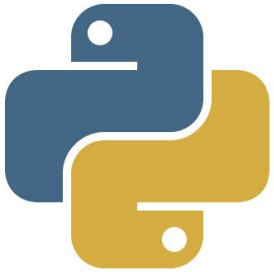
1. Введение в Python



Программа и программирование

Классическое определение, данное Н. Виртом:

«Алгоритмы + Структуры данных = Программы»

При разработке программы важно получить ответы на следующие вопросы:

- Можно ли решить поставленную задачу с помощью компьютера?
- Как можно решить поставленную задачу с помощью компьютера?
- Насколько эффективным будет решение (какие ресурсы для этого потребуются, как быстро будет решаться задача)?
- Каким образом можно оптимизировать решение, повысить эффективность (сэкономить ресурсы, ускорить решение)?

Понятие программы как способа записи алгоритма

Программа – это последовательность команд для ЭВМ, выполнение которых реализует заданный алгоритм.

Понятие программы обычно определяется в контексте некоторого *языка программирования*.

Программа есть алгоритм, реализованный таким способом и в такой форме, что вычисление алгоритма проводится автоматически. *Автоматическая вычислимость является неотъемлемым свойством программы*.

Программирование и программное обеспечение

Программирование (*кодирование*) – это перевод алгоритма на язык «понятных» ЭВМ команд. Таким образом, программа представляет собой способ записи алгоритма с использованием системы команд компьютера (машинных команд) или языка программирования.

Программное обеспечение (ПО) в современном понимании включает совокупность собственно программных средств, связанных с ними данных и программной документации.

Трансляторы: компиляторы и **интерпретаторы**

- *Транслятор* — это программа, которая переводит входную программу на исходном (входном) языке в эквивалентную ей выходную программу на результирующем (выходном) языке.
- *Компилятор* — это транслятор, который осуществляет перевод исходной программы в эквивалентную ей объектную программу на языке машинных команд или на языке ассемблера.
- *Интерпретатор* — это программа, которая воспринимает входную программу на исходном языке и выполняет ее.

Язык программирования

Язык программирования определяет правила представления программы в виде текста в конечном алфавите символов. Описание языка определяет:

- типы данных
- структуру памяти исполняемой программы
- виды языковых конструкций программы
- правила исполнения конструкции каждого вида и программы в целом.

История языка Python

- Язык программирования Python был создан примерно в 1991 году голландцем Гвидо ван Россумом.
- Для распределённой ОС Amoeba требовался расширяемый скриптовый язык, и Гвидо начал писать Python на в свободное от работы время.

Преимущества языка Python

Pascal	C++	Python

program number;
var i:integer;
begin
  for i:=0 to 9 do
  writeln(i);
end.
	#include <iostream>
	using namespace std;
	void main()
	{
	  int i;
	  for (i=0;i<=9;i++)
	  cout<<i<<“\n”;
	}
		for i in range(9):
		  print(i)

Особенности Python

- Скриптовый язык. Код программ определяется в виде скриптов.
- Поддержка самых различных парадигм программирования, в том числе объектно-ориентированной и функциональной парадигм.
- Интерпретация программ. Для работы со скриптами необходим интерпретатор, который запускает и выполняет скрипт.
- Кроссплатформенность. Не имеет значения ОС, главное наличие интерпретатора.

Влияние других языков на Python

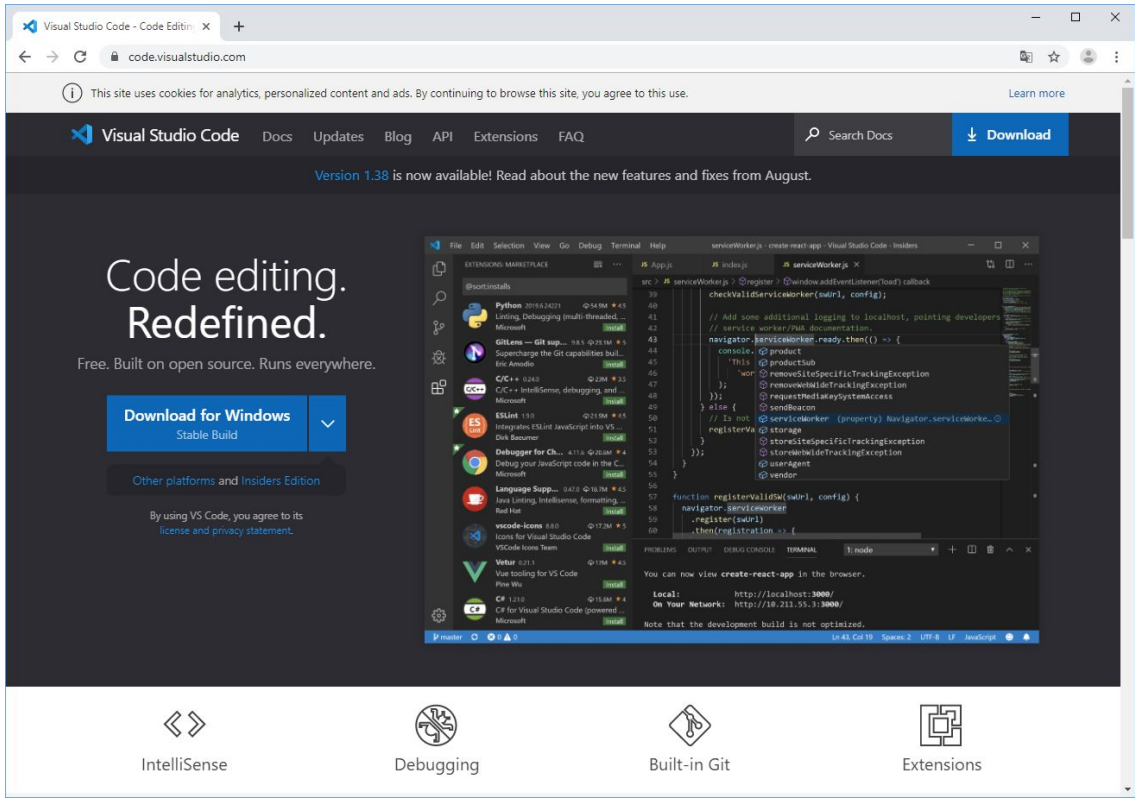
- ABC — отступы для группировки операторов, высокоуровневые структуры данных (map) (Python фактически создавался как попытка исправить ошибки, допущенные при проектировании ABC);
- Modula-3 — пакеты, модули, использование else совместно с try и except, именованные аргументы функций (на это также повлиял Common Lisp);
- C, C++ — некоторые синтаксические конструкции (как пишет сам Гвидо ван Россум — он использовал наиболее непротиворечивые конструкции из C, чтобы не вызвать неприязнь у C-программистов к Python);
- Smalltalk — объектно-ориентированное программирование;
- Lisp — отдельные черты функционального программирования (lambda, map, reduce, filter и другие);
- Fortran — срезы массивов, комплексная арифметика;
- Miranda — списочные выражения;
- Java — модули logging, unittest, threading (часть возможностей оригинального модуля не реализована), xml.sax стандартной библиотеки, совместное использование finally и except при обработке исключений, использование @ для декораторов;
- Icon — генераторы.

