Н. А. ЛЕСТЕНКО, И. Д. МАМАЕВ

ТЕХНОЛОГИЯ ОБРАБОТКИ ДАННЫХ ДЛЯ ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЗАДАЧ НА ЯЗЫКЕ PYTHON

Практическое пособие

Санкт-Петербург
Издательство БГТУ «ВОЕНМЕХ» им. Д. Ф. Устинова
2024

УДК 004.434.032.2(076)
Л51

Лестенко, Н. А.

Л51 Технология обработки данных для предметно-ориентированных задач на языке Python : практическое пособие / Н. А. Лестенко, И. Д. Мамаев. – Санкт-Петербург : Изд-во БГТУ «ВОЕНМЕХ» им. Д. Ф. Устинова, 2024. – 62 с.

ISBN 978-5-00221-085-5

Рассмотрены введение в синтаксис языка программирования Python, основные структуры данных, алгоритмы и принципы объектно-ориентированного программирования. Приведены отдельные библиотеки и фреймворки для продвинутых пользователей.

Предназначено для студентов, обучающихся по направлениям подготовки «Программная инженерия», «Информационные системы и технологии», «Теоретическая и прикладная лингвистика», «Интеллектуальные системы в гуманитарной среде».

УДК 004.434.032.2(076)

Р е ц е н з е н т канд. техн. наук, доц. *А.А. Тропченко*
(Университет ИТМО)

Утверждено
редакционно-издательским
советом университета

ISBN 978-5-00221-085-5

ВВЕДЕНИЕ

В современном мире Python имеет два основных значения. Во-первых, Python – язык программирования общего назначения, который отличается своей простотой и универсальностью. Он стал одним из самых популярных языков программирования. Его можно использовать для решения множества задач в различных областях: обработка текстов на естественном языке, разработка веб-приложений и др. В большей степени Python известен благодаря богатой библиотеке по обработке больших массивов данных. В этом контексте Python часто рассматривается не только как язык программирования, но и как комплекс пакетов, созданных различными разработчиками.

Python был разработан в конце 80-х гг. XX в. голландским программистом Гвидо ван Россумом. Название языка было взято из телешоу BBC «Летающий цирк Монти Пайтона», большим поклонником которого являлся программист. Python регулярно обновляется, появляются новые функции, но основная философия осталась неизменной. Этот язык программирования отличается простым синтаксисом и хорошо структурированным кодом. «Красивое лучше, чем уродливое», «Простое лучше, чем сложное» и «Читаемость имеет значение» – говорится в Zen of Python – сборнике основных принципов для написания кода.

Цель настоящего пособия – изучение основ программирования на Python и приобретение основных навыков разработки программ для анализа данных. В пособии также представлены специализированные разделы, которые будут интересны специалистам определенных областей знаний. Так, например, подразд. 2.1 и 2.2 ориентированы на обучающихся по направлениям подготовки «Теоретическая и прикладная лингвистика» и «Интеллектуальные системы в гуманитарной среде», подразд. 2.3 – по направлениям подготовки «Программная инженерия» и «Информационные системы и технологии».

1. ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON

1.1. Общий обзор языка Python. Основы работы в IDLE. Базовый синтаксис и основные команды языка Python

Язык программирования Python – язык высокого уровня, в котором используется динамическая строгая типизация. Термин ***«динами-***

ческая типизация» означает автоматическое подстраивание переменной под тот тип значения, который закладывается в переменную. Все составляющие данного языка являются объектами. Python сформировался под влиянием таких языков программирования, как C/C++, Java, Ada и др.

Одним из преимуществ данного языка является довольно простой синтаксис, что обеспечивает низкий порог входа и обширный функционал, доступный «из коробки», а также огромное количество библиотек для данного языка.

Чтобы любой код выполнился, необходима определенная программа, которую будет принимать код на Python и переводить его в код, понятный компьютеру. Такие программы называются IDE – ***интегрированная среда разработки (Integrated Development Environment),*** которая содержит, помимо множества полезных инструментов, очень важный элемент – ***интерпретатор,*** под которым понимают транслятор, запускающий код построчно. Интерпретатор получает некоторую команду, написанную на языке программирования, считывает ее и сразу же выполняет. Другой известный транслятор – ***компилятор.*** Он получает файл с кодом, расшифровывает его и переводит в машинные коды целиком. На выходе пользователь получает исполняемый файл, который можно запустить внутри операционной системы.

В данном пособии рассматривается стандартная IDE – IDLE Python. На официальном сайте пользователь может скачать актуальную версию Python (рис. 1).

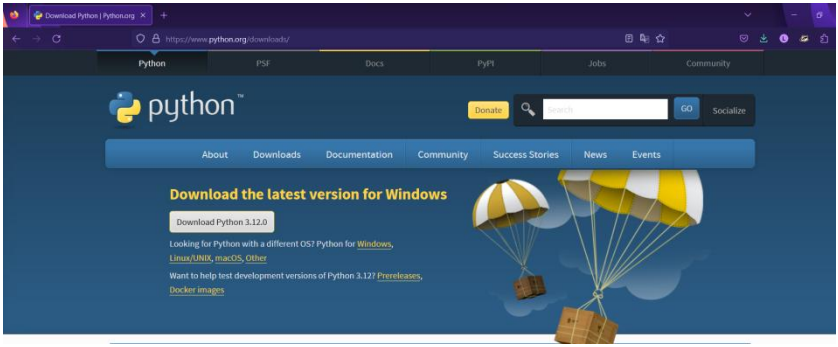

Рис. 1. Главная страница

Далее пользователь выбирает предпочитаемый путь установки с добавлением прав администратора и добавлением исполняемого файла *python* в системную переменную *PATH* (рис. 2).

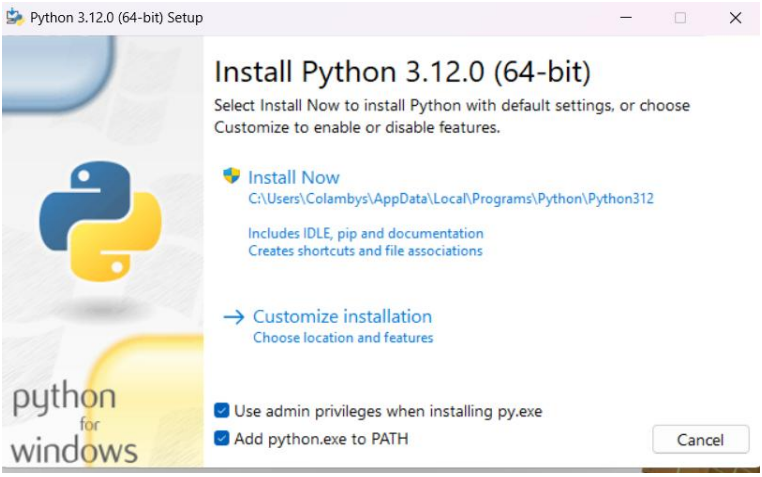

Рис. 2. Установка Python

При открытии приложения появляется *python console* (рабочая среда, консоль), в которой возможно выполнение небольших скриптов (рис. 3).

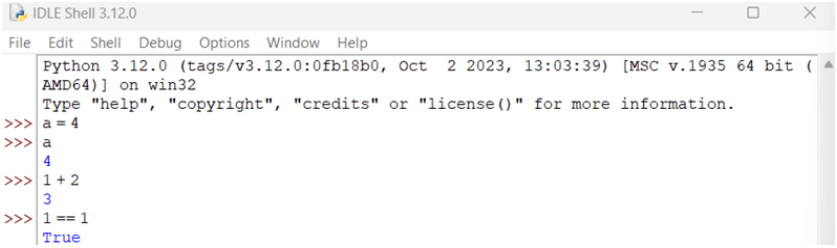

Рис. 3. Python console

Для разработки программ создается файл (рис. 4).

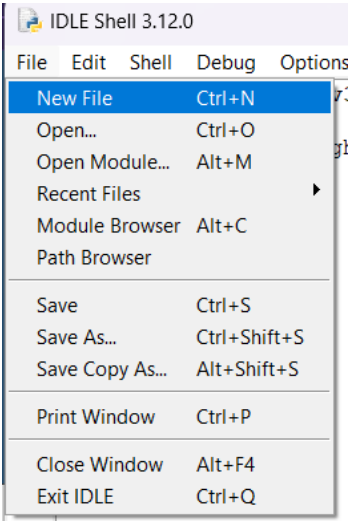

Рис. 4. Создание файла

Далее происходит создание программы и после сохранения – ее запуск. В представленной на рис. 5 программе a = 4 – создание переменной со значением 4, print(a) – функция вывода данного значения. В последующих разделах пособия рассматриваются другие основные команды языка Python.

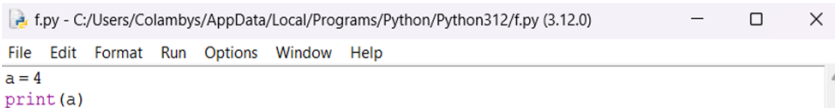

Рис. 5. Главное окно созданного файла

Вопросы к подразд. 1.1

1. Что такое Python?
2. Что такое динамическая типизация?
3. Отличие интерпретатора от компилятора.
4. Что такое IDE?

1.2. Переменные и имена. Минимальная программа на языке Python

Пользователи, начинающие свое знакомство с Python, зачастую применяют его для вычислений некоторых простых количественных значений или работы с постоянными значениями, такими как строки. Результаты этих операций необходимо сохранить в памяти компьютера, именно с этой целью используются ***переменные,*** которые можно определить как именованные области памяти, в которых значения сохраняются для последующего доступа к ним. В Python хорошей практикой считается присвоение переменной имени, которое будет описывать ее содержимое.

В переменных можно хранить практически все, что угодно, для этого необходимо присвоить именованной переменной значение с помощью оператора =. Согласно рекомендациям по оформлению кода PEP8, необходимо до и после знака присваивания поставить по одному пробельному символу

```
month = "May"
```

Переменной month было присвоено строковое значение "May", которое хранится в памяти компьютера. Для того чтобы подобрать удачное имя для переменной, необходимо следовать следующим правилам. Во-первых, имя начинается с буквы или знака подчеркивания, за которыми следуют буквы, цифры или символы подчеркивания (user_name, _user, user1). Во-вторых, имя не может начинаться с цифры. В-третьих, имя не должно входить в стандартный «вокабуляр» языка программирования Python, т. е. оно не должно быть ключевым словом, которое обладает специальным назначением (например, ключевое слово break отвечает за выход из цикла). Со списком основных ключевых слов можно ознакомиться в документации Python на официальном сайте. Наконец, имя переменной чувствительно к регистру, т.е. переменные month и Month не равны между собой. Тем не менее, пользователь может в зависимости целей программы использовать различные стили переменных. Например, для передачи константных величин может использоваться UPPERCASE: PI = 3.14.

Вывести значение на экран можно с помощью функции, представленной ниже

```
print(month)  # May
```

Обратите внимание, что в правой части кода прописан вывод результата. Подобная запись известна как ***комментарий,*** который

обычно поясняет, что делает функция или какой результат будет получен. Для верного оформления комментария на уровне строки кода необходимо как минимум два пробела до символа # и один пробел после него.

Даже при создании такого простого кода необходимо следовать требованиям синтаксиса языка программирования, иначе могут возникнуть ошибки. К ним относятся следующие случаи:

1) наличие опечатки при написании функции;

2) добавление дополнительного отступа перед функцией;

3) изменение регистра функции;

4) опущение открывающих и/или закрывающих скобок для записи аргументов;

5) опущение открывающих и/или закрывающих двойных/одинарных кавычек при записи строковых данных.

Вопросы к подразд. 1.2

1. Что такое переменная?
2. Какие требования предъявляются к наименованию переменных?
3. Что обозначает символ # в Python?
4. Какие ошибки могут возникать при создании кода?

