данных, в котором только некоторые образцы имеют явные выходные метки, а другие образцы не имеют меток. Это позволяет модели использовать информацию из помеченных и непомеченных данных для обучения. В Python для реализации обучения с частичным привлечением учителя можно использовать библиотеку scikit-learn.

Например, есть набор данных о пациентах, включающий медицинские показатели и информацию о заболеваниях. Некото-

рые пациенты имеют явно указанные диагнозы, а другие пациенты – нет. Задача состоит в том, чтобы построить модель, которая может классифицировать пациентов на основе их медицинских показателей и предсказывать диагнозы. Можно использовать обучение с частичным привлечением учителя (рис. 138), где помеченные данные используются для обучения модели, а непомеченные данные помогают выявить скрытые зависимости между показателями и диагнозами.

```
from sklearn.semi_supervised import LabelPropagation
from sklearn.metrics import accuracy_score
import numpy as np

# Загрузка данных
X_labeled = np.array([[показатели и информация о пациентах с явно указанными
    диагнозами]])
y_labeled = np.array([диагнозы для помеченных пациентов])

X_unlabeled = np.array([[показатели и информация о пациентах без указанных
    диагнозов]])

# Создание модели и обучение на помеченных данных
model = LabelPropagation()
model.fit(X_labeled, y_labeled)

# Предсказание диагнозов для непомеченных данных
predicted_labels = model.predict(X_unlabeled)

# Оценка точности модели (если у вас есть фактические диагнозы для непомеченных
    данных)
# y_true = [фактические диагнозы для непомеченных пациентов]
# accuracy = accuracy_score(y_true, predicted_labels)

# Вывод предсказанных диагнозов
print(predicted_labels)
```

Рис. 138. С частичным привлечением учителя

Обучение с подкреплением (Reinforcement Learning). Здесь модель обучается на основе взаимодействия с окружающей средой. Модель принимает решения и выполняет действия, а затем получает положительные или отрицательные награды за свои действия. Цель модели состоит в том, чтобы максимизировать суммарную награду, используя обратную связь от окружающей среды. В Python

для реализации обучения с подкреплением можно использовать библиотеку TensorFlow или фреймворк OpenAI Gym.

Например, рассмотрим задачу обучения агента компьютерной игре. Агент начинает с некоторого состояния в игре и принимает действия, чтобы максимизировать свою награду, которая может быть связана с достижением определенного уровня, набором очков или победой в игре. Агент использует обратную связь от окружающей среды, чтобы корректировать свои действия и улучшать свою стратегию во времени (рис. 139).

```
import gym

# Создание среды игры
env = gym.make('название_игры')

# Параметры обучения
num_episodes = 1000
max_steps_per_episode = 100

# Цикл обучения
for episode in range(num_episodes):
    # Сброс среды и получение начального состояния
    state = env.reset()

    # Цикл внутри эпизода
    for step in range(max_steps_per_episode):
        # Визуализация среды (если необходимо)
        env.render()

        # Выбор действия агента
        action = выбор_действия(state)

        # Применение действия в среде и получение нового состояния, награды и
            флага завершения
        new_state, reward, done, _ = env.step(action)

        # Обновление стратегии агента на основе обратной связи от среды (например,
            с помощью обучения с подкреплением)
        обновление_стратегии(state, action, reward, new_state)

        # Обновление текущего состояния
        state = new_state

        # Проверка флага завершения эпизода
        if done:
            break

# Завершение игры и закрытие среды
env.close()
```

Рис. 139. С подкреплением

Классификация на примере алгоритма k-ближайших соседей (kNN)

Алгоритм k-ближайших соседей (kNN) является простым и популярным методом классификации. Он основывается на идее, что объекты, близкие в пространстве признаков, склонны иметь одинаковый класс.

Алгоритм k-ближайших соседей основывается на принципе «похожие находят похожих». Он относит новый объект к классу, который является наиболее распространенным среди k в пространстве признаков.

Для классификации алгоритм kNN выполняет ряд шагов.

• Загрузка данных: сначала необходимо загрузить данные, на которых будет выполняться классификация. Набор данных должен содержать примеры объектов и соответствующие им метки классов.

• Подготовка обучающей и тестовой выборок: для оценки производительности модели необходимо разделить данные на обучающую и тестовую выборки. Обучающая выборка используется для обучения модели, а тестовая выборка используется для оценки ее точности.

• Определение числа соседей k: параметр k указывает, сколько ближайших соседей будет использовано для классификации нового объекта. Значение k должно быть положительным целым числом, обычно выбирается экспериментально или с помощью кросс-валидации.

• Вычисление расстояний: для определения ближайших соседей необходимо вычислить расстояние между новым объектом и каждым объектом обучающей выборки. Расстояние может быть измерено различными метриками, такими как евклидово расстояние, манхэттенское расстояние и другие.

• Определение k-ближайших соседей: выбираются k объектов из обучающей выборки, наиболее близких к новому объекту на основе вычисленных расстояний.

• Прогнозирование класса: определяется класс, который является наиболее распространенным среди k-ближайших соседей. Этот класс становится предсказанным классом для нового объекта.

• Оценка точности: сравнивается предсказанный класс с фактическим классом из тестовой выборки для оценки точности модели. Это может быть выполнено с помощью различных метрик, таких как точность (accuracy), матрица ошибок (confusion matrix) и другие.

На рис. 140 представлен пример кода на Python, демонстрирующий классификацию с использованием алгоритма k-ближайших соседей.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Загрузка набора данных Iris
iris = load_iris()
X = iris.data  # Матрица признаков
y = iris.target  # Вектор классов

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Создание модели k-ближайших соседей
k = 3  # Число соседей
model = KNeighborsClassifier(n_neighbors=k)

# Обучение модели на обучающей выборке
model.fit(X_train, y_train)

# Классификация тестовых данных
y_pred = model.predict(X_test)

# Оценка точности модели
accuracy = accuracy_score(y_test, y_pred)
```

Рис. 140. Классификация с использованием алгоритма k-ближайших соседей

В этом примере нами используется набор данных Iris из библиотеки scikit-learn. Сначала загружаем данные и разделяем их на обучающую и тестовую выборки с помощью функции train_test_split(). Затем создаем модель KNeighborsClassifier с числом соседей k = 3. Далее обучаем модель на обучающей выборке с помощью метода fit(), а затем классифицируем тестовые данные с помощью метода predict(). Наконец, оцениваем точность модели с помощью функ-

ции accuracy_score(), сравнивая предсказанные классы y_pred с фактическими классами y_test.

Рассмотрим основные метрики оценки классификации.

✓ Точность (Precision): отношение числа верно классифицированных положительных примеров к общему числу примеров, которые были предсказаны как положительные.

✓ Полнота (Recall): отношение числа верно классифицированных положительных примеров к общему числу истинно положительных примеров.

✓ F1-мера (F1-score): сбалансированная метрика, которая объединяет точность и полноту. F1-мера является гармоническим средним между точностью и полнотой.

✓ ROC-кривая (Receiver Operating Characteristic curve): график, который отображает зависимость между частотой истинных положительных классификаций и частотой ложных положительных классификаций при изменении порога классификации.

✓ AUC-ROC (Area Under the ROC Curve): метрика, которая вычисляет площадь под ROC-кривой и представляет собой меру качества классификатора. Значение AUC-ROC находится в диапазоне от 0 до 1, где 1 означает идеальный классификатор.

Для осуществления классификации применяются валидационная и тестовая выборки:

– валидационная выборка используется для настройки параметров модели и выбора наилучших моделей. Она отделяется от обучающей выборки и используется для оценки производительности модели на разных комбинациях параметров;

– тестовая выборка используется для финальной оценки производительности модели после того, как параметры были настроены с использованием валидационной выборки. Тестовая выборка должна быть независимой от обучающей и валидационной выборок и представлять реальные данные, на которых модель будет использоваться.

Для оценки производительности модели применяется кросс-валидация, которая позволяет использовать все доступные данные для обучения и оценки модели. В процессе кросс-валидации данные разбиваются на несколько фолдов (подвыборок). Затем модель

обучается и оценивается несколько раз, каждый раз используя разные комбинации фолдов для обучения и оценки.

Кросс-валидация позволяет получить более надежные оценки производительности модели и снизить возможность переобучения.

Кластеризация. Метрики оценки кластеризации

Кластеризация – задача машинного обучения, которая относит объекты к группам (кластерам) на основе их сходства или близости друг к другу.

Рассмотрим некоторые популярные алгоритмы кластеризации и метрики оценки качества кластеризации.

K-средних (k-means) – один из наиболее распространенных алгоритмов кластеризации. Он разбивает набор данных на заранее заданное количество кластеров (k). Алгоритм начинается с инициализации случайных центров кластеров, а затем выполняет итеративные шаги перераспределения объектов между кластерами и обновления центров кластеров до сходимости.

На рис. 141 приведен пример использования алгоритма K-средних (k-means) с помощью библиотеки scikit-learn.

```
from sklearn.cluster import KMeans
import numpy as np

# Генерируем случайные данные
X = np.random.rand(100, 2)

# Создаем экземпляр модели KMeans с количеством кластеров равным 3
kmeans = KMeans(n_clusters=3)

# Выполняем кластеризацию
kmeans.fit(X)

# Получаем метки кластеров для каждого объекта
labels = kmeans.labels_

# Получаем координаты центров кластеров
centers = kmeans.cluster_centers_

# Выводим результаты
print("Метки кластеров:", labels)
print("Центры кластеров:", centers)
```

Рис. 141. Пример использования алгоритма K-средних

K-средних++ (k-means++) является улучшенной версией алгоритма K-средних. Он предлагает улучшенный метод инициализации начальных центров кластеров, что помогает избежать проблемы попадания в локальные оптимумы.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – это алгоритм, который основывается на плотности объектов в пространстве. Он определяет кластеры как плотные регионы в пространстве объектов, разделенные областями более низкой плотности. DBSCAN способен обнаруживать кластеры произвольной формы и обрабатывать выбросы (шум) в данных.

На рис. 142 приведен пример использования алгоритма DBSCAN с помощью библиотеки scikit-learn.

```
from sklearn.cluster import DBSCAN
import numpy as np

# Генерируем случайные данные
X = np.random.rand(100, 2)

# Создаем экземпляр модели DBSCAN с параметрами epsilon и минимальным числом соседей
dbscan = DBSCAN(eps=0.3, min_samples=5)

# Выполняем кластеризацию
dbscan.fit(X)

# Получаем метки кластеров для каждого объекта (-1 означает выбросы)
labels = dbscan.labels_

# Выводим результаты
print("Метки кластеров:", labels)
```

Рис. 142. Пример использования алгоритма DBSCAN

После проведения кластеризации важно оценить качество полученных кластеров. К наиболее распространенным метрикам оценки кластеризации относятся:

– Adjusted Rand Index (ARI): измеряет сходство между фактическими метками и метками, присвоенными алгоритмом кластеризации. Значение ARI может находиться в диапазоне от –1 до 1, где 1 указывает на идеальное совпадение меток, а отрицательное значение указывает на случайное совпадение;

– Silhouette Coefficient: рассчитывает качество кластеризации, учитывая внутрикластерное сходство и разделение между кластерами. Значение Silhouette Coefficient находится в диапазоне от –1 до 1, где значения ближе к 1 указывают на хорошую кластеризацию, а значения ближе к –1 указывают на неправильную классификацию;

– Calinski-Harabasz Index: использует межкластерное разделение и внутрикластерную дисперсию для оценки качества кластеризации. Высокое значение индекса Calinski-Harabasz указывает на хорошую кластеризацию;

– Dunn Index: измеряет отношение межкластерного разделения к внутрикластерной дисперсии. Чем выше значение индекса Dunn, тем лучше кластеризация.

Тема 9. Системы глубокого обучения

Нейронные сети

Нейронные сети являются мощным инструментом машинного обучения, вдохновленным работой нейронов в человеческом мозге. Они состоят из искусственных нейронов, которые объединены в слои и позволяют моделировать сложные функции.

Определение функции ошибки (или функции потерь) является важной частью обучения нейронных сетей. Функция ошибки измеряет расхождение между предсказанными значениями модели и фактическими значениями (целевыми метками) и используется для настройки параметров модели в процессе обучения.

Рассмотрим наиболее распространенные функции ошибки, которые часто используются при обучении нейронных сетей.

✓ Среднеквадратичная ошибка (Mean Squared Error, MSE): данная функция ошибки вычисляет среднеквадратичную разницу между предсказанными значениями и целевыми метками. Она часто используется в задачах регрессии.

✓ Перекрестная энтропия (Cross-Entropy): эта функция ошибки широко применяется в задачах классификации. Она вычисляет сумму логарифмических разниц между предсказанными вероятностями и целевыми метками.

✓ Категориальная перекрестная энтропия (Categorical Cross-Entropy): эта функция ошибки также используется в задачах классификации, когда у нас есть несколько классов. Она обобщает перекрестную энтропию на случай многоклассовой классификации.

✓ Логистическая потеря (Log Loss): эта функция ошибки вычисляет сумму логарифмических разниц между предсказанными вероятностями и целевыми метками. Она также используется в задачах бинарной классификации.

Конкретный выбор функции ошибки зависит от типа задачи и особенностей данных. Важно выбрать подходящую функцию ошибки, которая будет подходить для конкретной задачи и поможет модели достичь желаемых результатов при обучении.

Одним из ключевых аспектов нейронных сетей является их способность к обучению на основе данных. **Обучение нейронной сети** заключается в настройке весов и смещений нейронов, чтобы минимизировать ошибку между прогнозируемыми значениями модели и фактическими значениями целевой переменной.

Для оптимизации весов нейронов используется *метод обратного распространения градиента*. Этот метод основан на градиентном спуске, который позволяет найти локальный минимум функции ошибки. Обратное распространение градиента работает путем вычисления градиента функции ошибки по каждому весу в сети и последующего обновления весов в направлении, противоположном градиенту.

Процесс обучения нейронной сети обычно выполняется на основе батчей и эпох.

Батч – подмножество обучающих примеров, которое используется для вычисления градиента и обновления весов. Выбор размера батча зависит от доступных ресурсов и размера обучающего набора данных.

Эпоха – один проход по всем обучающим примерам в обучающем наборе. В процессе обучения нейронной сети обычно выполняется несколько эпох, чтобы модель могла улучшить свои прогнозы и снизить ошибку.

Процесс обучения нейронной сети с использованием обратного распространения градиента идет через следующие шаги.

1. Инициализация весов и смещений: веса и смещения нейронов инициализируются случайными значениями или другими методами инициализации.

2. Прямое распространение (forward propagation): входные данные передаются через сеть, и каждый нейрон вычисляет свой выход на основе взвешенной суммы входов и активационной функции.

3. Вычисление функции ошибки: сравниваются прогнозируемые значения сети с фактическими значениями целевой переменной и вычисляется функция ошибки.

4. Обратное распространение (backward propagation): градиент функции ошибки вычисляется по каждому весу в сети путем применения правила цепочки. Градиент показывает, как изменение веса влияет на ошибку.

5. Обновление весов: веса нейронов обновляются в направлении, противоположном градиенту функции ошибки, с использованием градиентного спуска или других оптимизационных алгоритмов.

6. Повторение шагов 2–5 на каждом батче: процесс прямого и обратного распространения выполняется несколько раз на каждом батче данных из обучающего набора.

7. Повторение шагов 2–6 на нескольких эпохах: процесс обучения повторяется на нескольких эпохах, чтобы модель могла улучшить свои прогнозы и снизить ошибку.

8. Оценка модели: после завершения обучения модель оценивается на отложенном тестовом наборе данных, чтобы оценить ее производительность.

Процесс обучения нейронной сети с помощью обратного распространения градиента может быть дополнен различными методами оптимизации, регуляризацией, выбором активационных функций и другими техниками, чтобы улучшить производительность модели и предотвратить переобучение.

Ниже представлен программный код примера определения нейронной сети с одним скрытым слоем.