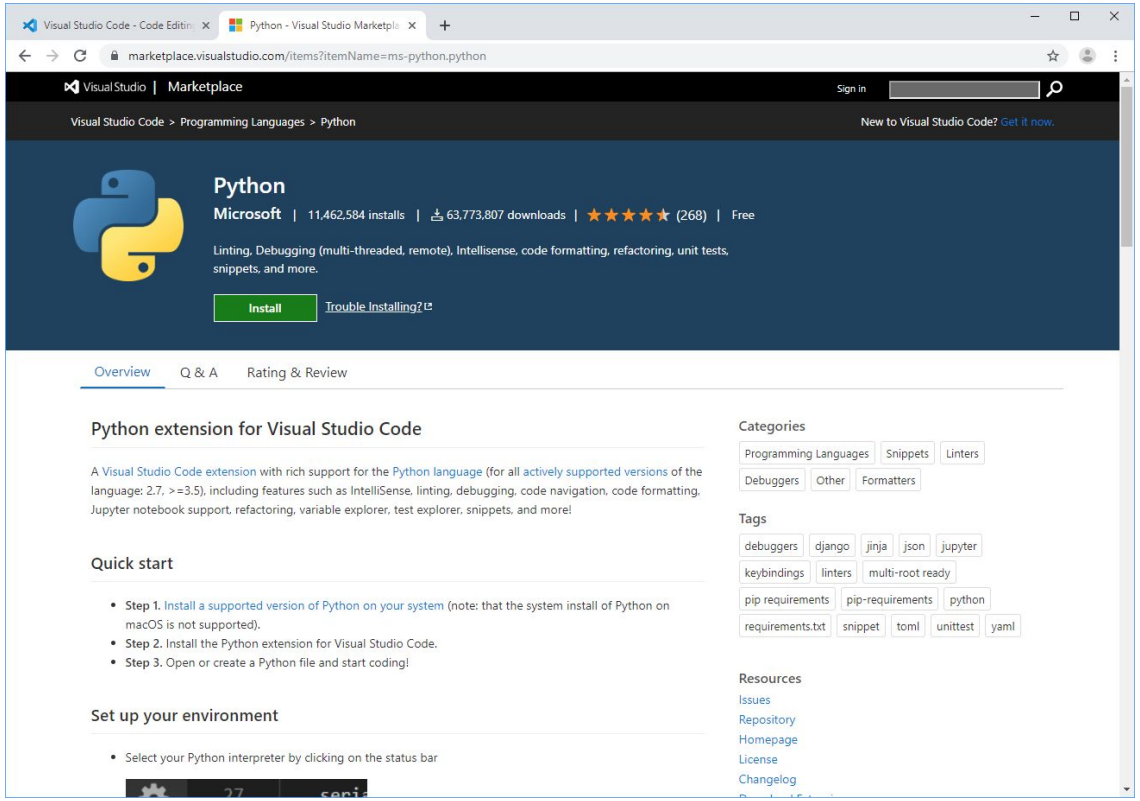
Дзен Python

1. Красивое лучше, чем уродливое.
2. Явное лучше, чем неявное.
3. Простое лучше, чем сложное.
4. Сложное лучше, чем запутанное.
5. Плоское лучше, чем вложенное.
6. Разреженное лучше, чем плотное.
7. Читаемость имеет значение.
8. Особые случаи не настолько особые, чтобы нарушать правила.
9. Должен существовать один — и, желательно, только один — очевидный способ сделать это.
10. Если реализацию сложно объяснить — идея плоха.

```
import this
```