1.3. Встроенные типы данных

Тип данных – набор свойств переменной, определяющих множество допустимых значений некоторого объекта (например, объект типа *целое число* может принимать только целочисленные значения в определенном диапазоне), а также набор операций, допустимых для определенного типа данных (например, строковые объекты могут умножаться на целое число).

Все типы данных в языке Python являются объектами, для которых вызывается особая функция – ***конструктор.***

В Python ***целочисленный тип данных*** представлен встроенным типом int (*integer*). Этот тип данных используется для хранения чисел без десятичной части. Размер целого числа может принимать любые отрицательные и положительные значения, ограничиваясь лишь объемом памяти компьютера:

```
x = 5
y = −10
z = 0
```

Примеры различной записи целочисленных значений приведены на рис. 6. В первом случае выводится целое число, во втором случае – запись в двоичном коде, в третьем – в восьмеричном коде, а с помощью префикса 0x – в шестнадцатеричном коде.

```
>>> 12
12
>>> 0b101
5
>>> 0o123
83
>>> 0xfff
4095
```

Рис. 6. Пример целых чисел

Операции, связанные с целочисленными значениями, включают в себя сложение, вычитание, умножение, деление, целочисленное деление, взятие остатка, а также операции сравнения: больше, меньше, равно и др.

```
a = 10
b = 3
sum_result = a + b  # 13
difference_result = a - b  # 7
product_result = a * b  # 30
division_result = a / b  # 3.333...
integer_division_result = a // b  # 3
remainder_result = a % b  # 1
power_result = a ** b  # 1000
```

В Python основным типом ***вещественного числа*** является тип данных float. Он представляет числа с плавающей точкой двойной точности, диапазон значений которых зависит от компилятора, применявшегося для компиляции интерпретатора Python. Числа типа float записываются с десятичной точкой (рис. 7).

В машинном представлении такие данные хранятся как двоичные числа. Это означает, что одни дробные значения могут быть представлены точно (такие как 0.5), а другие – только приблизительно. Например, сумма 0.1 и 0.2 будет равна не 0.3, а 0.30000000000000004. Для представления используется фиксированное число битов, поэтому

существует ограничение на количество цифр в представлении таких чисел.

```
>>> 0.4
0.4
>>> -.2
-0.2
>>> .4
0.4
>>> 4.
4.0
>>> 0.1+0.2
0.30000000000000004
>>> 0.4+0.2
0.6000000000000001
>>> 0.4+0.4
0.8
```

Рис. 7. Пример вещественных чисел

Строка – неизменяемая последовательность символов Unicode, которая представлена в Python классом str. Параметры объекта выглядят следующим образом:

```
class str(object=b'', encoding='utf-8', errors='strict')
```

Данная последовательность обрамляется кавычками или апострофами, причем важно следить, чтобы на концах были кавычки одного и того же типа. Например, 'Hello' – верный вариант, 'Hello" – уже нет. Также используются строки в тройных кавычках, т.е. строки, которые начинаются и заканчиваются тремя символами, однако данный вариант больше подходит для обозначения документации внутри кода.

Вторым параметром строки является кодировка символов, которая отвечает за правильное отображение символов, в том числе русского алфавита, а также влияет на правила сравнения строк. По умолчанию Python хранит строки в *UTF-8*, но могут использоваться *ASCII, CP1251, KOI-8* и др.

Следующий параметр отвечает за экранирование символов, т.е. замену служебных символов на соответствующую им последовательность, в которые входит перенос или табуляция. Например, последовательность \n – символ перевода строки, \t – символ табуляции, \\ – обратный слеш (\) и др. Основные последовательности приведены в табл. 1.

Таблица 1

Последовательность	Обозначение
\'	Апостроф (остается один ')
\"	Кавычка (остается один символ ")
\r	Возврат каретки
\t	Горизонтальная табуляция
\u ...	16-битный символ Unicode в 16-ричном представлении
\U ...	32-битный символ Unicode в 32-ричном представлении
\x ...	16-ричное значение
\o ...	8-ричное значение
\0	Символ *Null* (не признак конца строки)

Некоторые примеры:

```
# простой текст
"Привет, мир"
# текст с апострофами
'Привет, мир'
# тройные кавычки
"""Привет, мир, но
этот текст будем писать на
нескольких строчках"""

# экранированные последовательности начинаются с символа \
"Привет, мир, \"мир\""
'Привет, мир, \'мир\''
"Привет, мир, 'мир'"
"\tПривет,\n мир"
# через формат "r" можно указать строку в точном выводе всех символов
r"Привет\tи\nмир и еще текст\\"
```

Логический тип данных представлен встроенным типом bool. Логические значения могут быть либо True (истина), либо False (ложь).

```
is_true = True
is_false = False
```

Логические значения часто возникают при сравнении операндов с использованием операторов сравнения. Операторы сравнения возвращают логические значения:

```
x = 5
y = 10
result_greater_than = x > y  # False
result_less_than = x < y  # True
result_equal_to = x == y  # False
result_not_equal_to = x != y  # True
```

Логические значения также возникают в результате логических операций, таких как логическое И (and), логическое ИЛИ (or) и логическое НЕ (not):

```
a = True
b = False
result_and = a and b  # False
result_or = a or b  # True
result_not = not a  # False
```

Для проверки типа данных переменной используется функция type(). Эта функция возвращает объект типа переменной:

```
x = 5
y = 'Hello, World!'
z = 3.14
is_true = True
print(type(x))  # <class 'int'>
print(type(y))  # <class 'str'>
print(type(z))  # <class 'float'>
print(type(is_true))  # <class 'bool'>
```

Наконец, в Python существует специальное значение None типа NoneType, обозначающее нейтральное или «нулевое» поведение. Присвоение такого значения ничем не отличается от других: a = None, обозначает, что идентификатор a задан, но ни с чем не связан. Наиболее часто используется для защитного программирования: «если что-то не None, можно продолжать работу программы».

Вопросы к подразд. 1.3

1. Что такое целочисленный тип данных? Чем он отличается от вещественных?
2. Что такое строковый тип данных?
3. Какие параметры существуют у строкового типа данных?
4. Что такое логический тип данных?
5. Что такое «нечисло»?

1.4. Операции со строками

Строки в Python являются неизменяемым типом данных. Это означает, что операторы, функции и методы не могут ее изменить. Если необходимо сохранить результат изменения существующей строки, нужно создать новую переменную. Так, например, результат ***конкатенации (сложения)*** двух строк с помощью оператора + нужно записать в новую переменную, как показано в примере ниже:

```
syllable1 = 'pine'
syllable2 = 'apple'
word = syllable1 + syllable2
print(word)  # pineapple
```

Обратите внимание, что результат конкатенации позиционно зависим, т.е. при записи word = syllable1 + syllable2 в переменной word будет храниться строка applepine.

В Python не существует операции вычитания строк, которая могла бы быть аналогична операции вычитания чисел, т.е. оператор не применяется к строковым данным. Вместо этого можно использовать метод string.replace(arg1, arg2) для создания новой строки путем замены частей исходной строки. В качестве первого аргумента выступает подстрока исходной строки, которую необходимо заменить, а в качестве второго – заменяющая подстрока. Преобразуем приведенный выше пример, в качестве второго аргумента будет выступать пустая подстрока:

```
word = 'pineapple'
syllable1 = 'pine'
syllable2 = word.replace(syllable1, '')
print(syllable2)  # apple
```

В Python можно умножать строки на целые числа, что приводит к повторению содержимого переменной нужное количество раз:

```
word = 'pine' * 3
print(word)  # pinepinepine
```

Для определения длины строки применяется функция len():

```
word = 'pineapple'
len_word = len(word)
print(len_word)  # 9
```

Для вывода одного символа строки используется ***индексация,*** под которой понимают соответствие каждого символа в строке номеру, начиная с нуля.

Символы строки	p	i	n	e	a	p	p	l	e
Индексы	0	1	2	3	4	5	6	7	8

Чтобы вывести символы по индексу, его нужно указать в квадратных скобках:

```
word = 'pineapple'
print(word[0])        # p
print(word[1])        # i
print(word[8])        # e
print(word[-1])       # e
print(word[9])        # IndexError: string out of range
```

В приведенном примере необходимо обратить внимание на два последних вывода. В Python пользователь может выводить символ строки, проведя индексацию с правого края. Для этого используют отрицательные индексы, при этом подсчет начинается с индекса -1, а не 0, как это делается при индексации с левого края. Последний пример выведет ошибку IndexError, так как пользователь попытался обратиться к элементу строки с несуществующим индексом (см. приложение).

Если пользователю требуется получить часть строки, а не один символ, применяют ***срезы.*** Срез извлекает подстроку, начиная с определенного индекса и заканчивая последним индексом. Символ с указанным конечным индексом не входит в срез. Синтаксис срезов сводится к следующему: string[init_ind:final_ind:step]. Первый параметр

указывает на начальный индекс, второй – на конечный индекс, а третий – на шаг, который по умолчанию равен единице. Использование срезов с шагом позволяет получить элементы с некоторым «прореживанием». Третий параметр является факультативным:

```
word = 'pineapple'
print(word[2:5])      # nea
print(word[4:])       # apple
print(word[:4])       # pine
print(word[2:7:2])    # nap
```

В первом выводе пользователь увидит подстроку без элемента p, так как конечный индекс 5 не включается. Во втором выводе не указан конечный индекс, т.е. вывод подстроки будет осуществляться с указанного элемента до конца исходной строки. Опущение начального индекса в третьем выводе указывает на вывод исходной строки с самого начала вплоть до символа с индексом 3. Последний вывод nap получился следующим образом: на первом этапе извлеклась подстрока neapp, затем в ней остались все элементы через один от начального: ~~n~~e~~a~~p~~p~~ → nap.

Пользователь также может найти индекс подстроки в Python. Для этой цели существуют два метода: string.find(str, [start], [end]) и string.rfind(str, [start], [end]). Первый метод возвращает индекс первого совпавшего значения подстроки, второй – последнего совпавшего значения. Факультативными аргументами являются второй и третий, они указывают на поиск в некотором срезе:

```
word = 'pineapple'
print(word.find('ap'))  # 4
print(word.find('pl'))  # 6
print(word.find('p', 6, 8))  # 6
```

Чтобы разбить строку на части с помощью разделителя, можно использовать метод split(), в качестве аргумента он принимает символ-разделитель или подстроку-разделитель:

```
sentence = 'It was a dark, starless night.'
print(sentence.split(','))
# ['It was a dark', ' starless night.']
```

В приведенном выше примере результатом вывода стал список, при этом последний элемент-подстрока списка начинается с пробела, так как разделитель представлен единственным символом – запятой.

Сам разделитель не включается в выдачу. Особенности списков будут рассмотрены в подразд. 1.8.

Наконец, пользователь может изменять регистр строк. Метод upper() преобразовывает все символы в верхний регистр, lower() – в нижний регистр:

```
sentence = 'I can see a toy.'
print(sentence.upper())  # I CAN SEE A TOY.
print(sentence.lower())  # i can see a toy.
```

Вопросы к подразд. 1.4

1. Что такое конкатенация?
2. Как реализовать вычитание строк?
3. Дайте определение понятию индексации.
4. Что такое срез и шаг?
5. Как вычислить длину строки?
6. Какой метод позволяет разделить исходную строку на список подстрок?

Практическое задание к подразд. 1.4

Создать программу, в которой задается переменная sentence – русскоязычное предложение следующего содержания: «*Документ, опубликованный на сайте Смольного, был размещен жилищным комитетом для общественного обсуждения*». Выполните следующие задания, перед выводом результата нужно напечатать поясняющую строку.