```
import numpy as np

# Инициализация весов
def initialize_parameters(input_size, hidden_size, output_size):
    np.random.seed(0)
    W1 = np.random.randn(hidden_size, input_size) * 0.01
    b1 = np.zeros((hidden_size, 1))
    W2 = np.random.randn(output_size, hidden_size) * 0.01
    b2 = np.zeros((output_size, 1))
    parameters = {“W1”: W1, “b1”: b1, “W2”: W2, “b2”: b2}
    return parameters

# Прямое распространение
def forward_propagation(X, parameters):
    W1 = parameters[“W1”]
    b1 = parameters[“b1”]
    W2 = parameters[“W2”]
    b2 = parameters[“b2”]

    Z1 = np.dot(W1, X) + b1
    A1 = np.tanh(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = {“Z1”: Z1, “A1”: A1, “Z2”: Z2, “A2”: A2}
    return A2, cache

# Обратное распространение
def backward_propagation(X, Y, cache, parameters):
    m = X.shape[1]
    A1 = cache[“A1”]
    A2 = cache[“A2”]
    W2 = parameters[“W2”]

    dZ2 = A2 – Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis=1, keepdims=True) / m
    dZ1 = np.dot(W2.T, dZ2) * (1 – np.power(A1, 2))
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis=1, keepdims=True) / m
    grads = {“dW1”: dW1, “db1”: db1, “dW2”: dW2, “db2”: db2}
    return grads
```

```
    # Обновление весов
    def update_parameters(parameters, grads, learning_rate):
        W1 = parameters["W1"]
        b1 = parameters["b1"]
        W2 = parameters["W2"]
        b2 = parameters["b2"]

        dW1 = grads["dW1"]
        db1 = grads["db1"]
        dW2 = grads["dW2"]
        db2 = grads["db2"]

        W1 -= learning_rate * dW1
        b1 -= learning_rate * db1
        W2 -= learning_rate * dW2
        b2 -= learning_rate * db2

        parameters = {"W1": W1, "b1": b1, "W2": W2, "b2": b2}
        return parameters

    # Обучение модели
    def train_model(X, Y, hidden_size, num_iterations, learning_rate):
        input_size = X.shape[0]
        output_size = Y.shape[0]
        parameters = initialize_parameters(input_size, hidden_size,
output_size)

        for i in range(num_iterations):
            A2, cache = forward_propagation(X, parameters)
            grads = backward_propagation(X, Y, cache, parameters)
            parameters = update_parameters(parameters, grads, learning_
rate)

            if i % 100 == 0:
                cost = compute_cost(A2, Y)
                print(f"Iteration {i}: Cost = {cost}")

    return parameters
```

```
    # Пример использования
    X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])
    Y = np.array([[0, 1, 1, 0]])
    hidden_size = 4
    num_iterations = 1000
    learning_rate = 0.1

    parameters = train_model(X, Y, hidden_size, num_iterations, learning_rate)
```

На рис. 143 показан пример использования библиотеки Keras для обучения нейронной сети.

```python
import numpy as np
from keras.models import Sequential
```python
from keras.layers import Dense

# Создание модели
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

# Компиляция модели
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

# Генерация случайных входных данных и меток
data = np.random.random((1000, 100))
labels = np.random.randint(10, size=(1000, 1))

# Преобразование меток в бинарное кодирование
one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

# Обучение модели
model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

Рис. 143. Пример определения нейронной сети
с одним скрытым слоем

Эти примеры на языке Python показывают обучение нейронной сети с помощью обратного распространения градиента.
```

Работа с изображениями с помощью нейронных сетей

Это одна из важных областей глубокого обучения. Для работы с изображениями часто используются **сверточные нейронные сети** (Convolutional Neural Networks, CNN), которые специально разработаны для анализа и обработки двумерных данных, таких как изображения. Сверточные нейронные сети состоят из нескольких слоев, включая сверточные слои, слои пулинга и полносвязные слои.

Сверточные нейронные сети (CNN) используют операции свертки и пулинга для изучения иерархических признаков изображений.

Операция свертки является одной из ключевых операций в сверточных нейронных сетях. Она применяет фильтры (ядра свертки) к входным данным и вычисляет свертку между фильтром и соответствующей областью изображения. Это позволяет нейронной сети выделять различные признаки, такие как границы, текстуры и формы.

На рис. 144 представлен пример кода на Python, который демонстрирует операцию свертки с использованием библиотеки TensorFlow.

```
import tensorflow as tf

# Входные данные
input_data = tf.constant([[[[1], [2], [3]],
                          [[4], [5], [6]],
                          [[7], [8], [9]]]], dtype=tf.float32)

# Фильтр
filter_data = tf.constant([[[[1]], [[1]]],
                           [[[1]], [[1]]]], dtype=tf.float32)

# Применение операции свертки
convolution = tf.nn.conv2d(input_data, filter_data, strides=[1, 1, 1, 1], padding
    ='VALID')

# Вывод результата
with tf.Session() as sess:
    result = sess.run(convolution)
    print(result)
```

Рис. 144. Применение свертки к входному изображению

Операция пулинга (например, max-pooling) используется для уменьшения размерности и сокращения количества параметров в сверточных нейронных сетях. Она выбирает максимальное значение в заданной области и создает уменьшенное представление признаков. На рис. 145 представлен пример кода на Python.

```
import tensorflow as tf

# Входные данные
input_data = tf.constant([[[[1], [2], [3], [4]],
                          [[5], [6], [7], [8]],
                          [[9], [10], [11], [12]],
                          [[13], [14], [15], [16]]]], dtype=tf.float32)

# Применение операции max-pooling
pooling = tf.nn.max_pool2d(input_data, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
    padding='VALID')

# Вывод результата
with tf.Session() as sess:
    result = sess.run(pooling)
    print(result)
```

Рис. 145. Применение max-pooling к входному изображению

Рассмотрим наиболее популярные архитектуры сверточных нейронных сетей, которые демонстрируют высокую производительность на задачах обработки изображений.

✓ AlexNet: это одна из первых глубоких сверточных нейронных сетей, которая привлекла широкое внимание в сообществе машинного обучения. Архитектура AlexNet состоит из пяти сверточных слоев, которые чередуются со слоями пулинга, а также нескольких полносвязных слоев.

✓ VGG: архитектура VGG (Visual Geometry Group) также имеет выдающуюся производительность в задачах классификации изображений. Она состоит из множества сверточных слоев, включая несколько блоков с несколькими сверточными слоями и слоями пулинга. Архитектура VGG отличается от других моделей своей глубиной, что позволяет ей извлекать более сложные признаки.

✓ Inception (GoogLeNet): архитектура Inception, также известная как GoogLeNet, разработана Google. Она использует модули Inception, которые объединяют свертку разных размеров и слои пулинга для извлечения признаков на разных уровнях детализации. Архитектура Inception является эффективной и успешно использует вычислительные ресурсы.

✓ ResNet: архитектура ResNet (Residual Network) предложила новый подход к глубоким сверточным нейронным сетям с использованием блоков соединения с остаточными связями. Это позво-

ляет обучать глубокие модели без проблемы затухания градиентов и позволяет добавлять больше слоев, улучшая производительность.

В контексте сверточных нейронных сетей предобученные модели, которые были обучены на больших наборах данных, могут быть использованы как источник знаний для новых задач. Путем удаления последних слоев и замены их новыми слоями, специфичными для конкретной задачи, можно достичь хорошей производительности с меньшими вычислительными затратами.

Трансферное обучение (Transfer Learning) – методика, которая позволяет использовать предварительно обученные модели на одной задаче для решения другой задачи. Вместо обучения модели с нуля можно использовать предварительно обученные веса нейронной сети на большом наборе данных и дообучить модель на относительно небольшом наборе данных для новой задачи. Трансферное обучение позволяет извлечь высокоуровневые признаки из предварительно обученной модели, которые могут быть полезны для новой задачи. Это особенно полезно, когда у вас есть ограниченное количество данных для обучения новой модели.

В трансферном обучении используются два подхода:

– Fine-tuning (тонкая настройка): в этом подходе предварительно обученная модель размораживается, и последние слои (или несколько последних слоев) модели переобучаются на новом наборе данных. Это позволяет модели адаптироваться к новым признакам и классам, учитывая уже изученные низкоуровневые признаки;

– Feature Extraction (извлечение признаков): в этом подходе предварительно обученная модель используется как фиксированный «экстрактор признаков». Веса модели замораживаются, и входные данные проходят через модель для извлечения признаков. Затем эти признаки могут быть переданы в новую модель (обычно полносвязные слои) для решения новой задачи.

Трансферное обучение позволяет сократить время обучения, улучшить обобщающую способность модели и достичь лучших результатов на новом наборе данных, особенно когда у вас ограниченное количество данных для обучения.

Обработка текстов

Обработка текстов с использованием нейронных сетей является важной областью обработки естественного языка (Natural Language Processing, NLP). Нейронные сети позволяют моделировать сложные зависимости и структуры в текстовых данных, что делает их мощным инструментом для обработки и анализа текста.

Нейронные сети широко применяются во многих задачах обработки текстов, включая следующие:

– классификация текста: нейронные сети могут использоваться для классификации текстовых документов на разные категории или метки. Например, задача определения тональности текста (положительная, отрицательная, нейтральная) или классификация спама;

– извлечение информации: нейронные сети могут быть применены для извлечения структурированной информации из текстовых данных. Например, извлечение именованных сущностей (имен людей, мест, организаций) или извлечение ключевых фраз из текста;

– машинный перевод: нейронные сети, особенно модели на основе трансформера, показывают отличные результаты в задачах машинного перевода, позволяя переводить текст между различными языками;

– генерация текста: нейронные сети могут быть использованы для генерации текста на основе заданного контекста. Например, генерация описаний изображений или автоматическое создание текстовых статей;

– вопросно-ответные системы: нейронные сети могут быть применены для создания систем вопросно-ответных пар, позволяя отвечать на вопросы пользователей на основе доступной информации;

– сентимент-анализ: нейронные сети могут использоваться для анализа эмоциональной окраски текста и определения тональности выражений (позитивная, негативная, нейтральная).

Нейронные сети широко применяются во многих задачах обработки текстов. Векторные представления для текста играют значительную роль в этих задачах, поскольку позволяют представить слова или фразы в виде числовых векторов. Это упрощает обработку текста нейронными сетями, поскольку они могут работать с числовыми данными и использовать их для обучения моделей

и принятия решений. Векторные представления помогают сетям улавливать семантические и синтаксические свойства текста, а также выявлять сходства и различия между словами и фразами, что способствует более точным и эффективным результатам в задачах обработки естественного языка.

Векторные представления для текста позволяют представить слова или фразы в виде числовых векторов, что упрощает их обработку нейронными сетями. Наиболее популярными методами векторизации текста являются:

– Word2Vec: метод, который позволяет преобразовать слова в плотные векторы фиксированной размерности. Он основан на идее, что слова, используемые в схожих контекстах, имеют похожие значения. Word2Vec включает две основные модели: Skip-gram и Continuous Bag-of-Words (CBOW). Skip-gram предсказывает соседние слова на основе текущего слова, а CBOW предсказывает текущее слово на основе его соседей;

– FastText: метод, разработанный Facebook Research, который расширяет идеи Word2Vec, учитывая также подсловные (subword) информации. Он строит векторное представление для каждого слова путем усреднения векторов его подслов. Это позволяет справляться с неизвестными словами и редкими словами лучше, чем методы, основанные только на словах.

Векторные представления для текста позволяют представить слова или фразы в виде числовых векторов, что упрощает их обработку нейронными сетями. Это особенно полезно при работе с последовательными данными, такими как текст. В этом контексте для обработки текста можно использовать рекуррентные нейронные сети (RNN). RNN представляют собой класс нейронных сетей, специально созданных для работы с последовательными данными, включая текст. Они имеют внутреннюю память, которая позволяет учитывать контекст и зависимости между элементами последовательности. Это делает RNN мощным инструментом для анализа и обработки текста, например, в задачах машинного перевода, генерации текста, распознавания речи и других приложениях обработки естественного языка, где важен контекст.

