ГПОУ ТО «Тульский экономический колледж»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по организации, планированию и проведению практических работ, внеаудиторной самостоятельной работы по учебной дисциплине «Информатика» профессионального модуля «Аналитика и визуализация данных на Python» для студентов всех специальностей

для средних специальных учебных заведений

Щёкино 2024

Одобрена предметной (цикловой) комиссией №1 Председатель ПЦК №1 _____ М.И.Хейфец «_____»_____2024г. Регистрационный № «_____»_____2024г. Методист _____Е.В. Яковлева	Утверждаю Заместитель директора по учебной работе _____Е.В. Кошелева «_____»_____2024г.

Авторы:
И.В.Васильева - преподаватель высшей категории ГПОУ ТО «Тульский экономический колледж»
Е.В.Темерева – преподаватель ГПОУ ТО «Тульский экономический колледж»

Преподаватели подготовили материал, методические рекомендации по выполнению практических работ по теме «Язык программирования Python».

В работе представлен разнообразный материал теоретических и практических вопросов и заданий, который применяется при изучении дисциплины «Информатика» профессионального модуля «Аналитика и визуализация данных на Python».

Данный материал работы необходим студенту, так как показаны приёмы, методы и формы самостоятельного изучения материала.

Содержание

Тема 1.1. Введение в ЯЗЫК программирования Python

Практическая работа № 1

Интерактивная среда программирование на Python. Ввод и вывод данных. Функции print(), input(). Типы данных. Математические операции с целыми и вещественными числами.

Hello, world! Установка и настройка среды

Цель этого занятия – установка, настройка среды и написание первой программы. Приступим к установке. Убедитесь, что на вашем компьютере есть 25 Гб свободного места. Изучить типы данных и операции. Изучить основы работы с переменными и операции над ними при работе с языком Python.

Для начала найдём и установим PyCharm – эта программа понадобится для работы на языке Python.

Важно!

PyCharm – кроссплатформенная среда, позволяющая работать на объектно-ориентированном языке программирования Python.

1 часть.

Загрузка PyCharm

Скачать PyCharm можно на официальном сайте программы по этой ссылке – https://pycharm-community-edition.softonic.ru/. Данный сайт можно легко найти в интернете. Для этого откроем любой браузер, например, Google (рис. 1.1) и введём в поисковую строку запрос «PyCharm скачать». Далее перейдём на первую полученную вкладку официального сайта JetBrains (рис. 1.2).

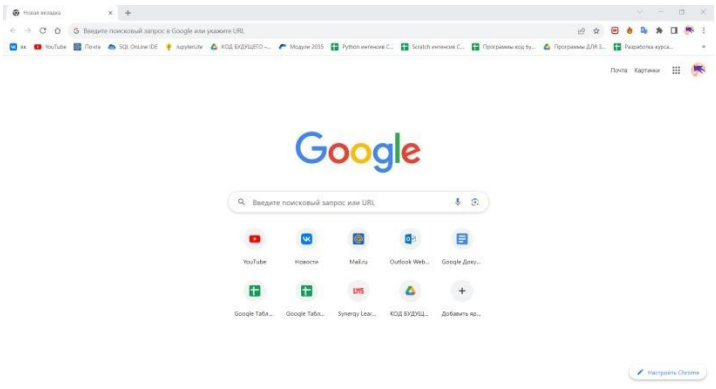


Рис. 1.1. Открываем любой браузер для выполнения запроса

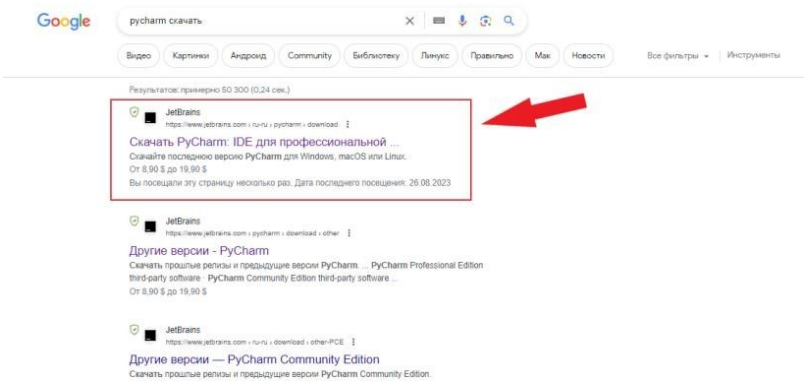


Рис. 1.2. Переходим по ссылке на официальный сайт JetBrains

После перехода откроется официальный сайт JetBrains. На данной странице размещены ссылки на загрузку установочного файла (Windows, macOS). Если операционная система – Windows, достаточно нажать кнопку «Скачать» для установки версии «Professional» (рис. 1.3, п. 1). Для установки версии «Community Edition» достаточно нажать на кнопку «Скачать» (рис. 1.3, п. 2). Рекомендуется скачивать версию «Community Edition».

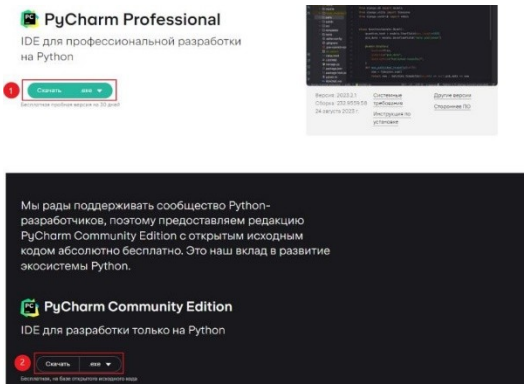


Рис. 1.3. Загрузка установочного файла для Windows

Далее нужно выбрать место сохранения установочного файла (рис. 1.4).

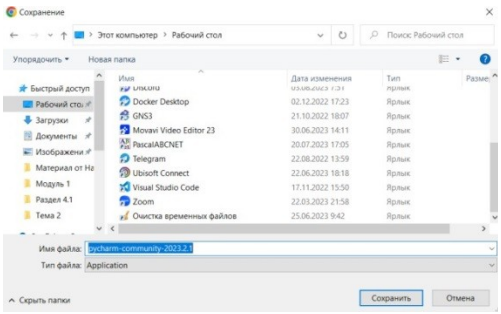


Рис. 1.4. Выбор директории для загрузки установочного файла

Для установки нужно нажать два раза левой кнопкой мыши (ЛКМ) по загруженному установочному файлу (рис. 1.5).

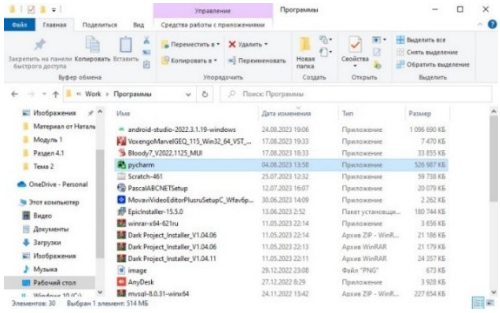


Рис. 1.5. Начало установки среды PyCharm

Установка и настройка PyCharm

Для начала нам нужно выбрать и установить среду PyCharm. Процесс установки интуитивен.

Важно!

Отладка – проверка работоспособности кода.

Компиляция – запуск и выполнение написанного кода.

После установки среды PyCharm пользователя встретит стартовое меню (рис. 1.6).

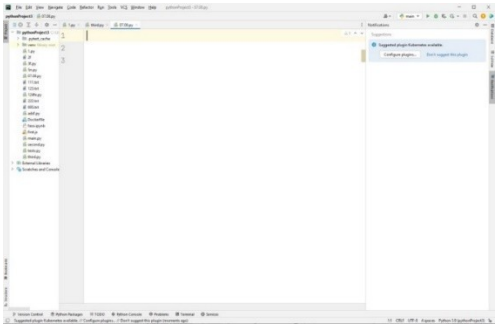


Рис. 1.6. Стартовое меню среды PyCharm

Для изменения размера шрифта нужно нажать на вкладку «File» (рис. 1.7).

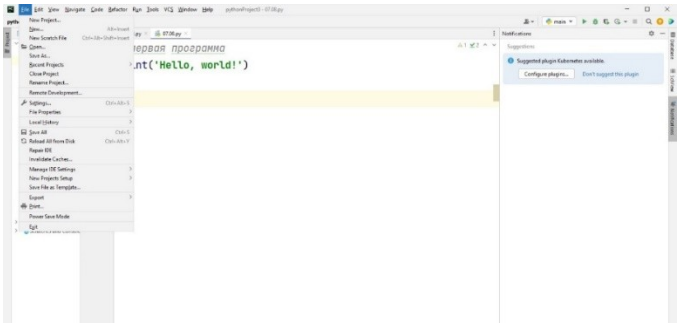


Рис. 1.7. Переход во вкладку «File»

Далее выбрать пункт «Settings» (рис. 1.8).

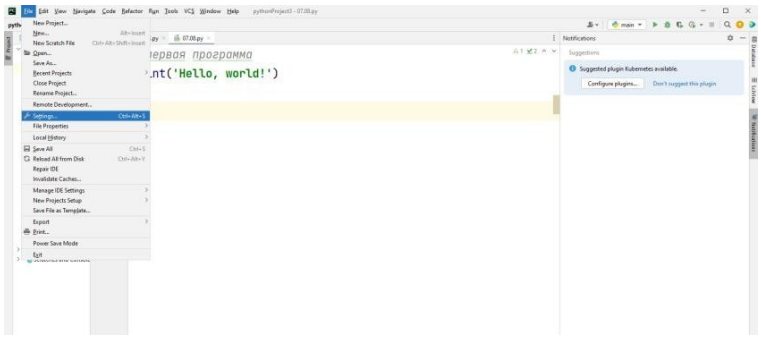


Рис. 1.8. Выбор пункта «Settings»

При переходе в данный пункт откроется диалоговое окно (рис. 1.9).

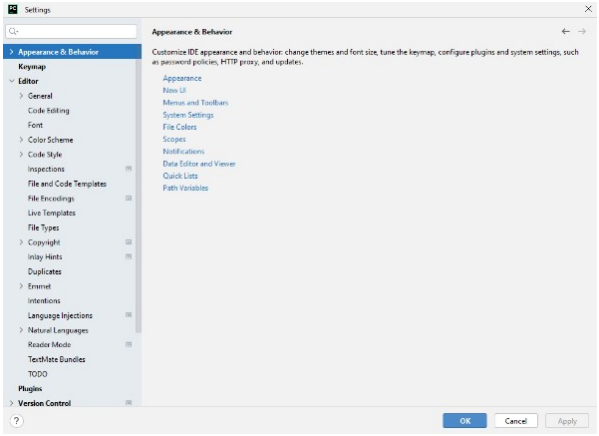


Рис. 1.9. Вид диалогового окна пункта «Settings»

Далее нужно перейти в подпункт «Editor» (рис. 1.10).

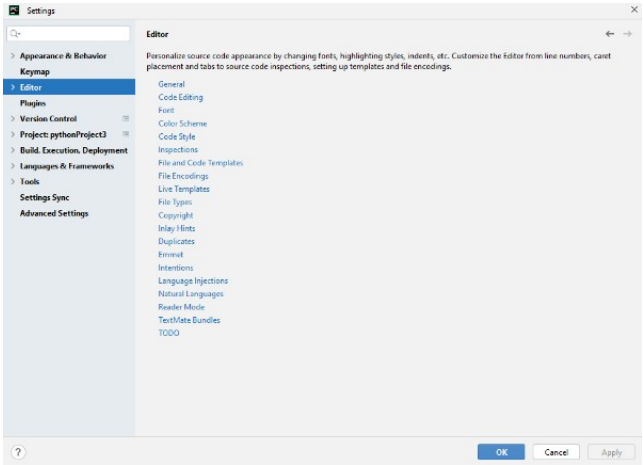


Рис. 1.10. Переход в подпункт «Editor»

Здесь необходимо выбрать раздел «Font» (рис. 1.11).

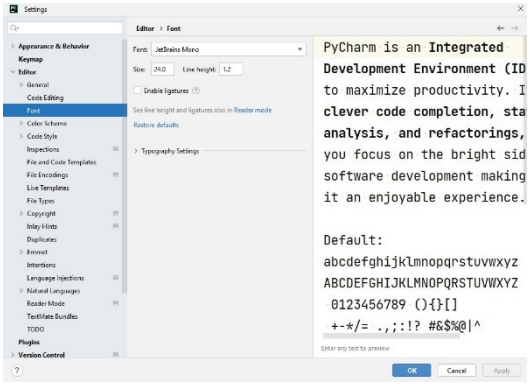


Рис 1.11. Содержимое раздела «Font»

Число возле надписи «Size» отвечает за размер шрифта программы (рис. 1.12).

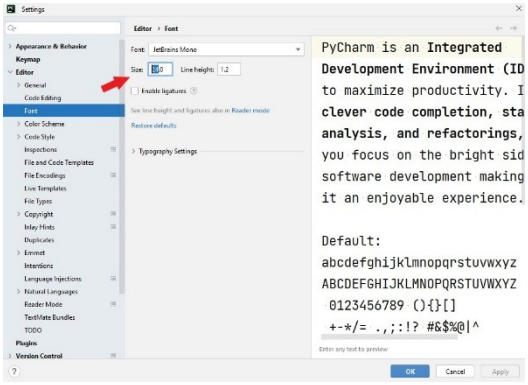


Рис. 1.12. Изменение размера шрифта в PyCharm

Создание первого проекта

Для создания проекта нужно перейти во вкладку «File» (рис. 1.13).

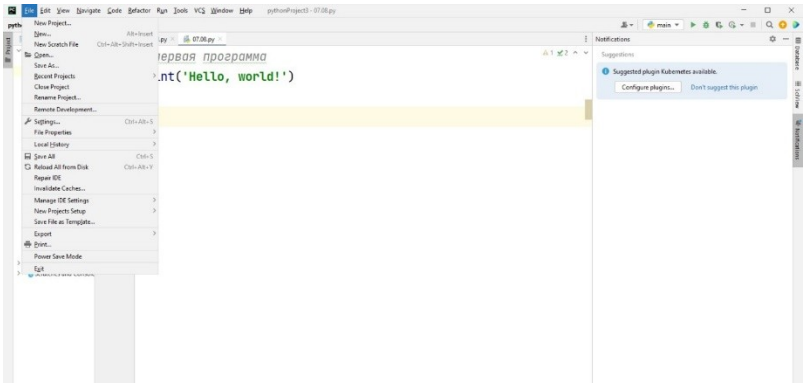


Рис. 1.13. Переход во вкладку «File»

Здесь нужно выбрать подраздел «New» (рис. 1.14).

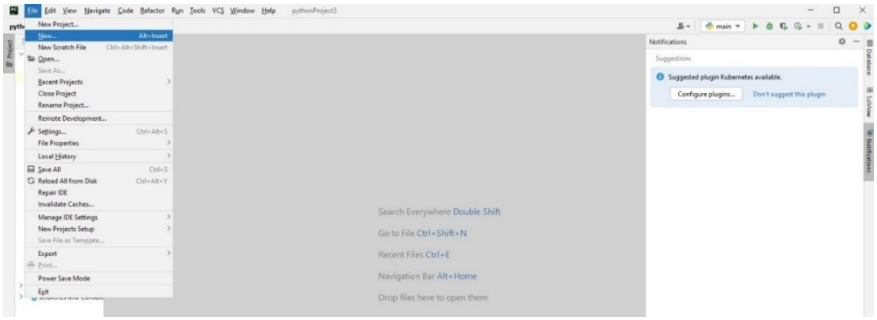


Рис. 1.14. Переход в подраздел «New»

Откроется следующее меню (рис. 1.15).

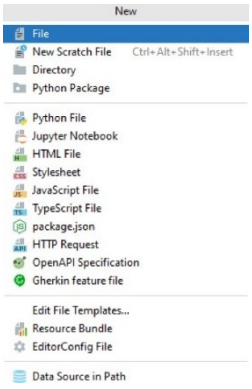


Рис. 1.15. Содержимое подраздела «New»

Здесь нужно выбрать «Python File» (рис. 1.16).

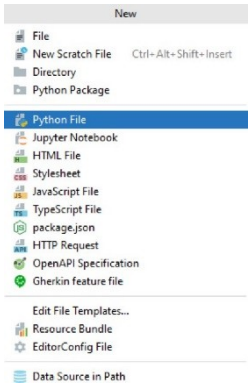


Рис. 1.16. Выбор формата файла для проекта

Нажмем левой кнопкой мыши по этому виду и дадим имя (рис. 1.17).

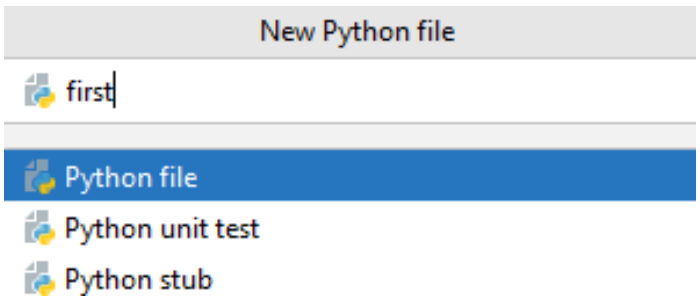


Рис. 1.17. Ввод имени файла

Нажмем кнопку «Enter», данный файл появится в рабочей области среды (рис. 1.18).

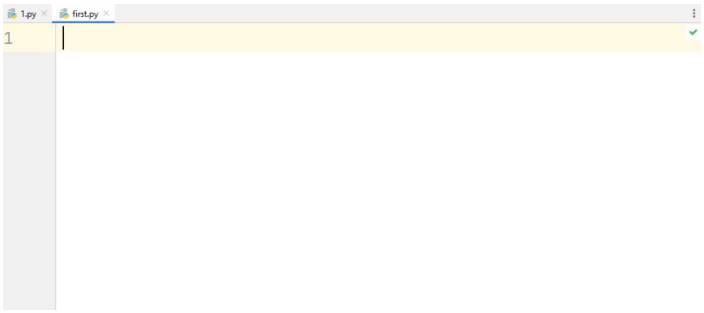


Рис. 1.18. Расположение нового файла в рабочей области среды PyCharm

Знакомство с интерфейсом

Создание нового проекта окончено, теперь давайте познакомимся с интерфейсом PyCharm поближе. Все окно можно разделить на 5 основных частей (рис. 1.19). В первой части находится **панель закладок**, во второй части – **рабочая область среды**, в третьей – **дерево исполнительных файлов**, в четвертой – **область уведомлений**, в пятой – **набор инструментов**.

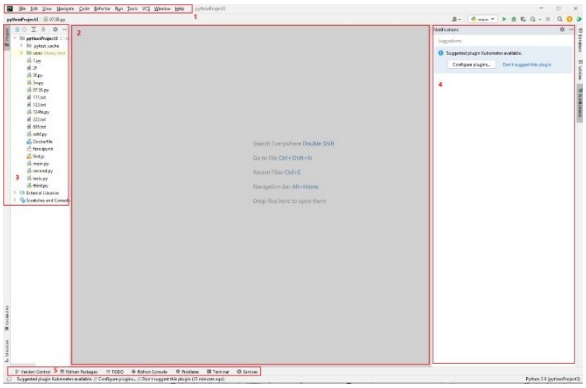


Рис. 1.19. Области взаимодействия среды PyCharm

Напишем нашу первую программу с выводом простого сообщения на консоль (рис. 1.20).

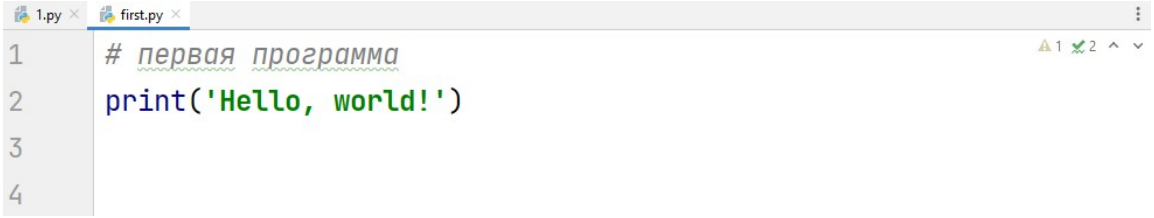


Рис. 1.20. Листинг первой программы

Для запуска программы нужно нажать на данную кнопку (рис. 1.21).