1. Подсчитайте количество символов в предложении.
2. Выведите последний символ двумя разными способами.
3. Выведите девятый символ.
4. Выведите подстроку, начиная с 26-го символа.
5. Выведите строку, применив срез с шагом, который равен трем.
6. Разделите строку по пробелу.
7. Разделите строку по сочетанию «*ен*».
8. Примените конкатенацию, добавив к строке точку, пробел и предложение следующего содержания: «*Самый высокий взнос – 13,63 рубля за квадратный метр – установлен для домов с лифтом двух типов: дореволюционных и построенных после 1980 года*».
9. Для новой строки найдите индекс первой и последней подстроки «*ны*».
10. Переведите новую строку в верхний регистр, разделите ее по предлогу «для».

1.5. Условный оператор

Оператор if-elif-else в Python – это конструкция, которая позволяет некоторой программе принимать решения на основе определенных условий. С его помощью можно выполнить один набор инструкций, если условие истинно, и другой набор, если условие ложно. Синтаксис условного оператора можно описать следующим образом:

```
if условие_1:
   # Этот блок кода выполнится, если условие_1 истинно
   ...
elif условие_2:
   # Этот блок кода выполнится, если условие_1 ложно, но условие_2 истинно
   ...
else:
   # Этот блок кода выполнится, если ни одно из условий не истинно
   ...
```

При создании условия важно следить за синтаксисом. Обратите внимание, что формулировка условия в каждой ветви заканчивается двоеточием, а код, который будет выполняться при активации условия, создается с отступом в четыре пробельных символам, например:

```
age = 18
if age >= 18:
   print("Вы совершеннолетний.")
else:
   print("Вы несовершеннолетний.")
```

В начале вводят переменную age, которая содержит значение 18, затем – условную конструкцию if-else. Обратите внимание, что ветвь elif в данном примере опущена (она ***факультативна***). Так как ответ бинарный, то пользователь использует двухместный оператор, который начинается с ключевого слова if, за которым следует выражение сравнения. Это выражение проверяет, является ли значение переменной age больше или равным 18. Если условие внутри if истинно (если возраст больше или равен 18), то выполняется блок кода, следующий за if. В данном случае этот блок кода содержит вывод текстовой информации о совершеннолетии человека. Если условие внутри if ложное (возраст меньше 18), то выполнится блок кода после else. В дан-

ном случае этот блок кода выведет информацию о несовершеннолетии пользователя.

Если пользователю необходимо расширить количество условий для проверки, то он внедряет ветвь elif столько раз, сколько предусмотрено конкретной ситуацией. Ниже приведен пример кода, который переводит первичные баллы студента за тест в итоговую оценку:

```
points = 85
if points >= 90:
   print("Отлично!")
elif points >= 70 and points <= 89:
   print("Хорошо.")
elif points >= 50 and points <= 69:
   print("Удовлетворительно.")
else:
   print("Плохо.")
```

Ветвь else тоже факультативна. Другими словами, можно использовать следующие комбинации для создания условий: if, if-else, if-elif, if-elif-else.

Вопросы к подразд. 1.5

1. Что такое условный оператор, для чего он используется?
2. Что обозначает ключевое слово else?
3. Какие основные синтаксические особенности условных операторов нужно учитывать?
4. Что происходит, если условие внутри if является истинным?
5. Перечислите особенности ветви elif.

Практические задания к подразд. 1.5

1. Введите целое число, определите, является ли оно четным или нечетным.

2. Напишите программу для вычисления бонуса сотрудника на основе его стажа. Если стаж больше 5 лет, бонус составляет 10%, в противном случае – 5%.

3. Введите свой возраст, определите возрастную категорию (ребенок, подросток, взрослый и т. д.).

4. Напишите программу, которая преобразует температуру из градусов Цельсия в градусы Фаренгейта и наоборот в зависимости от выбора пользователя.

5. Создайте программу для проверки банковского счета. Если баланс меньше нуля, выведите сообщение о долге. При балансе больше нуля выведите сообщение о том, что счет в порядке.

6. Проверьте, что пользователь ввел корректный пароль при регистрации на сайте (например, длина пароля не менее 8 символов и содержит буквы и цифры).

7. Введите слово и определите, является ли оно палиндромом (читается одинаково справа налево и слева направо).

8. Создайте программу, которая определяет, является ли заданный год високосным.

9. Реализуйте программу для вычисления бонусов сотрудников на основе их производительности. Сотрудники с высокой производительностью получают более высокие бонусы.

10. Выберите геометрическую фигуру (круг, квадрат, треугольник) и рассчитайте ее площадь в зависимости от введенных данных.

11. Напишите программу для проверки допустимости веса ручной клади в аэропорту с выводом необходимых сообщений.

12. Создайте программу, которая просит пользователя ввести температуру и условия (солнечно, облачно, дождь) и рекомендует, что надеть.

13. Реализуйте программу для проверки, принадлежит ли пользователь к группе риска для определенного заболевания на основе его возраста и медицинской истории.

14. Введите показатели артериального давления и пульса и определите состояние здоровья человека.

15. Напишите программу, которая просит пользователя ввести возраст и определяет, сколько стоит билет в кино для него.

16. Реализуйте программу, которая предлагает скидку на товар в зависимости от его стоимости (например, 10%-ная скидка на товары дороже 1000 рублей).

17. Разработайте систему анкетирования для абитуриентов. В зависимости от ответов на вопросы, программа определяет, подходит ли кандидат для поступления.

18. Введите номер месяца. Программа должна определить, к какому времени года этот месяц относится (зима, весна, лето, осень).

19. Введите дату рождения (месяц и день), программа должна определить знак зодиака.

20. Введите небольшой текст, затем введите слово. Программа определяет, присутствует ли это слово в тексте, и выводит результат.

1.6. Циклы

Циклы – это важная концепция в программировании, позволяющая выполнять один и тот же блок кода многократно. В Python существует два основных вида циклов: while и for.

Цикл while выполняет блок кода, пока заданное условие истинно (True). Он может быть бесконечным, если условие всегда истинно:

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

Этот цикл будет выполняться до тех пор, пока count меньше 5. Каждую итерацию count увеличивается на 1. Как и в случае с условным оператором, необходимо следить за двоеточием в конце строки, задающей цикл, а также за пробельными отступами при задании блока кода в теле цикла.

Цикл for используется для перебора элементов в последовательности, такой как список, кортеж или строка. Цикл, приведенный ниже, будет перебирать элементы списка fruits и выводить каждый элемент последовательно:

```
fruits = ["яблоко", "банан", "апельсин"]
for fruit in fruits:
    print(fruit)
```

Если пользователю необходимо изменить поведение циклов, то он может воспользоваться несколькими операторами, которые позволяют менять поведение циклов по умолчанию. Оператор break используется для досрочного завершения цикла, например, при выполнении определенного условия. В приведенном ниже примере вводится функция range(10), которая создает объект в виде арифметической последовательности чисел от 0 до 9. Последовательно будут выводится цифры 0, 1, 2, 3 и 4, при переходе к значению 5 условие станет истинным, что приведет к досрочному выходу из цикла, значения 5, 6, 7, 8 и 9 не будут напечатаны:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Пустой оператор pass не выполняет никаких действий. Он часто используется как заполнитель, когда синтаксически необходимо указать действие, но ничего выполнять не требуется. При запуске кода, который представлен ниже, будут выведены все элементы заданной арифметической последовательности:

```
for i in range(5):
    if i == 2:
        pass  # Пропустить выполнение на этой итерации
    print(i)
```

Оператор continue используется для перехода к следующей итерации цикла без завершения всего цикла:

```
for i in range(5):
    if i == 2:
        continue  # Перейти к следующей итерации
    print(i)
```

Из заданной арифметической последовательности не будет выведено только значение 2.

Вопросы к подразд. 1.6

1. Какие циклы существуют в Python? В чем их различие?
2. Назовите основное назначение оператора break.
3. Назовите основное назначение оператора pass.
4. Назовите основное назначение оператора continue.

Практические задания к подразд. 1.6

1. Подсчитать количество гласных букв в тексте, введенном с клавиатуры.

2. Проверить, является ли введенный с клавиатуры текст русскоязычной панграммой (текст, который содержит все буквы алфавита некоторого языка).

3. Разбить введенный с клавиатуры текст по пробелам и вывести каждый элемент списка.

4. Во введенном с клавиатуры тексте заменить все гласные буквы на X.

5. Создать некоторый список из 15 элементов с дублетами. С помощью цикла for удалить дублеты.

6. Вводить числа с клавиатуры и выводить их накопленную сумму, пока пользователь не введет ноль.

7. Вывести все четные числа от 1 до 20 с использованием цикла while.

8. Перевернуть введенное с клавиатуры число в обратном порядке с использованием цикла while.

1.7. Работа с текстовыми файлами

Большие объемы информации обычно хранят не в структурах данных, а в файлах или базах данных. На первых этапах знакомства с Python пользователи начинают работать с текстовыми файлами, которые рассматриваются как набор символов и строк.

Для начала работы с текстовым файлом в Python его нужно открыть с помощью встроенной функции open("example.txt", "r"). Она принимает два основных аргумента: имя файла (или путь к нему) и режим доступа:

```
file = open("example.txt", "r")
```

В данном случае "example.txt" – это имя файла, а "r" – режим доступа, который означает чтение (*read*). Существуют и другие режимы доступа, такие как "w" (запись), "a" (добавление, создаст новый файл для записи, если не найдет с указанным именем), "rb" (чтение в бинарном режиме), "r+" (чтение и запись), "rb+" (чтение и запись в бинарном режиме) и т.д. В качестве факультативного аргумента выступает указание кодировки текстового файла. В таком случае приведенный выше пример будет выглядеть следующим образом:

```
file = open("example.txt", "r", encoding = "utf-8")
```

После открытия файла можно считать его содержимое с помощью различных методов. Наиболее распространенный метод – read(), который считывает весь файл или заданное количество символов:

```
content = file.read()  # Считать весь файл
print(content)
```

Если нужно прочитать файл построчно, можно использовать метод readline() или перебрать файл в цикле for:

```
for line in file:
    print(line)
```

После завершения работы с файлом важно закрыть его с помощью метода close(). Закрытие файла освобождает ресурсы и предот-

вращает возможные проблемы с доступом к файлу другими программами:

```
file.close()
```

В качестве альтернативы можно использовать блок with для автоматического закрытия файла после завершения работы с ним:

```
with open("example.txt", "r") as file:
    content = file.read()
    # Файл автоматически закроется после завершения блока
```

Для записи данных в файл используется режим доступа "w" или "a" в зависимости от того, хочет ли пользователь перезаписать существующий файл или добавить данные в конец файла. Для записи текста в файл используется метод write():

```
with open("output.txt", "w") as file:
    file.write("Это текст, который будет записан в файл.")
```

Иногда пользователю может понадобиться работать с файлами в бинарном режиме, например, для чтения или записи изображений, аудиофайлов и т.д. В этом случае нужен режим доступа "rb" (чтение бинарного файла) или "wb" (запись бинарного файла):

```
with open("image.jpg", "rb") as binary_file:
    data = binary_file.read()
```

Наконец, при работе с файлами важно обрабатывать исключения, которые могут возникнуть, например, если файл не существует или нет прав на доступ к нему. Для этого служит конструкция try...except:

```
try:
    with open("file_that_may_not_exist.txt", "r") as file:
        content = file.read()
except FileNotFoundError:
    print("Файл не найден.")
except PermissionError:
    print("Нет прав доступа к файлу.")
```

Вопросы к подразд. 1.7

1. Какие аргументы принимает встроенная функция open()?
2. Перечислите известные режимы доступа, которые используются при открытии файлов.

3. Каким образом можно считывать файл построчно?
4. Для чего используется работа в бинарном режиме?
5. Какая конструкция отвечает за обработку исключений?

Практические задания к подразд. 1.7

1. Напишите программу, которая откроет текстовый файл, прочитает его содержимое и выведет его на экран.

2. Создайте программу, которая подсчитает количество строк в текстовом файле.

3. Напишите программу, которая откроет текстовый файл, выполнит поиск определенной строки и заменит ее на другую.

4. Создайте скрипт, который объединит содержимое нескольких текстовых файлов в один.