Рекуррентные нейронные сети (RNN) представляют собой класс нейронных сетей, специально созданных для работы с последовательными данными, такими как текст. RNN имеют внутреннюю память, которая позволяет им учитывать контекст и последовательность входных данных.

LSTM (Long Short-Term Memory) и **GRU** (Gated Recurrent Unit) являются двумя расширениями RNN, которые были разработаны для преодоления проблемы затухания и взрыва градиентов. Они добавляют дополнительные механизмы, которые помогают сети сохранять и забывать информацию внутри себя, что позволяет им лучше улавливать долгосрочные зависимости в тексте.

На рис. 146 показан пример, демонстрирующий базовую обработку текста с использованием библиотеки NLTK (Natural Language Toolkit).

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Загрузка необходимых ресурсов NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Пример текста для обработки
text = "Пример предложения, которое нужно обработать."

# Токенизация (разделение текста на отдельные слова)
tokens = word_tokenize(text)

# Приведение слов к нижнему регистру
tokens = [token.lower() for token in tokens]

# Удаление стоп-слов (например, предлогов и союзов)
stop_words = set(stopwords.words('russian'))
tokens = [token for token in tokens if token not in stop_words]

# Лемматизация (приведение слов к их базовой форме)
lemmatizer = WordNetLemmatizer()
tokens = [lemmatizer.lemmatize(token) for token in tokens]

# Вывод обработанных токенов
print(tokens)
```

Рис. 146. Базовая обработка текста

В примере мы использовали nltk.tokenize.word_tokenize для разделения текста на отдельные слова (токены). Затем привели все слова к нижнему регистру с помощью генератора списка. Далее использовали nltk.corpus.stopwords для загрузки стоп-слов на русском языке и удалили их из списка токенов. Наконец, применили лемматизацию с помощью nltk.stem.WordNetLemmatizer для приведения слов к их базовой форме.

Обработанные токены были сохранены в переменной tokens и выведены на экран. В результате был получен список токенов, прошедших обработку, который может быть использован для дальнейшего анализа или обработки текста.

Контрольные вопросы

1. Что такое искусственный интеллект (ИИ) и какие задачи он решает?
2. Какие основные методы машинного обучения используются в системах искусственного интеллекта?
3. Что такое экспертные системы и как они используются в ИИ?
4. Какие задачи решает система обработки естественного языка (NLP)?
5. Что такое системы компьютерного зрения и какие задачи они могут выполнять?
6. Каково сочетание искусственного интеллекта и робототехники?
7. Какие задачи решает система автоматического планирования и принятия решений?
8. Какие преимущества предоставляют системы искусственного интеллекта?
9. Каковы основные задачи классификации, кластеризации и регрессии в ИИ?
10. Какие библиотеки машинного обучения широко используются для классификации в Python?
11. Что такое задача кластеризации и какие библиотеки могут использоваться для ее выполнения?
12. Какую информацию можно получить из задачи кластеризации данных о покупках в интернет-магазине?

13. Что такое задача регрессии в ИИ и какие данные она предсказывает?
14. Какие характеристики домов могут быть использованы для задачи регрессии в предсказании их цен?
15. Как машинное обучение отличается от искусственного интеллекта?
16. Что такое обучение с учителем в машинном обучении и какие примеры задач относятся к этому типу?
17. Что такое обучение без учителя в машинном обучении и какие примеры задач относятся к этому типу?
18. Что такое обучение с подкреплением в машинном обучении и какие примеры задач относятся к этому типу?
19. Какие преимущества предоставляет машинное обучение?
20. Какие типы данных могут быть обработаны с помощью методов машинного обучения?

Тесты для самоконтроля

(ответы см. в приложении)

1. Что такое искусственный интеллект? (*один вариант ответа*)

а) область компьютерных наук, занимающаяся созданием систем, способных имитировать интеллектуальное поведение человека
б) методика программирования компьютеров
в) система искусственного производства
г) технология создания физических роботов

2. Какие задачи могут решать системы искусственного интеллекта? (*один вариант ответа*)

а) анализ данных, обучение, принятие решений
б) печать документов, сканирование изображений
в) обработка платежей, управление складом
г) чтение электронных писем, отправка сообщений

3. Что представляют собой системы обработки естественного языка (NLP)? (*один вариант ответа*)

а) системы, позволяющие компьютерам взаимодействовать с людьми на естественном языке
б) системы для распознавания образов и классификации объектов

в) системы для решения оптимизационных задач
г) системы для управления автономными транспортными средствами

4. Что такое системы машинного обучения? (*один вариант ответа*)
а) системы, использующие алгоритмы и статистические модели для обучения на основе данных и принятия решений без явного программирования
б) системы для автоматического планирования и принятия решений
в) системы для обработки и анализа больших объемов данных
г) системы для распознавания речи и выполнения задач, связанных с языком

5. Что делают системы компьютерного зрения? (*один вариант ответа*)
а) позволяют компьютерам «видеть» и анализировать изображения и видео
б) позволяют компьютерам обрабатывать и анализировать большие объемы данных
в) позволяют компьютерам взаимодействовать с людьми на естественном языке
г) позволяют компьютерам автоматически принимать оптимальные решения

6. В каких областях могут использоваться системы искусственного интеллекта? (*несколько вариантов ответа*)
а) логистика, автономные транспортные средства, управление ресурсами
б) печать документов, сканирование изображений
в) чтение электронных писем, отправка сообщений
г) обработка платежей, управление складом

7. Что позволяют системы искусственного интеллекта делать с данными? (*несколько вариантов ответа*)
а) обрабатывать и анализировать большие объемы данных
б) распознавать образы и классифицировать объекты

в) принимать решения на основе логики и опыта
г) взаимодействовать с людьми на естественном языке

8. Что такое нейронные сети? (*один вариант ответа*)
а) методы кластеризации данных
б) алгоритмы для распознавания образов и классификации объектов
в) алгоритмы и статистические модели для обучения на основе данных
г) методы работы с текстовыми данными

9. Что такое глубокое обучение? (*несколько вариантов ответа*)
а) подход к машинному обучению, основанный на нейронных сетях с большим количеством слоев
б) методика программирования компьютеров
в) технология создания физических роботов
г) системы для обработки и анализа больших объемов данных

10. Что такое алгоритмы кластеризации? (*один вариант ответа*)
а) алгоритмы, используемые для распознавания образов и классификации объектов
б) алгоритмы и статистические модели для обучения на основе данных
в) алгоритмы, используемые для группировки данных на основе их сходства
г) алгоритмы для распознавания речи и выполнения задач, связанных с языком

11. Что такое обучение с подкреплением? (*один вариант ответа*)
а) подход к машинному обучению, в котором агент осуществляет действия в среде и получает награду или штраф в зависимости от результатов
б) методика программирования компьютеров
в) системы для обработки и анализа больших объемов данных
г) системы, использующие базы знаний для принятия решений в определенной области

12. Что представляют собой методы обработки естественного языка (Natural Language Processing)? (*один вариант ответа*)
а) методы для распознавания образов и классификации объектов
б) методы для обработки и анализа больших объемов данных

в) методы для обработки и анализа текстов на естественных языках
г) методы для управления автономными транспортными средствами

13. Что такое обучение с учителем? (*один вариант ответа*)

а) подход к машинному обучению, в котором модель обучается на основе предоставленных ей примеров входных данных и соответствующих им выходных меток
б) методика программирования компьютеров
в) системы для обработки и анализа текстов на естественных языках
г) системы, использующие базы знаний для принятия решений в определенной области

14. Что такое обучение без учителя? (*один вариант ответа*)

а) подход к машинному обучению, в котором модель самостоятельно находит закономерности и структуру в данных без предоставления выходных меток
б) алгоритмы для распознавания образов и классификации объектов
в) методы для обработки и анализа больших объемов данных
г) системы для обработки языка

15. Что такое переобучение (overfitting) в контексте машинного обучения? (*один вариант ответа*)

а) ситуация, когда модель слишком хорошо запоминает обучающие данные и плохо обобщает новые данные
б) ситуация, когда модель не может достичь высокой точности на обучающих данных
в) ситуация, когда модель не может адекватно анализировать большие объемы данных
г) ситуация, когда модель не может принимать решения на основе логики и опыта

16. Какие методы машинного обучения используются для задач классификации? (*один вариант ответа*)

а) логистическая регрессия, метод опорных векторов (SVM), случайный
б) методы для обработки и анализа текстов на естественных языках
в) методы для распознавания образов и классификации объектов
г) методы для управления автономными транспортными средствами

17. Что такое регрессия в контексте машинного обучения? (*один вариант ответа*)

а) метод для классификации объектов на основе их признаков и выделения классов

б) метод для определения оптимальной стратегии принятия решений в условиях неопределенности

в) метод для предсказания числовых значений на основе зависимостей между переменными

г) метод для обработки и анализа текстов на естественных языках

18. Какие методы машинного обучения используются для задач кластеризации? (*один вариант ответа*)

а) k-средних, иерархическая кластеризация, DBSCAN

б) методы для распознавания образов и классификации объектов

в) методы для обработки и анализа текстов на естественных языках

г) методы для управления автономными транспортными средствами

19. Что такое сверточные нейронные сети (Convolutional Neural Networks)? (*один вариант ответа*)

а) методы для распознавания речи и выполнения задач, связанных с языком

б) методы для обработки и анализа текстов на естественных языках

в) методы для обработки и анализа изображений, использующие сверточные слои для извлечения признаков

г) методы для управления автономными транспортными средствами

20. Какие методы машинного обучения используются для обработки и анализа текстов на естественных языках? (*один вариант ответа*)

а) методы глубокого обучения, рекуррентные нейронные сети, методы обработки последовательностей

б) методы для распознавания образов и классификации объектов

в) методы для определения оптимальной стратегии принятия решений в условиях неопределенности

г) методы для управления автономными транспортными средствами

21. Какие методы машинного обучения используются для обработки речи и выполнения задач, связанных с языком? (*один вариант ответа*)

а) рекуррентные нейронные сети (RNN), сверточные нейронные сети (CNN), трансформеры
б) методы для обработки и анализа текстов на естественных языках
в) методы для распознавания образов и классификации объектов
г) методы для управления автономными транспортными средствами

22. Что такое глубокое обучение (deep learning)? (*один вариант ответа*)

а) подход к машинному обучению, использующий глубокие нейронные сети с большим количеством слоев для извлечения и представления сложных зависимостей в данных
б) методы для обработки и анализа текстов на естественных языках
в) методы для распознавания образов и классификации объектов
г) методы для управления автономными транспортными средствами

23. Что такое алгоритмы кластеризации? (*один вариант ответа*)

а) методы для распознавания речи и выполнения задач, связанных с языком
б) методы для обработки и анализа текстов на естественных языках
в) методы для группировки объектов на основе их схожести в некотором пространстве признаков
г) методы для управления автономными транспортными средствами

24. В чем заключается основная задача искусственного интеллекта (ИИ)? (*один вариант ответа*)

а) производство высококачественных компьютеров
б) имитация интеллектуального поведения человека
в) разработка новых операционных систем
г) создание электронных устройств

25. Какие системы относятся к системам искусственного интеллекта? (*один вариант ответа*)

а)системы обработки естественного языка
б) системы автоматического планирования и принятия решений

в) системы компьютерного зрения
г) все вышеперечисленное

26. Что позволяют делать системы обработки естественного языка (Natural Language Processing, NLP)? (*один вариант ответа*)
а) взаимодействовать с компьютерами на естественном языке
б) распознавать образы и классифицировать объекты
в) анализировать и генерировать текст на естественном языке
г) планировать и принимать оптимальные решения

27. Какие методы используются в системах машинного обучения? (*один вариант ответа*)
а) алгоритмы и статистические модели
б) базы знаний, созданные экспертами
в) алгоритмы для распознавания образов
г) библиотека scikit-learn

28. Что позволяют делать системы компьютерного зрения? (*один вариант ответа*)
а) обрабатывать и анализировать большие объемы данных
б) распознавать образы и классифицировать объекты
в) взаимодействовать с компьютерами на естественном языке
г) принимать оптимальные решения и планировать действия

29. Как называются системы, способные воспринимать окружающую среду, принимать решения и взаимодействовать с ней? (*один вариант ответа*)
а) системы обработки естественного языка
б) робототехника
в) системы автоматического планирования и принятия решений
г) системы машинного обучения

30. Как называются системы, использующие алгоритмы и модели для принятия оптимальных решений и планирования действий? (*один вариант ответа*)
а) экспертные системы
б) системы обработки естественного языка
в) системы компьютерного зрения
г) системы автоматического планирования и принятия решений

Практические задания по темам модуля 3

Тема «Введение в искусственный интеллект и основные методы машинного обучения»

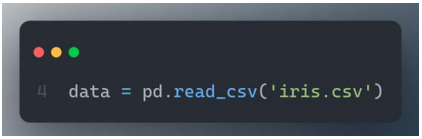
Задание 1. Создайте и проанализируйте набор данных о цветах ириса. Используйте библиотеки Pandas для работы с данными и Matplotlib для визуализации результатов агрегации данных.

Ход работы

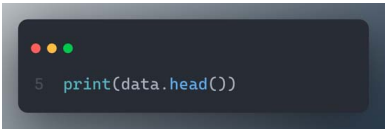
1. Изучите методические рекомендации.

2. В проекте создайте файл iris.csv со следующим содержимым:

3. Загрузите данные из CSV-файла в Pandas DataFrame. Это возможно с помощью метода pd.read_csv().



4. Посмотрите и оцените данные, которые были загружены. Чтобы взглянуть на загруженные данные, выведите первые 5 строк.



В результате вывода получится, что есть 4 числовых столбца с признаками iris (длина и ширина лепестков и чашелистиков) и один столбец class с видом цветка (setosa, versicolor или virginica).