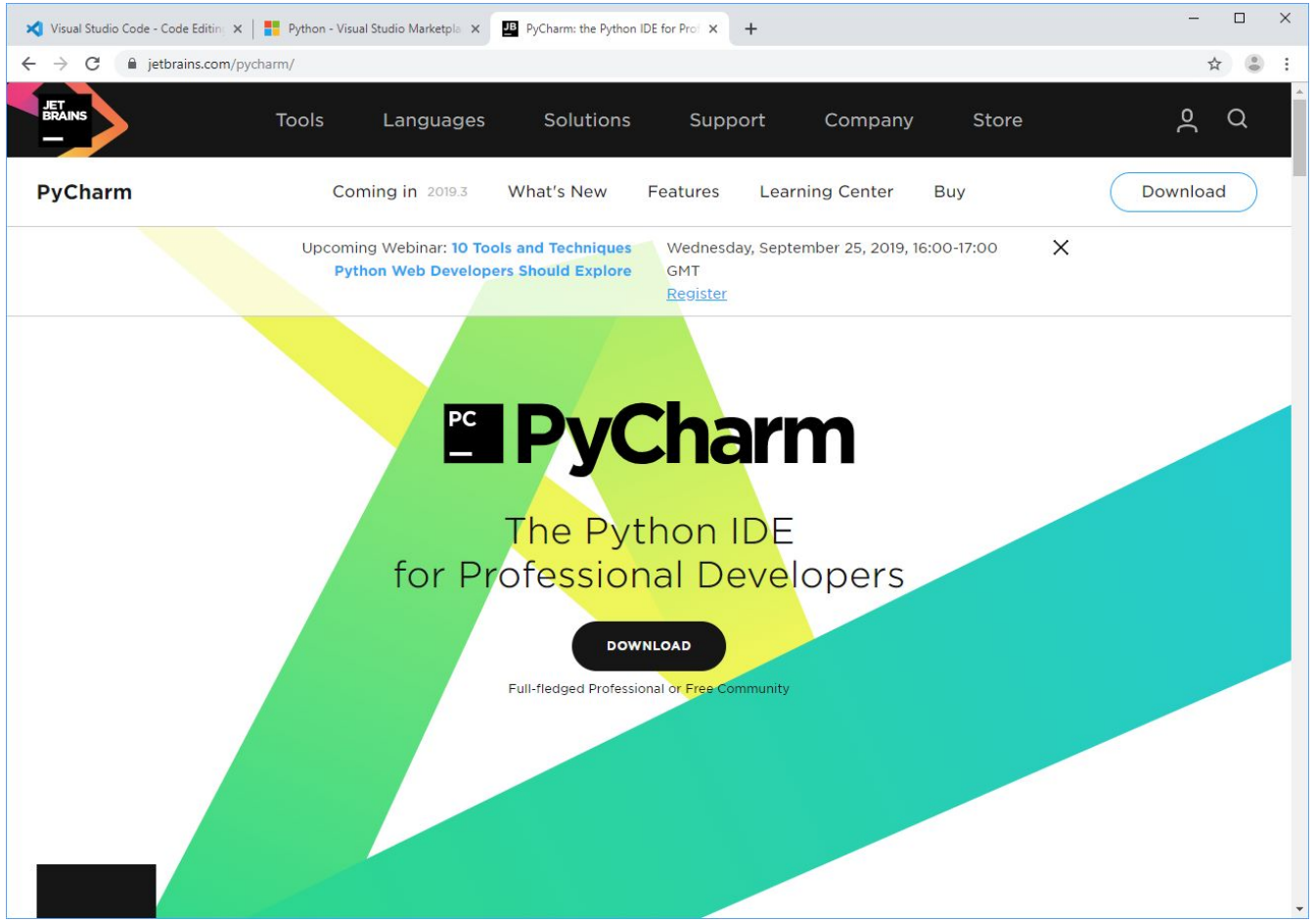
Visual Studio Code



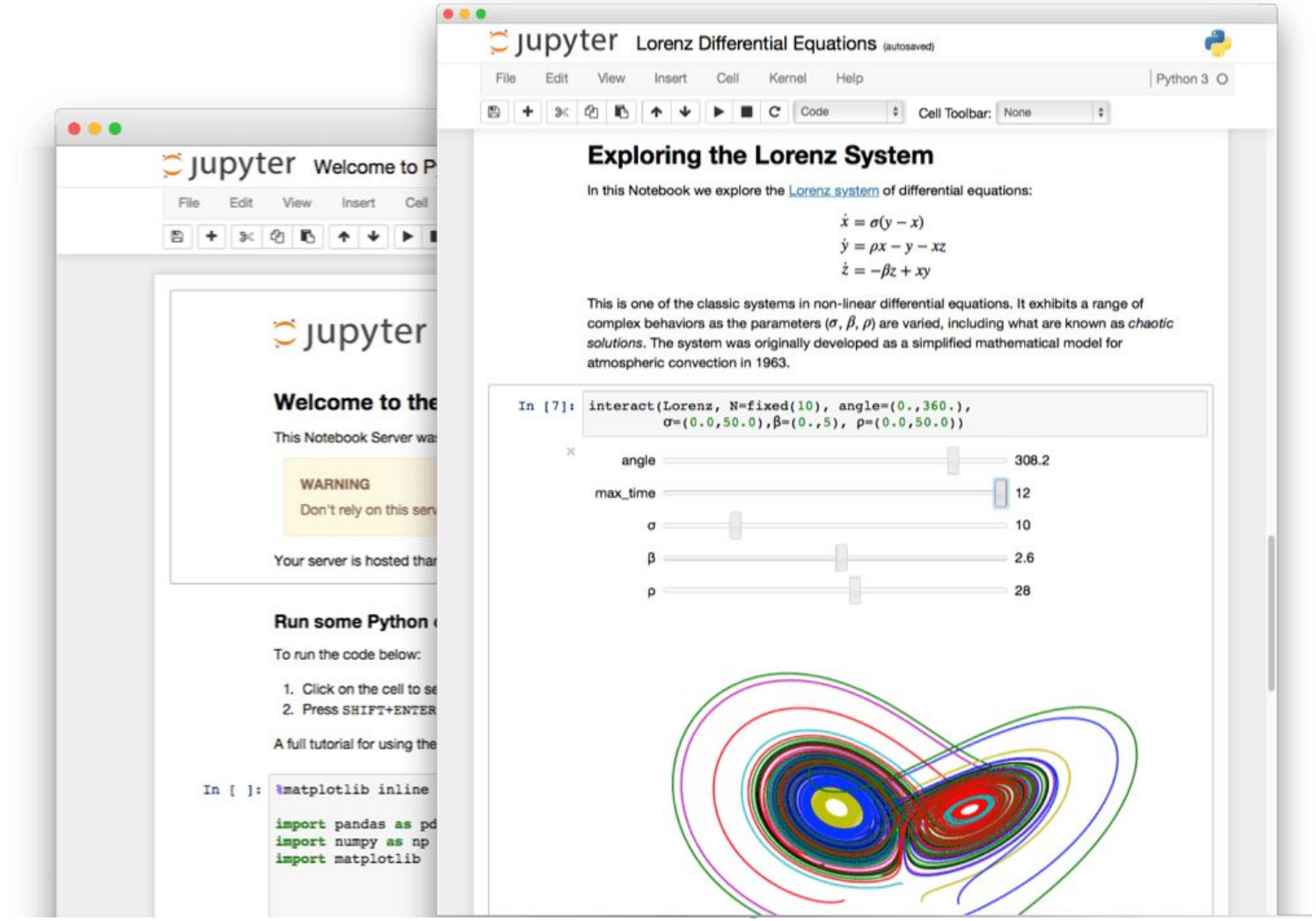


Python extension for Visual Studio Code https://marketplace.visualstudio.com/items?itemName=ms-python.python