5. Напишите программу, которая откроет текстовый файл, разобьет его на слова и подсчитает, сколько раз каждое слово встречается в тексте.

6. Напишите программу, которая сравнит два текстовых файла и определит, совпадают ли их содержания.

1.8. Списки

Списки – это структура данных, которая хранит объекты произвольных типов данных. Чтобы работать со списками, требуется создать их с помощью функции list():

```
list("abcde")  # ['a', 'b', 'c', 'd', 'e']
```

Также списки можно создать через операцию [], что приведет к появлению пустого списка. Еще один способ создать список – это использовать генераторы списков. ***Генератор списков*** – способ построить новый список, применяя выражение к каждому элементу последовательности. Генераторы списков очень похожи на цикл for:

```
c = [c * 2 for c in 'Python']
print(c)  # ['PP', 'yy', 'tt', 'hh', 'oo', 'nn']
```

Списки являются коллекцией, в которую могут входить другие списки, например, c = ['a', 'b', ['c', 'd']] – список списков.

Для обращения к конкретному элементу требуется указать имя списка, а также через операцию [] индекс элемента. Принципы индексации списков такие же, как и у строк:

```
c = [1, 2, 3, 4, 5]
print(c[0]) # 1
print(c[-1]) # 5
print(c[7])  # IndexError: list index out of range
```

Два списка считаются равными, если они содержат один и тот же набор элементов, например,

```
numbers1 = [1, 2, 3, 4, 5]
numbers2 = list([1, 2, 3, 4, 5])
if numbers1 == numbers2:
   print("numbers1 равен numbers2")
else:
   print("numbers1 не равен numbers2")
```

Если необходимо получить какую-то определенную часть списка, то применяются срезы, принцип работы которых аналогичен тем, которые были описаны для строк:

```
people = ["Кирилл", "Иван", "Юрок", "Андрей", "Костя", "Антонина"]
slice_people1 = people[:3]  # с 0 по 3, не включая элемент 3
print(slice_people1)  # ["Кирилл", "Иван", "Юрок"]
slice_people2 = people[1:3]  # с 1 по 3, не включая элемент 3
print(slice_people2)  # ["Иван", "Юрок"]
slice_people3 = people[1:6:2]  # с 1 по 6 с шагом 2
print(slice_people3)  # ["Иван", "Андрей", "Антонина"]
```

Можно использовать отрицательные индексы:

```
people = ["Кирилл", "Иван", "Юрок", "Андрей", "Костя", "Антонина"]
slice_people1 = people[:-1]  # с предпоследнего по нулевой элемент
print(slice_people1)  # ["Кирилл", "Иван", "Юрок", "Андрей", "Костя"]
slice_people2 = people[-3:-1]  # с третьего с конца по предпоследний элемент
print(slice_people2)  # [ "Андрей", "Костя"]
```

Основные методы работы со списками приведены в табл. 2.

Таблица 2

Метод	Назначение
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L

Окончание табл. 2

Метод	Назначение
list.insert(i, x)	Вставляет на *i*-й элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. Возникает ValueError, если такого элемента не существует
list.pop([i])	Удаляет *i*-й элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Примеры работы с некоторыми методами:

```
people = ["Кирилл", "Иван"]
# добавляем в конец списка
people.append("Юрок")  # ["Кирилл", "Иван", "Юрок"]
# добавляем набор элементов ["Андрей", "Костя"]
people.extend(["Андрей", "Костя"])        # ["Кирилл", "Иван", "Юрок",
"Андрей", "Костя"]
# добавляем на вторую позицию
people.insert(1, "Антонина")  # ["Кирилл", "Антонина", "Иван", "Юрок",
"Андрей", "Костя"]
# удаляем все элементы
people.clear()  # []
```

В языке Python элементы также можно удалять с помощью оператора del. В качестве параметра этому оператору передается удаляемый элемент или набор элементов:

```
people = ["Кирилл", "Иван"]
del people[1]
print(people)  # ["Кирилл"]
```

Метод отличается от ***функции*** тем, что обращение к нему происходит с помощью операции. Примеры разных функций для работы со списками представлен в табл. 3.

Таблица 3

Название функции	Пояснение
list([object])	Создание списка
abs(x)	Возвращение абсолютной величины числа (модуль числа)
slice([start=0], stop, [step=1])	Объект среза от start до stop с шагом step
bytes([источник [, кодировка [ошибки]]])	Возвращение объекта типа bytes, который является неизменяемой последовательностью целых чисел в диапазоне $0 \leq X < 256$. Аргументы конструктора интерпретируются как для bytearray()
int([object], [основание системы счисления])	Преобразование к целому числу
input([prompt])	Возвращение введенной пользователем строки, prompt – подсказка пользователю
len(x)	Возвращение числа элементов в указанном объекте
max(x)	Нахождение максимального значения
min(x)	Нахождение минимального значения
sorted(x)	Сортировка коллекции

Вопросы к подразд. 1.8

1. Как создать пустой список в Python?
2. Как добавить элемент в конец списка?
3. Как проверить, содержится ли определенный элемент в списке?
4. Как получить длину списка?
5. Как удалить элемент из списка по индексу?
6. Как скопировать список?
7. Как объединить два списка в один?
8. Как отсортировать список?
9. Как инвертировать порядок элементов в списке?
10. Как найти максимальный и минимальный элементы в списке?
11. Как создать список с использованием генератора списка?

12. Как удалить все вхождения определенного элемента из списка?

Практические задания к подразд. 1.8

1. Управление оценками:

• создайте список, представляющий оценки студентов (целые числа);

• добавьте новую оценку в конец списка;

• удалите оценку, которая не удовлетворяет минимальному требованию;

• посчитайте средний балл;

• выведите все оценки выше среднего.

2. Соревнование по времени:

• создайте список временных результатов (в секундах) для участников забега;

• отсортируйте список по возрастанию времени;

• выведите первые три места;

• посчитайте среднее время.

3. Работа с числами:

• создайте список чисел;

• умножьте каждый элемент списка на два;

• найдите сумму всех чисел, которые делятся на три;

• замените все отрицательные числа на их абсолютные значения;

• выведите только уникальные числа.

1.9. Словари

Словари в Python представляют собой неупорядоченные коллекции, состоящие из пар ***«ключ – значение»***. Словарь создается с использованием фигурных скобок {}. Каждая пара «ключ–значение» отделяется от другой запятой, а элементы пар – двоеточием:

```
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}
```

Элементы словаря могут быть получены по ключу

```
print(my_dict['name'])  # John
```

Новые элементы добавляются присвоением значения по ключу. Существующие элементы изменяются аналогично:

```
my_dict['occupation'] = 'Engineer'
my_dict['age'] = 26
```

Элемент удаляется оператором del:

```
del my_dict['city']
```

С помощью оператора in можно проверить, существует ли ключ в словаре:

```
if 'name' in my_dict:
   print('Key "name" exists')
```

В словарях можно итерироваться по ключам, значениям или парам «ключ – значение»:

```
for key in my_dict:
   print(key, my_dict[key])
```

Наконец, необходимо рассмотреть основные методы работы со словарями.

Метод clear() удаляет все элементы из словаря:

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'Wonderland'}
my_dict.clear()
print(my_dict)  # {}
```

Метод get(key, default) возвращает значение для указанного ключа. Если ключ отсутствует, то сохраняется значение по умолчанию:

```
age = my_dict.get('age')
print(age) # 25
```

Метод items() возвращает представление всех пар «ключ – значение», метод keys() – всех ключей в словаре, а метод values() – всех значений в словаре. Метод pop(key, default) удаляет и возвращает значение для указанного ключа. Если ключ отсутствует и указано значение по умолчанию, то можно вернуть это значение. Метод update() в Python используется для обновления элементов словаря, добавляет элементы из другого словаря или итерируемого объекта. Если второй словарь содержит ключи, уже присутствующие в первом словаре, значения будут обновлены на соответствующие значения из второго словаря:

```
# Исходный словарь
original_dict = {'name': 'Alice', 'age': 25, 'city': 'Wonderland'}
# Новые данные для обновления
new_data = {'age': 26, 'occupation': 'Engineer', 'country': 'Fantasia'}
# Обновление словаря
```

```
original_dict.update(new_data)
# Вывод обновленного словаря
print(original_dict)
# {'name': 'Alice', 'age': 26, 'city': 'Wonderland', 'occupation': 'Engineer',
'country': 'Fantasia'}
```

Вопросы к подразд 1.9

1. Что такое словарь?
2. Какой оператор удаляет элементы словаря?
3. За что отвечает метод update()?

Практические задания к подразд. 1.9

1. Создайте словарь, учитывающий студентов в университете. Каждый студент должен быть представлен в виде словаря с ключами «имя», «возраст», «курс» и «средний балл». Реализуйте следующие задачи: добавьте несколько студентов в словарь, рассчитайте средний балл по всем студентам, найдите студента с самым высоким средним баллом, удалите одного из студентов из словаря, выведите информацию о каждом студенте в удобочитаемом формате.

2. Создайте словарь, представляющий некоторую телефонную книгу. Каждый контакт должен быть представлен в виде словаря с ключами «имя», «номер телефона», «электронный адрес» и «адрес проживания». Реализуйте следующие задачи: добавьте несколько контактов в словарь, проверьте наличие контакта в телефонной книге по имени, обновите номер телефона у одного из контактов, удалите один из контактов из телефонной книги, выведите список всех контактов в телефонной книге.

3. Создайте словарь, учитывающий продукты в магазине. Каждый продукт должен быть представлен в виде словаря с ключами «название», «цена» и «количество единиц на складе». Реализуйте следующие задачи: добавьте несколько продуктов в словарь, проверьте наличие конкретного продукта в магазине, обновите цену у одного из продуктов, удалите один из продуктов из словаря, выведите информацию о каждом продукте в читаемом формате.

1.10. Кортежи и множества

В Python ***кортеж* (*tuple*)** – это неизменяемая (immutable) последовательность элементов. Кортежи похожи на списки, но имеют не-

сколько ключевых различий. Кортежи создаются с использованием круглых скобок () или функции tuple():

```
# Создание пустого кортежа
empty_tuple = ()
# Создание кортежа из элементов
my_tuple = (1, 2, 3, 4, 5)
# Кортеж из различных типов данных
mixed_tuple = ('a', 1, 'b', 2.5)
# Кортеж из одного элемента (обратите внимание на запятую)
single_element_tuple = (10,)
# Использование функции tuple()
another_tuple = tuple([1, 2, 3])
```

Основные особенности кортежей сводятся к следующим. Во-первых, кортежи ***не могут изменяться.*** Пользователь не может добавлять, удалять или изменять элементы кортежа после его создания. Во-вторых, кортежи можно ***индексировать.*** Наконец, Python позволяет ***упаковывать*** и ***распаковывать*** значения для присваивания и передачи значений:

```
# Индексация кортежа
print(my_tuple[0])  # Выводит: 1
print(mixed_tuple[2])  # Выводит: 'b'
# Упаковка и распаковка
a, b, c = my_tuple  # Распаковка кортежа
print(a, b, c)  # Выводит: 1 2 3
```

Множество **(*set*)** – это неупорядоченная коллекция уникальных элементов. Оно создается с использованием фигурных скобок {} или функции set():

```
# Создание пустого множества
empty_set = set()
# Множество из элементов
my_set = {1, 2, 3, 4, 5}
# Множество из списка (с удалением дубликатов)
another_set = set([1, 2, 2, 3, 4, 4, 5])
```

Необходимо отметить основные особенности множеств. В множестве ***не может быть дублирующихся элементов.*** Элементы множества ***не имеют определенного порядка.*** При выводе элементы могут располагаться в разном порядке. Множества ***поддерживают такие операции, как объединение, пересечение, разность и проверка на подмножество.***