```
Run                                                    7s on 23:36:43, 10/25 ✓

   sepallength  sepalwidth  petallength  petalwidth        class
0          5.1         3.5          1.4         0.2  Iris-setosa
1          4.9         3.0          1.4         0.2  Iris-setosa
2          4.7         3.2          1.3         0.2  Iris-setosa
3          4.6         3.1          1.5         0.2  Iris-setosa
4          5.0         3.6          1.4         0.2  Iris-setosa
```

5. На основе получившихся данных постройте график.

Для этого необходимо задать само построение графика на основе имеющихся данных.

```
# Визуализация агрегированных данных
mean_sepal_length.plot.bar()
plt.ylabel('Средняя длина чашелистика')
plt.show()
```

Данный код строит столбчатую диаграмму на основе рассчитанных ранее агрегированных данных о средней длине чашелистика для разных видов ирисов.

mean_sepal_length.plot.bar() – строит столбчатую диаграмму из данных в переменной mean_sepal_length. Каждый столбец будет соответствовать одному виду iris.

plt.ylabel(‘Средняя длина чашелистика’) – добавляет подпись к оси Y, чтобы было понятно, что значат данные по вертикали.

plt.show() – отображает получившуюся диаграмму.

В результате итоговый код должен выглядеть следующим образом.

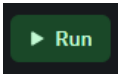
```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('iris.csv')
print(data.head())

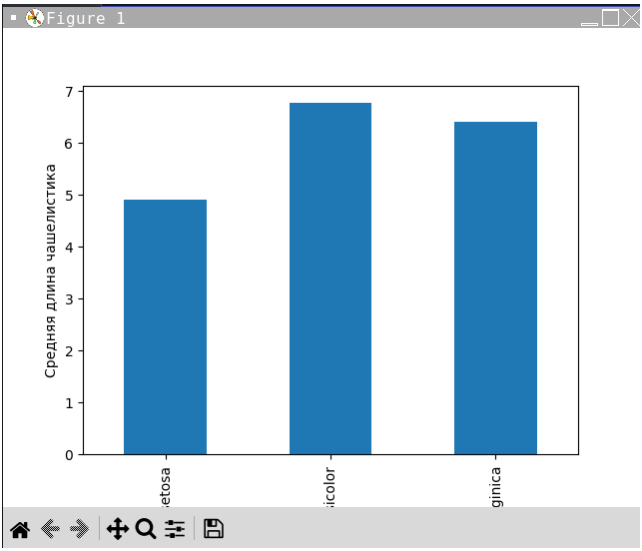
# Агрегация - группировка по виду и расчет средней длины
mean_sepal_length = data.groupby('class')['sepallength'].mean()
print(mean_sepal_length)

# Визуализация агрегированных данных
mean_sepal_length.plot.bar()
plt.ylabel('Средняя длина чашелистика')
plt.show()
```

6. Запустите код с помощью кнопки Run.



В результате выполнения данного кода будет получен следующий график.



На графике отражена столбчатая диаграмма, которая показывает соотношение длины чашелистиков для разных видов ирисов.

Методические указания

Для работы с табличными данными в Python используется библиотека Pandas. Она позволяет загружать данные из разных источников, преобразовывать и анализировать их.

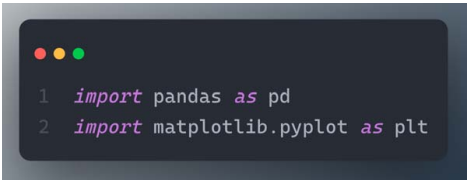
Для визуализации данных может использоваться библиотека matplotlib. Matplotlib – библиотека для визуализации и построения графиков в Python. Основные возможности Matplotlib:

– построение разнообразных типов диаграмм: линейных, столбчатых, круговых, точечных, гистограмм и так далее;
– возможность настройки внешнего вида графиков: цвета, маркеры, стиль линий и так далее;
– отображение графиков в различных форматах: в окне программы, в веб-браузере, сохранение в файл;

– большое количество функций для настройки аннотаций на графике: заголовки, подписи осей, легенда и т. п.;
– возможность построения графиков в 3D;
– широкий набор математических функций для визуализации различных зависимостей;
– интеграция с такими библиотеками как NumPy, Pandas, SciPy и другими.

Основное преимущество Matplotlib в том, что с ее помощью можно быстро визуализировать данные и строить графики publication quality. Это удобный и гибкий инструмент для наглядного представления результатов анализа данных в Python.

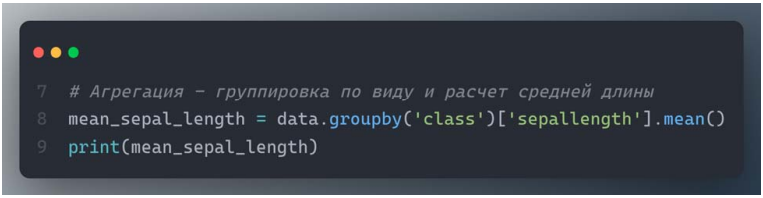
Чтобы работать с данными в Python, нам нужно импортировать соответствующие библиотеки. В данном случае нам понадобятся Pandas для работы с табличными данными и Matplotlib для визуализации.



Процесс суммирования и обобщения данных с целью получения более компактного и информативного представления называется **агрегацией данных** (data aggregation).

Основные виды агрегации данных:
– группировка (grouping) – объединение строк в группы по определенным критериям (например, по странам или годам);
– суммирование (summing) – расчет суммы значений в группе (например, сумма продаж по странам);
– усреднение (averaging) – расчет среднего значения в группе (например, средняя зарплата по отделам);
– подсчет значений (counting) – подсчет количества записей в группе;
– поиск экстремумов (finding minimums/maximums) – нахождение минимального и максимального значения в группе.

Агрегация позволяет сжать большие объемы данных, выявить в них тенденции и закономерности. Это важный этап предварительного анализа данных (EDA).



```
# Агрегация - группировка по виду и расчет средней длины
mean_sepal_length = data.groupby('class')['sepallength'].mean()
print(mean_sepal_length)
```

Производим группировку данных методом groupby по столбцу ‘class’, который содержит виды ирисов (setosa, versicolor и др.). Внутри каждой группы по виду берем столбец ‘sepallength’ – длина чашелистика. Затем вычисляем среднее значение (mean) длины чашелистика для каждой группы.

Таким образом, «сжимаем» детальные данные по каждому цветку, чтобы получить только одно число – среднюю длину чашелистика для каждого вида.

Задание 2. Создайте и сравните модели классификации для определения состояния здоровья пациентов, используя алгоритмы kNN, деревья решений и логистическую регрессию. Для этого решите задачу.

Имеются данные о пациентах с признаками: возраст, пол, наличие хронических заболеваний (да/нет), температура. Необходимо построить модели для классификации состояния пациента на «здоров» и «болен» с помощью алгоритма kNN, дерева решений и логистической регрессии. Сравнить точность полученных моделей на тестовой выборке. Выбрать наиболее точную модель.

Для решения можно использовать любые библиотеки машинного обучения в Python. Данная задача позволит сравнить простоту построения модели, интерпретируемость и точность рассматриваемых алгоритмов классификации на практическом примере.

Ход работы

1. Изучите методические рекомендации.
2. Импортируйте необходимые библиотеки.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

В данной работе самостоятельно создается выборка данных, которая будет проанализирована.

3. Создайте данные в виде DataFrame в Pandas с 4 столбцами:
‘age’ – случайные значения от 18 до 65;
‘sex’: 0 – для женского пола, 1 – для мужского;
‘chronic’: 0 – если нет хронических заболеваний, 1 – если есть;
‘temp’ – случайные значения с нормальным распределением вокруг 36.6 и стандартным отклонением 0.5.

4. Добавьте столбец ‘condition’:
0 – если пациент здоров;
1 – если пациент болен.

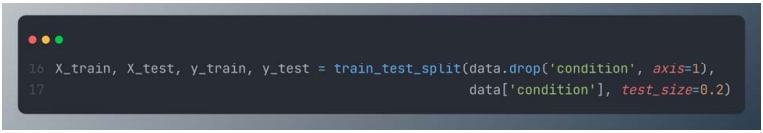
Значения генерируются случайно.

В итоге будут получены данные со 100 строками и 5 столбцами, которые можно использовать для последующего моделирования.

Таким образом была сымитирована выборка данных о пациентах с признаками и классом «здоров/болен». Эти сгенерированные данные будут использоваться далее для сравнения алгоритмов классификации.

```
data = pd.DataFrame({
    'age': np.random.randint(18, 65, 100),
    'sex': np.random.randint(0, 2, 100),
    'chronic': np.random.randint(0, 2, 100),
    'temp': np.random.normal(36.6, 0.5, 100)})
data['condition'] = np.random.randint(0, 2, 100)
```

5. Разделите данные на обучающую и тестовую выборки.



```
X_train, X_test, y_train, y_test = train_test_split(data.drop('condition', axis=1),
                                                    data['condition'], test_size=0.2)
```

– data.drop(‘condition’, axis=1) – изначальный датафрейм data, из которого нами удален столбец ‘condition’. Это будут признаки X для обучения модели;

– data[‘condition’] – целевой столбец y, который мы предсказываем.

Далее с помощью train_test_split разделяем X и y на обучающую и тестовую выборки с соотношением 80 % и 20 % соответственно.

Получится:

– X_train, X_test – признаки для обучающей и тестовой выборок;

– y_train, y_test – целевые значения для обучающей и тестовой выборок.

То есть исходный датасет был разделен на 4 части:

– признаки и цель для обучения;

– признаки и цель для тестирования.

Это позволяет корректно оценить качество модели по отдельным данным, которые не использовались для обучения.

6. Обучите модель kNN.

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print('Точность kNN:', accuracy_score(y_test, knn_pred))
```

7. Обучите дерево решений.

```
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
tree_pred = tree.predict(X_test)
print('Точность дерева решений:', accuracy_score(y_test, tree_pred))
```

8. Обучите логистическую регрессию.

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
logreg_pred = logreg.predict(X_test)
print('Точность лог. регрессии:', accuracy_score(y_test, logreg_pred))
```

Таким образом последовательно обучаются и тестируются 3 модели классификации на сгенерированных данных, сравнивается их точность и выбирается лучшая.

Title

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

data = pd.DataFrame({
    'age': np.random.randint(18, 65, 100),
    'sex': np.random.randint(0, 2, 100),
    'chronic': np.random.randint(0, 2, 100),
    'temp': np.random.normal(36.6, 0.5, 100)})
data['condition'] = np.random.randint(0, 2, 100)

X_train, X_test, y_train, y_test = train_test_split(data.drop('condition', axis=1),
  data['condition'], test_size=0.2)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
print('Точность kNN:', accuracy_score(y_test, knn_pred))

tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
tree_pred = tree.predict(X_test)
print('Точность дерева решений:', accuracy_score(y_test, tree_pred))

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
logreg_pred = logreg.predict(X_test)
print('Точность лог. регрессии:', accuracy_score(y_test, logreg_pred))
```

В результате выполнения кода будет получена точность kNN, точность дерева решений и точность логической регрессии.

```
Run                                    5s on 09:59:01, 10/26 ✓
Точность kNN: 0.5
Точность дерева решений: 0.55
Точность лог. регрессии: 0.5
```

Методические указания

Классификация – это задача машинного обучения, в которой необходимо отнести объект к одному из заранее определенных классов. Алгоритмы классификации используются в различных приложениях – распознавании изображений, детектировании спама, прогнозировании и др. Рассмотрим несколько популярных алгоритмов классификации и сравним их особенности.

k-ближайших соседей (kNN) – простой метод классификации, основанный на сравнении нового объекта с k-ближайшими соседями из обучающей выборки. Гиперпараметр – k. Преимущества – простота, нет необходимости в обучении. Недостатки – высокая вычислительная сложность на больших наборах данных.

Деревья решений – метод, использующий иерархическую структуру для построения модели на основе признаков. Гиперпараметры – глубина дерева, число листьев и др. Плюсы – интерпретируемость, работа с категориальными признаками. Минусы – склонность к переобучению.

Задание 3. Используя методы регрессии, проведите оценку и подбор оптимальных параметров для прогнозирования цен на недвижимость. Обучите и оцените производительность моделей линейной регрессии, регрессионного дерева и случайного леса на предоставленных данных.

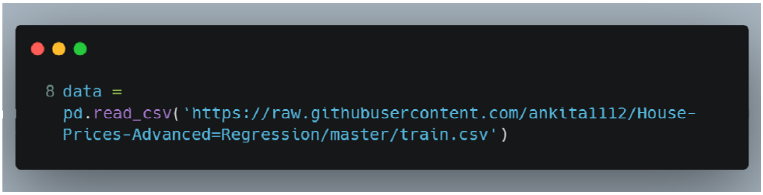
Ход решения

1. Изучите методические указания.

2. Для решения задачи прогнозирования цен на недвижимость сначала импортируйте библиотеки для работы программы.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

3. Загрузите csv-файл, который содержит необходимые данные. Можно использовать данные из CSV-файла с помощью Pandas.



4. Произведите кодирование категориальных признаков с использованием one-hot encoding:

– pd.get_dummies(data). Используется для создания бинарных (дамми) переменных для каждого уникального значения в категориальных признаках. Он автоматически создает столбцы для каждой уникальной категории и присваивает им значения 0 или 1, в зависимости от наличия или отсутствия данной категории в исходных данных;

– drop_first=True – параметр, указывающий на то, что нужно удалить первый столбец бинарных переменных для каждого категориального признака. Это делается, чтобы избежать ловушки фиктивных переменных, когда одну из бинарных переменных можно предсказать по остальным. Например, если у вас был категориальный признак «Color» с возможными значениями «Red,» «Green,» и «Blue», кодирование one-hot encoding создаст три новых столбца: «Color_Red,» «Color_Green,» и «Color_Blue.» Если объект имеет красный цвет, то в столбце «Color_Red» будет стоять 1, а в остальных – 0. В результате этого шага все категориальные признаки заменяются бинарными столбцами, которые теперь можно использовать для обучения моделей машинного обучения. Это важный шаг, так как модели, такие как линейная регрессия, работают только с числовыми данными, а не с текстовыми категориями.