Рис. 1.21. Вид кнопки для запуска программы

Посмотрим на вывод программы (рис. 1.22).

```
# первая программа
print('Hello, world!')
```

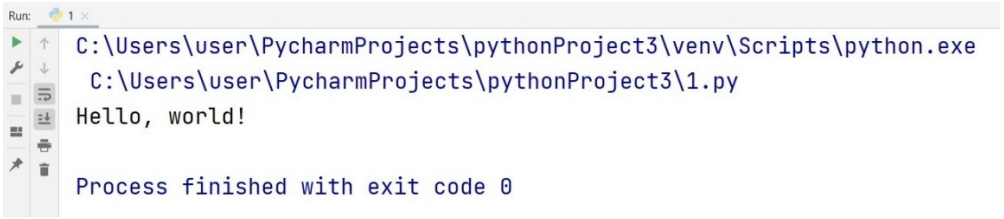


Рис. 1.22. Вывод консоли

Комментарии в коде очень важны. Они используются для того, чтобы на человеческом языке сообщить информацию, а также отключить (закомментировать) части кода программы, если нужно, чтобы они временно не работали. Комментарии в программе можно добавлять при помощи символа # (рис. 1.23).

Рис. 1.23. Добавление комментариев в программу

При помощи команды **input()** можно вводить информацию с клавиатуры. Посмотрим на пример работы такой программы (рис. 1.24).

```
# ввод слова
word = input('Введите слово: ')
# вывод слова
print(word)
```

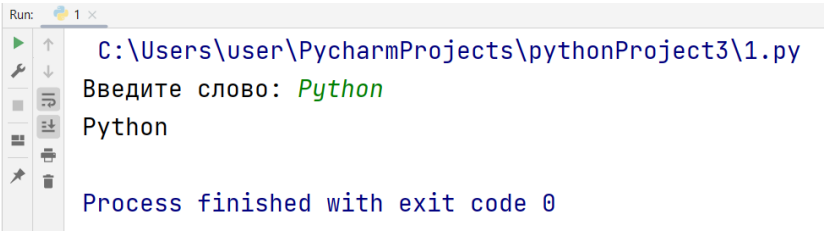


Рис. 1.24. Пример работы программы с вводом данных с клавиатуры

При помощи оператора **“+”** можно слитно выводить несколько строк (рис. 1.25).

```
# использование оператора +
print('Аква' + 'парк')
```

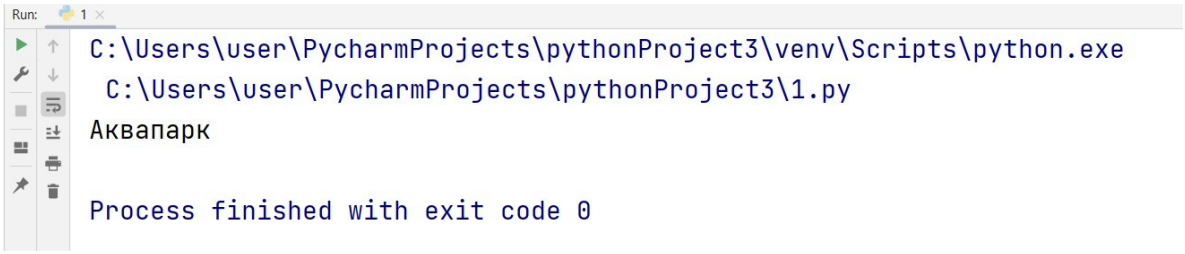


Рис. 1.25. Пример использования оператора “+”

Важно!

Программа – последовательность действий или операций, которая приводит к решению конкретной задачи.

Программирование – процесс создания компьютерных программ.

Переменная – область памяти, которая предназначена для хранения данных (значений).

Операция – конструкция, аналогичная по записи математическим операциями, специальный способ записи некоторых действий.

Контрольные вопросы.

1. Какую роль выполняет команда print() в языке Python?
2. Что такое программа?
3. Как можно передавать данные программе?
4. Какую роль выполняет команда input() в языке Python?
5. Какое максимальное количество аргументов можно передать программе на языке Python за один раз?

2 часть. Типы данных и операции

Типы данных и простые команды

Основные типы данных, которыми можно пользоваться на языке Python:

Числа:

- **int** – целые числа;
- **long** – целые числа произвольной точности;
- **bool** – логические числа;
- **float** – числа с плавающей точкой;
- **complex** – комплексные числа.

Последовательности:

- **str** – строки;
- **dict** – словари.

Тип данных **int** отвечает за целые числа, команда **type()** позволяет узнать тип данных (рис. 1.26).

```
# ввод данных
c = 7
d = 13
# вывод данных
Print ('Переменная "c":', c)
print('Переменная "d":', d)
print('Тип данных переменной "c":', type(c))
print('Тип данных переменной "d":', type(d))
```

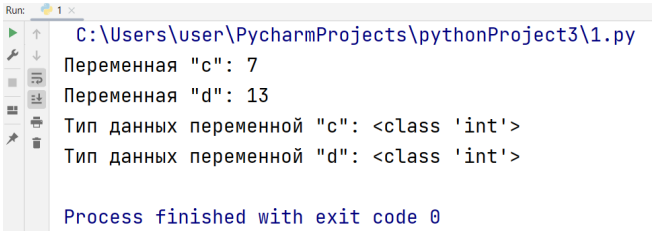


Рис. 1.26. Пример использования типа данных ‘int’ и команды ‘type()’

Тип данных **float** отвечает за числа с дробной частью (рис. 1.27.).

```
# ввод данных
c = 7.12
d = 4.1243
```

 # вывод данных

print(**'Переменная "c":'**, c)

print(**'Переменная "d":'**, d)

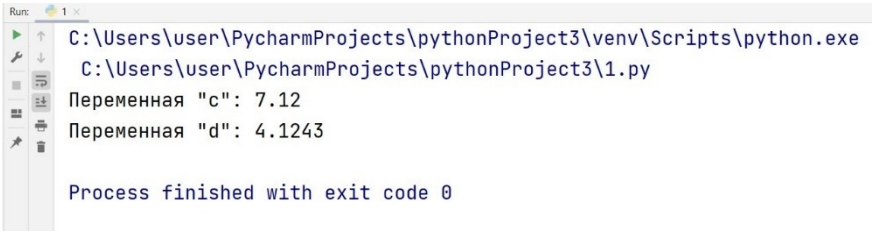


Рис. 1.27. Пример использования типа данных ‘float’

Команда **bool()** имеет два значения: **False**, если переменная равна нулю, и **True** (в других случаях) (рис. 1.28).

 # ввод данных

c = 6

d = 0

 # вывод данных

print(**'Применение команды bool() к "c":'**, bool(c))

print(**'Применение команды bool() к "d":'**, bool(d))

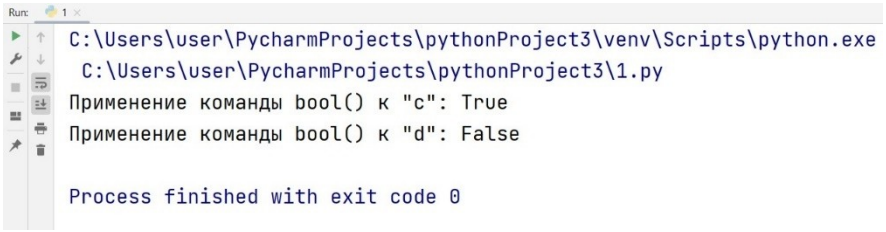


Рис. 1.28. Пример использования команды ‘bool()’

Команда **bool()** может применяться в тех ситуациях, когда необходимо подключить логику к программе. Это могут быть простые логические выражения или сложные конструкции, объединенные в блоки.

Команда **chr()** переводит число в знак таблицы Unicode (рис. 1.29).

 # ввод данных

c = 123

d = 246

 # вывод данных

print(**'Применение команды chr() к "c":'**, chr(c))

print(**'Применение команды chr() к "d":'**, chr(d))

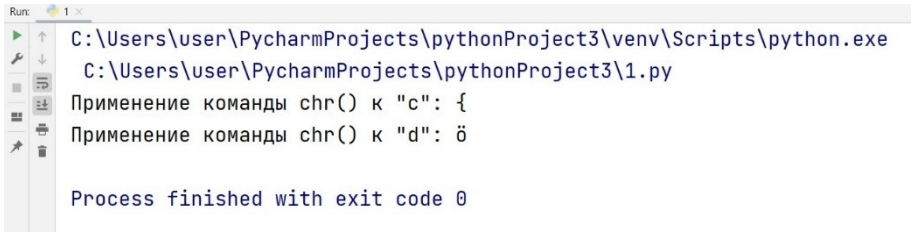


Рис. 1.30. Пример использования команды ‘chr()’

Тип данных **str** отвечает за строки, команда **list()** преобразует последовательность с разным содержимым в список (рис. 1.31).

 # ввод данных

first_one = [1, 3, 5, 7]

second_one = list(**'abc'**)

 # вывод данных

print(**'Переменная "first_one":'**, first_one)

print(**'Переменная "second_one":'**, second_one)

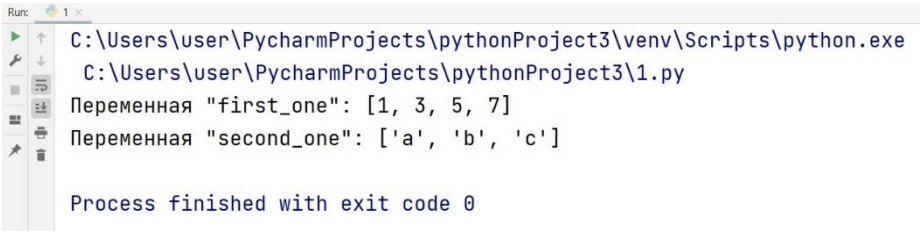


Рис. 1.31. Пример использования типа данных «str» и команды «list()

Словари позволяют хранить информацию в парах (рис. 1.32.).

 # ввод словаря

phonebook = {

"Mark": 1243,

"John": 8723,

"Alex": 7234

}

 # вывод данных

print(**'Словарь:'**, phonebook)

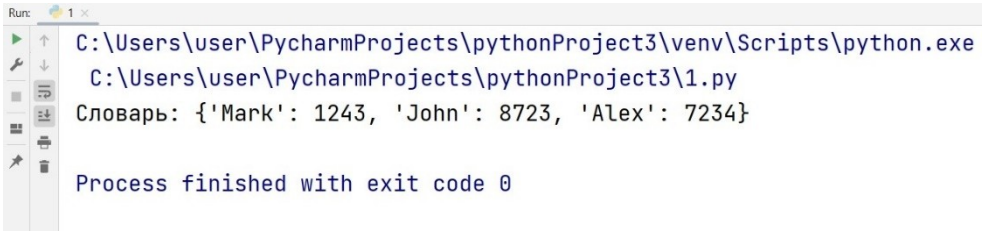


Рис. 1.32. Пример использования словаря

При помощи такой команды можно вывести значение одной строки словаря (рис. 1.33).

```
# ввод словаря
phonebook = {
"Mark": 1243,
"John": 8723,
"Alex": 7234
}
# вывод данных
print('Словарь:', phonebook)
print('Первое значение:', phonebook["Mark"])
print('Второе значение:', phonebook["John"])
```

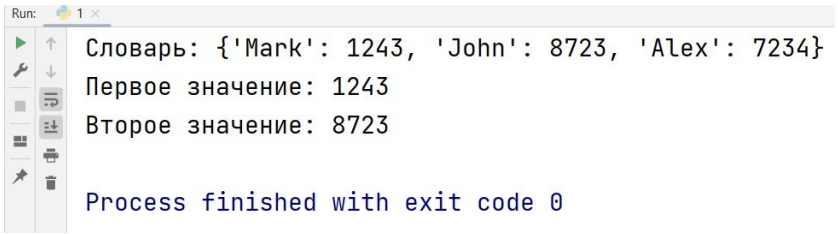


Рис. 1.33. Пример использования команды по выводу части информации

При помощи такой команды можно добавить пару значений в словарь (рис. 1.34).

```
# ввод словаря
phonebook = {
"Mark": 1243,
"John": 8723,
"Alex": 7234
}
# вывод данных
print('Словарь до изменения:', phonebook)
# добавление данных
```

phonebook[**"Andrew"**] = 6142

print(**'Словарь после изменения:'**, phonebook)

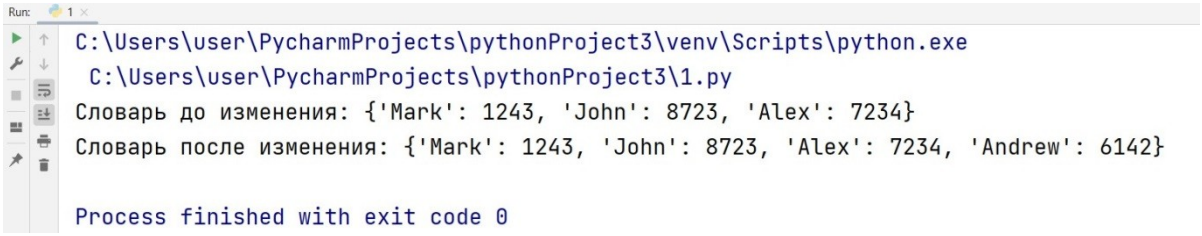


Рис. 1.34. Пример использования команды по добавлению информации в словарь

При помощи команды **update()** можно изменять значения в словаре (рис. 1.35).

 # ввод словаря

phonebook = {

"Mark": 1243,

"John": 8723,

"Alex": 7234

}

 # вывод словаря до изменения

print(**'Словарь до изменения:'**, phonebook)

 # изменение значений

phonebook.update (Mark = 777, Alex = 412)

print(**'Словарь после изменения:'**, phonebook)

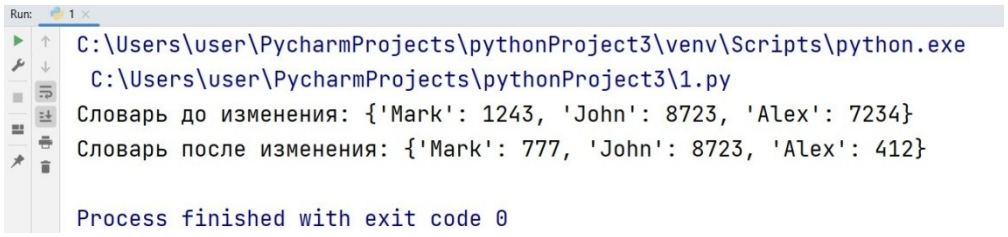


Рис. 1.35. Пример использования команды ‘update()’

При помощи команды **pop()** можно удалять значения из словаря (рис. 1.36).

 # ввод словаря

phonebook = {

"Mark": 1243,

"John": 8723,

"Alex": 7234

}

 # вывод словаря до изменения

print(**'Словарь до изменения:'**, phonebook)

 # удаление значений

phonebook.pop(**"Alex"**)

print(**'Словарь после изменения:'**, phonebook)

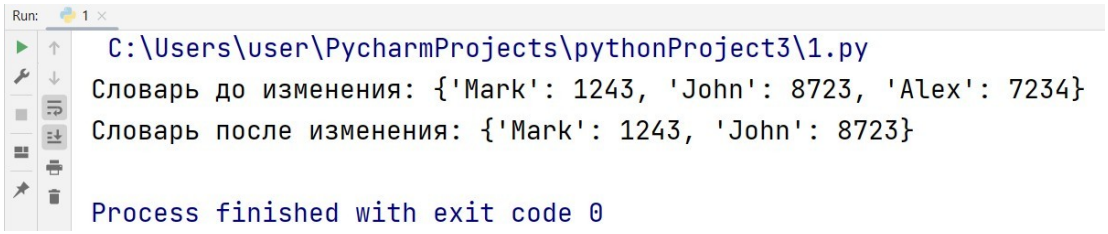


Рис. 1.36. Пример использования команды ‘pop()’

При помощи типа данных **dict** можно описывать словари. **Словари** – изменчивая структура данных для хранения пар **ключ** – **значение**, где значение определяется ключом. В качестве ключа могут выступать разные типы данных (числа, строки и т. п.).

Контрольные вопросы

1. Какую роль выполняет команда int в языке Python?
2. Что такое оператор?
3. Как можно передавать данные программе?
4. Какую роль выполняет функция float() в языке Python?
5. Какое минимальное количество аргументов можно передать программе на языке Python за один раз?

3 часть. Простые операции с переменными.

Например, при помощи таких двух строк можно вывести на консоль одно слово или однострочное сообщение (рис. 1.37).

 # ввод данных

x = **'Мама и папа'**

 # вывод данных

print(**'Результат:'**, x)

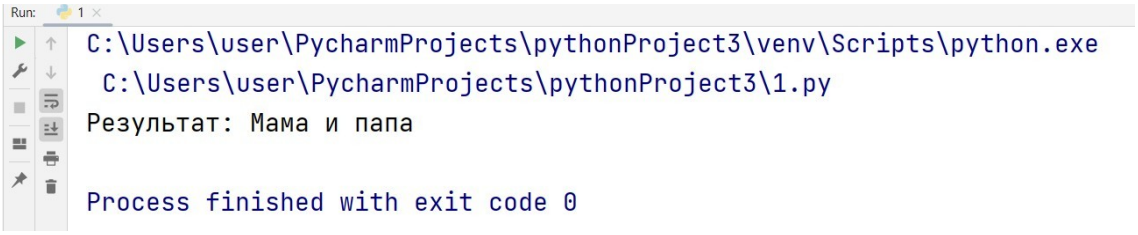


Рис. 1.37. Пример вывода значения переменной на консоль

В языке Python можно производить простые операции с числами (как с целыми, так и с вещественными) и давать названия переменным, в которых уже что-то было записано (рис. 1.38, рис. 1.39).

```
# ввод переменных
first = 100
second = first * 1.3
third = second * 2
# вывод данных
print('Переменная "first":', first)
print('Переменная "second":', second)
second) print('Переменная "third":', third)
third)
```

```
# ввод переменных
first = 100
second = first + 1.3
third = second + 2
# вывод данных
print('Переменная "first":', first)
print('Переменная "second":',
print('Переменная "third":',
```

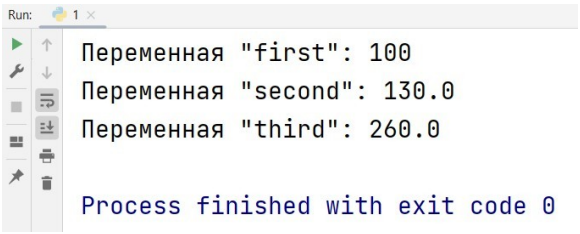


Рис. 1.38. Использование простых операций

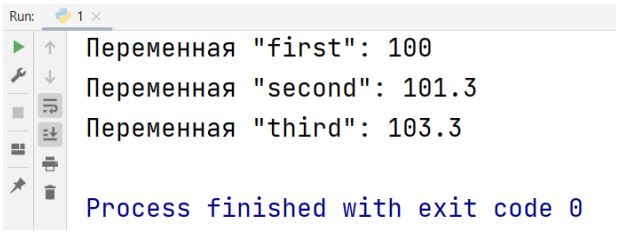


Рис. 1.39. Использование простых операций

При помощи особого форматированного вывода можно добавлять значение переменной в строку (рис. 1.40).

```
# ввод переменной
t = 3
# вывод строки
print('Это наше занятие №{}!'.format(t))
```

Рис. 1.40. Пример использования форматированного вывода

Посмотрим на пример использования операторов умножения, возведения в степень и получения остатка от деления (рис. 1.41).

```
# операция умножения
a = 2 * 5
```

```
# операция возведения в степень
b = 3 ** 3
# операция получения остатка от деления
c = 74 % 7
# вывод результатов
print('Переменная "a":', a)
print('Переменная "b":', b)
print('Переменная "c":', c)
```

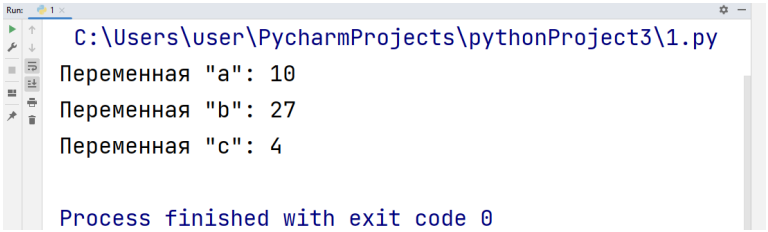


Рис. 1.41. Пример использования математических операторов

Порядок выполнения команд

Рассмотрим примеры использования сложных операторов присваивания в программах ‘+=’ и “-=” (рис. 1.42, рис. 1.43).

```
# объявление переменной 'y'
y = 10
print('Переменная "y" до изменения:', y)
# оператор '+='
y += 20
print('Переменная "y" после изменения:', y)
```

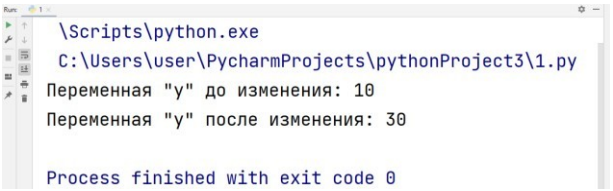


Рис. 1.42 Использование оператора «+=»

```
# объявление переменной 'y'
y = 10
print('Переменная "y" до изменения:', y)
# оператор '-='
y -= 4
print('Переменная "y" после изменения:', y)
```

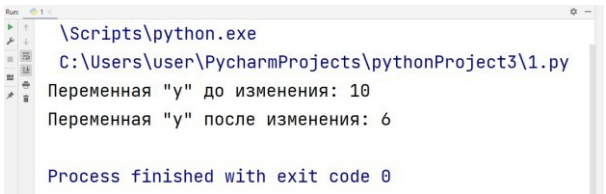


Рис. 1.43 Использование оператора «-=»

Бывают такие ситуации, когда необходимо выполнять ввод данных с клавиатуры и выводить данные на консоль (рис. 1.44).

```
# ввод переменных
a = int(input('Введите переменную "a": '))
c = int(input('Введите переменную "c": '))
# вывод данных
print('Переменная "a":', a)
print('Переменная "c":', c)
```

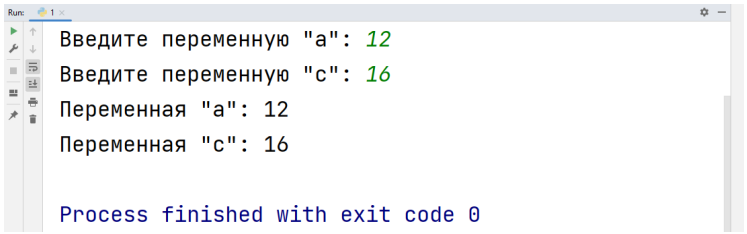


Рис. 1.44. Пример использования команды ‘input()’

Коммутативные операции

От перемены мест слагаемых, сумма не меняется. Так трактуется один из базовых законов арифметики – коммутативный закон. Данный закон встроен в язык программирования Python (рис. 1.45).