```
# Операции над множествами
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
# Объединение множеств
union_set = set1.union(set2)  # или set1 | set2
print(union_set)  # {1, 2, 3, 4, 5, 6}
# Пересечение множеств
intersection_set = set1.intersection(set2)  # или set1 & set2
print(intersection_set)  # {3, 4}
# Разность множеств
difference_set = set1.difference(set2)  # или set1 - set2
print(difference_set)  # {1, 2}
```

Вопросы к подразд. 1.10

1. Что такое кортеж? Перечислите его особенности?
2. Что такое множество? Перечислите его особенности?

Практические задания к подразд. 1.10

1. Напишите программу, которая объединяет два кортежа элементов в один, содержащий все элементы обоих исходных кортежей без дубликатов.
2. Напишите программу, которая принимает два кортежа и определяет, одинаковы ли они.
3. Проверьте, существует ли определенный элемент в кортеже, и выведите сообщение о его наличии или отсутствии.
4. Создайте два множества и проверьте, пересекаются ли они (имеют ли общие элементы).
5. Создайте два множества и найдите разность между ними (элементы, которые есть в первом множестве, но отсутствуют во втором).

1.11. Функции

Функцию в программировании можно рассматривать как небольшую автономную подпрограмму, которая встроена в основную программу. Функция содержит код, решающий конкретную задачу. Например, в некоторой компьютерной игре могут существовать отдельные функции для подсчета очков, отрисовки игрового поля, управления движением фигур и пр. Использование функций имеет ряд достоинств. Во-первых, переменные, определенные внутри функции, видны только внутри этой функции. Это позволяет избежать конфликтов и ошибок, связанных с именами переменных, либо встречающихся в разных частях программы. Во-вторых, функции позволяют изолировать повторяющийся код и использовать его многократно. Это снижает объем кода, так как, если возникает необходимость в изменении логики программы, внести корректировки можно в одном месте (внутри функции). Наконец, применение функциональных блоков внутри кода делает его отладку более эффективной.

Функция в Python определяется с использованием ключевого слова def, за которым следует имя функции и круглые скобки, содержащие параметры функции:

```
def say_hello(name):
    print("Привет, "+name+"!")
```

В данном примере say_hello – имя функции, name – параметр функции. Для выполнения кода внутри функции необходимо вызвать ее с помощью имени функции и передать ей аргументы (значения) в параметры, если они определены. В нижеприведенном примере пользователь передает строку "Анна", а результатом исполнения функции будет строка именного приветствия:

```
say_hello("Анна")  # Привет, Анна!
```

Функции могут возвращать результат своей работы за счет ключевого слова return. Такой подход позволяет сохранить результат выполнения функции в переменной:

```
def add(a, b):
    result = a + b
    return result
```

Функция add принимает два аргумента a и b, она выполняет сложение и возвращает результат:

```
sum_result = add(3, 4)
print(sum_result)  # 7
```

Параметры функции могут быть обязательными или иметь значения по умолчанию. Параметры со значениями по умолчанию необязательны при вызове функции:

```
def greet(name, greeting="Привет"):
    print(f"{greeting}, {name}!")
greet("Анна")  # Привет, Анна!
greet("Петр", "Здравствуйте")  # Здравствуйте, Петр!
```

Переменные, объявленные внутри функции, локальны и видны только внутри этой функции. Переменные, объявленные вне функции, являются глобальными и доступны из любой части программы.

Функция может вызывать саму себя. Этот метод называется ***рекурсией*** и используется для решения задач, которые могут быть разделены на подзадачи:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Функция factorial принимает один аргумент n, который представляет собой число, для которого нужно вычислить факториал. Факториал числа n (обозначается как n!) – это произведение всех целых чисел от 1 до n. Например, факториал 5 равен 5! = 5 * 4 * 3 * 2 * 1 = 120. В рекурсивной функции factorial базовый случай – тот, в котором n равно 0. В таком случае функция возвращает 1, так как факториал 0 равен 1. В противном случае функция вызывает саму себя с аргументом n - 1 и умножает результат на n, чтобы получить факториал n:

```
result = factorial(5)
print(result)  # 120
```

Отдельным видом функций являются ***анонимные функции* (*lambda-функции*).** Они представляют собой специальный вид функций, которые могут быть определены без использования ключевого слова def. Анонимные функции часто используются для создания не-

больших одноразовых функций, которые не требуют именования и объявления в коде. Основной синтаксис lambda-функции выглядит следующим образом:

```
lambda arguments: expression
```

В представленной выше структуре arguments – это список аргументов, передаваемых в функцию, а expression – это выражение, которое должно быть выполнено и возвращено как результат работы функции. Приведем простейшую lambda-функцию, которая возвращает сумму двух чисел:

```
add = lambda x, y: x + y
result = add(3, 4)
print(result)  # 7
```

Наконец, функции в Python можно задокументировать, т.е. подвергнуть подробному описанию для удобства других пользователей. Для этой цели используются ***строки документации*** **(*docstrings*),** которые располагаются внутри функции и описывают ее назначение и параметры:

```
def my_function(param1, param2):
    """
    Описание функции
    :param param1: Описание параметра 1
    :param param2: Описание параметра 2
    :return: Описание возвращаемого значения
    """
    # Код функции
```

Вопросы к подразд. 1.11

1. Что такое функция?
2. Каким образом задается функция в Python?
3. Объясните понятия рекурсии.
4. Что такое анонимная функция?
5. Что такое строки документации?

Практические задания к подразд. 1.11

1. Создайте функцию, которая определяет, является ли заданное число простым.

2. Создайте функцию, которая принимает строку и возвращает количество слов в ней.

3. Напишите функцию, которая принимает строку и возвращает ее в обратном порядке.

4. Напишите функцию, которая находит максимальный элемент в списке.

5. Создайте функцию, которая принимает список чисел и возвращает среднее значение их суммы.

6. Напишите функцию, которая находит уникальные элементы в списке.

7. Напишите функции для конвертации температуры из градусов Цельсия в Фаренгейта и наоборот.

8. Разработайте функцию-калькулятор, которая принимает выражение в виде строки и выполняет арифметические операции.

9. Напишите функцию, которая принимает коэффициенты квадратного уравнения и возвращает его корни.

10. Создайте функцию, которая считывает текстовый файл, выполняет над ним определенную операцию (например, поиск подстроки и ее замену) и записывает результат в новый файл.

1.12. Классы. Атрибуты и методы

В прошлых подразделах рассматривались различные варианты встроенных типов данных: целочисленные, вещественные, строки, а также булев тип данных. Но также в языке Python можно создавать собственные типы данных. Для этого используют классы. ***Класс*** – это шаблон или чертеж, описывающий состояние и поведение какой-либо сущности, которая может быть представлена в виде объекта. ***Объект*** – это конкретная реализация какого-либо класса.

Для понимания, что такое класс и объект, можно провести аналогию. Человека можно рассматривать как сущность с набором характеристик: имя, фамилия, возраст, цвет глаз или волос и др. Человек может выполнять различные действия: говорить, бегать, прыгать и др. Набор данных характеристик и действий человека можно считать классом, так как у каждого человека они индивидуальны, а конкретный человек является экземпляром этого класса.

Класс определяется с помощью ключевого слова class:

```
class название_класса:
    атрибуты_класса
    методы_класса
```

В приведенном выше коде атрибутами являются характеристики человека, а методами – его действия:

```
class Human:
    pass
```

В данном случае установлен класс Human, который условно представляет человека. В классе не определяются ни методы, ни атрибуты, однако поскольку в нем должно быть что-то определено, то в качестве заменителя функционала класса применяется оператор pass. Этот оператор нужен, когда синтаксически необходимо найти некоторый код, однако в ходе работы его использовать не нужно, и вместо конкретного кода вставляют оператор pass. Теперь создаются объекты данного класса Human:

```
class Human:
    pass
alice = Human()
pete = Human()
```

В данном коде после определения класса были созданы два объекта – alice и pete. Для этого вызывается особая функция, которая называется конструктором. ***Конструктор*** – это метод, который вызывается при создании какого-либо экземпляра класса (объекта). У него такое же имя, как и у класса. Каждый класс по умолчанию имеет конструктор без параметров.

В Python ***метод класса*** – это функция, определенная внутри класса и выполняющая операции над атрибутами класса. В отличие от методов экземпляра, методы класса имеют доступ к самому классу, а не к конкретному экземпляру этого класса. Метод класса может использоваться для создания или модификации атрибутов класса, выполнения операций, зависящих от класса в целом, или для возвращения новых экземпляров класса. Он полезен в тех случаях, когда операция не зависит от конкретного экземпляра класса, но требует доступа к его атрибутам:

```
class Human:  # определение класса Human
    def say_hello(self):
        print("Привет")
alice = Human()
alice.say_hello()
```

Здесь определен метод say_hello(), который условно выполняет приветствие – выводит строку на консоль. При выборе методов любого класса следует учитывать, что все они должны принимать в качестве первого параметра ссылку на текущий объект, который, согласно условностям, называется self. Через эту ссылку внутри класса можно обратиться к функциональности текущего объекта. Но при самом вызове метода этот параметр не учитывается.

Для обращения к методу через имя класса требуется оператор . (точка):

```
объект.метод(параметры)
```

Если метод должен принимать другие параметры, то они определяются после параметра self, и при вызове подобного метода для них необходимо передать значения:

```
class Human:      # определение класса Human
   def say(self, message):    # метод
      print(message)
alice = Human()
alice.say("Привет Мир")
```

В коде выше определяется метод say() с параметром self как ссылка на объект и метод message, при вызове которого необходимо передать значение. Параметр self – это просто соглашение о наименовании, которое обозначает первый параметр методов в классе. self указывает на экземпляр класса, с которым в данный момент работает метод. Это позволяет методам класса получать доступ к переменным экземпляра и другим методам этого экземпляра:

```
class MyClass:
   def __init__(self, value):
      self.var = value
   def display (self):
      print("Значение var:", self.var)
# Создаем экземпляр класса
obj = MyClass(value=42)
# Вызываем метод, который выводит значение instance_variable
obj.display()
```

В этом примере self в методе __init__ и self.var обозначают, что работа происходит с переменной экземпляра класса. Когда создается

экземпляр класса obj = MyClass(value=42), self автоматически ссылается на этот экземпляр, а переменная var становится частью этого экземпляра.

При вызове метода obj.display() self ссылается на экземпляр obj. Это позволяет получить доступ к var для данного экземпляра и вывести его значение.

Важно отметить, что self – это просто имя, возможно и другое имя, но соглашение о наименовании предполагает self. Также в Python концепция self применяется к методам экземпляра, а не к методам класса или статическим методам, которые могут использовать другие имена (например, cls для методов класса). Пример метода класса:

```
class MyClass:
    class_var = 0
    def display (cls):
        print("Значение class_var:", cls.class_var)
# Вызываем метод класса
MyClass.display()
```

В этом примере cls заменяет self, потому что это метод класса. В методе класса display cls ссылается на сам класс, а не на экземпляр.

Теперь более подробно рассмотрим ***конструктор*** – метод класса, который автоматически вызывается при создании экземпляра класса. Существует три типа конструкторов: ***по умолчанию*** (без параметров), ***копирования*** и ***конструктор с параметрами.*** В коде выше использовался конструктор по умолчанию, который не принимает параметров и который неявно имеют все классы. Он служит для инициализации объекта, устанавливая начальные значения атрибутов. В Python конструктор можно явным образом определить через __init__:

```
class MyClass:
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2
# Создаем экземпляр класса, вызывая конструктор
obj = MyClass(param1="Значение1", param2="Значение2")
# Доступ к атрибутам, установленным в конструкторе
print("Значение param1:", obj.param1)
print("Значение param2:", obj.param2)
```

В этом примере конструктор __init__ принимает два параметра: param1 и param2 – и устанавливает соответствующие атрибуты объекта: self.param1 и self.param2. При создании экземпляра класса obj, конструктор вызывается автоматически, а атрибуты param1 и param2 инициализируются переданными значениями.

Конструктор часто используется для установки начальных значений переменных экземпляра, создания и инициализации объектов, а также для выполнения других действий, которые должны произойти при создании объекта.

Атрибуты – переменные, связанные с объектами (экземплярами классов, классами или модулями):

```
class Human:
   def __init__(self, name):
      self.name = name    # имя человека
      self.age = 1        # возраст человека
alice = Human ("Alice")
# обращение к атрибутам
# получение значений
print(alice.name)    # Alice
print(alice.age)     # 1
# изменение значения
alice.age = 21
print(alice.age)     # 21
```

Теперь конструктор класса Human принимает еще один параметр – name. Через этот параметр в конструктор будет передаваться имя создаваемого человека. Отметим, что атрибуты необязательно определять внутри класса, так как Python позволяет сделать это динамически вне кода.