5. Заполните пропущенные значения.

```
10 data = data.fillna(data.median())
```

В этом шаге обрабатываются пропущенные значения в данных, чтобы модели машинного обучения могли правильно работать:

– data.median(). Метод median() используется для вычисления медианы для каждого столбца в наборе данных. Медиана – среднее значение, которое делит данные на две равные половины, и это хорошая метрика центральной тенденции данных. Вычисление медианы позволяет нам найти «среднее» значение для каждого признака, которое будет использоваться для заполнения пропущенных значений;

– data.fillna(data.median()). Этот метод Pandas заменяет все пропущенные значения в исходных данных медианными значениями, которые вычислили на предыдущем шаге. Таким образом, если в данных были пропущенные значения в каких-либо столбцах, они будут заменены на соответствующие медианные значения.

Пропущенные значения могут возникать по разным причинам, и их обработка важна, так как многие модели машинного обучения не могут работать с данными, в которых есть пропуски. Заполнение их медианными значениями – один из способов обработки пропущенных данных, и это позволяет сохранить информацию в наборе данных и подготовить его для обучения моделей.

6. Разделите данные на признаки и целевую переменную.

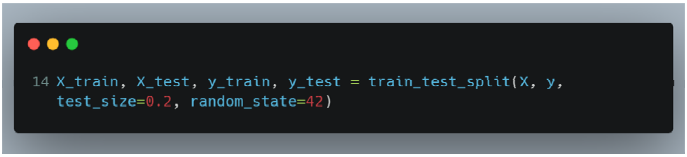
```
11 X = data.drop('SalePrice', axis=1)
12 y = data['SalePrice']
```

X = data.drop(‘SalePrice’, axis=1): создает матрицу признаков X путем удаления столбца ‘SalePrice’ из исходных данных data. По умолчанию метод drop() удаляет указанный столбец, и axis=1 указывает, что мы хотим удалить столбец (в Pandas 1 – это столбцы, 0 – это строки).

y = data[‘SalePrice’] создает вектор целевой переменной y, который состоит из значений столбца ‘SalePrice’ из исходных данных data. В данном контексте ‘SalePrice’ обычно представляет собой целевую переменную, которую мы хотим предсказать с использованием моделей регрессии.

В результате выполнения этого шага получится матрица признаков X, которая содержит все признаки, кроме целевой переменной, и вектор y, который содержит целевую переменную. Они необходимы для правильной подготовки данных перед обучением моделей регрессии, где X будет использоваться для предсказания y.

7. Разделите данные на обучающий и тестовый наборы. Разделение данных на обучающий и тестовый наборы – важный этап машинного обучения, который позволяет оценить производительность моделей на независимых данных. В данном случае данные разделяются на две части: обучающий набор (часть данных, на которой модель будет обучаться) и тестовый набор (часть данных, на которой модель будет тестироваться для оценки ее производительности).



X – матрица признаков, которую определили ранее, и она содержит все признаки из набора данных, кроме целевой переменной.

y – вектор целевой переменной, который также определен ранее и содержит значения, которые хотим предсказать.

test_size=0.2 – параметр, который указывает, какую долю данных мы хотим выделить для тестирования. Здесь 0.2 означает: 20 % данных будет использовано в тестовом наборе и 80 % будет в обучающем наборе.

random_state=42 – seed для генерации случайных чисел. Установив это значение, вы гарантируете, что разделение данных будет одинаковым при каждом запуске кода. Это полезно для воспроизводимости результатов.

После выполнения этой строки кода должны получиться следующие переменные:

X_train – обучающий набор признаков;

X_test – тестовый набор признаков;

y_train – обучающий набор целевой переменной;

y_test – тестовый набор целевой переменной.

8. Используйте X_train и y_train для обучения моделей и X_test для проверки и оценки их производительности.

9. Создайте экземпляры моделей регрессии, которые будут использоваться для обучения и предсказания.

```
15 linear_reg = LinearRegression()
16 decision_tree_reg = DecisionTreeRegressor()
17 random_forest_reg = RandomForestRegressor()
```

linear_reg = LinearRegression(). Здесь мы создаем экземпляр линейной регрессии (LinearRegression) из библиотеки scikit-learn. Линейная регрессия – модель, которая пытается установить линейную зависимость между признаками и целевой переменной.

decision_tree_reg = DecisionTreeRegressor(). Создает экземпляр регрессионного дерева (DecisionTreeRegressor). Регрессионное дерево – модель, которая разделяет данные на несколько подгрупп на основе признаков и прогнозирует целевую переменную на основе среднего значения в каждой подгруппе.

random_forest_reg = RandomForestRegressor(). Создает экземпляр случайного леса (RandomForestRegressor). Случайный лес – ансамблевая модель, которая объединяет несколько регрессионных деревьев для улучшения предсказательной способности.

Эти экземпляры моделей являются объектами, которые содержат методы и параметры для обучения и предсказания.

10. После создания экземпляров обучите их на обучающих данных и используйте для предсказания значений на тестовых данных.

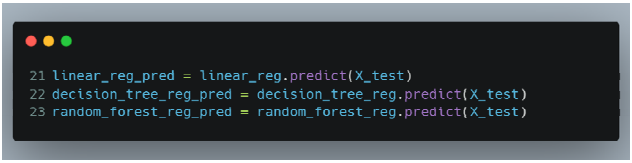
```
18 linear_reg.fit(X_train, y_train)
19 decision_tree_reg.fit(X_train, y_train)
20 random_forest_reg.fit(X_train, y_train)
```

linear_reg.fit(X_train, y_train). Здесь мы обучаем модель линейной регрессии (linear_reg), используя обучающие данные. Метод fit() принимает два аргумента: X_train – матрицу признаков из обучающего набора, и y_train – вектор целевой переменной из обучающего набора. Обучение модели линейной регрессии заключается в нахождении оптимальных весов (коэффициентов) для линейной комбинации признаков, которая наилучшим образом предсказывает целевую переменную.

decision_tree_reg.fit(X_train, y_train). Здесь мы обучаем модель регрессионного дерева (decision_tree_reg) на том же обучающем наборе данных. Регрессионное дерево строит дерево решений, которое разделяет данные на подгруппы и предсказывает целевую переменную в каждой подгруппе.

random_forest_reg.fit(X_train, y_train). В этой строке мы обучаем модель случайного леса (random_forest_reg) на обучающих данных. Случайный лес – ансамбль регрессионных деревьев, где каждое дерево обучается на подмножестве данных и затем их результаты комбинируются для улучшения предсказаний.

11. Примените обученные модели для создания предсказаний на тестовом наборе данных.



```
21 linear_reg_pred = linear_reg.predict(X_test)
22 decision_tree_reg_pred = decision_tree_reg.predict(X_test)
23 random_forest_reg_pred = random_forest_reg.predict(X_test)
```

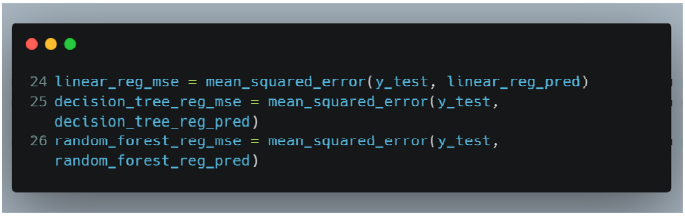
linear_reg_pred = linear_reg.predict(X_test). Используем обученную модель линейной регрессии (linear_reg), чтобы предсказать целевую переменную на тестовом наборе данных X_test. Метод predict() принимает матрицу признаков X_test в качестве аргумента и возвращает предсказанные значения для целевой переменной. linear_reg_pred будет содержать предсказанные значения для линейной регрессии.

decision_tree_reg_pred = decision_tree_reg.predict(X_test). Аналогично в этой строке мы используем обученную модель регрессионного дерева (decision_tree_reg) для предсказания целевой переменной на тестовом наборе данных X_test. decision_tree_reg_pred будет

содержать предсказанные значения для регрессионного дерева.

random_forest_reg_pred = random_forest_reg.predict(X_test). Здесь используем обученную модель случайного леса (random_forest_reg) для предсказания целевой переменной на тестовом наборе данных X_test. random_forest_reg_pred будет содержать предсказанные значения для случайного леса.

12. Оцените производительность каждой модели на тестовом наборе данных с использованием среднеквадратической ошибки (MSE).



```
24 linear_reg_mse = mean_squared_error(y_test, linear_reg_pred)
25 decision_tree_reg_mse = mean_squared_error(y_test,
   decision_tree_reg_pred)
26 random_forest_reg_mse = mean_squared_error(y_test,
   random_forest_reg_pred)
```

linear_reg_mse = mean_squared_error(y_test, linear_reg_pred) вычисляет среднеквадратическую ошибку (MSE) для предсказаний, полученных от модели линейной регрессии. MSE – метрика, которая измеряет среднеквадратичное отклонение между фактическими и предсказанными значениями целевой переменной. Чем меньше значение MSE, тем лучше модель.

decision_tree_reg_mse = mean_squared_error(y_test, decision_tree_reg_pred). Аналогично здесь вычисляем MSE для предсказаний, полученных от модели регрессионного дерева. Это позволяет оценить, насколько хорошо регрессионное дерево справляется с предсказанием целевой переменной.

random_forest_reg_mse = mean_squared_error(y_test, random_forest_reg_pred). Здесь мы вычисляем MSE для предсказаний, полученных от модели случайного леса. Сравнение MSE для разных моделей позволяет определить, какая из моделей лучше предсказывает целевую переменную на тестовых данных. Теперь у нас есть значения MSE для каждой модели, которые можем использовать для сравнения и определения того, какая модель дает наилучшие прогнозы. Модель с наименьшим значением MSE считается наиболее точной и хорошо предсказывающей.

13. Оцените производительность моделей на тестовом наборе данных с использованием коэффициента детерминации (R^2).

```
linear_reg_r2 = r2_score(y_test, linear_reg_pred)
decision_tree_reg_r2 = r2_score(y_test, decision_tree_reg_pred)
random_forest_reg_r2 = r2_score(y_test, random_forest_reg_pred)
```

linear_reg_r2 = r2_score(y_test, linear_reg_pred) вычисляет коэффициент детерминации (R^2) для модели линейной регрессии. R^2 – метрика, которая измеряет, насколько хорошо модель соответствует данным. Значение R^2 находится в диапазоне от 0 до 1, где 1 означает, что модель идеально соответствует данным, а 0 – что модель не объясняет вариацию в данных.

decision_tree_reg_r2 = r2_score(y_test, decision_tree_reg_pred) вычисляет R^2 для модели регрессионного дерева. Позволяет оценить, насколько хорошо регрессионное дерево объясняет вариацию в целевой переменной.

random_forest_reg_r2 = r2_score(y_test, random_forest_reg_pred) вычисляет R^2 для модели случайного леса. Сравнение R^2 для разных моделей позволяет определить, какая из моделей лучше соответствует данным и объясняет вариацию в целевой переменной.

После выполнения этих строк кода получились значения R^2 для каждой модели, которые можно использовать для сравнения и определения того, какая модель лучше объясняет данные. Модель с близким к 1 значением R^2 считается хорошей, так как она хорошо соответствует данным, в то время как модель со значением близким к 0 объясняет данные плохо.

14. Выведите параметры линейной регрессии MSE и R^2.

```
print("Линейная регрессия MSE: ", linear_reg_mse)
print("Решающее дерево регрессии MSE: ", decision_tree_reg_mse)
print("Случайный лес регрессии MSE: ", random_forest_reg_mse)

print("Линейная регрессия R^2: ", linear_reg_r2)
print("Решающее дерево регрессии R^2: ", decision_tree_reg_r2)
print("Случайный лес регрессии R^2: ", random_forest_reg_r2)
```

Результатом выполнения этого кода будут значения MSE и R^2 для линейной регрессии, регрессионного дерева и случайного леса. Эти метрики помогут вам оценить, насколько хорошо каждая модель предсказывает цену продажи домов, и сравнить их между собой.

```
Run                                                    16s on 11:30:42, 10/26 ✓
Линейная регрессия MSE:  2429423922.5307364
Решающее дерево регрессии MSE:  1475962033.5650685
Случайный лес регрессии MSE:  859220245.8132863
Линейная регрессия R^2:  0.683269804059395
Решающее дерево регрессии R^2:  0.8075750634722567
Случайный лес регрессии R^2:  0.887981264081286
```

Полный листинг кода

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Загрузка данных
data = pd.read_csv('https://raw.githubusercontent.com/ankita1112/House-Prices-Advanced-Regression/master/train.csv')

# Произведите кодирование всех категориальных признаков с использованием one-hot encoding
data = pd.get_dummies(data, drop_first=True)

# Заполнение пропущенных значений медианными значениями
data = data.fillna(data.median())

# Разделение данных на признаки (X) и целевую переменную (y)
X = data.drop('SalePrice', axis=1)
y = data['SalePrice']

# Разделение данных на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Создание экземпляров моделей
linear_reg = LinearRegression()
decision_tree_reg = DecisionTreeRegressor()
random_forest_reg = RandomForestRegressor()

# Обучение моделей
linear_reg.fit(X_train, y_train)
decision_tree_reg.fit(X_train, y_train)
random_forest_reg.fit(X_train, y_train)

# Предсказания моделей на тестовом наборе
linear_reg_pred = linear_reg.predict(X_test)
decision_tree_reg_pred = decision_tree_reg.predict(X_test)
random_forest_reg_pred = random_forest_reg.predict(X_test)