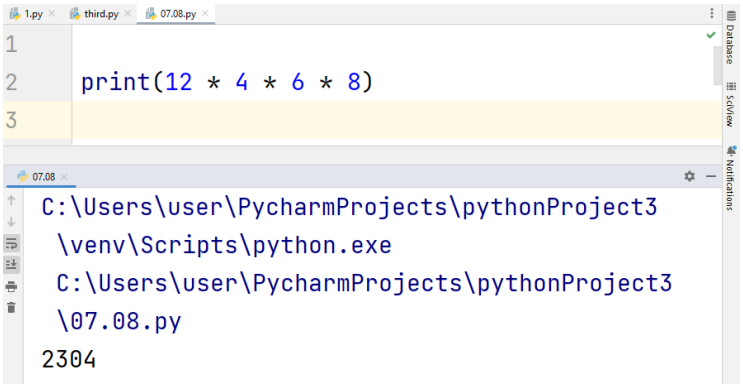


Рис. 1.45. Пример программы с использованием коммутативного закона

Математические операции

Рассмотрим еще один пример (рис. 1.46).

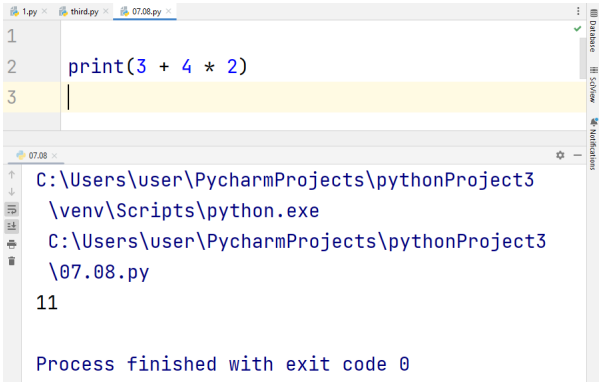


Рис. 1.46. Пример программы с использованием коммутативного закона

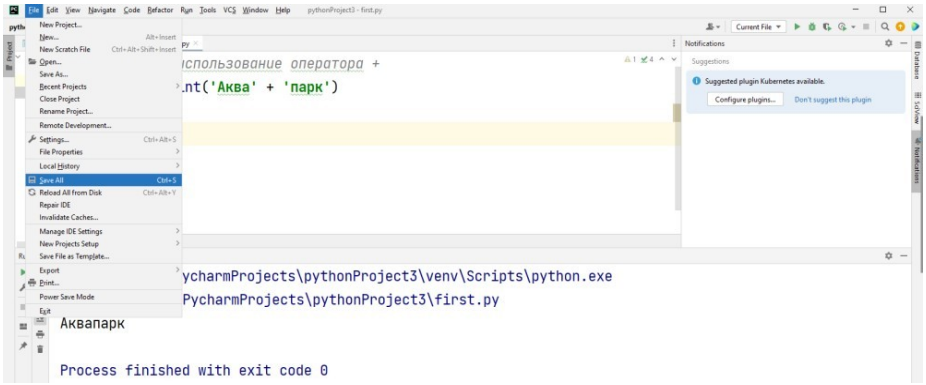


Рис. 1.47. Сохранение при закрытии проекта

Важно!

Оператор – элемент программы, позволяющий реализовывать математические и логические конструкции.

Контрольные вопросы.

1. Какую роль выполняет команда str в языке Python?
2. Что такое арифметическая операция?
3. Как можно передавать аргументы программе?
4. Какую роль выполняет функция dict() в языке Python?
5. Какое максимальное количество аргументов можно передать программе на языке Python за один раз?

Самостоятельная работа студента.

1. Составить программу вычисления значения функции $y(x) = x^2 - 7x + 8$ для заданного с клавиатуры значения аргумента x.

Указание. Примерный вид программы:

```
x=float(input("x= "))
y=x**2-7*x+8
```

```
print("y(x)= ", y)
```

2. Составить программу вычисления расстояния между двумя точками, заданными на плоскости своими координатами.

Указание. Примерный вид программы:

```
x1=float(input("x1=          "))
y1=float(input("y1=          "))
x2=float(input("x2=          "))
y2=float(input("y2= "))
r=((x1-x2)**2+(y1-y2)**2)**(1/2)
print("расстояние= ", r)
```

3. Составить программу вычисления суммы первых *n* членов арифметической прогрессии по любым двум её членам, номера которых известны.

Выводы

В ходе выполнения лабораторной работы вы получили представление о написании простых программ на языке программирования Python.

Контрольные вопросы

1. Какие основные операторы языка Python вы использовали при выполнении лабораторной работы?
2. Для чего используется оператор print при работе в языке Python?
3. Как можно возвести число в квадрат в языке Python?

Тема 1.2. Основные алгоритмические конструкции на Python

Практическая работа № 2

Понятие логических выражений и операций. Дизъюнкция, конъюнкция, отрицание. Таблица истинности. Проверка условия в Python. Синтаксис инструкций if, if-else, if- elif-else.

1 Часть.

Цель этого занятия – изучить логический тип данных.

Посмотрим на пример реализации оператора сравнения ‘==’ (рис. 2.1).

```
# использование оператора '=='
a = 10
b = 8 == 10
c = 10 == 10
# вывод данных
print('Переменная "a":', a)
print('Переменная "b":', b)
print('Переменная "c":', c)
```

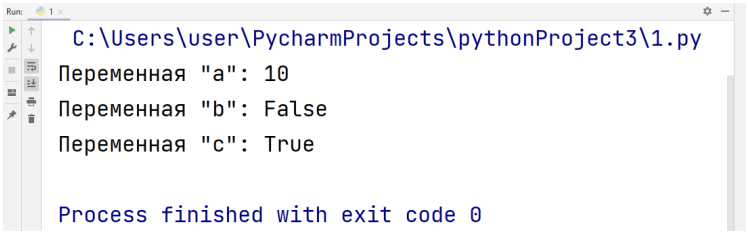


Рис. 2.1. Пример использования оператора сравнения ‘==’

Операторы способны выполнять несколько функций, например, сравнивать как переменные между собой, так и наборы переменных. В зависимости от типа и количества переменных, программа может решать различные задачи.

Оператор сравнения ‘!=’ работает с точностью наоборот (рис. 2.2).

```
# использование оператора '!='
a = 15
b = 8 != 10
c = 12 != 12
# вывод данных
print('Переменная "a":', a)
print('Переменная "b":', b)
print('Переменная "c":', c)
```

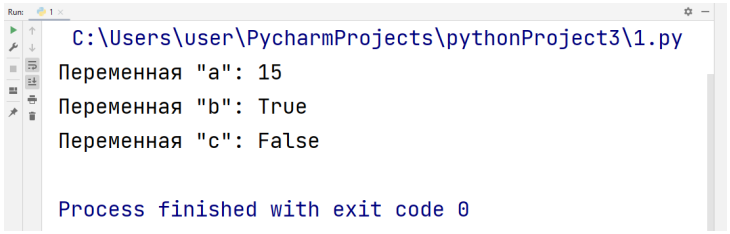


Рис. 2.2. Пример использования оператора сравнения ‘!=’

Посмотрим на реализацию операторов сравнения ‘>’ и ‘<’ (рис. 2.3).

```
# использование операторов '>' и '<'
a = 15
b = 8
# вывод данных
print('Переменная "a":', a)
print('Переменная  "b":',  b)
print('Переменная "a" больше, чем "b":', a>b)
print('Переменная "a" меньше, чем "b":', a<b)
```

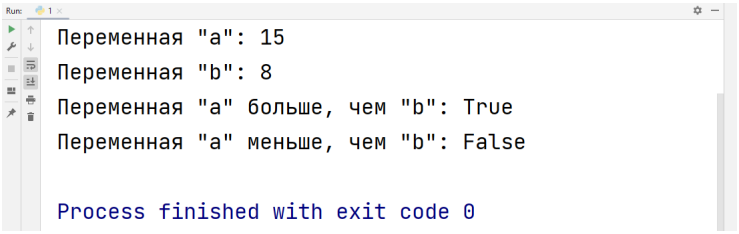


Рис. 2.3. Пример использования операторов сравнения ‘>’ и ‘<’

В языке Python могут применяться как строгие знаки, так и нестрогие (рис. 2.4).

```
# использование операторов '>=' и '<='
c = 13
d = 9
# вывод данных
print('Переменная "c":', c)
print('Переменная  "d":',  d)
print('Переменная "c" больше или равна переменной "d":', c>=d)
 print('Переменная "c" меньше или равна переменной "d":', c<=d)
```

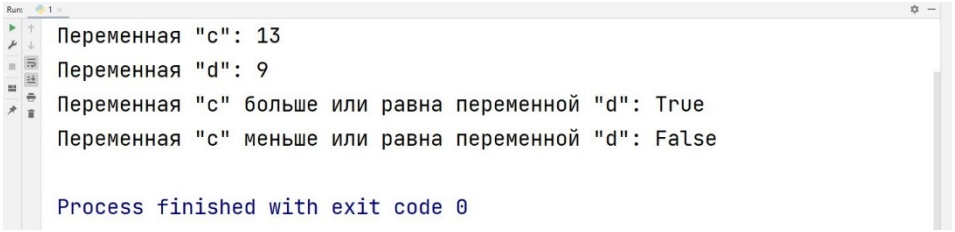


Рис. 2.4. Пример использования нестрогих операторов сравнения ‘>=’ и ‘<=’

Логические операции

Для создания составных условных выражений любой сложности применяются специальные операции, которые называются **логическими операциями**. Оператор **and (логическое умножение)** применяется для двух операндов (рис. 2.5).

```
# ввод данных
first = 23
second = 59
# использование лог. оператора 'and'
res = first > 20 and second == 59
# вывод данных
print('Переменная "first":', first)
print('Переменная "second":', second)
print('Переменная "res":', res)
```

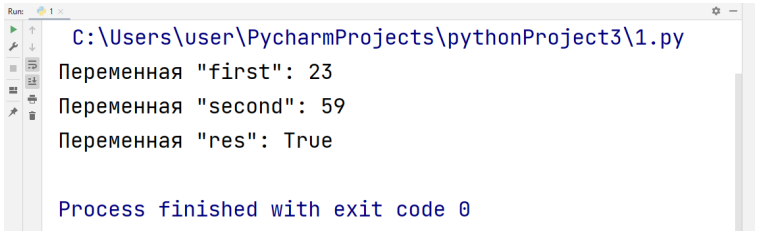


Рис. 2.5. Пример использования оператора ‘and’

Оператор **or (логическое сложение)** может применяться к двум операндам с одинаковым типом (рис. 2.6).

```
# ввод данных first = 23 second = False
# использование лог. оператора 'or'
res = first > 22 or second
# вывод данных
print('Переменная "first":', first)
print('Переменная "second":', second)
 print('Переменная "res":', res)
```

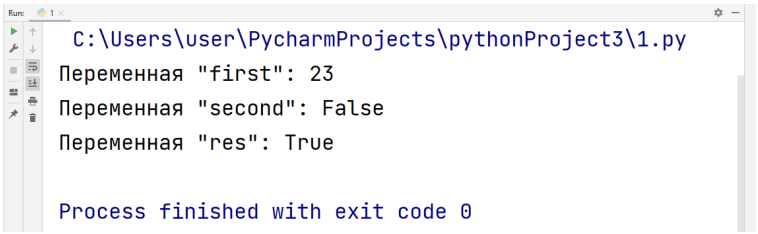


Рис. 2.6. Пример использования оператора ‘or’

Оператор **not (логическое отрицание)** возвращает значение **True**, если выражение равно **False** (данный оператор реализует операцию инверсии в языке Python) (рис. 2.7).

```
# ввод данных
first = 23 second = False
# использование лог. оператора 'not'
# вывод данных
print('Переменная "first":', first)
print('Переменная "second":', second)
print('Выражение "not 0":', not 0)
```

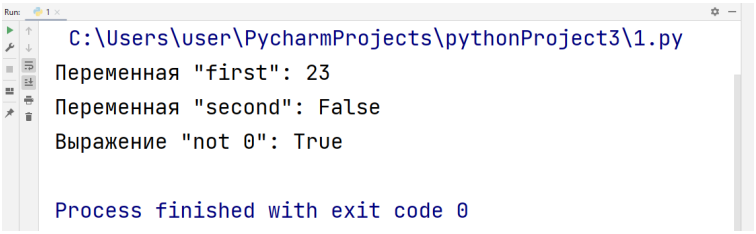


Рис. 2.7. Пример использования оператора ‘not’

Оператор **in** может проверить, входит ли подстрока (набор из меньшего числа символов, чем вся строка) в строку (рис. 2.8).

```
# ввод данных
mes = 'Python language'
first_op = 'Python'
second_op = 'SQL'
# использование оператора 'in'
# вывод данных
print('Строка "mes":', mes)
print('Входит ли "first_op":', first_op in mes)
print('Входит ли "second_op":', second_op in mes)
```

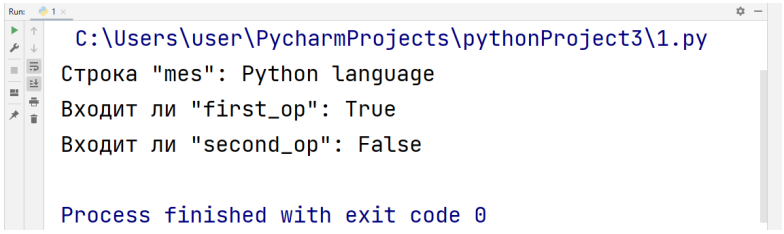


Рис. 2.8. Пример использования оператора ‘in’

Операторы if – else

В данном примере, если значение переменной **‘a’** больше, чем значение переменной **‘b’** (что является истиной), то выводится значение переменной **‘a’** (рис. 2.9.).

```
# ввод данных
a = 25
b = 15
# реализация оператора 'if'
    if a > b:
print('Переменная "a":', a)
```

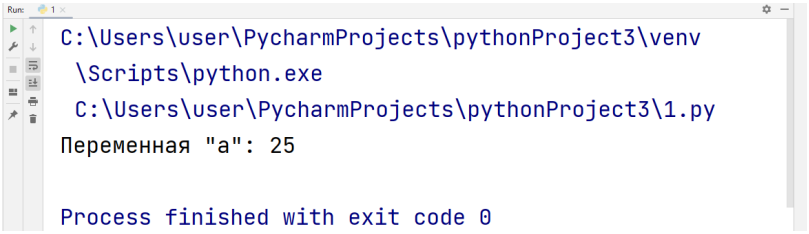


Рис. 2.8. Пример работы программы с операторами if – else

Добавим ветки **‘elif’** и **‘else’** (рис. 2.9.)

 # ввод данных

a = 5

b = 8

 # реализация операторов 'if', 'elif' и 'else'

if a > b: print(**'Переменная "a":'**, a)

elif a == b: print(**'Переменные равны!'**)

elif a**2 == 25: print(**'Значение переменной:'**, a)

else:

print(**'Переменная "b":'**, b)

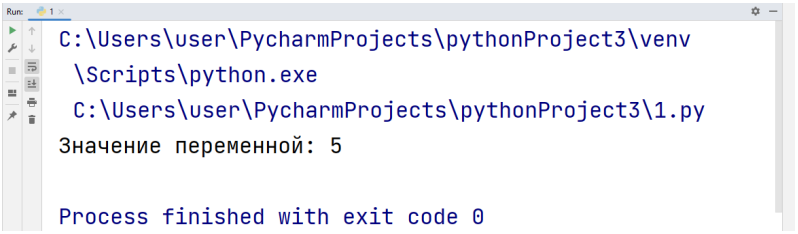


Рис. 2.9. Пример работы программы с операторами if – elif – else

Контрольные вопросы

1. Какую роль выполняет команда float в языке Python?
2. Что такое логическая операция?
3. Как можно передавать аргументы программе?
4. Какое минимальное количество аргументов можно передать программе на языке Python за один раз?

2 часть.

При решении задач важно реализовывать возможность выбора среди альтернативных операций на основе результатов проверки. В императивных языках программирования для этих целей используется оператор ветвления (условный оператор). В языке Python подобный оператор предусматривает возможность сделать выбор как из двух альтернативных ветвей программы, так и из трёх и более.

Общая форма условного оператора:

if <условие1>:

оператор1

elif <условие2>:

оператор2

else:

оператор3

Части else и elif являются необязательными. После части if указывается логическое условие, которое может быть истинным или ложным.

Как видно из описания условного оператора, он может содержать другие операторы, например операторы and (конъюнкция), or (дизъюнкция), not — отрицание, равенство ==, неравенство !=.

Также в Python можно записывать двойное условие, например 2<=a<=5, -10<v<=9. В том числе условный оператор может содержать внутри себя другой условный оператор.

Стоит обратить внимание на то, что после логического условия стоит двоеточие, для того чтобы показать, что далее идёт блок выражений. Блок выражений записывается с отступом.

Рассмотрим работу условного оператора более подробно. В самом простом случае оператор ветвления имеет вид:

if <условие>:

оператор1

Это неполная форма условного оператора.

Блок-схема работы данного оператора представлена на рисунке 2.10.

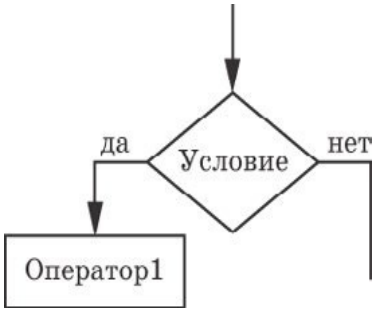


Рис. 2.10. Блок-схема неполного условного оператора

В случае истинности условия выполняется оператор1, а затем осуществляется выход из условного оператора (управление передаётся оператору, следующему за оператором if).

Пример 1

```
a=int(input())
if a<0:
   print('Ниже')
```

Результат работы программы представлен на рисунке 2.11.

Рис. 2.11. Результат работы программы

В данном примере в качестве условия используется сравнение a<0. Если это условие истинно, то на экран выводится текст «Ниже». Если же условие ложно, то программа ничего не выполняет.

Пример 2

```
a=int(input())
if (a<0) and (a>=-3):
    print('Ниже')
```

Результат работы программы представлен на рисунке 2.12.

Рис. 2.12. Результат работы программы

В данном примере условие составное: состоит из двух условий, объединённых операцией and (логическое «и»).

Рассмотрим далее более сложный вид оператора ветвления:

```
if <условие>:
    оператор1
else:
    оператор2
```

Обратите внимание на порядок отступов в формате оператора!

Это полная форма условного оператора.

Блок-схема работы данного оператора представлена на рисунке 2.13.

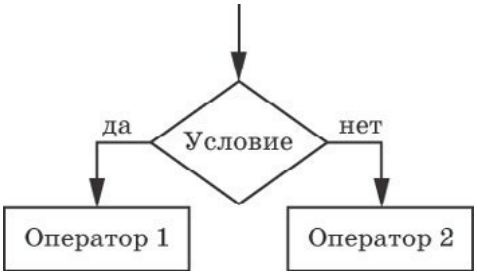


Рис. 2.13. Блок-схема полной формы условного оператора

Если условие истинно, то выполняется оператор1, в противном случае (если условие ложно), выполняется оператор2. Далее управление переходит к оператору, который следует за условным оператором.