```
class Human:
   def __init__(self, name):
      self.name = name    # имя человека
alice = Human ("Alice")
alice.age = "22"
print(alice.age)  # 22
```

В приведенном примере динамически устанавливается атрибут age, который хранит возраст человека. После присваивания значения пользователь может его получить. В то же время подобное определе-

ние чревато ошибками. Например, если обратиться к атрибуту до его определения, то программа сгенерирует ошибку:

```
class Human:
    def __init__(self, name):
        self.name = name    # имя человека
alice = Human("Alice")
print(alice.age)
# AttributeError: 'Human' object has no attribute 'age'
```

Для обращения к атрибутам объекта внутри класса в его методах также применяется слово self:

```
class Human:
    def __init__(self, name):
        self.name = name    # имя человека
        self.age = 19       # возраст человека
    def display_info(self):
        print(f"Name: {self.name}  Age: {self.age}")
alice = Human ("Alice ")
alice.display_info()
# Name: Alice   Age: 19
```

В данном примере определяется метод display_info(), который выводит информацию на консоль. И для обращения в методе к атрибутам объекта применяется слово self: self.name и self.age.

Вопросы к подразд. 1.12

1. Что такое класс в Python и чем он отличается от экземпляра класса?
2. Какие типы конструкторов существуют в Python?
3. Как можно создать экземпляр класса в Python и что происходит при его создании?
4. Какие способы доступа к атрибутам и методам объекта класса существуют в Python?
5. Что такое self?

Практическое задание к подразд. 1.12

1. Создать класс Person, представляющий человека. У класса должны быть следующие атрибуты:
   - name (строка) – имя человека;
   - age (целое число) – возраст человека;

• gender (строка) – пол человека.

2. Добавить метод __init__, который будет принимать значения name, age и gender и инициализировать соответствующие атрибуты объекта.

3. Добавить метод display_info, который будет выводить информацию о человеке в следующем формате: Имя: [name], Возраст: [age], Пол: [gender].

4. Создать несколько экземпляров класса Person с разными данными.

5. Использовать метод display_info() для отображения информации о созданных людях.

6. Добавить метод calculate_birth_year(), который будет печатать год рождения человека, основываясь на текущем годе и его возрасте.

1.13. Импортирование библиотек и модулей в Python

Язык программирования Python предоставляет большое количество встроенных и внешних ***модулей*** или ***библиотек*** для решения узкоспециализированных задач, таких как, например, работа с математическими операциями. Модуль – это файл, которые содержит определения функций, классов и переменных, а также исполняемый код. Библиотека – это набор связанных модулей, объединенных вместе для решения некоторой задачи.

Для импортирования встроенных библиотек или модулей служит ключевое слово import и имя модуля или библиотеки, которое записывается после него:

```
import math  # Импортирование модуля math для математических операций
```

После импортирования модуля можно использовать его функции, переменные и классы через синтаксис module.element:

```
x = math.sqrt(25)  # Использование функции sqrt() из модуля math
```

Пользователь может также импортировать только определенные функции, переменные или классы из стандартных библиотек с помощью from:

```
from math import sqrt  # Импортирование только функции sqrt из модуля math
```

Чтобы сократить запись и сделать код более читаемым, для библиотек при их импортировании можно задать псевдоним с помощью ключевого слова as:

```
import math as m  # Импортирование модуля math с псевдонимом m
x = m.sqrt(25)  # Использование псевдонима m для math
```

Python поддерживает сторонние библиотеки, которые не входят в стандартный набор. Чтобы использовать такие библиотеки, их нужно установить (например, через командную строку в ОС Windows помощью pip install ...) и затем импортировать в свой код. Для библиотеки, приведенной ниже, нужно в командной строке заранее прописать ее установку следующим образом: pip install requests. Если возникают сложности с установкой, то можно обратиться к документации библиотеки, так как разработчики обычно прописывают способы установки:

```
import requests  # Импортирование библиотеки requests для работы с HTTP-запросами
```

Наконец, пользователи могут импортировать и заранее созданные ими модули. Для этого нужно создать файл с расширением .py. В этом файле можно определять переменные, функции и классы:

```
# Модуль mymodule.py
def hello():
    print("Привет, мир!")
my_variable = 42
```

Затем другие программы могут импортировать модуль. После импортирования модуля можно использовать его функции, переменные и классы, обращаясь к ним через имя модуля, например, mymodule.hello().

Вопросы к подразд. 1.13

1. Чем модуль отличается от библиотеки?
2. Каким образом можно импортировать модули и библиотеки в Python?
3. Что обозначает ключевое слово from?
4. Что обозначает ключевое слово as?
5. Какие шаги нужно предусмотреть перед импортированием сторонней библиотеки?

Практические задания к подразд. 1.13

1. Импортируйте модуль math и используйте его функцию для вычисления некоторого значения, например, квадратного корня или синуса.

2. Импортируйте модуль datetime и используйте его для вывода текущей даты и времени.

3. Импортируйте модуль random и создайте программу, которая генерирует случайное число от 1 до 100 и предлагает пользователю угадать его.

4. Импортируйте модуль os и создайте программу, которая выводит список файлов в текущей директории.

5. Создайте свой собственный модуль с функцией или классом, а затем импортируйте его в другой файл и используйте его функциональность.

2. ОБЗОР СПЕЦИАЛИЗИРОВАННЫХ БИБЛИОТЕК И ФРЕЙМВОРКОВ ДЛЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON

2.1. Специализированные библиотеки для обработки текстовых массивов на естественном языке

Существует множество библиотек для обработки текстов на естественном языке (Natural Language Processing, NLP) в Python.

NLTK (Natural Language Toolkit) – пакет программ для статистической обработки текстовых массивов данных: деление на единицы анализа (токенизация), приведение к каноническим формам (лемматизация) или основам/псевдоосновам (стемминг), создание синтаксических структуры предложений и пр. Пример, демонстрирующий деление текста на токены:

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
text = "Это пример предложения. NLTK поможет его токенизировать."
words = word_tokenize(text)
print(words)
# ['Это', 'пример', 'предложения', '.', 'NLTK', 'поможет', 'его', 'токенизировать', '.']
```

Библиотека ***spaCy*** – это NLP-модуль, предназначенный для выполнения различных задач, таких как токенизация, частеречная разметка, извлечение именованных сущностей и пр. Пример вывода лексических единиц, знаков препинания и соответствующих морфологических тегов:

```
import spacy
nlp = spacy.load("en_core_web_sm")  # загрузка англоязычной модели
text = "SpaCy is an excellent library for Python."
doc = nlp(text)  # создание конвейера обработки текста
for token in doc:
   print(token.text, token.pos_)  # вывод токенов и частеречной разметки
# SpaCy PROPN
# is AUX
# an DET
# excellent ADJ
# library NOUN
# for ADP
# Python PROPN
# . PUNCT
```

Библиотека ***TextBlob*** предлагает расширить ряд задач, которые могут возникать в лингвистике: перевод текстов с одного языка на другой, поиск синонимов, а также тональный анализ в количественных единицах (положительно окрашенное предложение, нейтрально окрашенное предложение или отрицательно окрашенное предложение):

```
from textblob import TextBlob
text = "The food in the canteen of BSTU was great!"
blob = TextBlob(text)
print(blob.sentiment)
# Sentiment(polarity=1.0, subjectivity=0.75)
```

Gensim – это библиотека для моделирования тем и векторизации текста. Она позволяет работать с алгоритмами Word2Vec, Doc2Vec и пр. Ниже приведен пример для лексической единицы «первое» с тензором, состоящим из 100 значений (vector_size) при контекстном окне анализа 5:

```
from gensim.models import Word2Vec
sentences = [["это", "первое", "предложение"], ["а", "это", "второе",
"предложение"]]
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1,
sg=0)
print(model.wv['первое'])
# [-0.00713902  0.00124103 -0.00717672 -0.00224462  0.0037193 ...]
```

С 2016 г. в области компьютерной лингвистики начали использоваться нейронные сети. Одной из ведущих компаний является *HuggingFace*, предлагающая архитектуру ***Transformers***. Она предоставляет предварительно обученные языковые модели для различных задач NLP, такие как машинный перевод, извлечение информации, классификация текстов и т.д. Пользователь в коде, приведенном ниже, должен вывести количественное значение тональности предложения, при этом он заранее не конкретизировал тип языковой модели, с которой необходимо работать. В таком случае автоматически выбирается модель по умолчанию – distilbert base uncased-finetuned-sst-2-english. Для более точных результатов необходимо заранее прописать модель, которая обучена для решения определенного типа задач и на текстах определенного языка:

```
from transformers import pipeline
nlp = pipeline("sentiment-analysis")
result = nlp("The lectures in our university are really interesting!")
print(result)
# [{'label': 'POSITIVE', 'score': 0.9998194575309753}]
```

Более подробно с каждой из этих библиотек можно ознакомиться в соответствующей онлайн-документации.

Вопросы к подразд. 2.1

1. Что такое лемматизация и стемминг?
2. Что такое тональный анализ?
3. Какие библиотеки могут использоваться для представления текстов в векторной форме?
4. Что такое архитектура ***Transformers***?

Практические задания к подразд. 2.1

Выберите русскоязычный художественный текст не менее 100000 словоформ XX в. или начала XXI в. Воспользуйтесь комбинацией нескольких библиотек для представления следующих результатов:

1) вывод всех лемм текста;
2) вывод частеречной разметки каждой словоформы текста;
3) создание синтаксических структур зависимостей и непосредственно составляющих для трех любых сложноподчиненных предложений текста;
4) вывод результата тонального анализа для трех любых предложений текста;
5) вывод тематической модели текста (любого вида).

2.2. Парсинг Интернет-данных

BeautifulSoup – это библиотека для парсинга HTML- и XML-документов в Python. Она облегчает извлечение данных из веб-страниц. Прежде чем начать использовать библиотеку, нужно ее установить:

```
pip install beautifulsoup4
```

В проект библиотека импортируется следующим образом:

```
from bs4 import BeautifulSoup
```

В начале нужно создать объект BeautifulSoup, передавая ему HTML- или XML-документ. Это можно сделать, например, с помощью модуля requests для загрузки веб-страницы:

```
import requests
url = 'https://example.com'
response = requests.get(url)
html = response.text
soup = BeautifulSoup(html, 'html.parser')
```

Пользователь также может загрузить HTML из локального файла:

```
with open('example.html', 'r') as file:
   html = file.read()
soup = BeautifulSoup(html, 'html.parser')
```

BeautifulSoup предоставляет множество методов для поиска элементов на веб-странице. Один из наиболее часто используемых методов – это find() для поиска первого элемента, удовлетворяющего критериям:

```
title = soup.find('title')
print(title.text)
```

Если пользователь хочет найти все элементы, удовлетворяющие критериям, то можно обратиться к find_all():

```
links = soup.find_all('a')
for link in links:
   print(link['href'])
```

Пользователь получает доступ к атрибутам элементов с помощью квадратных скобок, как показано в примере выше. Например, чтобы получить значение атрибута href у ссылки, можно использовать link['href'].