PyCharm

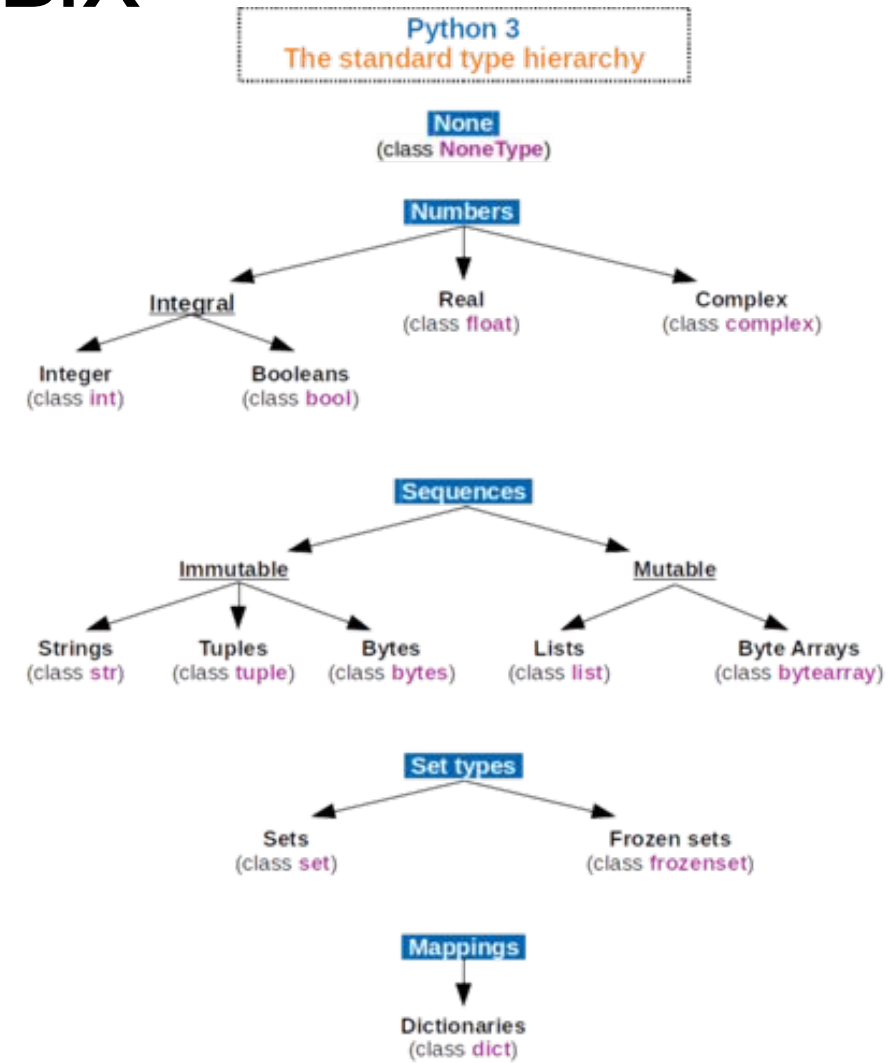

Visual Studio Code - Code Editing
Python - Visual Studio Marketpla
PyCharm: the Python IDE for Prof
jetbrains.com/pycharm/
JET BRAINS
Tools
Languages
Solutions
Support
Company
Store
PyCharm
Coming in 2019.3
What's New
Features
Learning Center
Buy
Download
Upcoming Webinar: 10 Tools and Techniques Python Web Developers Should Explore
Wednesday, September 25, 2019, 16:00-17:00 GMT
Register
PC PyCharm
The Python IDE for Professional Developers
DOWNLOAD
Full-fledged Professional or Free Community

Jupyter Notebook



Типы и структуры данных

- **Типы данных:**
  - числа (целые, вещественные, комплексные числа);
  - логические значения.
- **Структуры данных:**
  - строки;
  - кортежи;
  - списки;
  - множества;
  - словари.



Изменяемые и неизменяемые объекты

- **Неизменяемые (атомарные объекты):**
  - числа;
  - логические значения;
  - строки;
  - кортежи.
- **Изменяемые (ссылочные объекты):**
  - списки;
  - множества;
  - словари.

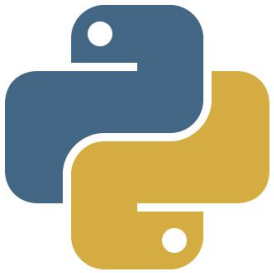
```
x = 0
y = x          #immutable
y+=1
print(x, y)   #0 1
x = []
y = x         #mutable
x.append(1)
print(x, y)   #[1] [1]
```

Динамическая типизация

- **Динамическая** (типы переменных определяются во время выполнения программы)
  строгая (нельзя смешивать в выражениях различные типы)
  неявная (задавать тип не надо).
- Конвертация типов.

```
a, b = '1', '2'
print(a + b)
#'12'
print(int(a) + int(b))
#3
print(int('12'))
#12
```

1.1. Базовые операции



Основные правила

- Программа на языке Python состоит из набора инструкций. Каждая инструкция помещается на новую строку.
- Python - регистрозависимый язык, поэтому выражения print, Print или PRINT представляют разные выражения.
- Используется нотация **under_score (snake_case)**, что слова в наименовании переменной разделяются знаком подчеркивания.
- Для отметки, что делает тот или иной участок кода, применяются комментарии знаком решетки **#**, для многострочного комментария ''' до и после.

```
user_name = 'Alex'
# Комментарий
print('Hello World')
'''
Многострочный
комментарий
'''
```

Арифметические операции

Арифметические выражения могут включать: переменные; знаки арифметических операций; вызовы функций; круглые скобки.

Арифметические операции

- Сумма: **+**
- Разница: **-**
- Произведение: ***
- Деление: **/**
- Остаток от деления: **%**
- Деление нацело: **//**
- Возведение в степень: ****

```
a=b=2
b=a+b+2
a=b*4
print(a)
#24
```

Оператор ввода

```
print('Введите число')
#Введите число
a=int(input())          # ввод значения переменной a
#25
c=a+2
print('c=', c)
#c=27
```

Оператор вывода

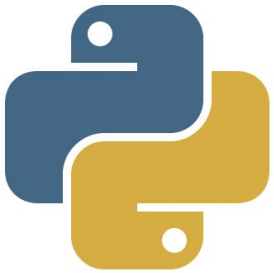
```
print(a)              # вывод значения переменной a
print('Hello!')       # вывод текста
print('Result: ', a)    # вывод текста и значения переменной a
print(a+b)              # вывод суммы чисел
```

Задачи

1. Сумма двух чисел.
2. Сумма трех чисел.
3. Напишите программу, которая считывает длины двух катетов в прямоугольном треугольнике и выводит его площадь. Каждое число записано в отдельной строке.
4. Напишите программу, которая приветствует пользователя, выводя слово Hello, введенное имя и знаки препинания по образцу: Hello, User!
5. N школьников делят k яблок поровну, неделящийся остаток остается в корзинке. Сколько яблок достанется каждому школьнику? Сколько яблок останется в корзинке?
6. Известно количество учащихся в каждом из трёх классов. Необходимо вычислить количество парт в кабинете, чтобы все учащиеся влезли в него. За каждой партой может сидеть не больше двух учеников.
7. Электронные часы. Дано число n. С начала суток прошло n минут. Определите, сколько часов и минут будут показывать электронные часы в этот момент. Программа должна вывести два числа: количество часов (от 0 до 23) и количество минут (от 0 до 59). Учтите, что число n может быть больше, чем количество минут в сутках.

```
a=int(input())
b=int(input())
c=a+b
print (c)
```

1.2. Конструкции управления потоком. Условия



Синтаксис условной инструкции

if <*Условие*>:
  <*Блок инструкций 1*>
else:
  <*Блок инструкций 2*>

```
x = int(input())
if x>0:
   print(x)
else:
   print(-x)
#Тернарный оператор
x = x if x>0 else -x
```