Приведём примеры работы такой формы условного оператора.

Пример 3

```
a=int(input())
b=int(input())
if a+b>10:
    print('Yes')
else:
    print('No')
```

Результат работы программы представлен на рисунке 2.14.

Рис. 2.14. Результат работы программы

В данном примере в случае истинности условия a+b>10 выполняется оператор print('Yes'), в противном случае — print('No').

После условия и после части **else** можно указывать несколько операторов, но тогда все они записываются с отступом! Для сравнения рассмотрим два примера.

Пример 4.1

```
a=10
b=15
c=3
if a+b>10:
    print('Yes')
else:
     print('No')
     c=a+b
print(c)
```

Результат работы программы представлен на рисунке 2.15.

Рис 2.15. Результат работы программы

Пример 4.2

```
a=10
b=15
c=3
if a+b>10:
   print('Yes')
else:
   print('No')
   c=a+b
print(c)
```

Результат работы программы представлен на рисунке 2.16.

Рис 2.16. Результат работы программы

Как видим, результаты работы программ различаются, потому что в примере 4.2 оператор c=a+b выполняется в любом случае, так как стоит вне оператора ветвления.

Третья форма оператора ветвления выглядит следующим образом:

```
if <условие1>:
    оператор1
elif <условие2>:
    оператор2
else:
    оператор3
```

Блок-схема работы данного оператора представлена на рисунке 2.17.

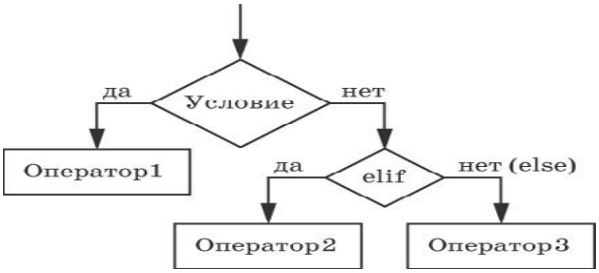


Рис. 2.17. Блок-схема условного оператора с двумя условиями

При использовании данной формы можно проводить проверку нескольких условий — после *if* и после *elif.* Оператор после *else* выполняется в том случае, если не выполнилось *условие2* после части *elif.*

Рассмотрим пример использования данной формы оператора *if.*

Пример 5

```
a=10
if a<-5:
   print('Yes')
elif -5<=a<=5:
    print('Maybe')
else:
    print('No')
```

Результат работы программы представлен на рисунке 2.18.

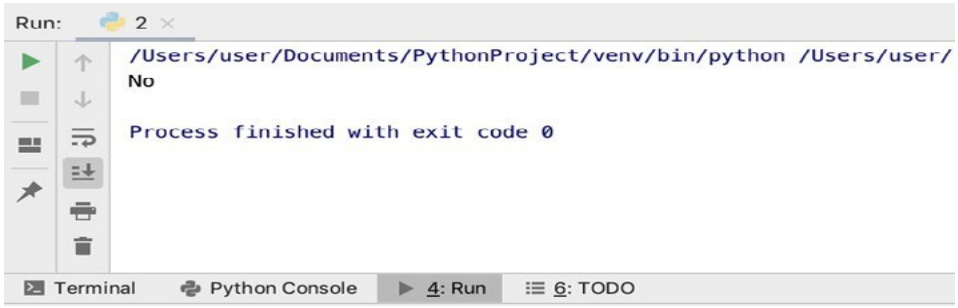


Рис. 2.18. Результат работы программы

Пример 6

Написать программу вычисления стоимости покупки с учётом скидки. Скидка в 3% предоставляется в том случае, если стоимость покупки превышает 1000 р., а скидка в 5% — если стоимость выше 3000 р.

```
s=int(input())
if s<1000:
   print(s)
elif 1000<=s<3000:
     print(s*0.97)
else:
      print(s*0.95)
```

Результат работы программы представлен на рисунке 2.19.

Рис. 2.19. Результат работы программы

Ещё раз обратимся к синтаксическим правилам языка Python.

В Python отсутствуют фигурные скобки (которые есть в языке C/C++) или разделители begin/end (которыми оперирует язык Pascal), окружающие блоки программного кода. Вместо этого принадлежность операторов к вложенному блоку определяется по величине отступов. Также операторы в языке Python обычно не завершаются точкой с запятой; признаком конца оператора служит конец строки с этим оператором.

Все составные операторы в языке Python оформляются одинаково: строка с заголовком завершается двоеточием, далее следуют вложенные операторы (один или более), обычно с отступом относительно заголовка. Эти операции с отступами называются блоком (или иногда набором).

if	
	Блок 1
elif	
	Блок 2
else	
	Блок 3

Рис.2.20. Схема вложенных блоков

Самостоятельная работа студента.

1. Написать программу вычисления оптимального веса: оптимальный вес вычисляется по формуле: рост (см) — 100.
2. Пользователю выдаётся рекомендация о снижении или наборе веса.

Указание. Примерный вид программы:

```
v=int(input("Ваш вес — "))
h=int(input("Ваш рост — "))
ideal_v=h-100
if v==ideal_v:
    print("Ваш вес в норме")
elif v>ideal_v:
     print("Ваш вес выше нормы! Необходимо сбросить вес")
else:
     print("Ваш вес ниже нормы! Необходимо набрать вес")
```

Результат работы программы представлен на рисунке 2.21.

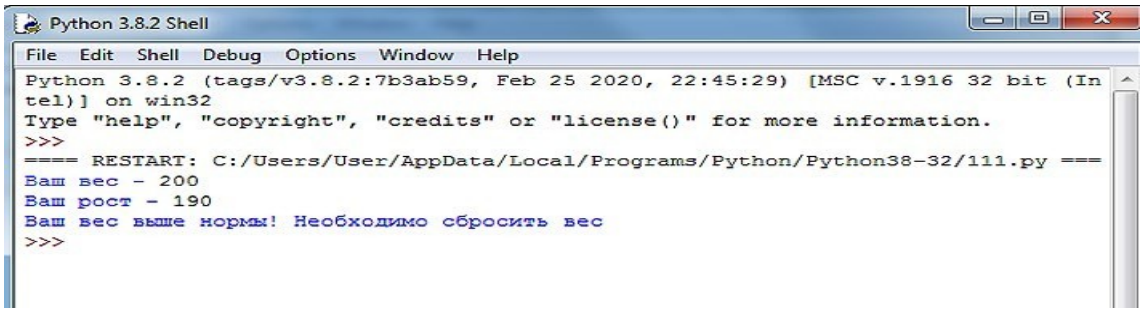


Рис. 2.21. Результат работы программы

3. Написать программу, определяющую вид треугольника (равносторонний, равнобедренный, разносторонний) по заданным длинам сторон.

4. Написать программу, которая считывает три вещественных числа и заменяет каж- дое чётное значение его частным от деления на 2, а единицу — числом 2.

5. Определить, является ли введённый пользователем год високосным.

Указание. Примерный вид программы:

```
y = int(input("Введите год "))
if (y%4==0 and y%100!=0) or (y%400==0):
    print("Високосный")
else:
    print("Обычный")
```

Написать программу, проверяющую, является ли введённое число чётным или оно нечётное.

Контрольные вопросы

1. Для чего используются условные операторы в программировании?
2. Как выглядит синтаксис условного оператора в языке Python?
3. Какие формы условного оператора можно использовать в языке Python?

3 Часть. Самостоятельная работа студента.

1. Среди трёх чисел найти среднее, т. е. которое больше одного числа, но меньше другого.
2. Даны длины трёх сторон одного треугольника и трех стороны другого треугольника. Определить, будут ли эти треугольники равновеликими, т. е. имеют ли они равные площади.
3. Ввести рост человека. Вывести на экран текст «ВЫСОКИЙ», если рост превышает 180 см, и «НЕ ОЧЕНЬ ВЫСОКИЙ» в противном случае.
4. Ввести 3 числа. Вывести их в порядке возрастания. (Пример: 12, 34, 56.)

Контрольные вопросы

1. Как выглядит полная форма оператора ветвления в языке Python?
2. Для чего используется часть else в операторе ветвления?
3. Какие основные операторы вы использовали при решении задач из лабораторной работы?

Тема 1.2. Основные алгоритмические конструкции на Python

Практическая работа № 3

Реализация циклических алгоритмов в Python. Функция range(). Синтаксис цикла for, цикла while.

Теоретическая часть

Справочник

Цикл в языке программирования представляет собой конструкцию, многократно выполняющую одну и ту же группу операторов. Число повторений (итераций) цикла может быть либо задано заранее, либо зависеть от истинности некоторого условия.

В реальной жизни постоянно применяются циклы, поэтому циклический алгоритм часто используются при решении задач по программированию.

В языке программирования Python может быть реализовано два вида цикла:

1) с предусловием — цикл *while*;
2) с параметром — цикл *for.*

Цикл *while* является часто используемым и универсальным циклом в Python. Полный формат данного цикла:

while <условие>:

<оператор1>

else:

<оператор2>

Часть *else* является необязательной. Блок-схема работы цикла *while* представлена на рисунке 2.22.

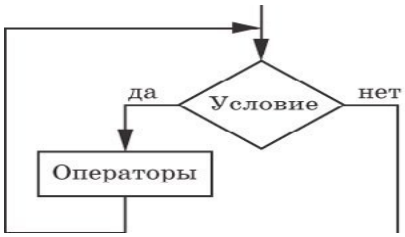


Рис. 2.22. Блок-схема цикла с предусловием while

Выполнение цикла *while* начинается с проверки условия. Если оно истинно (не равно *false*), выполняется оператор цикла. Если при первой же проверке выражение в условии равно *false*, цикл не выполнится ни разу. Если условие в цикле *while* никогда не станет ложным, то не будет причин остановки цикла и программа «зациклится». Чтобы этого не произошло, необходимо организовать момент выхода из цикла, т. е. ложность выражения в условии. Так,

например, изменяя значение какой-нибудь переменной в теле цикла, можно довести логическое выражение до ложности. Обратите внимание, что операторы тела цикла должны быть записаны с отступом.

Пример 1

```
i=5
while i<15:
print(i)
i+=2
```

Интерфейс программы PyCharm с введённой программой и результатом выполнения представлен на рисунке 2.23.

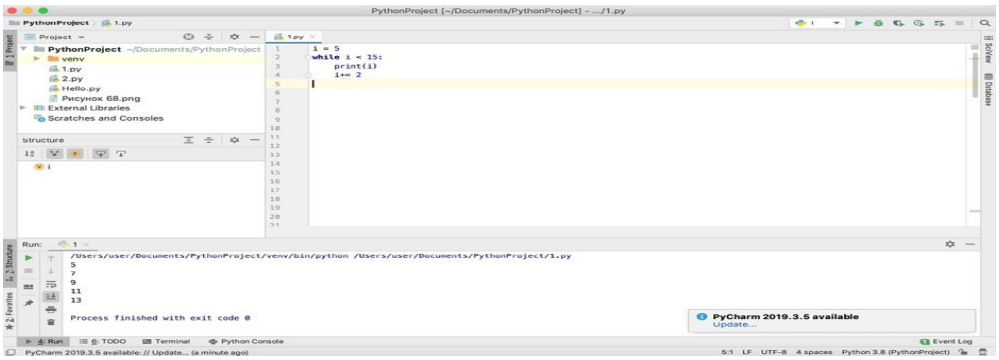


Рис.2.23 Результат работы программы

В данном примере организован перебор значений переменной i с шагом 2. Условие работы цикла: i<15. В теле цикла происходит изменение (увеличение) переменной i, поэтому цикл не будет бесконечным.

Пример 2

```
a=0
while a<7:
print("Python")
a+=1
```

В данном примере цикл выполняется, пока истинно условие a<7, значение перемен- ной a меняется в теле цикла.

Результат работы программы представлен на рисунке 2.24.

Рис. 2.24. Результат работы программы

После ключевого слова **while** указывается условное выражение, и пока это выражение возвращает значение **True**, будет выполняться блок инструкций. Ниже приведен пример программы, где цикл **while** будет выполняться до тех пор, пока значение переменной **‘number’** будет меньше значения 5 (рис. 2.25).

```
# применение цикла 'while'
number = 1
while number < 4:
print(f'number = {number}')
number += 1print('Работа программы завершена!')
```

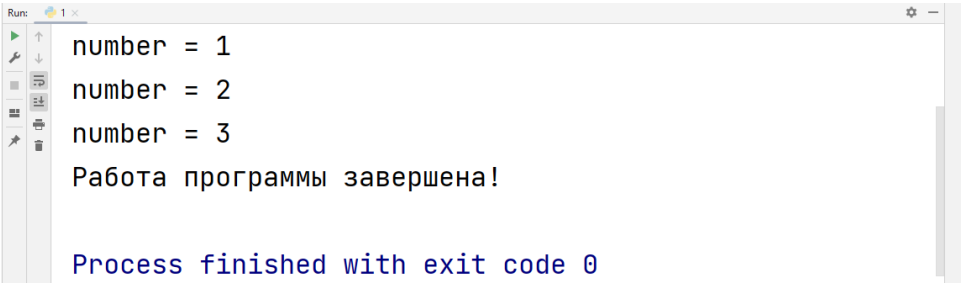


Рис. 2.25. Пример использования цикла ‘while’

Весь процесс цикла **while** можно представить так:

1. Проверяется значение переменной **‘number’** – больше ли оно 4 или нет. Поскольку вначале переменная равна единице, это условие возвращает значение **True**, и инструкции внутри цикла выполняются.

2. Инструкции цикла выводят н а консоль строку: number = 1. Далее значение переменной **‘number’** увеличивается на единицу. Так происходит до тех пор, пока значение переменной **‘number’** не становится равным 4.

3. Происходит проверка условия: number < 4. В таком случае, результатом будет значение **False**, так как значение переменной **‘number’** равно 4.

Важно!

Цикл – это часть программы, которая может выполняться несколько раз.

Контрольные вопросы

1. Какую роль выполняет команда float в языке Python?
2. Что такое логическая операция?
3. Как можно передавать аргументы программе?
4. Какую роль выполняет функция boolean() в языке Python?
5. Какое минимальное количество аргументов можно передать программе на языке

Python за один раз?

Второй цикл, используемый в языке Python, — цикл с параметром. Синтаксис данного цикла:

for <переменная> **in** <объект>:

<оператор1>

else:

<оператор2>

Блок-схема работы цикла представлена на рисунке 2.26.

Рис. 2.26. Блок-схема цикла с параметром

Этот цикл перебирает заданную последовательность значений любого итерируемого объекта (например, строки или списка) и для каждого значения выполняет тело цикла. Цикл выполняется заданное число раз. Для обращения к текущему элементу последовательности обычно используется переменная цикла, её иногда называют управляющей переменной.

Часто для организации работы цикла с параметром *for* используется функция *range.* Функция *range()* возвращает последовательность чисел, регулируемую переданными в неё аргументами. Возможны следующие варианты обращения к данной функции:

1) range(finish)
2) range(start, finish)
3) range(start, finish, step)

Здесь *start* — это первый элемент последовательности (включительно), *finish* — последний (не включительно), а *step* — разность между следующим и предыдущим членами последовательности.

Например, *range(5)* возвращает последовательность 0, 1, 2, 3, 4.

Вызов *range(2,8)* возвращает последовательность 2, 3, 4, 5, 6, 7. Рассмотрим примеры организации работы цикла с параметром.

Пример 3

for a **in** range(10):

```
print(a)
```

В данном примере цикл выводит на экран последовательность чисел от 0 до 9 включительно. Результат работы программы представлен на рисунке 2.27.

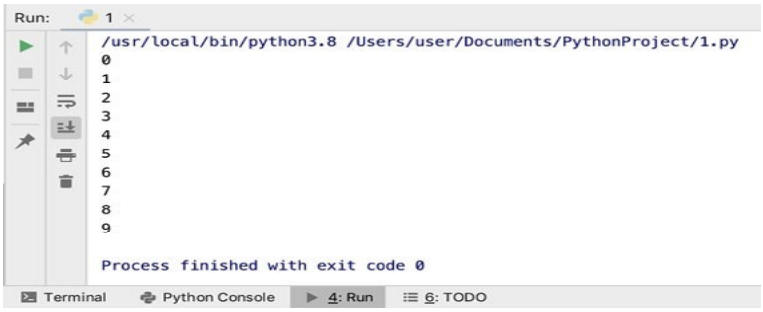


Рис. 2.27. Результат работы программы

Пример 4

```
for c in range(0,22,3):
    print(c,end=" ")
```

В данном примере на экран выводится последовательность чисел от 0 до 21 с шагом 3 в строку через пробел.

Результат работы программы представлен на рисунке 2.28.

Рис. 2.28. Результат работы программы

Также, говоря про работу циклов в языке Python, необходимо упомянуть про операторы continue, break, else.

Важно!

Оператор continue используется для перехода на следующую итерацию цикла, пропуская следующие после continue операторы тела цикла.

Пример 5

```
for i in range(10):
   if i==5:
        continue
   print(i*2,end=" ")
```

Результат работы программы представлен на рисунке 2.29.

Рис. 2.29. Результат работы программы

В данном примере при равенстве i==5 срабатывает оператор *continue*, в результате чего для *i*, равного 5, пропускается оператор *print(i*2,end=" ")*. Поэтому число 10 не выводится на экран.

Важно!

Оператор *break* используется для организации немедленного выхода из цикла. Это означает, что происходит досрочное завершение работы цикла.

Пример 6

```
for i in range(10):
   if i==5:
break
print(i*2, end=" ")
```

Результат работы программы представлен на рисунке 2.30.

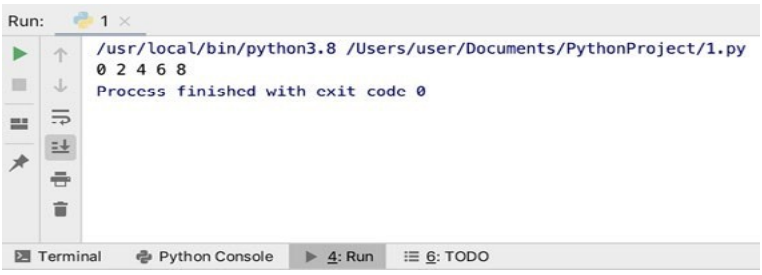


Рис. 2.30. Результат работы программы

В данном примере при равенстве i==5 срабатывает оператор break, в результате чего происходит завершение работы цикла. Значит, последнее значение i, рассмотренное в теле цикла, будет равно 4.

Оператор *else* используется для проверки, был ли произведён выход из цикла посредством оператора *break* или же цикл завершился иным образом.

Пример 7

```
for i in range(10):
    if i ==20:
```

```
        break
    print(i*2,end=" ")
else:
    print("значение не найдено")
```

Результат работы программы представлен на рисунке 2.31.

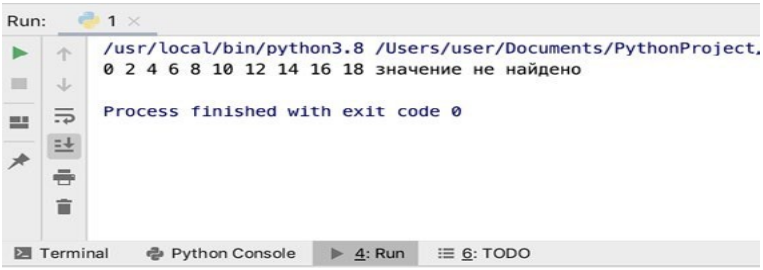


Рис. 2.31. Результат работы программы

В данном примере после вывода на экран последовательности от 0 до 18 на экран также выводится строка «значение не найдено», так как оператор *break* не сработал.

Также в языке Python возможно использование вложенных циклов, когда есть один внешний цикл и один или несколько вложенных. Стоит отметить, что использование вложенных циклов может замедлить работу программы.

Приведём ещё несколько примеров работы с циклами *for* и *while.*

Пример 8

На тренировке спортсмен ежедневно пробегает некоторую дистанцию, с каждым днём увеличивая её на 10%. Составить программу, определяющую по расстоянию, преодолённому спортсменом в первый день тренировки, длину дистанции на *k*-й день.

```
n= float(input("Введите начальную дистанцию "))
k= int(input("Введите количество дней "))
for i in range(k):
    n+=n*0.1
print("конечная дистанция ", n)
```

Результат работы программы представлен на рисунке 2.32.

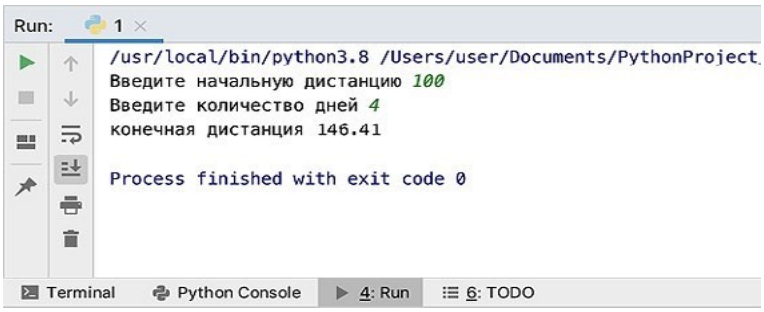


Рис. 2.32. Результат работы программы

В данной задаче используется цикл *for*, так как известно количество повторов цикла — k дней. Внутри цикла идёт увеличение переменной n, обозначающей длину дистанции: n+=n*0.1.