# Оценка производительности моделей
linear_reg_mse = mean_squared_error(y_test, linear_reg_pred)
decision_tree_reg_mse = mean_squared_error(y_test, decision_tree_reg_pred)
random_forest_reg_mse = mean_squared_error(y_test, random_forest_reg_pred)

linear_reg_r2 = r2_score(y_test, linear_reg_pred)
decision_tree_reg_r2 = r2_score(y_test, decision_tree_reg_pred)
random_forest_reg_r2 = r2_score(y_test, random_forest_reg_pred)

print("Линейная регрессия MSE: ", linear_reg_mse)
print("Решающее дерево регрессии MSE: ", decision_tree_reg_mse)
print("Случайный лес регрессии MSE: ", random_forest_reg_mse)

print("Линейная регрессия R^2: ", linear_reg_r2)
print("Решающее дерево регрессии R^2: ", decision_tree_reg_r2)
print("Случайный лес регрессии R^2: ", random_forest_reg_r2)
```

Методические указания

Регрессия – статистический метод анализа данных, который используется для моделирования отношений между зависимыми и независимыми переменными. Задачей регрессии является построение модели, которая предсказывает значение зависимой переменной на основе значений одной или нескольких независимых переменных. Эта модель может быть линейной или нелинейной, в зависимости от природы данных.

Линейная регрессия – один из наиболее простых и распространенных методов регрессии. Она предполагает, что связь между зависимой и независимыми переменными линейна. Модель линейной регрессии можно записать как:

$$Y = \beta0 + \beta1X1 + \beta2X2 + ... + \betanXn + \varepsilon,$$

где Y – зависимая переменная; Xi – независимые переменные; β0 – интерсепт (свободный член); βi – коэффициенты регрессии; ε – остатки (случайная ошибка).

Оценка параметров регрессии – процесс определения значений коэффициентов (β0, β1, β2, ...) модели, которые наилучшим образом соответствуют наблюдаемым данным. Оценка может производиться с использованием метода наименьших квадратов (МНК), который минимизирует сумму квадратов разностей между наблюдаемыми и предсказанными значениями.

В случаях когда связь между переменными не является линейной, используются нелинейные модели регрессии. Примерами могут служить экспоненциальные, логистические и полиномиальные модели. Для оценки параметров нелинейной регрессии используются различные методы, включая метод наименьших квадратов и метод максимального правдоподобия.

Подбор оптимальных параметров регрессии включает выбор наилучшей модели и настройку её параметров. Это важный шаг, так как плохой выбор модели или параметров может привести к низкой точности предсказаний. Для этого используются различные методы, включая кросс-валидацию и оптимизацию.

Кросс-валидация – метод оценки производительности модели и выбора оптимальных параметров. Он включает в себя разделение данных на обучающий и тестовый наборы, обучение модели на обу-

чающем наборе и оценку её производительности на тестовом наборе. Повторение этого процесса несколько раз позволяет получить усредненную оценку производительности модели.

Для подбора оптимальных параметров модели часто используется метод оптимизации, такой как метод градиентного спуска. Этот метод позволяет находить значения параметров, которые минимизируют функцию потерь, такую как среднеквадратичная ошибка.

Задания для самостоятельного выполнения

1. Напишите программу, которая генерирует случайные данные и использует метод линейной регрессии для предсказания значения зависимой переменной.

2. Создайте программу, которая обучает нейронную сеть распознавать рукописные цифры MNIST. Используйте библиотеку TensorFlow для реализации.

3. Реализуйте алгоритм классификации k-ближайших соседей (k-nearest neighbors) для набора данных Iris. Протестируйте его точность на тестовом наборе данных.

4. Напишите программу, которая использует метод опорных векторов (Support Vector Machines) для классификации изображений собак и кошек. Используйте набор данных CIFAR-10.

5. Создайте программу, которая применяет алгоритм кластеризации K-средних (k-means) к набору данных с информацией о покемонах. Визуализируйте полученные кластеры.

6. Напишите программу, которая использует линейную регрессию для предсказания стоимости дома на основе его площади.

7. Создайте программу, которая обучает модель машинного обучения для классификации писем на спам и не спам на основе набора данных SpamAssassin.

8.Реализуйте алгоритм K-средних (k-means) для кластеризации набора данных с информацией о покемонах.

9. Напишите программу, которая использует дерево решений (Decision Tree) для классификации пациентов на здоровых и больных на основе набора данных медицинских показателей.

10. Создайте программу, которая использует метод опорных векторов (Support Vector Machines) для классификации изображений собак и кошек на основе набора данных CIFAR-10.

Тема «Системы глубокого обучения»

Задание 1. Классификация изображений. Разработайте сверточную нейронную сеть для классификации изображений из датасета CIFAR-10, используя TensorFlow и Keras в среде Google Colab. Нормализуйте данные, создайте и обучите модель, оцените её точность на тестовых данных и визуализируйте результаты классификации нескольких изображений.

В рамках данного задания предлагается познакомиться с классификацией изображений на примере датасета CIFAR-10. Датасет CIFAR-10 состоит из 60 000 цветных изображений, поделенных на 10 классов. Каждый класс содержит 6000 изображений. Задача заключается в написании программы, которая сможет классифицировать изображения из этого датасета.

Ход работы

1. Изучите методические указания.

2. Перейдите в ресурсе Google Colab (https://colab.research.google.com/) и импортируйте ряд библиотек и инструментов.

```
1 import numpy as np
2 import tensorflow as tf
3 from tensorflow import keras
```

3. Загрузите набор готовых изображений, которые будут классифицированы (например, CIFAR-10).

```
4 # Загрузка датасета
5 (x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

4. Проведите предварительную обработку загруженных данных. Для этого необходимо нормализовать значения пикселей изображений, чтобы они находились в диапазоне от 0 до 1.

5. Создайте архитектуру нейронной сети для решения задачи классификации. В данном случае используется сверточная нейронная сеть (Convolutional Neural Network, CNN), которая является эффективным инструментом для обработки и классификации изображений.

```
model = keras.Sequential([
  keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3), activation='relu'),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Flatten(),
  keras.layers.Dense(64, activation='relu'),
  keras.layers.Dense(10)
])
```

Вариант Keras, называемый Sequential, позволяет строить модель путем последовательного добавления слоев друг за другом (строка 6).

6. Добавьте сверточные слои (строки 7, 9). Сверточные слои – основные строительные блоки сверточных нейронных сетей. Они обрабатывают изображения, выделяя из них характеристики при помощи сверточных операций. В данном случае мы добавляем сверточный слой с 32 фильтрами размером 3×3, функцией активации ReLU и указываем размерность входных данных (32×32 пикселя и 3 канала цветности).

7. Определите пулинговые слои (строки 8, 10). Пулинговые слои предназначены для уменьшения размерности данных и извлечения наиболее важных функций изображения. В данном случае мы добавляем слой пулинга с размером окна 2×2, что позволяет уменьшить размерность изображения в два раза.

8. Определите Flatten-слой (строка 11), который преобразует данные из двумерного массива (например, после сверточных слоев и слоев пулинга) в одномерный массив для передачи на следующие слои. Он выпрямляет данные в список пикселей.

9. Добавьте полносвязные слои (строки 12, 13). Полносвязные слои – обычные слои нейронной сети с полным соединением каждого нейрона с предыдущим слоем. В данном случае будут добавлены два полносвязных слоя. Первый слой имеет 64 нейрона и функцию активации ReLU, а второй слой имеет 10 нейронов, соответствующих количеству классов в задаче классификации изображений.

10. После определения полносвязных слоев произведите компиляцию модели. Компиляция модели – шаг, указывающий дополнительные настройки модели, которые позволяют определить, как будет выполняться обучение модели.

В данном коде модель компилируется с использованием следующих параметров:

– optimizer=’adam’. Оптимизатор ‘adam’ является распространенным методом оптимизации для моделей глубокого обучения. Он адаптивно настраивает скорость обучения на основе градиентов, что позволяет достигать более быстрого и точного обучения модели;

– функция потерь loss (строка 8). Функция потерь SparseCategoricalCrossentropy используется для задач многоклассовой классификации, где метки классов представлены в виде целых чисел вместо one-hot кодирования. Параметр from_logits=True указывает, что модель возвращает необработанные значения (logits) в последнем слое, а не значения, пропущенные через функцию активации softmax;

– метрики (строка 9). Указываем метрику ‘accuracy’ для оценки производительности модели. Метрика ‘accuracy’ определяет, как точно модель классифицирует данные, вычисляя отношение правильных предсказаний к общему количеству предсказаний.

11. Обучите модель на тренировочных данных с использованием заданных оптимизатора, функции потерь и метрики.

Модель обучается на тренировочных данных с использованием следующих параметров:

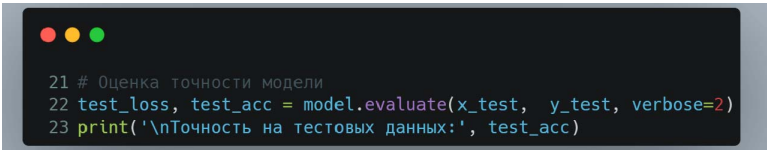
– обучающие данные x_train, y_train. x_train представляют собой массив тренировочных изображений, а y_train – метки классов соответствующих изображений. Эти данные используются для обучения модели;

– количество эпох epochs=10. Параметр epochs определяет, сколько раз модель пройдет через весь тренировочный набор данных. За одну эпоху модель проходит через все изображения и обновляет свои веса соответственно;

– валидационные данные validation_data=(x_test, y_test). x_test представляет собой массив тестовых изображений, а y_test – метки классов соответствующих изображений. Эти данные используются для оценки производительности модели во время обучения.

В результате выполнения этого кода модель будет обучаться на тренировочных данных в течение 10 эпох. Во время обучения модель будет обновлять свои веса, минимизируя функцию потерь, и оценивать свою производительность на валидационных данных.

12. После завершения обучения оцените точность получившейся модели.



```
21 # Оценка точности модели
22 test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
23 print('\nТочность на тестовых данных:', test_acc)
```

Функция evaluate (строка 22) принимает входные данные x_test и их метки y_test в качестве аргументов и возвращает значение потерь и точности модели на тестовых данных. В данном случае test_loss содержит значение потерь (ошибки) модели на тестовых данных, а test_acc содержит точность модели (процент правильных предсказаний).

На строке 23 выводится значение точности модели на тестовых данных с помощью функции print(). Точность показывает, какой процент изображений был правильно классифицирован моделью.

13. После обучения модели протестируйте её на реальном примере. Добавьте в конец файла следующий код.

```
import matplotlib.pyplot as plt

# Загрузка имен классов из датасета CIFAR-10
class_names = ['Самолет', 'Автомобиль', 'Птица', 'Кошка', 'Олень',
               'Собака', 'Лягушка', 'Лошадь', 'Корабль', 'Грузовик']

# Получение предсказаний модели для тестового набора данных
predictions = model.predict(x_test)

# Вывод классификации нескольких изображений
num_rows = 5
num_cols = 5
num_images = num_rows * num_cols

plt.figure(figsize=(10, 10))
for i in range(num_images):
    plt.subplot(num_rows, num_cols, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i], cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions[i])
    true_label = y_test[i][0]
    if predicted_label == true_label:
        color = 'green'
    else:
        color = 'red'
    plt.xlabel("{}".format(class_names[predicted_label]), color=color)

plt.show()
```

14. Для начала создайте список class_names, содержащий имена классов из датасета CIFAR-10. Эти имена соответствуют различным классам изображений, которые наша модель может классифицировать (строки 4, 5).

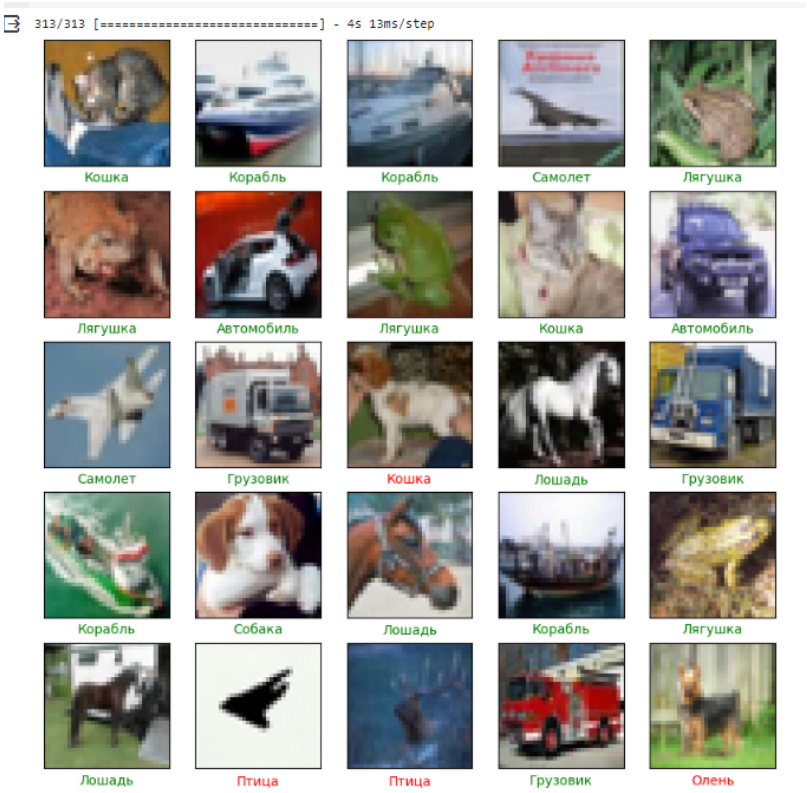
Затем, используя функцию predict, модель получает предсказания вероятностей для каждого класса для каждого изображения из тестового набора данных (строка 8).