HTML-документ представляет собой древовидную структуру. Можно перемещаться по этой структуре за счет методов навигации BeautifulSoup:

```
# Перейти к родительскому элементу
parent = link.parent
# Перейти к следующему элементу
next_sibling = link.find_next_sibling()
# Перейти к предыдущему элементу
prev_sibling = link.find_previous_sibling()
```

В BeautifulSoup также возможно фильтровать результаты поиска, используя функции-фильтры. Например, чтобы найти все элементы с определенным классом, можно воспользоваться следующим кодом:

```
elements_class = soup.find_all(class_='example-class')
```

Чтобы получить текст из элемента, используется .text:

```
paragraph = soup.find('p')
text = paragraph.text
```

Наконец, приведем небольшую цельную программу с применением BeautifulSoup. Предположим, у пользователя есть веб-сайт с новостными заголовками, которые необходимо извлечь их с помощью этой библиотеки:

```
import requests
from bs4 import BeautifulSoup
url = 'https://example-news-site.com'
response = requests.get(url)
html = response.text
soup = BeautifulSoup(html, 'html.parser')
```

```
headlines = soup.find_all('h2', class_='news-headline')
for headline in headlines:
    print(headline.text)
```

В этом примере пользователь отправляет запрос на веб-сайт, загружает HTML-страницу, создает объект BeautifulSoup и затем находит все элементы <h2> с классом news-headline, извлекая их текст и выводя на экран.

Вопросы к подразд. 2.2

1. Что такое BeautifulSoup и для чего он используется в Python?
2. В чем разница между find() и find_all().
3. Как можно фильтровать результаты поиска с использованием функций-фильтров в BeautifulSoup?
4. Как извлечь текст из элемента на веб-странице с помощью BeautifulSoup?

Практические задания к подразд. 2.2

Обратитесь к Интернет-порталу «Коммерсантъ». Извлеките названия из ленты новостей первой страницы, а также попытайтесь извлечь содержание этих новостей.

2.3. Разработка веб-приложения на основе Django

Django – это высокоуровневый веб-фреймворк, написанный на языке программирования Python, который позволяет быстро и эффективно разрабатывать веб-приложения. Он предоставляет множество удобных инструментов и абстракций, упрощающих процесс разработки, такие как работа с базами данных, обработка форм, управление сессиями, аутентификация, административный интерфейс и многое другое.

Основные черты и компоненты Django.

1. ***ORM (Object-Relational Mapping).*** Django предоставляет собой ORM, который позволяет работать с базами данных, используя объектно-ориентированный подход. В виде классов Python можно определить модели данных, а Django позаботится о создании и управлении соответствующими таблицами в базе данных.

2. ***Административный интерфейс.*** Django поставляется с встроенным административным интерфейсом, который автоматически создается на основе определенных моделей данных. Это позволяет администраторам управлять данными в приложении без написания дополнительного кода.

3. ***URL-преобразование и маршрутизация.*** Django использует систему URL-преобразования, с помощью которой легко определять структуру URL для некоторых приложений. URL-пути могут быть привязаны к определенным представлениям (views), обрабатывающим запросы.

4. ***Шаблоны.*** Django использует систему шаблонов для создания динамических HTML-страниц. Шаблоны позволяют вам вставлять переменные, логику и циклы прямо в HTML-код, упрощая создание динамического контента.

Фреймворк Django реализует архитектурный паттерн ***Model-View-Template (MVT),*** который по факту является модификацией распространенного в веб-программировании паттерна ***MVC (Model-View-Controller).*** Схематично архитектура MVT в Django приведена на рис. 8.

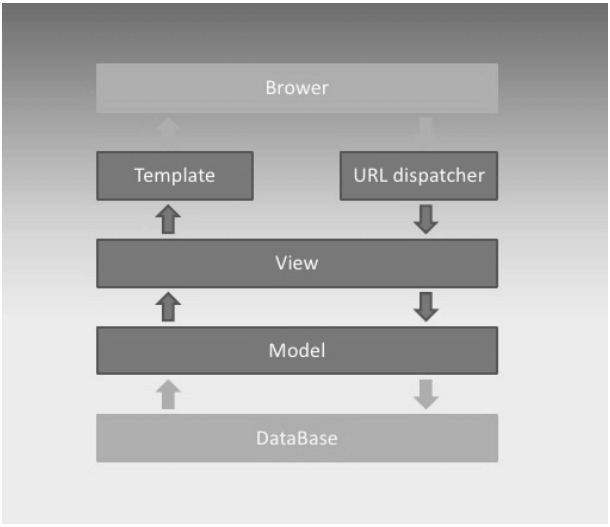

Рис. 8. Архитектура Model-View-Template

Основные элементы паттерна MVT:

1) ***URL dispatcher*** при получении запроса на основании запрошенного адреса URL определяет, какой ресурс должен обрабатывать данный запрос;

2) ***View*** получает запрос, обрабатывает его и отправляет пользователю некоторый ответ. Если для обработки запроса необходимо обращение к модели и базе данных, то View взаимодействует с ними.

Для создания ответа может применяться Template или шаблоны. В архитектуре MVC этому компоненту соответствуют контроллеры (но не представления);

3) ***Model*** описывает данные, используемые в приложении. Отдельные классы, как правило, соответствуют таблицам в базе данных;

4) ***Template*** представляет логику представления в виде сгенерированной html-разметки. В MVC этому компоненту соответствует View, т.е. представления.

Когда в приложение приходит запрос, то URL dispatcher определяет, с каким ресурсом сопоставляется данный запрос и передает этот запрос выбранному ресурсу. Ресурс фактически представляет функцию или View, который получает запрос и определенным образом обрабатывает его. В процессе обработки View может обращаться к моделям и базе данных, получать из нее данные или, наоборот, сохранять в нее данные. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет сгенерированный html-код, для генерации которого применяются шаблоны (Template).

После установки Django требуется создать проект, используя команду django-admin.py startproject LOG. С помощью django-admin предоставляется ряд команд для управления проектом Django. В частности, для создания проекта применяется команда startproject. Этой команде в качестве аргумента передается название проекта, в данном случае – LOG и будет создан проект с именем LOG, куда нужно перейти. Необходимо найти файл manage.py, через который будет управляться проект. Запуск проекта идет через команду python manage.py runserver. Если все выполнено верно, запустится веб-сервер с приветственной страницей Django (рис. 9).

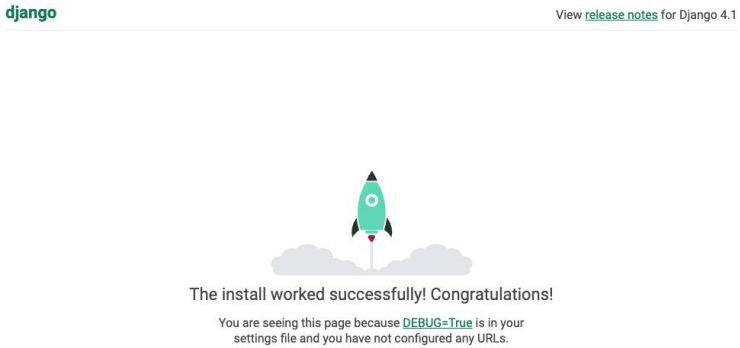

Рис. 9. Приветственная страница проекта

Созданный проект будет состоять из следующих элементов:

1. manage.py – выполняет различные команды проекта, например, создает и запускает приложение.

2. Собственно папка проекта LOG, которая содержит следующие файлы:

• файл __init__.py – указывает, что папка, в которой он находится, будет рассматриваться как модуль, это стандартный файл для программы на языке Python;

• settings.py – содержит настройки конфигурации проекта;

• urls.py – содержит шаблоны URL-адресов, определяет систему маршрутизации проекта;

• wsgi.py – содержит свойства конфигурации Web Server Gateway Inerface (WSGI), он используется при развертывании проекта;

• asgi.py – расширяет возможности WSGI, добавляя поддержку для взаимодействия между асинхронными веб-серверами и приложениями.

После открытия проекта нужно создать базу данных для хранения записей, они хранятся в файле db.sqlite3 (рядом с файлом manage.py), и для работы необходимо создать сами таблицы командой python manage.py migrate.

Так как проект на Django должен состоять из различных приложений (apps), то теперь нужно создать новое приложение в проекте, в данном случае оно называется logs. Делается это с помощью команды python manage.py startapp logs. В результате выполнения этих действий папка проекта LOG будет иметь содержимое, представленное на рис. 10.

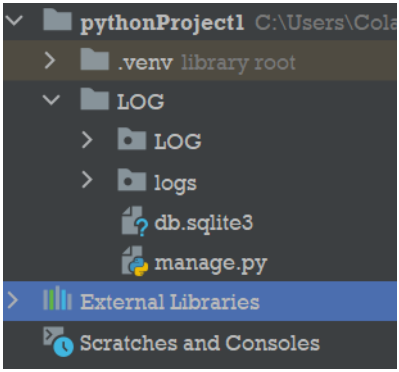

Рис. 10. Корневая структура проекта

Проект называется LOG, а само приложение – logs. Теперь требуется в файле settings.py найти список INSTALLED_APPS и добавить новый элемент с названием приложения – logs (рис. 11).

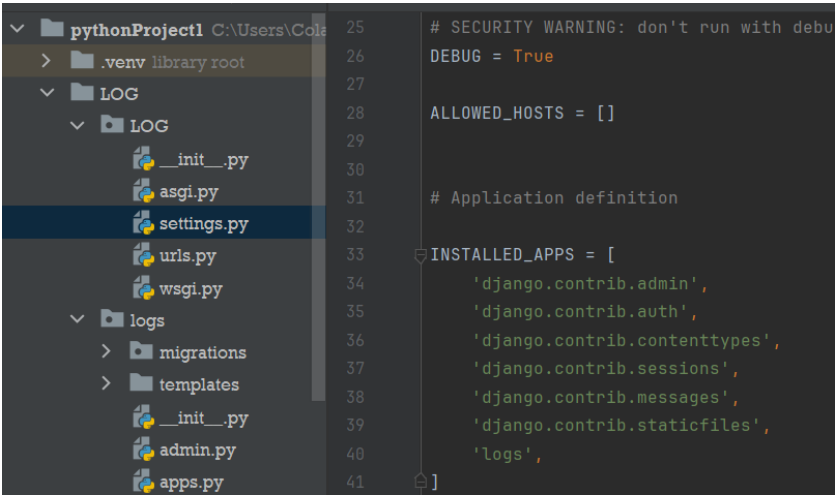

Рис. 11. Модификация файла settings.py

На следующем этапе нужно добавить модели Topic и Entry в файл models.py (папка LOG/logs). Модели будут содержать описания данных, которые будут храниться в таблицах баз данных проекта.

В файл models.py нужно добавить следующее содержимое:

```
from django.db import models
class Topic(models.Model):
        text = models.CharField(max_length=200)
        date_added = models.DateTimeField(auto_now_add=True)
        def __str__(self):
                return self.text
class Entry(models.Model):
        topic = models.ForeignKey(Topic, on_delete=models.PROTECT)
        text = models.TextField()
        date_added = models.DateTimeField(auto_now_add=True)
        class Meta:
                verbose_name_plural = 'entries'
```

```
    def __str__(self):
            return self.text[:50] + "..."
```

Далее создается миграция добавленных моделей и на этой основе присоединяются новые таблицы в базу данных с помощью команд python manage.py makemigrations logs и python manage.py migrate. Формируется суперпользователь командой python manage.py createsuperuser. Вносятся данные имени и пароль. При входе на сайт и ввода данных выводится админ-панель.