Пример 9

Перевести введённое пользователем десятичное число в двоичное. Известно, что число меньше 256.

```
dec=int(input("Введите десятичное число "))
v=128
for i in range(1,9):
    if dec>=v:
        print('1',end="")
        dec=dec-v
else:
```

Результат работы программы представлен на рисунке 2.33.

Рис. 2.33. Результат работы программы

В данном примере в переменной v задаётся вес старшего разряда двоичного числа. Так как по условию число должно быть меньше 256, то старший разряд будет иметь вес 128. Для перевода десятичного числа из него каждый раз вычитается вес старшего разряда, если это возможно, затем вес разряда уменьшается в 2 раза.

Пример 10

Разложить натуральное число на простые множители.

```
k=int(input("Введите число "))
print(k,'= ')
l=2
while not(k==1):
  if k%l==0:
     k=k/l
     print(l,end="")
   else:
     l+=1
```

Результат работы программы представлен на рисунке 2.34.

Рис. 2.34. Результат работы программы

В данном примере первое простое число l=2. В цикле while введенное число k несколько раз делится на потенциальный простой делитель l, если целочисленное деление не может быть выполнено, то ищется следующий простой делитель.

Самостоятельная работа студента.

1. Население города на 2021 г. насчитывало 620 тыс. человек. Считая темп прироста населения за год равным 3,7%, определить, в каком году оно превысит 1,5 млн человек.
2. Найти сумму нечётных делителей введённого с клавиатуры натурального числа.
3. Найти все натуральные числа из отрезка [1; 200], у которых количество делителей равно *n* (где *n* вводится с клавиатуры).

Указание. Примерный вид программы:

```
n=int(input("Введите кол-во делителей "))
for i in range(1,201):
    k=2
    for j in range (2,i):
        if i%j==0:
            k+=1
    if k==n:
        print(i)
```

4. Найти все четырёхзначные числа, у которых сумма крайних цифр равна сумме средних (например, 3221).
5. Найти все двухзначные числа, которые при умножении на 2 заканчиваются на 8, а при умножении на 3 — на 4.

В ходе выполнения самостоятельной работы вы получили представление о составлении циклических алгоритмов с использованием операторов while, for языка программирования Python.

Контрольные вопросы

1. Для чего используются циклы в языке программирования?
2. Какие виды циклов реализованы в языке Python?

3. Каков синтаксис оператора цикла while?

4. Каков синтаксис оператора цикла for?

5. Для чего используется оператор break внутри тела цикла?

Самостоятельная работа студента.

1. Два числа называются дружественными, если каждое равно сумме делителей другого, исключая само это число. Проверить, являются ли два введённых числа дружественными.

Указание. Примерный вид программы:

```
n=int(input("Введите первое число ")) m=int(input("Введите второе число "))
s1=0
for i in range(1,n):
   if n%i==0:
        s1+=i
s2=0
for i in range(1,m):
   if m%i==0:
      s2+=i
if (s1==m) and (s2==n):
   print("yes")
else:
   print("no")
```

2. Вывести на экран все четырёхзначные числа, у которых первая и последняя цифры равны.

3. На тренировке спортсмен ежедневно пробегает некоторую дистанцию, с каждым днём увеличивая её на 10%. Составить программу, определяющую по расстоянию, преодолённому спортсменом в первый день, количество тренировок, после которых ежедневная дистанция превысит *s* км.

5. Вывести на экран таблицу умножения на *n*, где *n* вводится с клавиатуры.

В ходе выполнения самостоятельной работы вы получили представление о составлении циклических алгоритмов с использованием операторов while, for языка программирования Python.

Контрольные вопросы

1. Какие основные операторы языка Python вы использовали при решении задач самостоятельной работы?

2. Какой оператор используется для организации цикла с предусловием в языке

Python?

3. Как организовать цикл с постусловием в языке Python?

Тема 1.3. Работа со списками и словарями

Практическая работа № 4-5.

Понятие словаря. Отличия словарей от списков. Создание словаря. Методы словарей. Применение списков и словарей в реальных задачах.

Цель этого занятия – изучить словари и их методы.

Словари

Словарь – структурная единица данных, в которой данные хранятся парами (ключ – значение).

Строка – текст, который пользователь хочет увидеть на консоли.

Словари создаются по следующему шаблону (рис. 2.49).

 # программа со словарем # создание словаря

phonebook = {

"Kevin": 123,

"Mark": 432,

"John": 643

}

 # вывод словаря

print(**'Словарь:'**, phonebook)

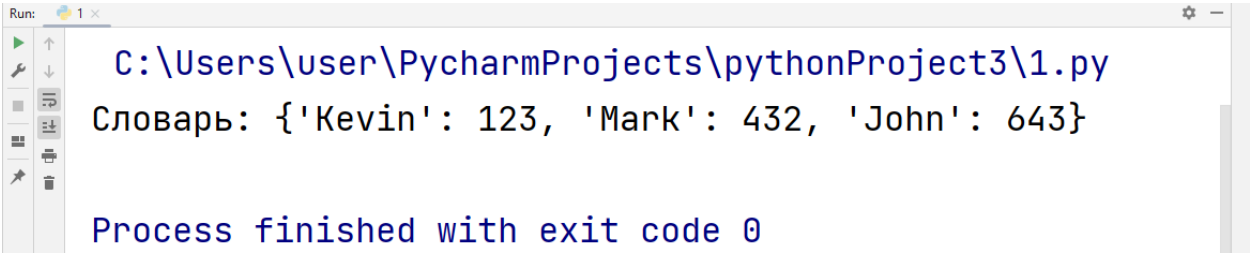


Рис. 2.49. Пример работы программы с использованием словаря

Для удаления данных можно использовать команду **del** (рис. 2.50).

 # программа со словарем # создание словаря

phonebook = {

"Kevin": 123,

"Mark": 432,

"John": 643

}

 # вывод словаря до удаления print(**'Словарь до:'**, phonebook) *# удаление элемента*

del phonebook[**"John"**]

 # вывод словаря после удаления

print(**'Словарь после:'**, phonebook)

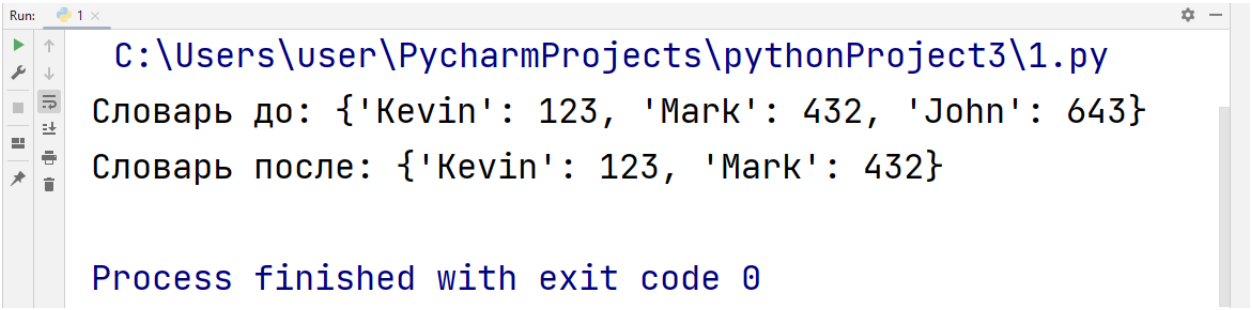


Рис. 2.50. Пример работы программы с использованием словаря

Простые команды словарей

Для получения данных из словаря используется команда **get()** (рис. 2.51).

 # программа со словарем # создание словаря

population = {

"SPB": 82319412,

"Frankfurt": 1734743,

"Colonge": 2184025

}

 # вывод данных print(population.get(**'SPB'**)) print(population.get(**'Colonge'**))

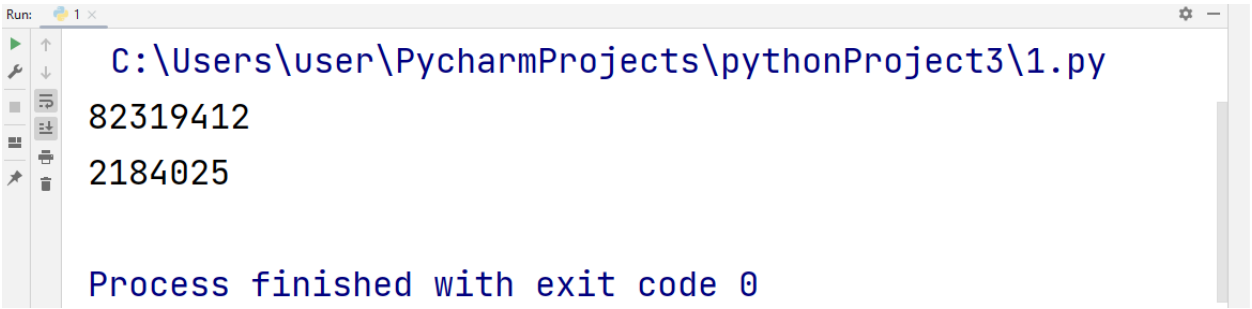


Рис. 2.51 Пример работы программы с использованием команды get()

Для добавления данных в словарь можно использовать команду **update()** (рис. 2.55).

 # программа со словарем # создание словаря

population = {

"SPB": 82319412,

"Frankfurt": 1734743,

"Colonge": 2184025

}

 # вывод словаря до добавления print(**'Словарь до:'**, population) *# добавление элементов*

population.update(Moscow = 123451, NSK = 1231451)

 # вывод словаря после добавления

print(**'Словарь после:'**, population)

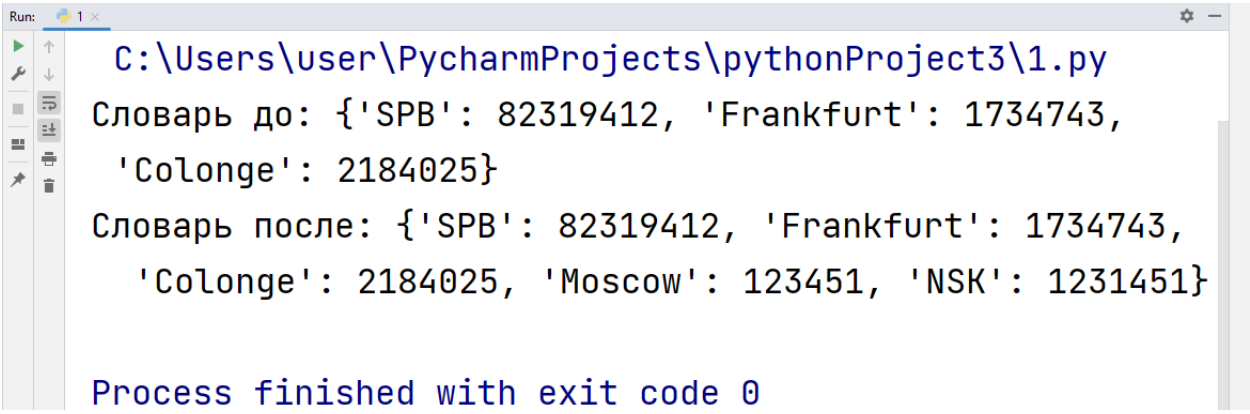


Рис. 2.55. Пример работы программы с использованием команды update()

Для определения количества пар данных используется команда **len()** (рис. 2.56).

 # программа со словарем # создание словаря

population = {

"SPB": 82319412,

"Frankfurt": 1734743,

"Colonge": 2184025

}

 # вывод данных

print(**'Словарь:'**, population)

print(**'Количество пар:'**, len(population))

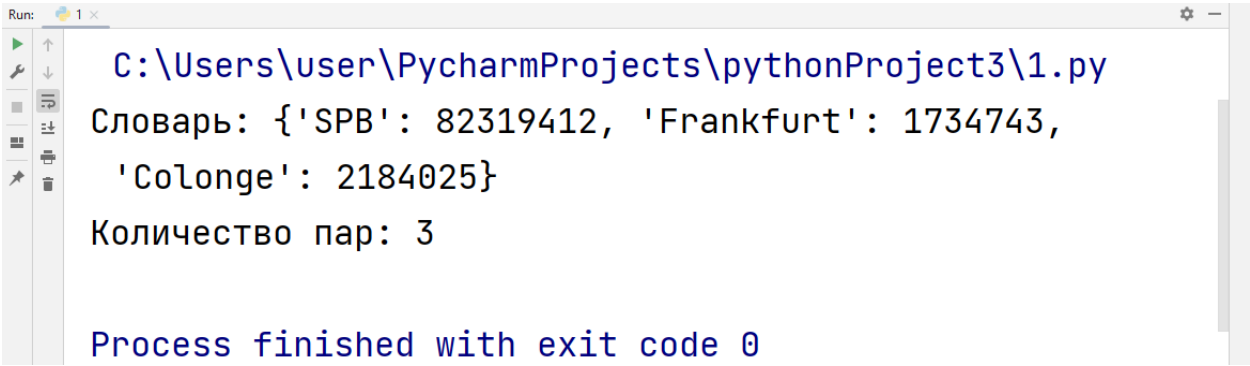


Рис. 2.56. Пример работы программы с использованием команды len()

Управляющие конструкции

С помощью конструкции **'\n'** можно перейти на следующую строку (рис. 2.57).

```
# программа с управляющей конструкцией
 # ввод строк
inf_f = 'Сообщение:' inf_s = 'Среда PyCharm'
 # вывод данных
print(inf_f, '\n', inf_s)
```

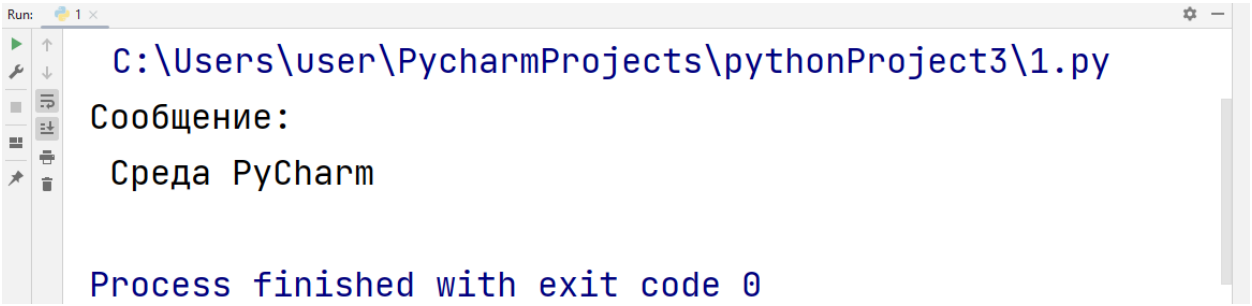


Рис. 2.56. Пример работы программы с использованием конструкции ‘\n’

При помощи конструкции **‘\t’** можно совершить горизонтальную табуляцию (рис. 2.57).

```
# программа с управляющей конструкцией # ввод строк
inf_f = 'Сообщение:' inf_s = 'Среда PyCharm'
# вывод данных
print(inf_f, '\t', inf_s)
```

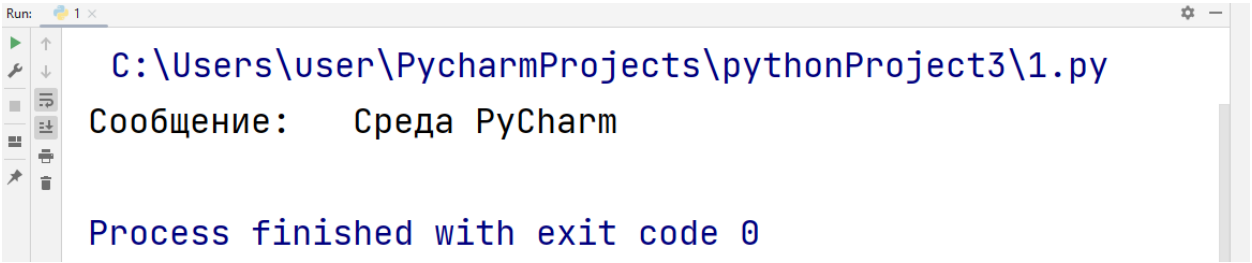


Рис. 2.57. Пример работы программы с использованием конструкции ‘\t’

С помощью фигурных скобок можно вставлять значения переменных в строки вывода (рис. 2.58).

```
# программа с {} # ввод данных Name = "Sam"
Age = 24
data = f"name: {Name} \n age: {Age}"
# вывод данных
print(data)
```

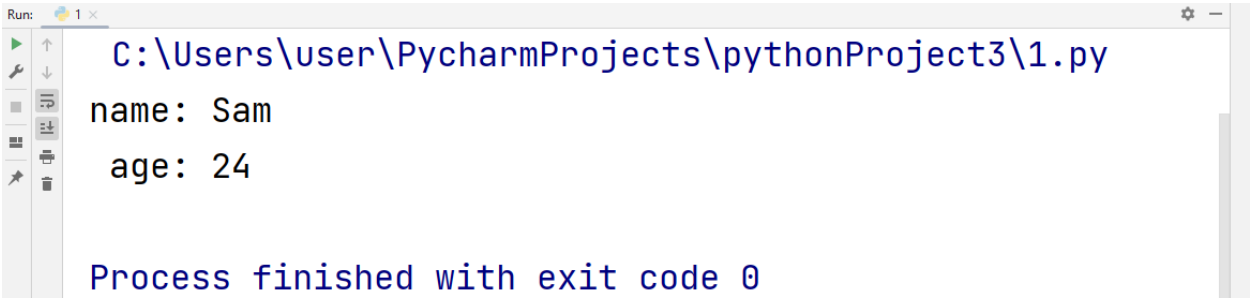


Рис. 2.58. Пример работы программы с использованием фигурных скобок

Важно!

Словарь – структурная единица данных, в которой данные хранятся парами (ключ – значение).

Строка – текст, который пользователь хочет увидеть на консоли.

Метод – функция или процедура, относящаяся к классу или объекту.

Контрольные вопросы

1. Какую роль выполняет функция len() в языке Python?
2. Что такое словарь?
3. Как можно указывать аргументы словарю?
4. Какую роль выполняет функция del в языке Python?
5. Какое максимальное количество аргументов можно передать словарю на языке Python за один раз?

Тема 1.4. Аналитика данных на Python

Практическая работа № 6-9

Понятие данных, больших данных. Наборы данных. Платформа Haggle. Библиотека Pandas. Объекты Series и DataFrame. Получение общей информации о данных. Индексация по условиям и изменение данных в таблицах

Строковые данные используются в языке Python для записи текстовой информации. Строки в Python представляют собой упорядоченные последовательности символов. Каждый символ имеет порядковый номер в таблице, которая называется кодовой таблицей. По умолчанию в языке Python используется стандарт кодовой таблицы Unicode. Строки в Python обрамляются кавычками или апострофами, при этом важно, чтобы с обоих концов строки использовались кавычки одного и того же типа.

Важно!

Важно отметить, что языке Python не используется символьный тип char, привычный для многих языков программирования, например Pascal, C++. То есть один символ, заключенный в апострофы, также представляет собой строку.

Строка – набор однотипных элементов, которые объединены под одним именем (рис. 2.59).

```
# программа со строкой # ввод строки
first = '123abc'
# вывод строки
print('Строка:', first)
```

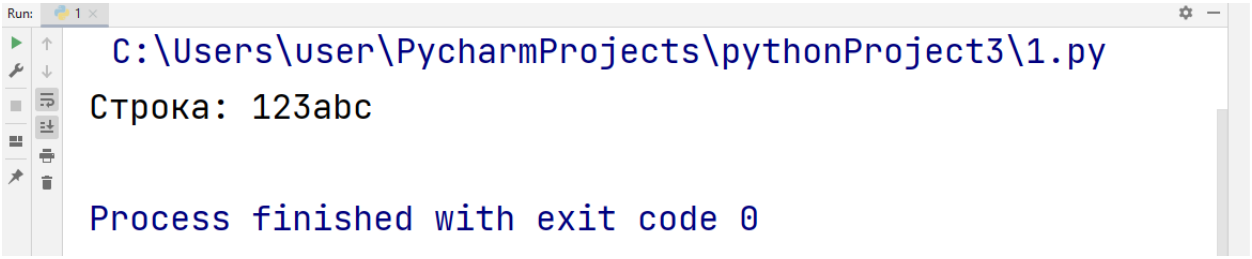


Рис. 2.59. Пример работы программы со строкой

В таблице 4 приведены примеры задания строк.