Вложенные условные инструкции

```
if x > 0:
   if y > 0:              # x > 0, y > 0
     print("Первая четверть")
   else:                  # x > 0, y < 0
     print("Четвертая четверть")
else:
   if y > 0:              # x < 0, y > 0
     print("Вторая четверть")
   else:                  # x < 0, y < 0
     print("Третья четверть")
```

Логические операторы

- Логическое И: **and**
- Логическое ИЛИ: **or**
- Логическое НЕ: **not**
- Больше: **>**
- Меньше: **<**
- Больше или равно: **>=**
- Меньше или равно: **<=**
- Равно: **==**
- Не равно: **!=**

```
a = int(input())
b = int(input())
if a % 10 == 0 or b % 10 == 0:
   result = True
else:
   result = False
print(result)
```

Каскадные условные инструкции

```
if x > 0 and y > 0:
   print("Первая четверть")
elif x > 0 and y < 0:
   print("Четвертая четверть")
elif y > 0:
   print("Вторая четверть")
else:
   print("Третья четверть")
```

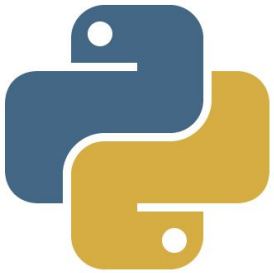
В такой конструкции условия if, ..., elif проверяются по очереди, выполняется блок, соответствующий первому из истинных условий. Если все проверяемые условия ложны, то выполняется блок else, если он присутствует.

Задачи

1. Минимум из двух чисел.
2. Минимум из трех чисел.
3. Шахматная доска.
   Заданы две клетки шахматной доски. Если они покрашены в один цвет, то выведите слово YES, а если в разные цвета — то NO. Программа получает на вход четыре числа от 1 до 8 каждое, задающие номер столбца и номер строки сначала для первой клетки, потом для второй клетки.
4. Сколько совпадает чисел.
   Даны три целых числа. Определите, сколько среди них совпадающих. Программа должна вывести одно из чисел: 3 (если все совпадают), 2 (если два совпадает) или 0 (если все числа различны).
5. Ход слона.
   Шахматный слон ходит по диагонали. Даны две различные клетки шахматной доски, определите, может ли слон попасть с первой клетки на вторую одним ходом.
6. Ход ладьи, короля, ферзя, коня.
7. Выполнить округление числа до тысячных в формате 123000 -> 123к.

```
a = int(input())
b = int(input())
if a<b:
   print(a)
else:
   print(b)
```

1.3 Строки



Строковый тип

- **str** – это последовательность символов.
- Определяется с помощью одинарных или двойных кавычек.

```
hse = "НИУ ВШЭ"
hse = 'НИУ ВШЭ'
hse = 'НИУ "ВШЭ"'
hse = "НИУ \"ВШЭ\""
print(hse)
#НИУ "ВШЭ"
```

Строковый тип

- Нумерация символов с нуля.
- Строки в языке Python невозможно изменить.

```
word = 'strength'
word[2] = 'r'
'''
TypeError: 'str' object does
not support item assignment
'''
```

Форматирование строк

```
a, b = 1, 2
print("%s != %s" % (a,b))
print("{} != {} ".format(a,b)
print("{0} != {1}".format(a,b)
print("{a} != {b}".format(a=a, b=b)
print(f"{a} != {b} ")
#1 != 2
```

Срезы

- Срез – это механизм управления строкой на основе индексации.
- substring = string[<начало среза>:<конец среза>].

```
word = 'strength'
print(word[0:2])
#st
print(word[2:4])
#re
```

Срезы

- Если в срезе опущен первый символ значит, он равен нулю.
- Если опущен последний символ – он равен длине строки.

```
word = 'strength'
print(word[:3])
#str
print(word[5:])
#gth
```

Срезы

- Можно выбирать подстроку с определенной шагом.
- substring = string[<начало>:<конец>:<**шаг обхода**>].

```
s = '1234567890'
print(s[::2])
#'13579'
print(s[1:10:2])
#'2468'
print( s[::-1])
# '0987654321'
```

Операции со строками

- Строк можно складывать и умножать.
- Строки можно сравнивать с помощью операторов <, <=, ==, !=, >, >= .

```
s = 'HSE' + ' Perm'
print(s)
# 'HSE Perm'
print('a' * 3)
# 'aaa'
```

Методы

- **len**(string) – длина строки;
- **string.find**(substring) – поиск подстроки в строке. Возвращает номер первого вхождения или -1;
- **string.rfind**(substring) – поиск подстроки в строке. Возвращает номер последнего вхождения или -1;
- **string.replace**('что', 'на что') – замена шаблона;
- **string.count**(substring) – подсчитывает количество вхождений одной строки в другую строку;
- **string.upper**() – преобразование строки к верхнему регистру;
- **string.lower**() – преобразование строки к нижнему регистр.

Байтовые строки

- Байт – минимальная единица хранения и обработки цифровой информации.
- **bytes** – это последовательность байтов.

```
print(b'bytes')
# b'bytes'
print('Байты'.encode('utf-8'))
#b'\xd0\x91\xd0\xb0\xd0\xb9\xd1\x82\xd1\x8b'
```

Задачи

1. Выведите строку в обратном порядке.
2. Выведите всю строку, кроме последних двух символов.
3. Выведите последние три символа строки.
4. Выведите символы с нечетными индексами, начиная со четвертого символа строки.
5. Разбить введенное число на разряды (тысячи, миллионы) с помощью пробела (10 234), точки (1.456.678), используя срезы.

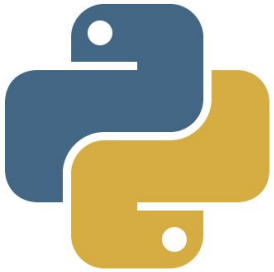
```
s = input()
print(s[::-1])
```

Задачи

6. Дана строка, заменить в ней подстроку 'Alex' на подстроку 'Bob'.
7. Дано предложение (состоящая из слов, разделенных пробелами). Выведите, сколько в предложении слов.
8. Дано словосочетание (состоящие из двух слов, разделенных пробелом). Переставьте эти слова местами.
9. Дана строка. Разделите ее на две части. Переставьте эти две части местами.
10. Вводится предложение с маленькой буквы, выведите предложение с большой буквы.
11. Перевести строку '+7-999-503-20-18' в строку '89995032018' и обратно.

```
s = input()
# Hello, Alex!
s = s.replace('Alex',
              'Bob'))
print(s)
#Hello, Bob!
```

1.4. Списки. Цикл for



Списки

- Для группировки множества элементов используются списки **list**.
- Для создания списка необходимо в квадратных скобках **[]** через запятую перечисляются все его элементы или в конструкторе **list()**.
- Списки имеют произвольную вложенность, поэтому могут включать в себя любые вложенные списки.

```
nums = [1, 3, 2, 1]
nums = list(1, 3, 2,1)
```