Таблица с двумя строчками для работы с группами пользователей (Groups) и с самими пользователями (Users) приведена на рис. 12. Для работы с текущими моделями их нужно зарегистрировать в файле admin.py (папка LOG/logs) через добавление следующего кода:

```
from django.contrib import admin
from logs.models import Topic, Entry
admin.site.register(Topic)
admin.site.register(Entry)
```

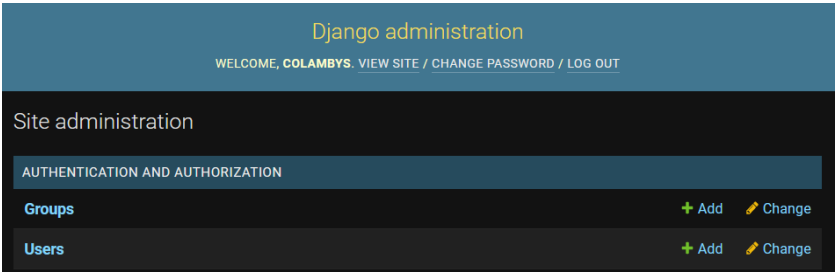

Рис. 12. Интерфейс админ-панели

После этого требуется обновить страницу админ-панели, можно увидеть две новые строчки: Entries и Topics.

На следующем этапе требуется добавить ***маршруты*** (urls), по которым будут открываться нужные страницы сайта. Для этого следует отредактировать два файла с одинаковым названием urls.py: это файл urls.py для всего проекта в целом (папка LOG/LOG) и для отдельного приложения logs (папка LOG/logs). В папке приложения logs файла urls.py нет, его нужно создать самостоятельно.

Для начала требуется открыть файл urls.py проекта (папка LOG/LOG) и в список urlpatterns добавить новую строчку:

```
from django.urls import path, include
urlpatterns = [
   path('admin/', admin.site.urls),
   path('', include('logs.urls', namespace='logs')),
]
```

Далее требуется создать файл urls.py в папке приложения (папка LOG/logs). Его содержимое должно быть таким:

```
from django.urls import path
from . import views
app_name = 'logs'
urlpatterns = [
            path('', views.index, name='index'),
            path('topics/<int:topic_id>/', views.topic, name='topic'),
]
```

В коде выше, в методе path(), первым параметром идет url страницы на сайте, второй параметр – это метод в файле views (в данном случае index или topic), который должен обрабатывать пришедший запрос от посетителя сайта, третий параметр – это произвольное имя данного маршрута, которое может применяться в дальнейшем для обращения к этому маршруту.

Представление (View) в Django – это контроллер в других фреймворках. Представление получает информацию от модели и отдает ее в ***шаблон (template),*** который нужен для размещения переданной ему информации на html-странице. Класс представления – views.py – можно найти в папке приложения logs (LOG/logs). Требуется ввести в него два метода из маршрутов, которые добавились выше, в файле urls.py приложения index() и topic(). В результате его содержимое должно быть следующим:

```
from django.shortcuts import render
from .models import Topic
def index(request):
            topics = Topic.objects.order_by('-date_added')
            context = {
                        'topics': topics,
            }
            return render(request,
                        'logs/index.html', context)
```

```
def topic(request, topic_id):
        topic = Topic.objects.get(id=topic_id)
        entries = topic.entry_set.order_by('-date_added')
        context = {
                'topic': topic,
                'entries': entries,
        }
        return render(request,
                'logs/topic.html', context)
```

В обоих методах присутствует метод render(). Он использует html-файлы для вывода полученных из модели данных на страницу в браузере. Это и есть шаблоны. Всего их будет три: один общий (base.html), который будут наследовать два других. Первый (index.html) – для вывода списка основных записей или тем (topics), второй (topic.html) – для вывода какой-то определенной темы и списка вложенных в нее заметок/записей (entries).

Для добавления нужных шаблонов (templates) в папке приложения logs (LOG/logs) создается папка templates, а внутри нее – папка с названием приложения logs. Затем внутри папки logs создаются три файла: base.html, index.html и topic.html. Содержимое файлов представляет html-код с включениями Django-директив.

Файл base.html выглядит следующим образом: вначале выводится ссылка на индексную страницу сайта, обратите внимание, чтобы создать ссылку здесь используется имя маршрута index, которое указано ранее в файле urls.py. Между Django-директивами {% block content %} и {% endblock content %} будет включено содержимое из двух других шаблонов: index.html и topic.html. Сам код имеет вид

```
<p><a href="{% url 'logs:index' %}">
        Журнал записей
</a></p>
{% block content %}{% endblock content %}
```

Код index.html.

```
{% extends 'logs/base.html' %}
{% block content %}
        <h3>Темы:</h3>
        {% for topic in topics %}
                <p>
```

```
                    <strong>
                        <a href="/topics/{{ topic.id }}/">
                            {{topic.text}}
                        </a>
                    </strong>
                    (Дата создания: {{topic.date_added|date:'H:i d/m/Y'}})
                </p>
            {% empty %}
                <p>Пока нет тем</p>
            {% endfor %}
{% endblock content %}
```

Код topic.html выглядит следующим образом.

```
{% extends 'logs/base.html' %}
{% block content %}
    <h3>Тема: {{topic}}</h3>
    {% for entry in entries %}
        <p>
            {{entry.text|linebreaks}}
            (Дата создания: {{entry.date_added|date:'H:i d/m/Y'}})
        </p>
    {% empty %}
        <p>Пока нет записей</p>
    {% endfor %}
{% endblock content %}
```

Итоговый интерфейс по пути http://127.0.0.1:8000/ приведен на рис. 13.

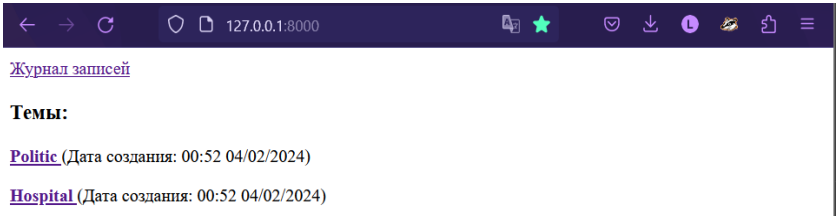

Рис. 13. Итоговый интерфейс проекта

Вопросы к подразд. 2.3

1. Что такое Django?
2. Как создать новый проект в Django с использованием django-admin?
3. Что такое Django ORM?
4. Как создать новое приложение внутри Django проекта с использованием manage.py?
5. Как запустить встроенный сервер Django?
6. Что представляет собой файл settings.py в Django-проекте?
7. Как добавить новую модель данных в Django-приложении?
8. Как применить миграции в Django?
9. Как создать административный пользователь в Django?
10. Как создать URL-маршрут в Django?
11. Что такое представление (view) в Django и как его создать?

Практическое задание к подразд. 2.3

На основе разобранного примера требуется сделать собственный новостной портал с иным интерфейсом (отображением) новостей, добавить категории постов.

Библиографический список

1. *Пол Д.*, Харви Д. Python: Искусственный интеллект, большие данные и облачные вычисления. СПб. : Питер, 2020. 864 с.

2. *Федоров Д. Ю.* Программирование на языке высокого уровня Python: учебное пособие для прикладного бакалавриата. М. : Изд-во Юрайт, 2017. 126 с.

3. *Beazley D.* Python Essential Reference. 4th Edition, Addison-Wesley Professional, 2009. 717 с.

4. *Forcier J.*, Bissex P., Chun W.J. Python web development with Django / Addison-Wesley Professional, 2008. 405 с.

ИЕРАРХИЯ КЛАССОВ ДЛЯ ИСКЛЮЧЕНИЙ

Иерархия классов для встроенных исключений в Python выглядит следующим образом:

```
BaseException
  SystemExit
  KeyboardInterrupt
  GeneratorExit
  Exception
    ArithmeticError
    AssertionError
    ...
    ...
    ...
    ValueError
    Warning
```

Все исключения в Python наследуются от базового BaseException:

• SystemExit – системное исключение, вызываемое функцией sys.exit() во время выхода из приложения;

• KeyboardInterrupt – системное исключение, которое возникает при завершении программы пользователем;

• GeneratorExit – системное исключение, которое вызывается методом close объекта generator;

• Exception – исключения, которые можно и нужно обрабатывать.

От Exception наследуются:

1) StopIteration – исключение, которое вызывается функцией next в том случае, если в итераторе закончились элементы;

2) ArithmeticError – ошибка, возникающая при вычислении, бывают следующие типы:

• FloatingPointError – ошибка при выполнении вычислений с плавающей точкой;

• OverflowError – ошибка, указывающая на большой результат вычислений для текущего представления (не появляется при операциях с целыми числами, но может появиться в некоторых других случаях);

• ZeroDivisionError – ошибка, которая возникает при попытке деления на ноль;

3) AssertionError – ошибка, указывающая на то, что выражение, используемое в функции assert, неверно;

4) AttributeError – ошибка, которая означает, что у объекта отсутствует нужный атрибут;

5) BufferError – ошибка, указывающая, что операция, для выполнения которой требуется буфер, не выполнена;

6) EOFError – ошибка чтения из файла;

7) ImportError – ошибка импортирования модуля;

8) LookupError – ошибка неверного индекса, делится на два типа:

• IndexError – ошибка, указывающая на то, что индекс выходит за пределы диапазона элементов;

• KeyError – ошибка, указывающая на то, что индекс отсутствует (для словарей, множеств и подобных объектов);

9) MemoryError – ошибка переполнения памяти;

10) NameError – ошибка отсутствия переменной с данным именем;

11) OSError – исключения, генерируемые операционной системой:

• ChildProcessError – ошибка, связанная с выполнением дочернего процесса;

• ConnectionError – исключение, связанное с подключениями (BrokenPipeError, ConnectionResetError, ConnectionRefusedError, ConnectionAbortedError);

• FileExistsError – исключение, которое возникает при попытке создания уже существующего файла или директории;

• FileNotFoundError – исключение, которое генерируется при попытке обращения к несуществующему файлу;

• InterruptedError – исключение, которое возникает в том случае, если системный вызов был прерван внешним сигналом;

• IsADirectoryError – исключение, которое возникает, если программа обращается к файлу, а это директория;

• NotADirectoryError – исключение, которое возникает, если приложение обращается к директории, а это файл;

• PermissionError – исключение, которое возникает, если прав доступа недостаточно для выполнения операции;

• ProcessLookupError – исключение, которое возникает, если процесс, к которому обращается приложение, не запущен или отсутствует;

• TimeoutError – исключение, которое возникает, если время ожидания истекло;

12) ReferenceError – ошибка, которая возникает, если осуществлена попытка доступа к объекту с помощью слабой ссылки, когда объект не существует;

13) RuntimeError – ошибка, которая генерируется в случае, когда исключение не может быть классифицировано или не подпадает под любую другую категорию;

14) NotImplementedError – ошибка, которая генерируется в случае, если абстрактные методы класса нуждаются в переопределении;

15) SyntaxError – ошибка синтаксиса;

16) SystemError – внутренняя ошибка;

17) TypeError – ошибка, указывающая на то, что операция не может быть выполнена с переменной этого типа;

18) ValueError – ошибка, которая возникает, когда в функцию передается объект правильного типа, но имеющий некорректное значение;

19) UnicodeError – исключение, связанное с кодирование текста в Unicode, бывает трех видов:

• UnicodeEncodeError – ошибка кодирования;

• UnicodeDecodeError – ошибка декодирования;

• UnicodeTranslateError – ошибка перевода *Unicode*;

20) Warning – предупреждение, некритическая ошибка.

С О Д Е Р Ж А Н И Е

Лестенко Никита Александрович, Мамаев Иван Дмитриевич

Технология обработки данных
для предметно-ориентированных задач на языке Python

Редактор *Г.В. Никитина*
Корректор *Л.А. Петрова*
Компьютерная верстка: *Н.А. Андреева*
Подписано в печать 15.05.2024. Формат 60×84/16. Бумага документная.
Печать цифровая. Усл. печ. л. 3,6. Тираж 100 экз. Заказ № 113.
Издательство БГТУ «ВОЕНМЕХ» им. Д. Ф. Устинова.
190005, С.-Петербург, 1-я Красноармейская ул., д. 1