15. Визуализируйте классификацию изображений (строки 11–30). Используя библиотеку Matplotlib, выводите классификацию нескольких изображений из тестового набора данных, чтобы увидеть, как модель классифицирует эти изображения. Цвет текста под каждым изображением отображает правильность предсказания: зеленым цветом обозначаются правильные предсказания, а красным – неправильные. Класс, предсказанный моделью, отображается под каждым изображением.

В результате должен получиться следующий код для классификации изображений:

```python
import numpy as np
import tensorflow as tf
from tensorflow import keras

# Загрузка датасета
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Предварительная обработка данных
x_train = x_train / 255.0
x_test = x_test / 255.0

# Определение архитектуры модели
model = keras.Sequential([
  keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Conv2D(64, (3, 3), activation='relu'),
  keras.layers.MaxPooling2D((2, 2)),
  keras.layers.Flatten(),
  keras.layers.Dense(64, activation='relu'),
  keras.layers.Dense(10)
])

# Компиляция модели
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Обучение модели
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Оценка точности модели
test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
print('\nТочность на тестовых данных:', test_acc)

import matplotlib.pyplot as plt

# Загрузка имен классов из датасета CIFAR-10
class_names = ['Самолет', 'Автомобиль', 'Птица', 'Кошка', 'Олень',
               'Собака', 'Лягушка', 'Лошадь', 'Корабль', 'Грузовик']

# Получение предсказаний модели для тестового набора данных
predictions = model.predict(x_test)

# Вывод классификации нескольких изображений
num_rows = 5
num_cols = 5
num_images = num_rows * num_cols

plt.figure(figsize=(10, 10))
for i in range(num_images):
    plt.subplot(num_rows, num_cols, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i], cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions[i])
    true_label = y_test[i][0]
    if predicted_label == true_label:
        color = 'green'
    else:
        color = 'red'
    plt.xlabel("{}".format(class_names[predicted_label]), color=color)

plt.show()
```

16. Запустите написанную сверточную нейронную сеть для классификации изображений, в результате выполнения которой получится набор классифицированных изображений из датасета CIFAR-10, где зеленым цветом отмечен текст для верно классифицированных изображений, а красным – прогноз с погрешностью.



Методические указания

Классификация изображений является важной задачей в области компьютерного зрения. Она позволяет автоматически определять и классифицировать содержимое изображений, что имеет широкое применение в различных областях, включая медицину, робототехнику, автоматическое распознавание лиц и многое другое.

В данном случае мы импортируем 3 библиотеки, которые необходимы для классификации изображений:

– NumPy. Библиотека очень полезна при классификации изображений, поскольку позволяет эффективно работать с массивами и матрицами чисел. В контексте классификации изображений происходит работа с пикселями изображений, которые могут быть представлены в виде массивов чисел. NumPy предоставляет удобные функции и операции для выполнения математических операций с этими массивами, таких как вычисление среднего значения, нормализация или обработка данных, а также инструменты для создания, изменения и переформатирования массивов, что поможет подготовить данные для классификации;

– TensorFlow. Является мощным инструментом для классификации изображений. Обеспечивает создание, тренировку и использование моделей машинного обучения для классификации изображений. С помощью TensorFlow вы можете построить различные типы моделей, включая нейронные сети, которые позволяют точно классифицировать изображения. TensorFlow предоставляет эффективные методы для обучения моделей, включая метод градиентного спуска, который автоматически обновляет параметры модели, чтобы минимизировать ошибку классификации. Это позволяет моделям классифицировать изображения с высокой точностью;

– Keras. Удобный высокоуровневый интерфейс, который работает поверх TensorFlow. Он делает процесс создания и обучения моделей глубокого обучения более простым и интуитивным. Keras предоставляет простые функции для определения архитектуры модели, таких как слои сверточных нейронных сетей и полносвязные слои. Он также предлагает инструменты для компиляции модели с функцией потерь и оптимизатором. Keras позволяет быстро определить и обучить модель классификации изображений без необходимости погружаться в детали реализации.

Нормализация значений пикселей изображений до диапазона от 0 до 1 является важным шагом предварительной обработки данных перед обучением модели классификации изображений.

Можно выделить несколько причин необходимости нормализации.

1. Стабильность обучения. Нормализация позволяет обеспечить стабильность обучения модели. Входные данные, содержащие числа большого масштаба, могут затруднить сходимость алгоритма оптимизации при обучении модели. Нормализация данных путем масштабирования значений пикселей от 0 до 1 помогает обеспечить более стабильное и эффективное обучение модели.

2. Совместимость с функциями активации. Некоторые активационные функции, такие как сигмоида или гиперболический тангенс, работают лучше с входными данными, находящимися в диапазоне от 0 до 1. Нормализация позволяет сохранять значения пикселей в этом диапазоне, что может способствовать улучшению производительности модели.

3. Предотвращение доминирования. Если значения пикселей входных данных не нормализованы, существует риск доминирования некоторых пикселей над другими. Это может привести к неправильному влиянию на обучение модели, где более яркие пиксели могут играть более существенную роль в принятии решения, а менее яркие пиксели могут быть незаметными. Нормализация позволяет уравновесить вклад каждого пикселя в общий контекст изображения.

Задание 2. Подготовьте обучающий набор данных с текстовыми отзывами, разделенными на категории «положительный», «отрицательный», «нейтральный».

Ход работ

1. Изучите методические указания.

2. Импортируйте необходимые библиотеки для работы классификатора:

– Pandas;
– Sklearn;
– Nltk;
– Random.

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,
classification_report
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import random
```

3. Загрузите необходимые модели категоризации отзывов. Здесь загружаются готовые категории отзывов, где stopwords – запрещенные слова, punkt – знаки пунктуации и movie_reviews – готовый набор шаблонных отзывов.

```python
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('movie_reviews')
```

4. Создайте некоторый набор данных из меток-отзывов с распределением на категории:

– positive. Означает, что отзыв о фильме положительный;

– negative – отзыв отрицательный;

– neutral – отзыв нейтральный.

```python
data = {
    "text": [
        "This product is amazing!",
        "I'm extremely disappointed with this service.",
        "The movie was fantastic.",
        "This movie is ok",
        "Terrible customer support experience.",
        "The restaurant exceeded my expectations.",
        "Worst movie i've ever seen",
    ],
    "sentiment": [
        "positive",
        "negative",
        "neutral",
        "positive",
        "negative",
        "positive",
        "negative",
    ]
}
```

5. Создайте DataFrame из набора представленных отзывов и сохраните его в csv-файл.

```
# Создаем DataFrame
df = pd.DataFrame(data)

# Сохраняем в CSV-файл
df.to_csv('dataset.csv', index=False)
```

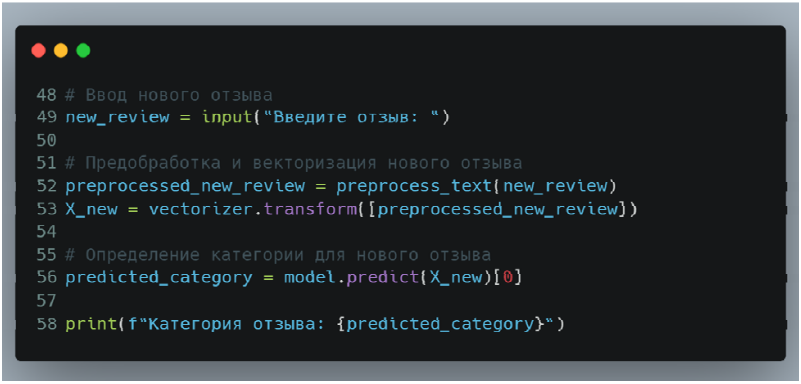
В данном случае DataFrame состоит из двух столбцов:
– «text»;
– «sentiment».

Задание 3. Работа с текстами и их векторными представлениями текстов. Разработайте программу для автоматической классификации текстовых отзывов на английском языке в категории «положительный», «отрицательный» и «нейтральный». Используйте методы векторизации текста, такие как TF-IDF и Word2Vec, и алгоритмы машинного обучения для анализа и классификации текстов. Создайте и сохраните обучающий набор данных в формате CSV, загрузите данные, обучите модель и оцените её эффективность на новых примерах.

1. Загрузите сохраненные отзывы из файла dataset.csv из предыдущего задания и выведите их на экран.

```
# Загрузка отзывов и их меток
reviews = pd.read_csv('dataset.csv')

# Вывод отзывов
for index, review in df.iterrows():
    print(f'Index: {index}')
    print(f'Sentiment: {review["sentiment"]}\n')
    print(review["text"])
    print('-' * 80)
```

2. Теперь, когда у вас есть набор текстовых данных, прочитайте новый отзыв и проведите его векторизацию.



```
48 # Ввод нового отзыва
49 new_review = input("Введите отзыв: ")
50
51 # Предобработка и векторизация нового отзыва
52 preprocessed_new_review = preprocess_text(new_review)
53 X_new = vectorizer.transform([preprocessed_new_review])
54
55 # Определение категории для нового отзыва
56 predicted_category = model.predict(X_new)[0]
57
58 print(f"Категория отзыва: {predicted_category}")
```

Векторизованный отзыв хранится в переменной X_new (строка 53). Перед использованием модели отзыв проходит через процесс предобработки текста, который включает удаление стоп-слов (stopwords), лемматизацию или другие шаги для подготовки текста к векторизации. Затем отзыв векторизуется с использованием переменной vectorizer, которая предварительно инициализирована и обучена на соответствующем наборе данных. Дальше векторизованный отзыв передается в модель для предсказания категории. Модель, которая была предварительно инициализирована и обучена, используется для выполнения предсказания. Она возвращает предсказанную категорию, которая сохраняется в переменную predicted_category (строка 56).

По итогам выполнения работы должен получиться следующий код для работы с текстами и их векторными представлениями:

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,
classification_report
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import random
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('movie_reviews')
# Создаем список с данными
data = {
    "text": [
        "This product is amazing!",
        "I'm extremely disappointed with this service.",
        "The movie was fantastic.",
        "This movie is ok",
        "Terrible customer support experience.",
        "The restaurant exceeded my expectations.",
        "Worst movie i've ever seen",
    ],
    "sentiment": [
        "positive",
        "negative",
        "neutral",
        "positive",
        "negative",
        "positive",
        "negative",
    ]
}
# Создаем DataFrame
df = pd.DataFrame(data)
# Сохраняем в CSV-файл
df.to_csv('dataset.csv', index=False)
# Загрузка отзывов и их меток
reviews = pd.read_csv('dataset.csv')
# Вывод отзывов
for index, review in df.iterrows():
    print(f'Index: {index}')
    print(f'Sentiment: {review["sentiment"]}\n')
    print(review["text"])
    print('-' * 80)

# Ввод нового отзыва
new_review = input("Введите отзыв: ")

# Предобработка и векторизация нового отзыва
preprocessed_new_review = preprocess_text(new_review)
X_new = vectorizer.transform([preprocessed_new_review])

# Определение категории для нового отзыва
predicted_category = model.predict(X_new)[0]

print(f"Категория отзыва: {predicted_category}")
```

В результате выполнения данной программы на выходе будет получена категория введенного отзыва.

```
# Ввод нового отзыва
new_review = input("Введите отзыв: ")

# Предобработка и векторизация нового отзыва
preprocessed_new_review = preprocess_text(new_review)
X_new = vectorizer.transform([preprocessed_new_review])

# Определение категории для нового отзыва
predicted_category = model.predict(X_new)[0]

print(f"Категория отзыва: {predicted_category}")
```

```
Введите отзыв: This movie is brilliant
Категория отзыва: positive
```

Методические указания

В настоящее время текстовые данные являются одним из основных источников информации, и умение эффективно работать с ними является важным навыком в сфере искусственного интеллекта и машинного обучения.

Векторные представления текстов позволяют использовать методы машинного обучения для анализа и обработки текстовой информации.

Одним из методов получения векторных представлений текстов является word2vec, который был описан в 2013 году. Word2vec использует нейронную сеть для изучения ассоциаций между словами на основе большого корпуса текста. После обучения модель может определять синонимичные слова или предлагать дополнительные слова для неполного предложения. Каждое отдельное слово представляется определенным набором чисел, выбранных таким образом, чтобы улавливать семантические и синтаксические особенности слов. Простая математическая функция, такая как косинусное сходство, может указывать на уровень семантической близости между словами, представленными этими векторами.

Другим методом получения векторных представлений текстов является GloVe (Global Vectors for Word Representation). GloVe использует матрицу совстречаемости слов для извлечения семантической информации из текста и создания векторных представлений слов.

Для работы с текстами и их векторными представлениями в модуле используются следующие инструменты и методы:

– токенизация. Позволяет разбивать текст на отдельные слова или токены;
– стемминг и лемматизация. Приведение слов к их основной форме для уменьшения размерности пространства признаков и улучшения качества модели;
– TF-IDF. Метод оценки важности слова в контексте документа и корпуса текстов;
– Bag-of-Words. Модель представления текста, в которой каждый документ представляется в виде мешка слов, игнорируя порядок слов;
– Word2Vec и GloVe. Методы получения векторных представлений слов и текстов;
– методы машинного обучения. Применение алгоритмов машинного обучения, таких как классификация, кластеризация и регрессия, для анализа и обработки текстовой информации.

DataFrame – структура данных, предоставляемая библиотекой pandas. Она представляет собой двумерную таблицу с данными, состоящую из рядов и столбцов.

Каждый столбец содержит последовательность значений определенного типа данных (строки).

Столбец text содержит текстовые отзывы, а столбец sentiment содержит метки, указывающие на эмоциональную окраску отзыва (positive, negative, neutral).