Списки

- Цикл **for** предназначен для перебора элементов.
- Для получения индекса элемента в цикле используется функция **enumerate**.

```
buses = ['13', '27', '18',
   '36', '30', '68']
for bus in buses:
   print(bus)
for i, bus in enumerate(buses):
   print(i, bus)
```

Функция range

- Функция **range** является генератором списков (начало последовательности, конец последовательности, [шаг, с которым генерируется значения]).
- lst = list(range(1, 3)) приведение к типу **list**.

```
for i in range(1, 3):
    print(i, end=' ')
#1 2
for i in range(0, 6, 2):
    print(i, end=' ')
#0 2 4
for i in range(0, -5, -1):
    print(i, end=' ')
#0 -1 -2 -3 -4
```

Встроенные функции

- **list.append(value)** — добавление элемента;
- **list.extend(list2)** — добавление списка;
- **list.insert(index, value)** — вставка;
- **list.index(value)** — найти индекс первого вхождения конкретного элемента;
- **list.count(value)** — подсчет повторов элемента;
- **list.remove(value)** — удаление элемента по значению;

- **list.pop(index)** — удаление элемента по индексу;
- **list.sort()** — сортировка;
- **list.reverse()** — перевернуть;
- **sorted(list)** — сортировка;
- **reversed(list)** — перевернуть;
- **len(list)** — длина списка;

Методы split и join

- Тип **str** имеет метод **split(),** который возвращает список подстрок по заданному разделителю (по умолчанию пробел).

```
s = input()
# '1 2 3'
print(s.split())
# ['1', '2', '3']
s = input()
# '127.0.0.1'
print(s.split('.'))
# ['127', '0', '0', '1']
```

Методы split и join

- Метод **join()** возвращает строку, полученную соединением элементов переданного списка с помощью разделителя, к которому применяется.

```
a = ['1', '2', '3']
print('!'.join(a))
# 1!2!3
```

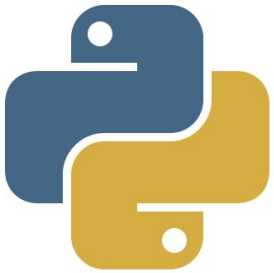
Задачи

1. Даны два целых числа A и B. Выведите все числа от A до B включительно, в порядке возрастания, если A < B, или в порядке убывания в противном случае.
2. Даны два целых числа A и B. Выведите все нечётные числа от A до B включительно, в порядке убывания. В этой задаче можно обойтись без инструкции if.
3. Дано несколько чисел. Вычислите их сумму. Сначала вводите количество чисел N, затем вводится ровно N целых чисел.
4. Факториалом числа n называется произведение 1 × 2 × ... × n. Обозначение: n! По данному натуральному n вычислите значение n!
5. Дано N чисел: сначала вводится число N, затем вводится ровно N целых чисел. Подсчитайте количество нулей среди введенных чисел и выведите это количество. Вам нужно подсчитать количество чисел, равных нулю, а не количество цифр.
6. По данному натуральному n ≤ 9 выведите лесенку из n ступенек, i-я ступенька состоит из чисел от 1 до i без пробелов.

```
a = int(input())
b = int(input())
arr = list(range(a,b+1))
for i in arr:
   print(i, end = ' ')
```

```
1
12
123
```

1.5. Цикл while



Цикл while

- Цикл **while** (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно.

```
while <условие>:
    <блок инструкций>
```

Цикл while

- Используется, когда невозможно определить точное значение количества проходов исполнения цикла.
- Пример использования цикла **while** для определения количества цифр натурального числа *n.*

```
n = int(input())
length = 0
while n > 0:
    n //= 10
    length += 1
print(length)

#length = len(input())
```

Управление циклами for и while

- Оператор **break** досрочно прерывает цикл.

```
while True:
    n = int(input())
    if n ==0:
        break
    print(n)
```

Управление циклами for и while

- Оператор **continue** начинает следующий проход цикла, минуя оставшееся тело цикла.

```
n = int(input())
while n != 0:
    if n % 2 == 0:
        n = int(input())
        continue
    print(n)
    n= int(input())
```

Управление циклами for и while

- После тела цикла можно написать команду **else**. Данный блок операций будет выполнен *1* раз после окончания цикла, если цикл завершен не командой **break**.

```
for i in 'hello world':
    if i == 'a':
        break
else:
    print("Буквы 'a' в строке нет")
```

Задачи

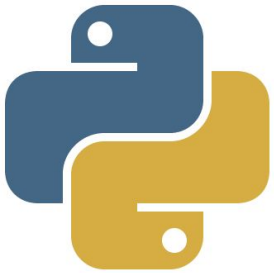
1. По данному целому числу N распечатайте все квадраты натуральных чисел, не превосходящие N, в порядке возрастания.
2. Программа получает на вход последовательность целых неотрицательных чисел, каждое число записано в отдельной строке. Определите кол-во, сумму и среднее значение всех элементов последовательности, завершающейся числом 0.
3. Дано целое число, не меньшее 2. Выведите его наименьший натуральный делитель, отличный от 1.
4. В первый день спортсмен пробежал *X* километров, а затем он каждый день увеличивал пробег на 10% от предыдущего значения. По данному числу *Y* определите номер дня, на который пробег спортсмена составит не менее *Y* километров.

```
n = int(input())
i = 1
while i**2 <= n:
    print(i**2)
    i+=1
```

Задачи

5. Последовательность состоит из различных натуральных чисел и завершается числом 0. Определите значение второго по величине элемента в этой последовательности. Гарантируется, что в последовательности есть хотя бы два элемента.
6. Последовательность состоит из натуральных чисел и завершается числом 0. Определите, сколько элементов этой последовательности равны ее наибольшему элементу.
7. Дана последовательность натуральных чисел, завершающаяся числом 0. Определите, какое наибольшее число подряд идущих элементов этой последовательности равны друг другу.
8. По данному числу n определите n-е число Фибоначчи.
9. Дано натуральное число A. Определите, каким по счету числом Фибоначчи оно является. Если A не является числом Фибоначчи, выведите число -1.

1.6. Генераторы списков



Списки

- Заполнение списков с помощью создания пустого списка.
- Списки поддерживают операции сложения и умножения.
- Для создания списка, заполненного одинаковыми элементами, можно использовать оператор повторения списка (умножения).

```
a = []
n = int(input())
for i in range(n):
    new_element = int(input())
    a.append(new_element)
print([7, 8] + [9])
print([0, 1] * 3)
```

Генераторы списков

- Другими словами, **списочные выражения**.
- Для создания списков, заполненных по более сложным формулам можно использовать *генераторы*: выражения, позволяющие заполнить список некоторой формулой.