Примеры строк в Python

Таблица 4

Задание строки	Описание
S=″	Пустая строка
S=′A′	Строка, состоящая из одного символа
S=′Hello′	Строка, состоящая из нескольких символов, заключена в апострофы
S="good"	Строка, состоящая из нескольких символов, заключена в кавычки

Строки можно соединять друг с другом при помощи оператора **‘+’** (рис. 2.60.).

```
# программа со строками  #ввод строк
Str_1 = 'This is '
str_2 = 'another lesson about Python'
# вывод строк с оператором '+'
print(str_1 + str_2)
```

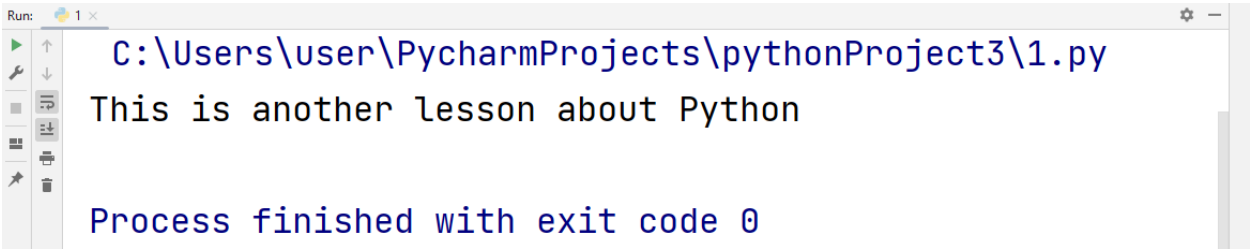


Рис. 2.60. Пример работы программы с использованием оператора ‘+’

Также в строках в языке Python могут использоваться специальные непечатаемые символы. В таблице 5 представлен перечень таких символов, которые носят название экранированных последовательностей: это последовательности, начинающиеся с символа «\», за которым следует один символ или более.

Экранированные последовательности

Таблица 5

Экранированная последовательность	**Описание**
\n	Перевод строки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\r	Возврат в начало строки

С помощью конструкции **‘\n’** можно перейти на следующую строку (рис. .2.61).

```
# программа с управляющей конструкцией # ввод строк
inf_f = 'Сообщение:'
inf_s = 'Среда PyCharm'
# вывод данных
print(inf_f, '\n', inf_s)
```

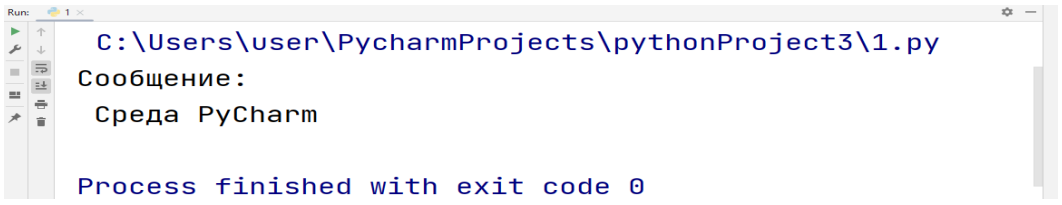


Рис. 2.61. Пример работы программы с использованием конструкции ‘\n’

При помощи конструкции **‘\t’** можно совершить горизонтальную табуляцию (рис. 2.62).

```
# программа с управляющей конструкцией # ввод строк
inf_f = 'Сообщение:'
inf_s = 'Среда PyCharm'
# вывод данных
```

print(inf_f, **'\t'**, inf_s)

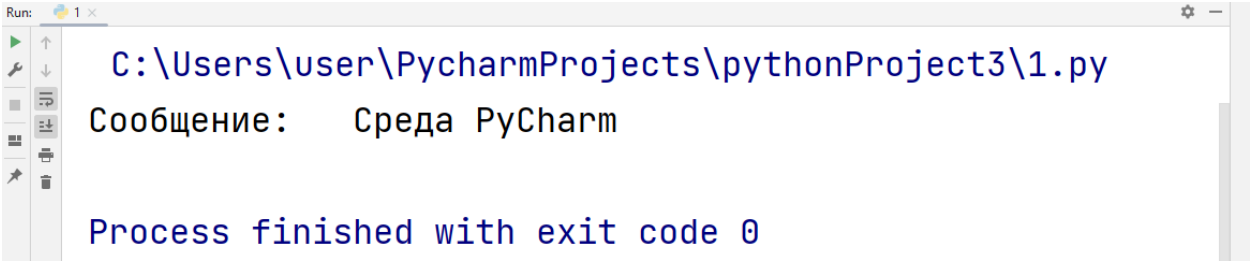


Рис.2 .62. Пример работы программы с использованием конструкции ‘\t’

С помощью фигурных скобок можно вставлять значения переменных в строки вывода (рис. 2.63).

 # программа с {} # ввод данных

Name = **"Sam"**

Age = 24

data = **f"name:** {Name} **\n age:** {Age}**"**

 # вывод данных

print(data)

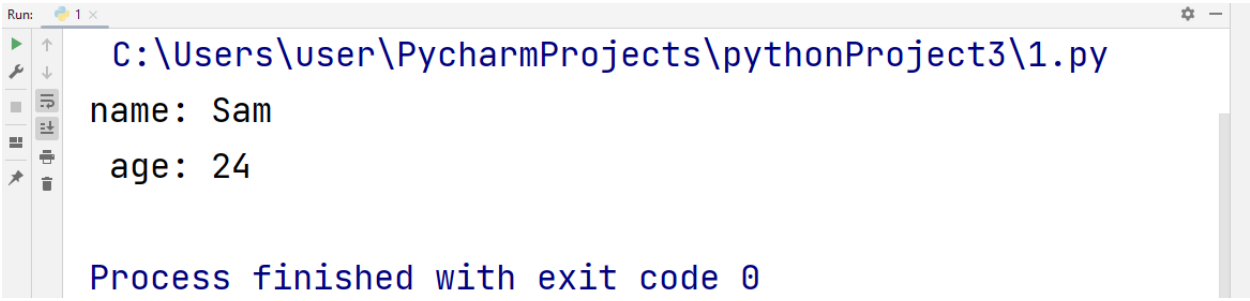


Рис. 2.63. Пример работы программы с использованием фигурных скобок

Аналогично спискам, строки в языке Python являются упорядоченными последовательностями символов, следовательно, могут быть проиндексированы. Для обращения к отдельному символу в строке можно указать имя строки, за которым следует число (индекс) в квадратных скобках [ ]. Так же, как и в списках, нумерация символов начинается с индекса 0. Например, обращение s[2] к строке s="hello" возвращает символ «l». Индексы также могут быть представлены и отрицательными значениями (табл. 6).

Варианты индексации строк в Python

Таблица 6

Строка S	**l**	**e**	**s**	**s**	**o**	**n**
Индекс	0	1	2	3	4	5
Отрицательный индекс	–6	–5	–4	–3	–2	–1

Важно!

Следует обратить внимание на то, что строки в языке Python, в отличие от списков, являются неизменяемыми!

После того как строка будет создана, её нельзя будет изменить, т. е. все операции над строками производятся путём создания новых строк. Например, нельзя изменить отдельный символ строки: обращение s[3]='x' вызовет ошибку.

Так же, как и для списков, для строк в языке Python доступны различные срезы. Формат среза строки схож с форматом среза списка:

Str [начало:конец:шаг], где:

- начало указывает, с какого элемента нужно начать (по умолчанию равно 0);
- конец указывает, по какой элемент берутся элементы (по умолчанию равен длине списка);
- шаг определяет, с каким шагом берутся элементы, например каждый второй или третий (по умолчанию каждый первый).

Приведём примеры работы со срезами строк.

Пример 1

```
word='strength'
print(word[4:6])
```

Выводит на экран 2 символа.

Результаты работы программы представлен на рисунке 2.64.

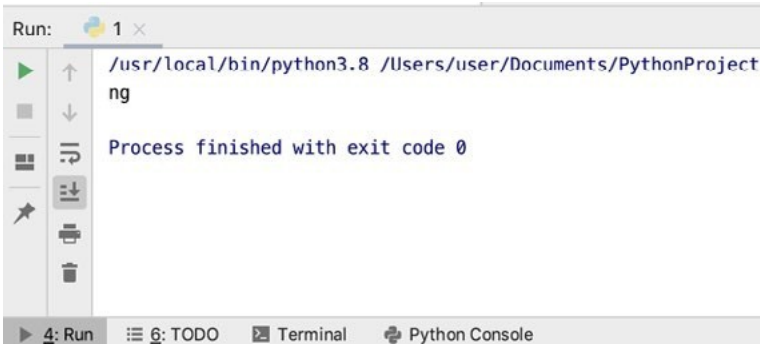


Рис. 2.64. Результат работы программы

Пример 2

```
word='strength'
print(word[:2])
```

Выводит на экран 2 символа.

Результат работы программы представлен на рисунке 2.65.

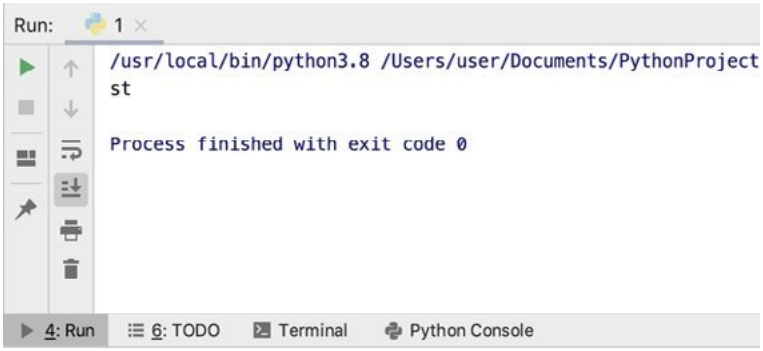


Рис. 2.65. Результат работы программы

Рассмотрим основные операции над строками, разрешённые в языке Python (табл. 7).

Операции над строками в Python

Таблица 7

Оператор	**Описание**	**Пример**
+	Сложение (конкатенация) строк. В результате применения возвращается строка, равная «склейке» указанных строк	a=′py′ b=′th′ b=′on′ s=a+b+c
*	Умножение строк. Оператор создаёт несколько копий строки, формат оператора: s*n или n*s, где s — это строка, а n — натуральное число	s=′ab′ sn=s*4
in	Оператор принадлежности, который возвращает True, если подстрока входит в строку, и False, если не входит	**if** ′z′ **in** s: print(5)
>, <, >=, <=, ==, !=.	Сравнение строк	S1="ab" S2="xy" if S1>S2: print("Ok")

Пример 3

```
s1="abc"
s2="123"
s3=s1+s2
s4=s1*3
print(s3,' ',s4)
if s3>s4:
    print(s3)
else:
```

print(s4)

Результат работы программы представлен на рисунке 2.66.

Рис. 2.66 Результат работы программы

Также в языке Python используется достаточное число встроенных функций для обработки строк. Опишем некоторые из них в таблице 8.

Основные функции (методы) для обработки строк

Таблица 8

Функция	**Описание**
ord(x)	Возвращает код символа x в ASCII
chr(n)	Возвращает символ, код которого в ASCII равен n
len()	Возвращает длину строки
isnumeric()	Возвращает True, если строка представляет собой число
lower()	Переводит строку в нижний регистр
upper()	Переводит строку в верхний регистр
find(str[,start[,end]])	Возвращает индекс подстроки в строке. Если подстрока не найдена, возвращается число –1
replace(old,new[,num])	Заменяет в строке одну подстроку (old) на другую (new). num ограничивает количество замен
split([d[,num]])	Разбивает строку на подстроки в зависимости от разделителя d. num определяет максимальное количество частей для разбиения
join(strs)	Объединяет строки в одну строку, вставляя между ними определённый разделитель strs

(Квадратные скобки в синтаксисе функций означают необязательные элементы.) Полный список методов строк можно увидеть, если вызвать функцию dir( ).

Пример 4

Заменить в строке все символы «о» на «а».

```
s1="good morning"
s2=s1.replace('o','a')
print(s2)
```

Результат работы программы представлен на рисунке 2.67.

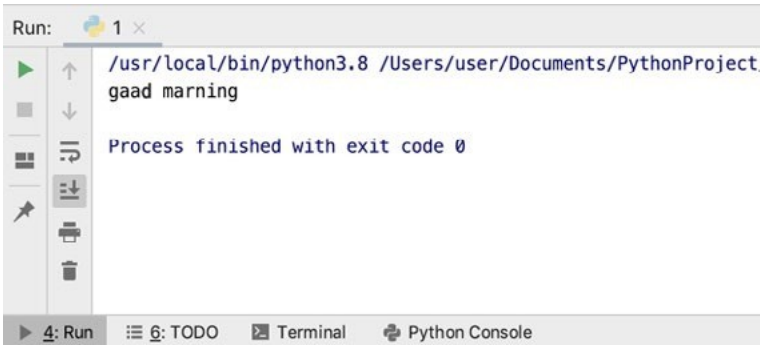


Рис. 2.67. Результат работы программы

В данном примере используется встроенная функция replace, которая заменяет все требуемые по условию символы.

Пример 5

В строке заменить пробелы знаком тире «–». Если встречается подряд несколько пробелов, то их следует заменить одним знаком «–», пробелы в начале и конце строки удалить.

```
s=input("Введите строку: ") i=0
while s[i]==' ':
    i+=1
s=s[i:]
i=len(s)
while s[i-1]==' ':
    i-=1
s=s[:i]
sn=s[0] i=1
while i<len(s):
    if s[i]!=' ':
        sn+=s[i]
    elif s[i-1]!=' ':
        sn+='-'
    i+=1
print(sn)
```

Результат работы программы представлен на рисунке 2.68.

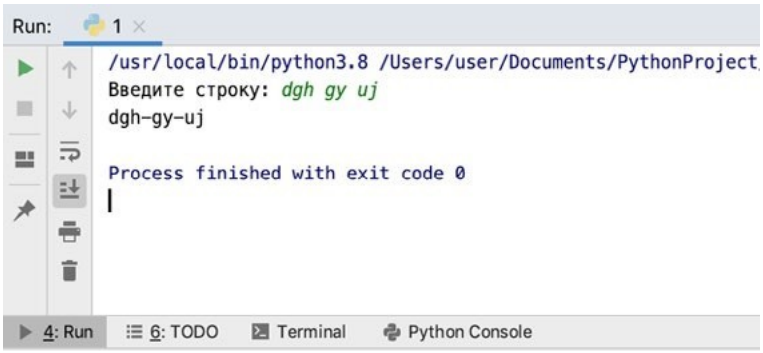


Рис. 2.68. Результат работы программы

В данном примере используется встроенная функция len, которая возвращает длину строки, также используются индексы для обращения к отдельному символу строки. Здесь не рекомендуется использовать функцию replace, так как она заменит все пробелы на тире, что противоречит условию задачи.

Пример 6

Определить, является ли введённая строка палиндромом, т. е. читается ли одинаково в прямом и обратном направлении (например, КАЗАК).

```
s=input("Введите строку: ")
sp=''
for i in range(len(s)-1,–1,-1):
    sp=sp+s[i]
if s==sp:
   print("палиндром"
else:
   print("не палиндром")
```

Результат работы программы представлен на рисунке 2.69.

Рис. 2.69. Результат работы программы

В данном примере получаем строку sp, обратную к строке s в цикле for с уменьшением шага. Затем две строки сравниваются, если они равны, то строка является палиндромом.

Можно предложить и другой вариант, в котором получаем обратную строку с использованием среза.

```
s=input("Введите строку: ")
sp=s[::-1]
if  s==sp:
    print("палиндром")
else:
    print("не палиндром")
```

Результат работы программы представлен на рисунке 2.70.

Рис. 2.70. Результат работы программы

Эту задачу можно ещё усложнить, если предположить, что в строке могут встречаться пробелы. Обычно в таких фразах пробелы не учитываются при проверке. Например, в знаменитой фразе «А роза упала на лапу Азора» пробелы игнорируются. Предварительно перед проверкой пробелы удаляются с помощью функции replace.

```
s=input("Введите строку: ")
if ' ' in s:
        s1=s.replace(' ','')
sp=s1[::-1]
if s1==sp:
   print("палиндром")
else:
print("не палиндром")
```

Результат работы программы представлен на рисунке 2.71

Рис. 2.71. Результат работы программы

Индексы списков

В списках индексы начинаются с нуля, а не с единицы. У первого элементы индекс 0, у второго – единица и т.д. С помощью индексов можно выводить отдельные элементы строк (рис. 2.72).

```
# программа с индексами списков # ввод строки
str_1 = [2, 4, 6, 8, 10]
# вывод данных
print('Строка:', str_1)
print('Первый элемент строки:', str_1[0])
print('Третий элемент строки:', str_1[2])
print('Последний элемент строки:', str_1[4])
```

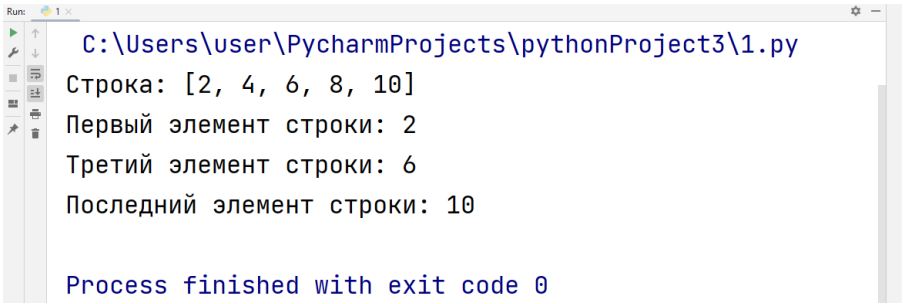


Рис. 2.72. Пример работы программы с использованием индексов списков

Здесь у строки **‘first’** выводятся элементы с индексами 0, 2 и 4 (рис. 2.73).

```
# программа с индексами списков
# ввод строки
first = [1, 3, 5, 7, 9, 11, 13]
# вывод данных
print('Строка:', first)
print('Первый элемент строки:', first[0])
 print('Третий элемент строки:', first[2])
```

print('**Пятый элемент строки:**', first[4])

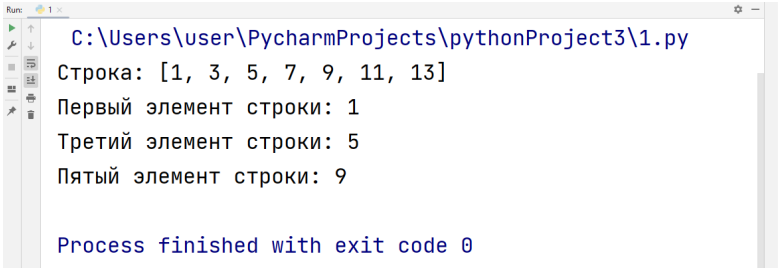


Рис. 2.73. Пример работы программы с использованием индексов списков

Контрольные вопросы

1. Какую роль выполняет функция max() в языке Python?
2. Что такое строка?
3. Как можно передавать аргументы строке?
4. Какую роль выполняет функция print() в языке Python?
5. Какое минимальное количество аргументов можно передать функции на языке Python за один раз?

Самостоятельная работа.

1. Заменить в строке все символы «а» на «mama».
2. Для каждого символа, введённого с клавиатуры, указать, сколько раз он встречается строке. Сообщение об одном символе должно выводиться не более одного раза.

Указание. Примерный вид программы:

```
s=input("Введите строку: ")
i=0
l=list(s)
i=0
while i<len(l):
    k=l.count(l[i])
    print(l[i],k)
    t=l[i]
    while t in l:
        l.remove(t)
```

3. Отредактировать предложение, удаляя из него лишние пробелы, оставляя только по одному пробелу между словами.
4. Проверить, можно ли из букв слова *X* составить слово *Y*, используя каждую букву слова *X* не более одного раза.

5. Подсчитать количество гласных букв в строке.

Указание. Примерный вид программы:

```
s=input("Введите строку: ")
k=0
gl=′аоеиуяыёэю′
for x in s:
    if x in gl:
        k+=1
print("кол-во гласных ",k)
```

6. Найти процентное содержание цифр в исходном тексте. найти сумму чисел, встречающихся в строке.

7. Для строки и указанного символа определить номер первого и последнего вхождения этого символа в строку.

8. Заменить в самом длинном слове строки буквы «а» на «b».

9. Удалить из строки «лишние» пробелы, т. е. оставить только один пробел в последовательности пробелов.

10. Удалить из строки все символы, заключённые в скобки. Например, для строки "abcd(123)efg" удалить подстроку "123" и получить в результате "adcdefg".