Метод to_csv() сохраняет данные в csv-файл с именем dataset.csv. Ниже представлен пример данных, хранящихся в файле.

dataset.csv ×

1 to 7 of 7 entries Filter

text	sentiment
This product is amazing!	positive
I'm extremely disappointed with this service.	negative
The movie was fantastic.	neutral
This movie is ok	positive
Terrible customer support experience.	negative
The restaurant exceeded my expectations.	positive
Worst movie i've ever seen	negative

Show 10 per page

Векторизация текста позволяет представить тексты в виде числовых векторов, где каждый элемент вектора представляет собой характеристику или признак текста.

Задания для самостоятельного выполнения

1. Напишите программу, которая создает и обучает нейронную сеть с одним скрытым слоем для решения задачи бинарной классификации на примере набора данных XOR.

2. Реализуйте программу, которая использует предобученную модель глубокого обучения (например, VGG или ResNet) для классификации изображений из набора данных ImageNet.

3. Создайте простую рекуррентную нейронную сеть (RNN) для генерации текста на основе набора данных с именами.

4. Напишите программу, которая использует автокодировщик (Autoencoder) для сжатия и восстановления изображений из набора данных MNIST.

5. Реализуйте модель глубокого обучения для определения типа цветка на основе набора данных Iris.

6. Создайте программу, которая использует сверточную нейронную сеть (CNN) для классификации изображений из набора данных FashionMNIST.

7. Напишите алгоритм глубокого обучения для определения настроения текста (например, позитивное или негативное) на основе набора данных отзывов.

8. Реализуйте простую генеративно-состязательную сеть (GAN), которая генерирует новые изображения цифр на основе набора данных MNIST.

9. Создайте программу, которая использует архитектуру глубокой нейронной сети для определения жанра музыкальных композиций на основе набора данных GTZAN.

10. Напишите алгоритм обучения с подкреплением (Reinforcement Learning) для создания агента, который обучается играть в игру «Камень, ножницы, бумага» с использованием алгоритма Q-обучения.

ЗАКЛЮЧЕНИЕ

В результате изучения учебно-методического пособия

✓ *студент*:

– освоит основные понятия языка программирования Python: от простой разработки до веб-разработки;
– рассмотрит принципы объектно ориентированного программирования, которые применяются при создании программ на языке программирования Python и фреймворке Django;
– выделит основные этапы разработки веб-приложения и создаст проект с применением БД;
– рассмотрит сферу применения языка программирования Python: освоит основные понятия систем искусственного интеллекта и машинного обучения;

✓ *преподаватель дисциплины*:

– получит методическую копилку для ведения дисциплины, связанной с разработкой программных продуктов на языке программирования Python и фреймворке Django;
– будет иметь примеры применения языка программирования Python в задачах искусственного интеллекта и машинного обучения;
– получит снижение трудоемкости при подготовке к занятиям;
– получит тестовый материал для проверки знаний студентов;

✓ *специалист*:

– получит методические рекомендации по созданию программных продуктов: от базовых понятий до полной инструкции создания программных проектов;
– систематизирует знания в области разработки программных продуктов с использованием языка программирования Python;
– получит методический набор инструментария, который можно применять в практической деятельности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Гуриков, С. Р. Основы алгоритмизации и программирования на Python : учеб. пособие / С. Р. Гуриков. – Москва : ИНФРА-М, 2023. – 341, [1] с. – (Высшее образование – Бакалавриат). – URL: znanium.com/catalog/product/1913856 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-16-102278-8.
2. Ермаков, С. Р. Основы веб-разработки : учебное пособие / С. Р. Ермаков, П. В. Беляев, А. В. Симонова. – Москва : РТУ МИРЭА, 2024. – 181 с. – ISBN 978-5-7339-2147-1. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/420965 (дата обращения: 31.01.2024). – Режим доступа: для авториз. пользователей.
3. Жуков, Р. А. Язык программирования Python : практикум : учеб. пособие / Р. А. Жуков. – Москва : ИНФРА-М, 2023. – 214, [1] с. – (Высшее образование – Бакалавриат). – URL: znanium.com/catalog/product/1915716 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-16-107207-3.
4. Карякин, М. И. Технологии программирования и компьютерный практикум на языке Python : учеб. пособие / М. И. Карякин, К. А. Ватульян, Р. М. Мнухин ; Южный федеральный университет. – Ростов-на-Дону [и др.] : Издательство Южного федерального университета, 2022. – 240 с. – URL: znanium.com/catalog/product/2057604 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-9275-4108-9.
5. Никитина, Т. П. Программирование. Основы Python для инженеров : учеб. пособие / Т. П. Никитина, Л. В. Королев. – Санкт-Петербург [и др.] : Лань, 2023. – 154 с. – URL: e.lanbook.com/book/302720 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-507-45284-2.
6. Разработка серверной части веб-ресурса : учеб. пособие / В. В. Никулин, А. А. Олейников, А. А. Сорокин, А. В. Олейникова. – Санкт-Петербург [и др.] : Лань, 2023. – 131 с. – URL: e.lanbook.com/book/356102 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-507-47868-2.

7. Рагимханова, Г. С. Программирование на Python : учебное пособие / Г. С. Рагимханова. – Махачкала : ДГПУ, 2022. – 126 с. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/330071 (дата обращения: 14.11.2024). – Режим доступа: для авториз. пользователей.
8. Советов, П. Н. Программирование на языке Питон : учеб. пособие / П. Н. Советов. – Москва : МИРЭА – Российский технологический университет, 2021. – 105 с. – URL: e.lanbook.com/book/226562 (дата обращения: 31.01.2024). – Режим доступа: по подписке.
9. Шевченко, Л. Г. Программирование на PYTHON в среде IDLE : учеб. пособие / Л. Г. Шевченко, Т. В. Дружинина. – Новосибирск : Новосибирский государственный технический университет, 2020. – 192, [2] с. – URL: znanium.com/catalog/product/1866915 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-7782-4215-9.
10. Янцев, В. В. Web-программирование на Python : учеб. пособие / В. В. Янцев. – Изд. 3-е, перераб. – Санкт-Петербург [и др.] : Лань, 2024. – 178 с. – URL: e.lanbook.com/book/310289 (дата обращения: 31.01.2024). – Режим доступа: по подписке. – ISBN 978-5-507-48364-8.

ГЛОССАРИЙ

Django – бесплатная и открытая библиотека для Python, предназначенная для создания веб-приложений.

Django Admin – встроенная веб-административная панель, которая предоставляет интерфейс для управления данными в Django-приложении.

ORM (Object-Relational Mapping) – библиотека, которая позволяет абстрагироваться от физической реализации базы данных и работать с объектами в памяти, а не с таблицами в базе данных.

Web-разработка – процесс создания веб-сайтов и веб-приложений, которые доступны через Интернет.

Авторизация пользователей – процесс проверки подлинности учетных данных пользователя и предоставления ему доступа к функциям приложения.

Атрибут – переменная, которая хранит данные объекта.

База данных в Python – организованная коллекция данных, которую можно хранить, обрабатывать и извлекать в программе на языке Python.

Библиотеки Python – наборы модулей и пакетов, предназначенные для решения определенных задач.

Виртуальное окружение в Python – изолированная среда, в которой можно устанавливать и использовать различные версии пакетов и модулей Python без влияния на другие проекты.

Вложенные функции – функции, которые могут быть определены внутри других функций.

Декораторы в Python – функции, которые принимают другую функцию в качестве аргумента и возвращают новую функцию, изменяющую поведение исходной функции без ее изменения.

Замыкание – функция, которая запоминает значения из окружающей среды даже после того, как эта среда была изменена.

Запуск веб-приложения на Python – процесс создания и запуска приложения, которое может принимать запросы от клиентов через Интернет.

Инкапсуляция – концепция, которая обеспечивает скрытие реализации объекта от пользователей объекта.

Исключения в Python – ошибки, которые возникают во время выполнения программы.

Искусственный интеллект (ИИ) – область компьютерных наук, которая занимается созданием систем, способных имитировать интеллектуальное поведение человека.

Класс – шаблон или описание объекта, который определяет свойства и методы объектов класса.

Классификация в ИИ – задача, в которой модель обучается присваивать новым данным определенные метки классов на основе предварительно известных классов.

Кластеризация в ИИ – задача, в которой модель группирует данные на основе их сходства без предварительно известных классов.

Коллекции (collections) – структуры данных в Python, которые позволяют хранить и обрабатывать данные в виде коллекций.

Кортежи в Python – упорядоченные коллекции объектов, которые могут содержать объекты разных типов.

Машинное обучение является подмножеством ИИ и представляет собой метод, который позволяет компьютерной системе автоматически извлекать знания из данных и использовать их для принятия решений или предсказания результатов на новых данных.

Метод – функция, которая определена в классе и может изменять данные объекта.

Миграции в Django – механизм, который позволяет автоматически создавать и обновлять схему базы данных на основе изменений, внесенных в модели Django.

Множества в Python – неупорядоченные коллекции уникальных элементов.

Множественное наследование – возможность класса наследовать свойства и методы сразу от нескольких родительских классов.

Модели в Django – объектно-реляционное отображение (ORM), которое позволяет определить структуру базы данных приложения.

Модули в Python – файлы, содержащие код, который можно использовать в других программах.

Наследование – механизм, который позволяет создавать новые классы на основе существующих классов, наследуя их свойства и методы.

Нейронные сети – мощный инструмент машинного обучения, вдохновленный работой нейронов в человеческом мозге.

Обучение нейронной сети заключается в настройке весов и смещений нейронов, чтобы минимизировать ошибку между прогнозируемыми значениями модели и фактическими значениями целевой переменной.

Объект – экземпляр класса, который содержит данные и функциональность, определенные в классе.

Объектно ориентированное программирование (ООП) – парадигма программирования, основанная на концепции объектов, которые объединяют данные и функциональность в единую сущность.

Переменные в Python – именованные области памяти, которые используются для хранения значений.

Полиморфизм – возможность объектов разных классов использовать одинаковое имя метода, но с различной реализацией.

Развертывание веб-приложения на Python с использованием Django – процесс подготовки и установки приложения, созданного с помощью Django фреймворка, на веб-сервере, чтобы пользователи могли получить доступ к нему через Интернет.

Регистрация пользователей – процесс создания учетной записи для нового пользователя в приложении.

Регрессия в ИИ – задача, в которой модель предсказывает непрерывные значения на основе предоставленных данных.

Сборка и запуск приложения на Python – процесс разработки и подготовки программного кода на языке Python, который может быть выполняемым на компьютере или другом устройстве.

Системы искусственного интеллекта (ИИ) представляют собой программные или аппаратные системы, способные выполнять задачи, которые требуют интеллектуальных способностей, обычно связанных с человеческим разумом.

Словари в Python – неупорядоченные коллекции объектов, которые хранятся в виде пар «ключ-значение».

Создание веб-страницы в Django – процесс создания интерфейса пользователя, который отображается в веб-браузере.

Списки – упорядоченные коллекции объектов, которые могут содержать объекты разных типов.

Строки в Python – последовательность символов, заключенных в кавычки.

Управление данными в Django приложении – процесс создания, чтения, обновления и удаления данных в базе данных, используя модели Django и ORM (объектно-реляционное отображение).

Установка библиотек в Python – процесс установки пакетов, которые предоставляют дополнительные функции и возможности для языка программирования Python.

Формы в Django – удобный способ создания HTML-форм для взаимодействия с пользователем на веб-сайте.

Функция в Python – блок кода, который может принимать входные аргументы и возвращать результаты.

Шаблоны в Django – файлы, которые содержат HTML-разметку и дополнительные теги шаблонов Django, позволяющие создавать динамические веб-страницы.

Шаблоны веб-страниц в Python – повторно используемые блоки HTML-кода, которые позволяют создавать динамические веб-страницы.

Ответы на тесты

№ вопроса	Вариант ответа	№ вопроса	Вариант ответа
Модуль 1. Введение в программирование на языке Python			
1	а	16	в
2	г	17	б
3	а, б, в	18	г
4	в	19	а, б, в
5	в	20	в
6	б	21	б
7	б	22	в
8	б	23	а
9	в	24	а
10	б	25	б
11	б	26	а
12	г	27	а
13	г	28	б
14	а	29	а, г
15	б	30	а, в
Модуль 2. Основы web-разработки на Python с использованием Django			
1	в	16	в
2	а	17	а, б, в
3	в	18	а
4	г	19	в
5	б	20	а
6	б, в	21	а, в
7	а, б, в	22	а
8	а	23	в
9	а, б	24	б
10	г	25	а
11	а, г	26	б
12	а, б, в	27	а, б
13	а	28	а, в
14	в	29	а
15	а, б	30	а, б

№ вопроса	Вариант ответа	№ вопроса	Вариант ответа
Модуль 3. Искусственный интеллект и машинное обучение. Примеры на Python			
1	а	16	а
2	а	17	в
3	а	18	а
4	а	19	в
5	а	20	а
6	а	21	а
7	а	22	а
8	в	23	в
9	а	24	б
10	в	25	г
11	а	26	в
12	в	27	а
13	а	28	б
14	а	29	б
15	а	30	г

Содержание

Учебное издание

Гущина Оксана Михайловна,
Герасимов Антон Владимирович

СРЕДСТВА ПРОГРАММНОЙ РАЗРАБОТКИ НА ЯЗЫКЕ PYTHON: ВЕБ-ПРИЛОЖЕНИЯ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И МАШИННОЕ ОБУЧЕНИЕ

Учебно-методическое пособие

Редактор *Т.М. Воропанова*
Технический редактор *Н.П. Крюкова*
Компьютерная верстка: *Л.В. Сызганцева*
Дизайн обложки: *И.И. Шишкина*

При оформлении пособия использованы иллюстрации от Freepik и kjpargeter на сайте ru.freepik.com

Подписано в печать 01.10.2025. Формат 60×84/16.
Печать оперативная. Усл. п. л. 21,09.
Тираж 100 экз. Заказ № 1-18-24.

Издательство Тольяттинского государственного университета
445020, г. Тольятти, ул. Белорусская, 14,
тел. 8 (8482) 44-91-47, www.tltsu.ru