[<выражение> for <переменная> in <последовательность>]

```
a = [input() for i in range(int(input()))]
```

Фильтрация

- Для фильтрации списка также можно использовать *генераторы*, указав условие по которому необходимо отобрать элементы.

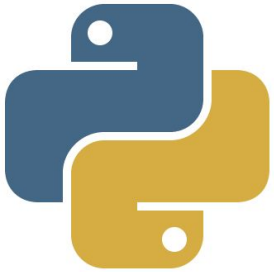
```
names = ['Alex', 'Bob', 'Mike', 'Alla']
names = [name for name in names if name.startswith('A')]
print(names)
#['Alex', 'Alla']
```

Задачи

1. Дан список чисел в одну строку. Вывеси отсортированный по убыванию список чисел в формате float.
2. Дан список чисел. Определите, сколько в этом списке элементов, которые больше двух своих соседей, и выведите количество таких элементов. Крайние элементы списка никогда не учитываются, поскольку у них недостаточно соседей.
3. Дан отсортированный список чисел и новое число, которое необходимо вставить в такое место в списке, чтобы он остался отсортированным.
4. Дан список чисел в одну строку через запятую. Подсчитать сумму модулей.

```
a = [float(s) for s in input().split()]
a.sort(reverse=True)
```

1.7. Двумерные массивы



Двумерный массив

- В Python матрицу можно представить в виде списка списков, каждый элемент которого является в свою очередь, например, числом.

```
n = int(input())
a = []
for i in range(n):
   a.append([])
   for j in range(n):
     a[i].append(input())
```

Двумерный массив

- Для создания двумерного массива можно также использовать *генераторы*, но такая запись имеет плохую читаемость.

```
n = int(input())
a = [[0 for i in range(n)] for i in range(n)]
```

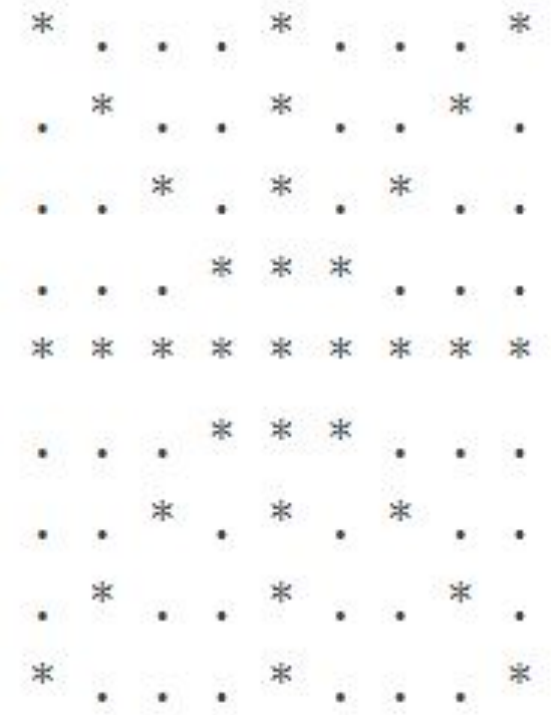
Обращение к массиву по индексам

- Для обращения к элементам списка используются индексы: [<первый уровень вложенности>] [<второй уровень вложенности>] и т.д.
- Для двумерного массива [<строка>][<столбец>].

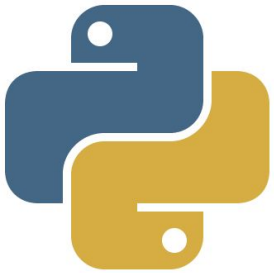
```
for i in range(n):
    for j in range(n):
    a[i][j] = 2
```

Задачи

1. Считать размерность двумерного массива и сам массив. Найти индексы максимального числа.
2. Вводится размерность массива. Вывести массив снежинки заданного размера.
3. Заполнить двумерных массив в шахматном порядке (черный – 1, белый – 0).
4. Поменять местами первый и последний столбец.
5. Сдвинуть строки на *n* строк вниз.



1.8. Функции



Функции

- Функции – это многократно используемые фрагменты программы.
- Функция начинается с ключевого слова **def**, названия функции и двоеточия.

```
def <название>:
    <блок инструкций>
```

Функции

- Ключевое слово **return** возвращает результат работы функции.
- Функция может возвращать любые типы объектов (переменные, списки, функции), а также произвольное их кол-во.
- Функция может и не заканчиваться ключевым словом **return**, при этом функция вернет значение **None**.

```
def sum_2(x, y):
    'summarizes two elements'
    return x + y

print(sum_2(1, 10))
#11
```

Аргументы функции

- Функция может принимать произвольное количество аргументов или не принимать их вовсе.
- Также распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

```
def sum_3 (a, b, c=2):
   return a + b + c


print(sum_3(1, 2, 3))
#6
print(sum_3(1, 2))
#5
print(sum_3(b=1, a=2))
#5
```

Аргументы функции

- Произвольное количество параметров обозначается звездочкой перед аргументом *args.
- Произвольное количество именованных аргументов обозначается двумя звездочками перед аргументом **kwargs.

```
def many(*args, **kwargs):
    print(args)
    print(kwargs)
many(1, 2, 3, name="Mike",
job="programmer")
'''(1, 2, 3)
{'job': 'programmer',
'name': 'Mike'} '''
```

Локальные переменные

- При объявлении переменных внутри функции, они не связаны с другими переменными с таким же именем за пределами функции т.е. являются *локальными* в функции.
- *Область видимости* всех переменных ограничена блоком, в котором они объявлены, начиная с точки объявления имени.

```
x = 50
def func():
   x = 2

func()
print(x)
#50
```

Глобальные переменные

- Для изменения переменной, определенной вне функции, необходимо указать, что данная переменная глобальная с помощью ключевого слова **global**.
- Используя одно ключевое слово global, можно объявить сразу несколько переменных: **global** x, y, z.

```
x = 50
def func():
   global x
   x = 2

func(x)
print(x)
#2
```

Передача аргументов в функцию

- Если вы передаете неизменяемый объект функции, вы по-прежнему не можете восстановить внешнюю ссылку, и вы не можете даже мутировать объект.

```
#immutable
def clear_int(num):
    num = 0

number = 10
clear_int(number)
print(number)
#10
```

Передача аргументов в функцию

- Если вы передаете изменяемый объект в функцию, функция получает ссылку на тот же объект, и вы можете мутировать его в памяти, но если вы переустановите ссылку в функции, внешний масштаб не будет знать ничего об этом, и после того, как вы закончите, внешняя ссылка все равно укажет на исходный объект.

```
#mutable
def clear_dict(d):
    #d = {}         #does not work
    d.clear()

dct = {1: '1'}
clear_dict(dct)
print(dct)
#{}
```