Контрольные вопросы

1. Дайте определение строки в языке программирования Python. Строки должны заключаться в кавычки или апострофы?
2. Можно ли изменять строку в языке программирования Python? Можно ли изменять отдельный символ строки?
3. Какие основные встроенные функции для работы со строками в языке Python вы можете назвать?
4. Какие основные функции для работы со строками в языке Python вы использовали при решении задач лабораторной работы?
5. Какие способы задания строк вы использовали в практической работе?
6. Возможен ли следующий фрагмент программы на языке Python?

```
s=′pqr′ s1=′123′ s+=s1
```

Графическая интерпретация данных. Библиотеки numpy, matplotlib, pandas

В Python есть много возможностей для графической интерпретации данных.

Библиотека *numpy* является одной из наиболее часто используемых библиотек для реализации алгоритмов вычислительной математики, анализа временных рядов и визуализации данных в форме массивов, в частности, представляемыми списками и кортежами.

Библиотека *matplotlib* предназначена для построения графиков, гистограмм различных видов и других способов графической интерпретации данных. Графики могут быть представлены в 2D и 3D-форматах.

Библиотека *pandas* широко используется для анализа данных (*Data Mining*). Данные в ней реализуются в виде структур *Series* и *DataFrame*. Пакет также содержит ряд методов для фильтрации данных и модули выполнения операций ввода-вывода.

Задача 14

Составить программу для построения семейства графиков функций:

$$f1(x) = \sin(0,8x)^2 + e^{\cos(x)}$$

$$f2(x) = e^{\cos(2x)} - 3\sin(0,5x) + 0,4$$

Программа

Текст модуля *Grafik.py*

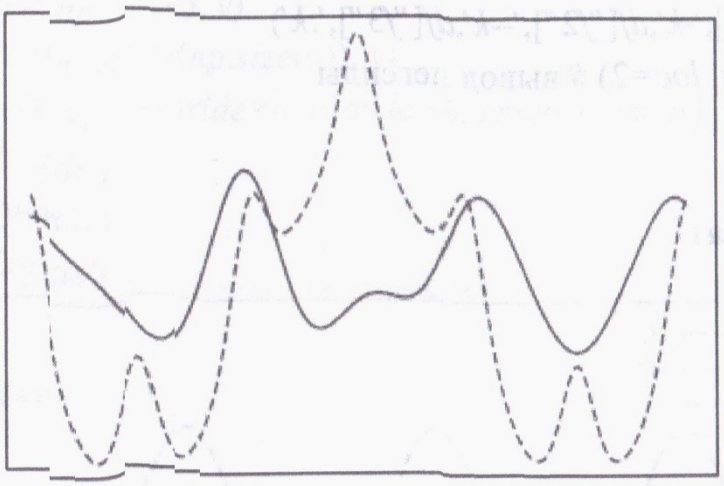
```
import numpy as np, matplotlib.pyplot as plt
# Семейство графиков
def grf1(x):
  return np.sin(0.8*x)**2+np.exp(np.cos(x))
def grf2(x):
  return np.exp(np.cos(2*x))-3*np.sin(0.5*x)+0.4
```

```
xn=0
xk=np.pi*6
nn=200
X=np.linspace(xn,xk,nn)
Y,Z=grf1(X),grf2(X)
plt.plot(X,Y)
plt.plot(X,Z)
plt.show()
```

Модуль *main.py*

```
import Grafik
```

Результат



Задача 15

Составить программу для построения семейства графиков функций:

$$f1 = -0.12x$$

$$f2 = 10\sin(0.2x)$$

$$f3 = 15/(1 + 4{,}1 \cdot x)$$

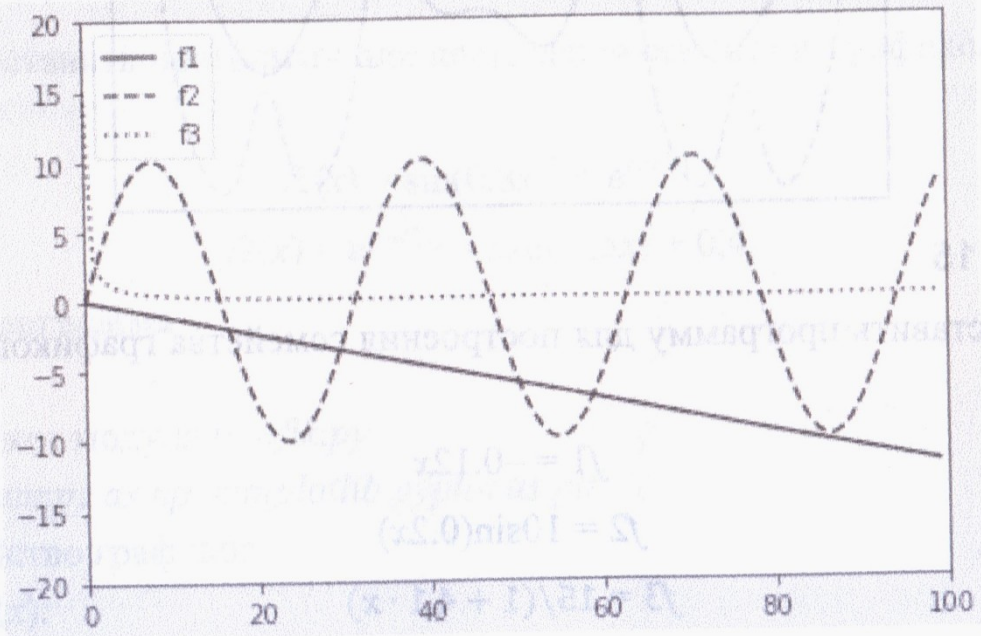
с использованием структуры *DataFrame*.

Программа

```
#с использованием фреймов
import pandas as pd
import matplotlib.lines
import matplotlib.pyplot as plt
import numpy as np
data = {'f1':[–0.12*x for x in range(100)],
        'f2':[10*np.sin(0.2*x) for x in range(100)],
        'f3':[15/(1+4.1*x) for x in range(100)]}
df = pd.DataFrame(data)
plt.axis([0,100,–20,20])

plt.plot(df["f1"],'-k',df["f2"],'--k',df["f3"],':k')
plt.legend(data, loc=2) # вывод легенды
plt.show()
```

Результат



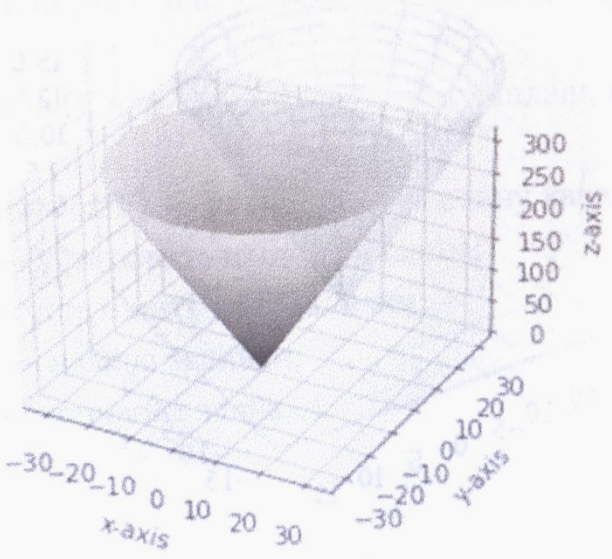
Задача 16

Составить программу для построения перевернутого конуса в 3D-формате.

Программа

```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 5 * np.outer(np.cos(u), v)
y = 10 * np.outer(np.sin(u), v)
z = 50 * np.outer(np.ones(np.size(u)), v)
ax.plot_surface(x, y, z,  rstride=4, cstride=4, cmap = cm.copper)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
plt.show()
```

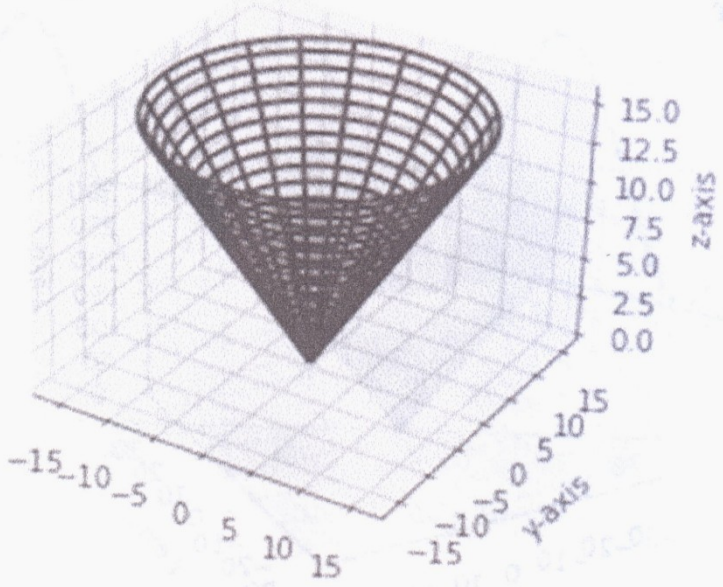
Результат



Программа

```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 5 * np.outer(np.cos(u), v)
y = 5 * np.outer(np.sin(u), v)
z = 5 * np.outer(np.ones(np.size(u)), v)
ax.plot_wireframe(x, y, z, rstride=5, cstride=5, color= (0.1,0.1,0.1),
linewidth=2)
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
plt.show()
```

Результат



Тема 1.5. Анализ данных на практических примерах

Практическая работа № 10-12.

Понятие статистики, описательной статистики. Описательный анализ данных. Основные описательные статистические величины (частота, среднее арифметическое, медиана, мода, размах, стандартное отклонение). Функции описательной статистики в Python Pandas. Практика вычисления описательных статистических величин в Python Pandas

Теоретическая часть

С функциями в языке Python мы встречаемся постоянно, например при работе со строками. Дадим определение функции: это именованный фрагмент программного кода, к которому можно обратиться из другого места вашей программы (но есть lambda-функции, у которых нет имени). Часто программист создаёт функции для работы с данными,
которые передаются ей в качестве аргументов, также функция может формировать некоторое возвращаемое значение.

Справочник

Функции в программировании используются для следующих целей: многократного использования в программе одного и того же фрагмента кода; разделения сложной программы на составные части — процедурной декомпозиции. Функции в Python используются для реализации вспомогательного алгоритма.

Рассмотрим создание функции. В общем виде функция имеет следующий формат:

def <имя_функции>(par1,par2,...,parN): операторы

 # программа с функцией

 # объявление функции **def** sum():

 # тело функции

x = 10

y = 15

 # вывод результата print(**'Сумма:'**, x+y)

 # вывод функции sum()

Аргументы функции – компоненты функции.

Посмотрим на пример использования функции, в выводе которой задаются значения переменных (рис. 2.74).

```
# объявление функции
def func_3(x, s):
# тело функции
# вывод результата
print('Результат:', x+s)
# вызов функции
func_3(10,15)
```

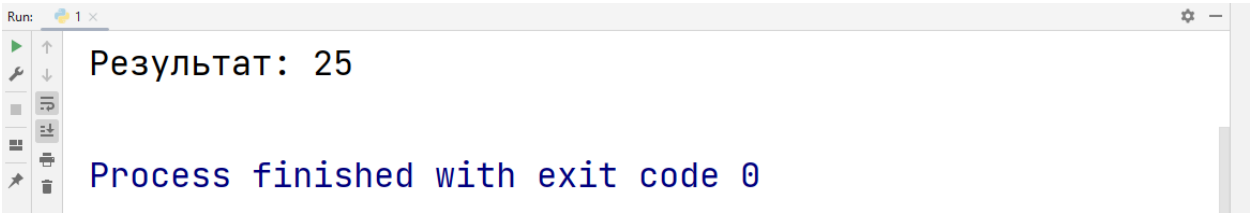


Рис. 2.74. Пример работы программы с функцией

Кроме этого, значения переменных могут задаваться в теле функции (рис. 2.75).

```
# объявление функции
def func_3():
# тело функции
x = 10
s = 14
# вывод результата
print('Результат:', x+s)
# вызов функции
func_3()
```

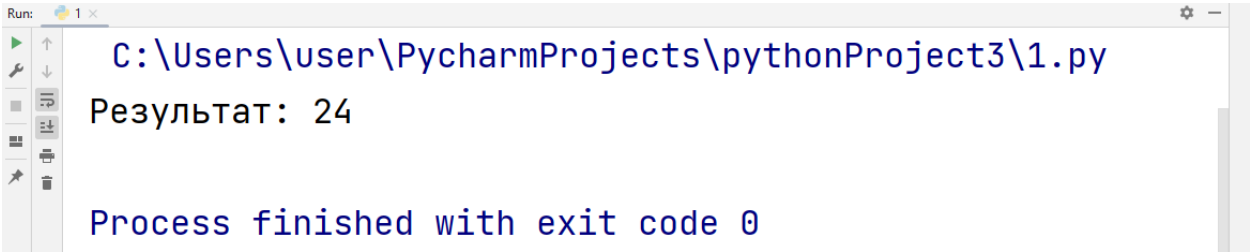


Рис. 2.75. Пример работы программы с функцией

Глобальные переменные могут быть объявлены вне функции, но использоваться ей (рис. 2.76).

```
# программа с функцией
# объявление глобальной переменной
```

```
n = 'Python'
# первая функция
def first():
print('This is', n)
# вторая функция
def second():
print('This was', n)
# вызовы функций
first() second()
```

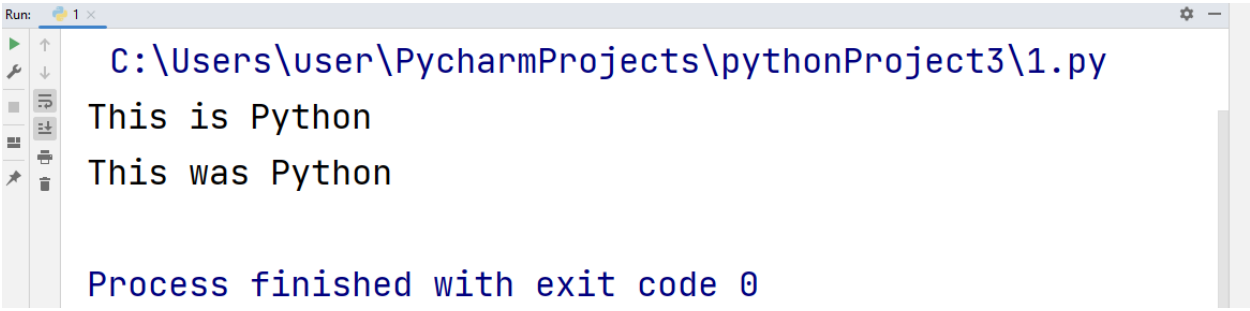
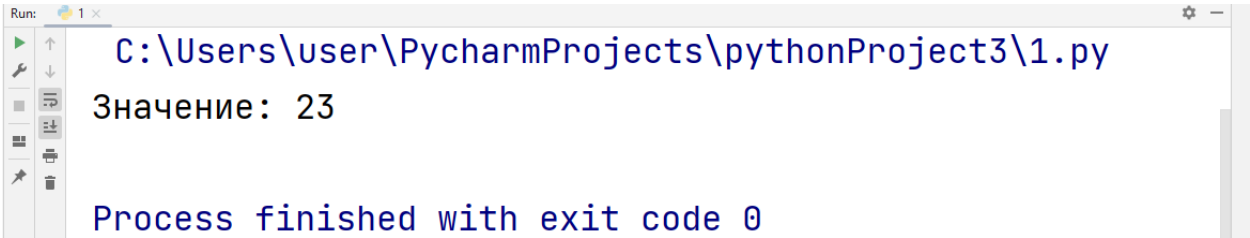


Рис. 2.76 Пример работы программы с глобальной переменной

Покажем пример программы с глобальной переменной в виде целого числа (рис. 2.77).

```
# программа с функцией
# объявление глобальной переменной
x = 23
# объявление функции
    def first(): print('Значение:', x)
# вызов функции
first()
```



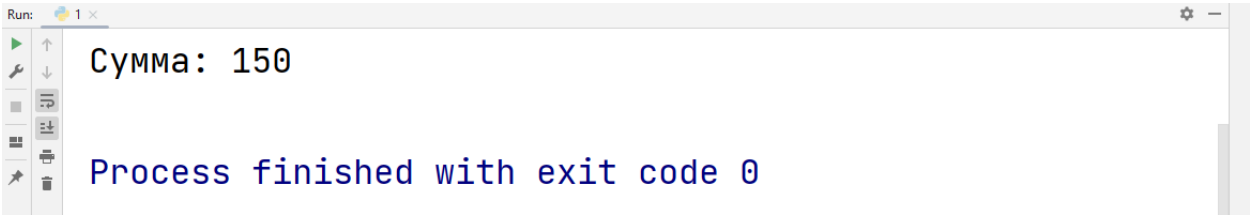


Рис. 2.77. Пример работы программы с функцией

Описание функции начинается со строки заголовка, в которой оператор def определяет имя функции, с которым будет связан объект функции, далее следует список параметров, который может быть пустым либо состоять из некоторого числа параметров в круглых скобках. Имена параметров в строке заголовка будут связаны с аргументами, передаваемыми в функцию в точке вызова. Возврат значения функцией осуществляется с помощью ключевого слова return, после которого указывается возвращаемое значение. Слово return может располагаться в любом месте в теле функции, этот оператор завершает работу функции и передает результат вызывающей программе.

Оператор return выглядит следующим образом:

return <выражение>

Оператор return является необязательным — если он отсутствует, работа функции завершается, когда достигается конец тела функции.

Вызов функции может осуществляться следующим образом:

<имя_функции>(arg1,arg2,...,argN)

В функцию передаются конкретные аргументы — записываются вместо параметров. Вызов функции может также быть включён в состав выражения.

Количество аргументов и параметров при вызове функции должно совпадать (но можно запрограммировать переменное количество принимаемых аргументов). В качестве аргументов могут выступать как конкретные значения, так и переменные.

Таким образом, общая структура работы с функцией выглядит, как показано на рисунке 2.78.

def <имя_функции>(par1,par2,...,parN):	
<имя_функции>(arg1,arg2,...,argN)	

Рис. 2.78. Схема работы с функцией

Приведём примеры описания и вызова простых функций.

Пример 1

```
def summ1(x,y):
        return x+2*y
x=int(input()
y=int(input())
print(summ1(x,y))
```

В этом примере описана функция с двумя параметрами, которая вычисляет выражение $x + 2 \cdot y$. Далее в основной программе с клавиатуры вводятся два целых числа, на экран выводится значение функции summ1, аргументами которой выступают введённые числа.

Результат работы программы приведен на рисунке 2.79.

Рис. 2.79. Результат работы программы

Пример 2

```
def mmm(a,b,c):
    if (a>b) and (a>c)
        return a
    elif (b>c) and (b>a):
        return b
    elif (c>a) and (c>b):
        return c
print(mmm(3,6,1))
```

В этом примере описана функция mmm, которая возвращает наибольшее из трёх чисел. В основной программе осуществляется вывод на экран значения функции от аргументов 3, 6, 1.

Результат работы программы представлен на рисунке 2.80.

Рис. 2.80. Результат работы программы

Самостоятельная работа

1. Написать функцию, которая возвращает процент от числа.

Указание. Примерное описание функции:

```
def procent(x,n):
    return x/100*n
```

Данная функция принимает два аргумента: само число и требуемый процент. Возвра- щает процент, рассчитанный по формуле.

Примерный вид программы:

```
def procent(x,n):
    return x/100*n
x=int(input())
n=int(input())
print(procent(x,n))
```

2. Написать функцию, которая вычисляет наибольший общий делитель двух целых чисел.

Указание

Для разработки функции можно использовать алгоритм Евклида, который представляет собой эффективный алгоритм нахождения наибольшего общего делителя. Данный алгоритм получил своё название в честь известного греческого математика Евклида. Алгоритм был описан автором в VII и X книгах «Начал». Это один из старейших численных алгоритмов, используемых в наше время. Данный алгоритм представлен на рисунке 2.81.

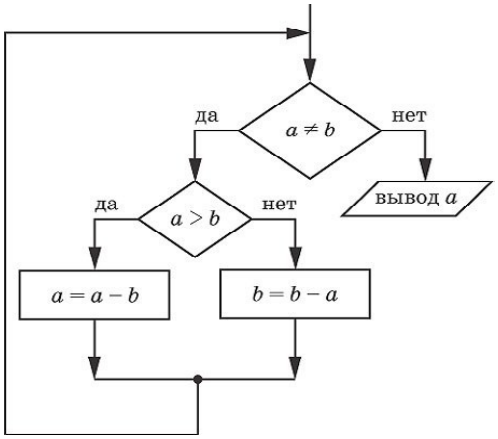


Рис. 2.81. Блок-схема алгоритма Евклида

Примерный вид описания функции:

```
def nod(a,b):
    while a!=b:
        if a>b:
            a-=b
        else:
            b- =a
    return a
x=int(input())
y=int(input())
print(nod(x,y))
```

3. Написать функцию, которая проверяет, является ли введённое число простым.

4. Написать функцию, которая выводит только гласные английские буквы введённой строки.

Указание. Примерный вид программы:

```
def glas(a):
    k=0
    while k<len(a):
        if a[k] in 'aeiouyAEIOUY':
            print(a[k])
        k+=1
a=str(input())
glas(a)
```

5. Написать функцию, которая для любого целого аргумента выводит количество цифр в нём.

6. Написать функцию нахождения длины отрезка по координатам его концов.

7. Написать функцию, определяющую, является ли заданное число палиндромом (например, число 12721 — палиндром, одинаково читается слева направо и справа налево).

9. Написать функцию нахождения длины отрезка по координатам его концов. Используя эту функцию, решить следующую задачу: даны координаты трёх вершин треугольника. Найти длины всех его сторон.

10. Написать функцию, которая сравнивает введённые числа и результат выдаёт в виде знаков «>», «<» или «=».

11.Написать функцию, которая по введённому *n* выводит на экран *n*-е число Фибоначчи.

Контрольные вопросы

1. Для чего используются функции в программировании?
2. Как выглядит описание функции в языке Python?
3. Для чего используется оператор return при описании функции в языке Python?
4. Какой вид имеет описание функции в языке Python?
5. Какие основные операторы языка Python вы использовали при решении задач практической работы?
6. Можно ли описать функцию в языке Python, не используя оператор return?

Тема 1.6. Основы визуализации данных

Практическая работа № 13-15.

Необходимость визуализации данных для анализа. Понятие научной графики. Библиотека Matplotlib. Понятие рисунка в Matplotlib. Основные виды графиков (гистограммы, диаграммы рассеяния, диаграмма размаха, линейный график, круговая диаграмма, тепловые карты). Основные графические команды в MatpLotlib

Модули и библиотеки

> **Словарь** – структурная единица данных, в которой данные хранятся парами (ключ – значение).
>
> **Библиотека** – файл с шаблонами кода.

Подключить библиотеку можно при помощи команды **import**. Команда **help()** позволит посмотрметь возможности библиотеки (рис. 2.82).

```
# программа с библиотекой
# подключение библиотеки
'scipy' import scipy help(scipy)
```

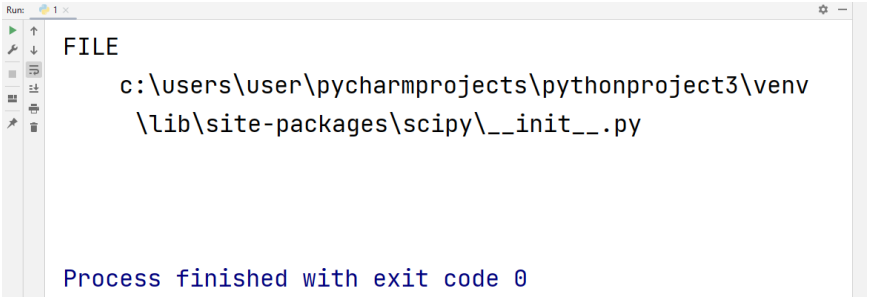


Рис. 2.82. Пример работы программы с использованием команды help()

Библиотеки **‘scipy’**, **‘numpy’** и **‘matplotlib’** позволяют строить различные графики (рис. 2.83).

```
# программа с библиотеками # подключение библиотек import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy import interpolate
# ввод данных
x = np.arange(3, 17) y = np.exp(x/2.0)
f = interpolate.interp1d(x, y) x_1 = np.arange(7, 14)
y_1 = f(x_1)
# вывод данных
plt.plot(x, y, 'o', x_1, y_1, '--') plt.show()
```

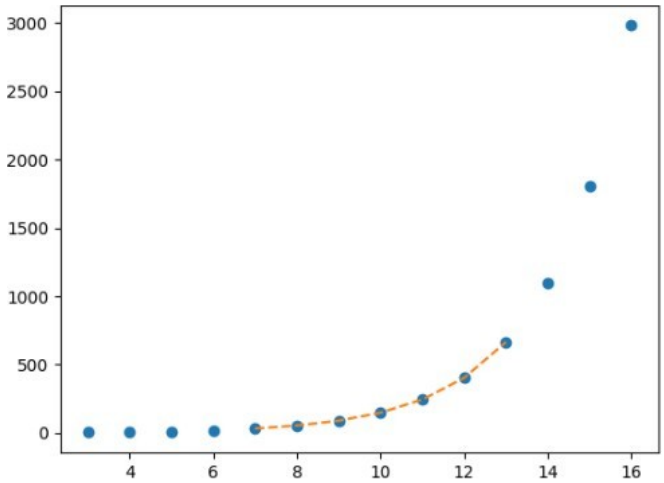


Рис. 2.83. Пример работы программы с использованием библиотек scipy, numpy и matplotlib

Построение рисунков. Библиотека turtle

Черепашья графика входила в язык программирования Logo, разработанный Уолли Фейрцейгом (Wally Feurzeig), Сеймуром Папертом (Seymour Papert) и Синтией Соломон (Cynthia Solomon) в 1967 году. Рабочим инструментом (карандашом) является черепашка (*turtle*), которая по умолчанию находится в точке с координатами (0, 0) в плоскости *x*-*y* и может выполнять ряд команд:

forward(), или *fd*() — черепашка перемещается вперед на заданное число пикселей;

backward(), или *bk*(), или *back*() — черепашка перемещается назад на заданное число пикселей;

right(), или *rt*() — повернуть черепашку вправо на заданное число градусов;

left(), или *lt*() — повернуть черепашку влево на заданное число градусов;

dot() — рисовать точку;

circle() — рисовать окружность;

position(), или *pos*() — текущее положение карандаша на холсте;

xcor(), *ycor*() — координата *x* карандаша на холсте и координата *y* соответственно;

pendown(), или *pd*(), или *down*() — опустить карандаш, начать рисовать;

penup(), или *pu*(), или *up*() — поднять карандаш, перестать рисовать;

pensize(), или *width*() — установить толщину карандаша;

pencolor() — установить цвет карандаша;

fillcolor() — установить цвет заливки;

begin_fill() — начало заливки;

end_fill() — окончание заливки.

Контрольные вопросы

1. Какую роль выполняет функция max() в языке Python?
2. Что такое библиотека?
3. Как можно указывать аргументы командам библиотеки?
4. Какую роль выполняет функция import в языке Python?
5. Какое максимальное количество команд библиотеки можно применить на языке Python в одной программе?

Тема 1.7. Проектная работа «Анализ больших данных в профессиональной сфере»

Практическая работа № 16-17

Характеристика основных этапов процесса анализа данных. Подготовка данных. Исследование и визуализация данных. Построение предсказательной модели. Интерпретация результатов анализа. Реализация основных этапов процесса анализа данных на примере набора данных из профессиональной сферы

Материалы для организации и проведения учебно-исследовательской и проектной деятельности школьников

Проект по программированию представляет собой проект, результатом которого является программа для решения той или иной задачи. Особенностью является то, что одна и та же задача в зависимости от уровня проработки может быть решена как начинающим, так и опытным программистом.

При выполнении проекта по программированию учащиеся имеют следующие возможности: выработать умение самостоятельно формулировать цели и задачи проекта, планировать свою деятельность, повысить уровень программирования на языке Python, получить умение представлять результаты своей деятельности.

Проект может разрабатываться индивидуально или группой учащихся. Если задача достаточно сложная, то проект может быть разбит на подзадачи, под проекты. Каждую подзадачу будут выполнять различные группы участников проекта. Например, одна группа занимается разработкой алгоритма, другая группа — непосредственно написанием и отладкой кода на языке Python, третья — подготовкой к презентации проекта.

План работы над проектом по программированию может совпадать с этапами разработки программы (рис.2.84).

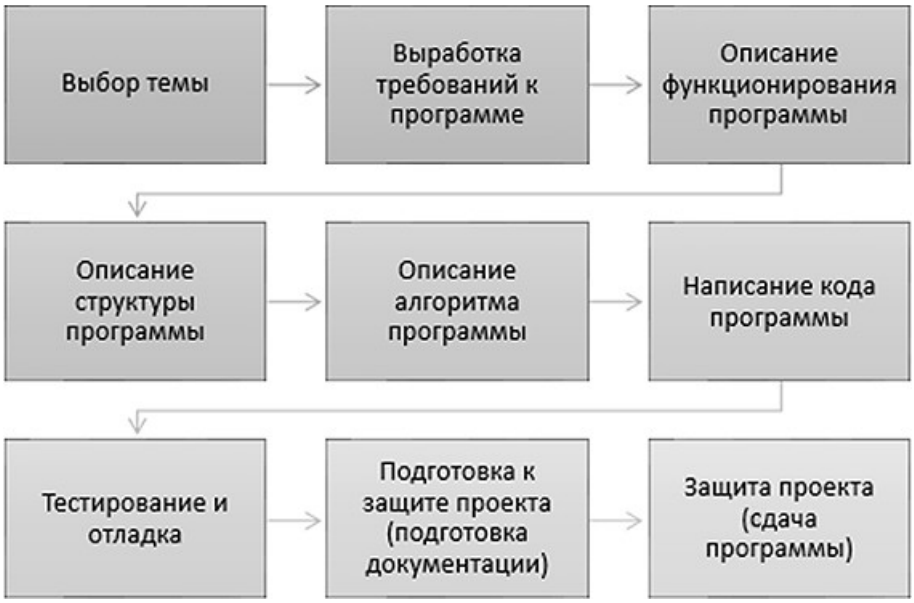


Рис. 2.84. Этапы проекта

В помощь участникам проекта можно предложить заполнить следующий учётный лист.

Проект по программированию

Тема проекта:

Творческое название (при наличии):

Основополагающий вопрос:

Авторы:

1.

2.

3.

…

Предметная область:

Краткая аннотация:

Этапы выполнения проекта:

При подготовке к защите проекта учащимся необходимо сделать презентацию и доклад, в котором отражаются основные этапы разработки программы, представлен алгоритм решения задачи, дан листинг программы, сформулированы основные результаты работы. Можно предложить в помощь учащимся заполнить следующий чек-лист.

1. Аннотация.
2. Содержание.
3. Постановка задачи:

а) возможности использования программы; б) описание интерфейса.

4. Формализация алгоритма:

а) перечень подпрограмм (при наличии);

б) описание алгоритма (блок-схема или подробное словесное описание алгоритма).

5. Листинг программы (текст программы).

6. Тестовые примеры:

а) результаты работы;

б) скриншоты результатов работы.

7. Описание размещения.

8. Требования к программным и аппаратным средствам.

Для оценивания проекта могут быть разработаны специальные оценочные листы.

В таблице 23 представлен пример оценочного листа.

Лист оценивания проекта

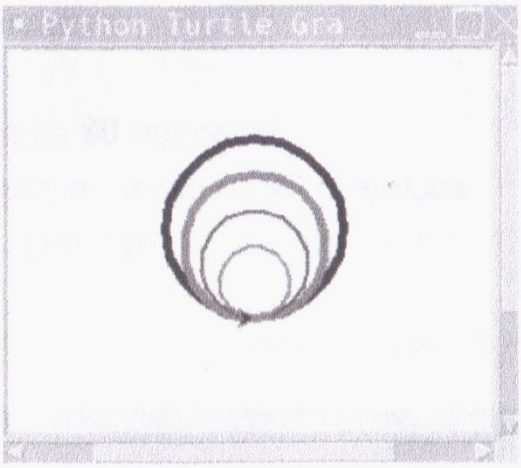
Таблица 9

Критерий оценивания	**1-я группа**	**2-я группа**	…
Актуальность темы			
Соответствие содержания проекта заявленной теме			
Техническая сложность разработанной программы			
Оригинальность алгоритма			
Дизайн интерфейса			
Степень разработанности программы			
Применение программы для решения аналогичных задач			
Итоговое количество баллов			

Темы проектов

```
turtle.circle(40)
turtle.pensize(2)
turtle.pencolor("blue")
turtle.circle(30)
turtle.pencolor("green")
turtle.circle(20)
```

Результат



Построение фракталов

У понятия «фрактал» нет строгого определения, поэтому слово «фрактал» не является математическим термином. Слово фрактал образовано от латинского fractus и в переводе означает «состоящий из фрагментов». Обычно так называют геометрическую фигуру. По определения Бенуа Мандельброта, «фракталом называется структура, состоящая из частей, которые в каком-то смысле подобны целому».

Роль фракталов в машинной графике сегодня достаточно велика. Они приходят на помощь, например, когда требуется, с помощью нескольких коэффициентов, задать линии и поверхности очень сложной формы. С точки зрения машинной графики, фрактальная геометрия незаменима при генерации искусственных облаков, гор, поверхности моря.

Из определения Мандельброта следует одно из основных свойств фракталов — самоподобие, то есть небольшая часть фрактала, содержит информацию обо всем фрактале. Фрактал строится рекурсивно из фигур, имеющих разный масштаб.

Понятие *L*-системы было введено А. Лидермайером. Формально *L*-система состоит из алфавита, аксиомы (инициализатора) и набора порождающих правил. В алфавит, в частности, входят символы:

F — перемещение на один шаг вправо, прорисовывая след;

b — перемещение на один шаг вправо, не прорисовывая след;

\+ — увеличение угла на заданную величину;

– — уменьшение угла на заданную величину.

Терл-графика является подсистемой вывода графического представления фрактального объекта.

Задача 20

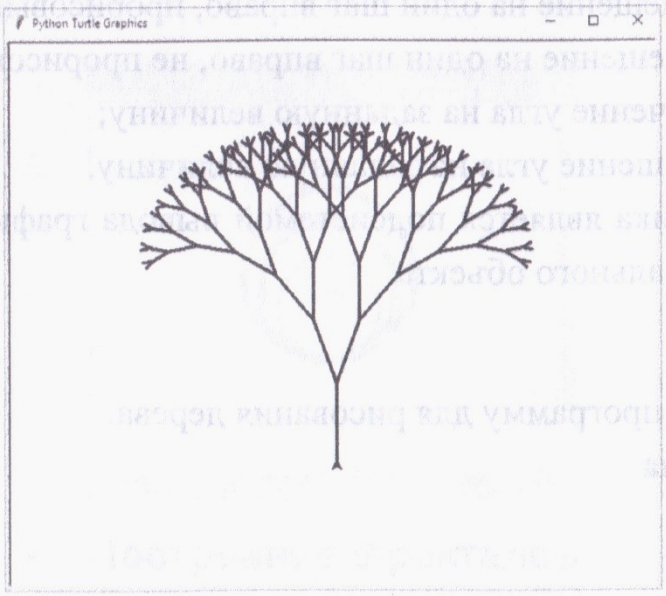
Составить программу для рисования дерева.

Программа

```
import turtle
def tree(branchLen,t):
    if branchLen > 3:
        t.forward(branchLen)
        t.right(20)
        tree(branchLen–10,t)
        t.left(40)
        tree(branchLen–10,t)
        t.right(20)
        t.backward(branchLen)
t = turtle.Turtle()
myWin = turtle.Screen() # определить холст
t.pencolor((0.001, 0.001, 0.001))
t.left(90)
t.up()
```

```
t.backward(125)
t.down()
t.color("green")
tree(80,t)
myWin.exitonclick()
```

Результат



Из рисунка видно, что дерево построено из однотипных элементов:

Задача 21

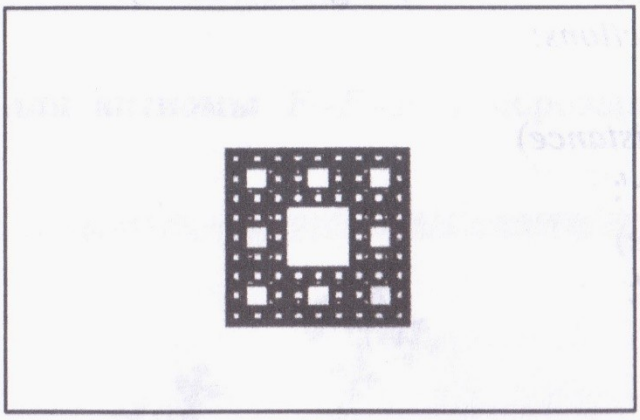
Составить программу для рисования ковра Серпинского.

Программа

```
import turtle
def s(n, m):
    if n == 0:
        turtle.color('deeppink')
```

```
        turtle.begin_fill()
        for _ in range(4):
            turtle.forward(m)
            turtle.left(90)
        turtle.end_fill()
    else:
        for _ in range(4):
            s(n – 1, m / 3)
            turtle.forward(m / 3)
            s(n – 1, m / 3)
            turtle.forward(m / 3)
            turtle.forward(m / 3)
            turtle.left(90)
turtle.ht()
turtle.tracer(10)
s(4, 100)
turtle.done()
```

Результат для числа итераций 4



Из рисунка видно, что ковер построен из однотипных элементов:

— число итераций 0:

Задача 23

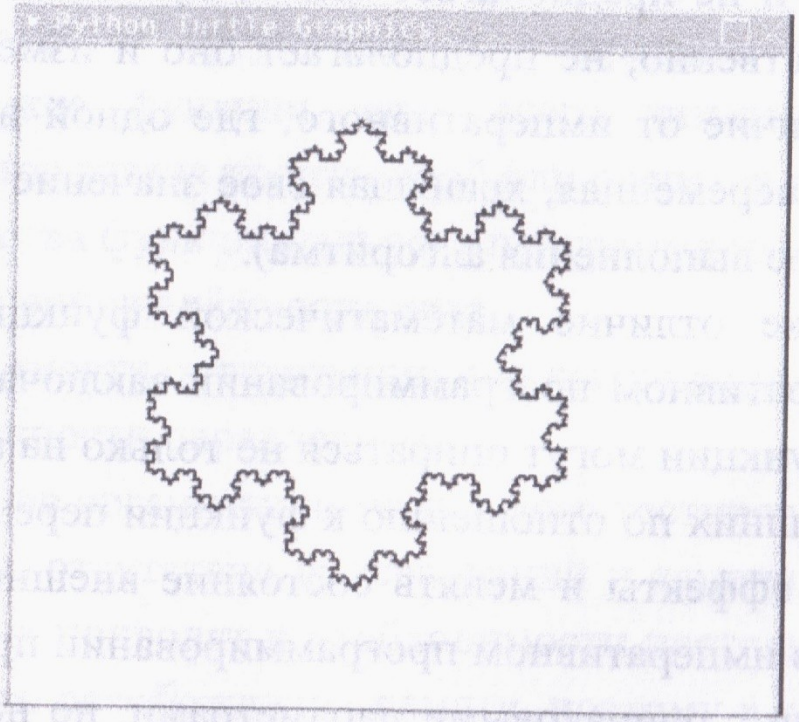
Составить программу для рисования снежинки Коха.

Программа

```
import turtle
def create_l_system(iters, axiom, rules):
    start_string = axiom
    if iters == 0:
        return axiom
    end_string = ""
    for _ in range(iters):
        end_string = "".join(rules[i] if i in rules else i for i in start_string)
        start_string = end_string
    return end_string
def draw_l_system(t, instructions, angle, distance):
    for cmd in instructions:
        if cmd == 'F':
            t.forward(distance)
        elif cmd == '+':
            t.right(angle)
        elif cmd == '-':
            t.left(angle)
axiom = "F++F++F"
rules = {"F":"F-F++F-F"}
iterations = 4
angle = 60
length=3
size=2
y_offset=-100
x_offset=-25
offset_angle=30
width=450
height=450
inst = create_l_system(iterations, axiom, rules)
t = turtle.Turtle()
```

```
wn = turtle.Screen()
wn.setup(width, height)
t.up()
t.backward(-x_offset)
t.left(90)
t.backward(-y_offset)
t.left(offset_angle)
t.down()
t.speed()
t.pensize(size)
draw_l_system(t, inst, angle, length)
t.hideturtle()
wn.exitonclick()
```

Результат



Литература:

1. Т.П. Никитина Программирование. Основы Python: учебное пособие для СПО/т.п. Никитина, Л.В. Королев. – Санкт-Петербург: Лань, 2023. – 156 с.: ил. – Текст: непосредственный. ISBN 978-5-507-45283-5
2. Р.А. Жуков Язык программирования Python: практикум: учебное пособие/ Р.А. Жуков. – Москва: ИНФРА-М, 2023.-216с. + Доп. материалы [Электронный ресурс]. – (Среднего профессиональное образование). ISBN 978-5-16-015638-5 (print) ISBN 978-5-16-108139-6 (online)
3. С.А. Чернышов Основы программирования на Python: учебное пособие для среднего профессионального образования/ С.А. Чернышов. – Москва: Издательство Юрайт, 2023. – 286с. – (Профессиональное образование). – Текст: непосредственный. ISBN 978-5-534-15160-2
4. https://elar.urfu.ru/bitstream/10995/28769/1/978-5-7996-1198-9_2014.pdf
5. https://topuch.com/moskovskij-priborostroitelenij-tehnikum-v10/index.html
6. https://habr.com/ru/articles/31180/