Анонимные функции

- Анонимная функция — особый вид функций, которые объявляются в месте использования и не получают уникального идентификатора для доступа к ним.
- Анонимные функции могут содержать лишь одно выражение, но и выполняются они быстрее. Анонимные функции создаются с помощью ключевого слова **lambda**.

```
square = lambda x: x**2
print(square(2))
#4
print((lambda x,y: x*y)(10,20))
#200
```

Функция sort()

- Функция **sort**() принимает атрибут **key** для сортировки. Этот ключ должен быть функцией, которая принимает один параметр, а возвращает значения по которому нужно

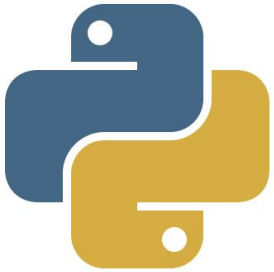
```
lst = ['a.2', 'b.1', 'c.3’]
print(sorted(lst, key=lambda x: int(x[2:])))
# ['b.1', 'a.2', 'c.3']
lst.sort(key=lambda x: int(x[2:]), reverse=True)
print(lst)
```

Задачи

1. Написать функцию вычисления площади прямоугольника. Входные параметры: ширина, высота. Выходной параметр: площадь.
2. Написать функцию вычисления площади треугольника. Входные параметры: основание, высота. Выходной параметр: площадь.
3. Написать функцию вычисления площади круга. Входные параметры: радиус. Выходной параметр: площадь.
4. Даны четыре действительных числа: x1, y1, x2, y2. Напишите функцию distance(x1, y1, x2, y2), вычисляющая расстояние между точкой (x1,y1) и (x2,y2). Выведите результат работы этой функции.
5. Напишите функцию capitalize(), которая принимает слово из маленьких латинских букв и возвращает его же, меняя первую букву на большую. На вход подаётся строка, состоящая из слов, разделённых одним пробелом. Выведите строку, где каждое слово начиналось с большой буквы.
6. Найти расстояние между точками в *n*-мерном пространстве.

```
def rectangle(a,b):
    return a*b
```

1.9. Файлы. Исключения



Файлы

При работе с файлами необходимо соблюдать некоторую последовательность операций:

1. Открытие файла с помощью метода **open().**
2. Чтение файла с помощью метода **read()** или запись в файл посредством метода **write().**
3. Закрытие файла методом **close().**

Файлы

Режимы открытия текстовых файлов:
'r' — открытие на чтение (является значением по умолчанию);
'w' — открытие на запись, содержимое файла удаляется, если файла не существует, создается новый;
'a' — открытие на дозапись, информация добавляется в конец файла;
'+' — открытие на чтение и запись.

```
f = open('text.txt', 'r')
```

Чтение из файла

Пример содержимого файла text.txt:

Hello world! \n

\n

The end. \n

```
f = open('text.txt')
f.read(1)
#'H'
f.read()
#'ello world!\nThe end.\n'
```

Указатель в файле

- Метод **tell()** сообщает в скольких байтах от начала файла сейчас находится указатель.
- Для перехода на нужную нам позицию, следует использовать метод **seek()**.

```
f = open('text.txt')
f.read(10)
print ('Текущая позиция:', f.tell())
f.close()
```

Построчное чтение

- Для построчного чтения файла можно использовать конструкцию **for**.

```
f = open('text.txt')
for line in f:
   print(line)

#'Hello world!\n '
# '\n'
#'The end.\n'
```

Построчное чтение

- Для чтения одной строки можно использовать метод **readline()**.
- Прочитать все строки и вернуть список можно с помощью метода **readlines()**.

```
f = open('text.txt')
lines = f.readlines()
f.close()
```

Запись в файл

- Для записи данных в файл используется метод **write()**.
- Метод **writelines()** позволяет записать сразу список значений в файл.

```
f = open('text.txt', 'w')
lst = [1, 2]
f.writelines(lst)
f.close()
```

Обработка исключительных ситуаций

- Уязвимый код заключается в блок **try**, после которого следует блок **except**, которому может задаваться возможная ошибка и реакция на нее.

```
try:
    a = float(input("Введите число:"))
except:
    print ("Это не число!")
```

Получение информации об исключении

- С помощью оператора **as** мы можем передать всю информацию об исключении в переменную, которую затем можно использовать в блоке **except**.

```
try:
    number = int(input("Введите число: "))
    print("Введенное число:", number)
except ValueError as e:
    print("Сведения об исключении", e)
print("Завершение программы")
```

Обработка исключительных ситуаций

- При открытии файла или в процессе работы с ним мы можем столкнуться с различными исключениями.

```
try:
    f = open('text.txt')
    lines = f.readlines()
except:
    print ("Ошибка при чтении файла")
```

Обработка исключительных ситуаций

- К блоку **except** можно добавить необязательный блок **else**, который сработает в случае, если программа выполнилась без ошибок.
- Еще один необязательный блок **finally**, который сработает независимо от того, выполнился код с ошибками или без.

```
try:
    a = float(input("Введите число: ")
    print (100 / a)
except:
    print ("Ошибка.")
else:
    print ("Без ошибок.")
finally:
    print ("Выполняется в любом случае!")
```

Обработка исключительных ситуаций

- В данном случае вся работа с файлом идет во вложенном блоке **try**. И если вдруг возникнет какое-либо исключение, то в любом случае в блоке finally файл будет закрыт.

```
try:
    f = open('text.txt')
    a = float(f.readline())
    print (100 / a)
except:
    pass
finally:
    f.close()
```

Менеджеры контекста

- Менеджеры контекста позволяют компактно управлять ресурсами (файлами) вместо конструкции **try-finally**.
- Основное преимущество использования **with** — это гарантия закрытия файла вне зависимости от того, как будет завершён вложенный код.

```
f = open("text.txt", "w")
try:
    f.write("Hello world!")
finally:
    f.close()

with open("text.txt", "w") as f:
    f.write("Hello world!")
```

Задачи

1. Считать текст из файла input.txt и записать в файл output.txt.
2. Создать файл results.txt, куда записать результаты игроков считая имя игрок и его результат с клавиатуры.
3. Считать результаты игроков из файла results.txt и вывести на экран лучшего игрока по количеству баллов.
4. Считать результаты игроков из файла results.txt и записать в файл output.txt. Рейтинг игроков по возрастанию их количеству баллов.

Ввод:

Alex
55
Bob
32

Входной файл:

Alex: 55
Bob: 32
Nick: 88

Выходной файл:

Alex: 55
Bob: 32

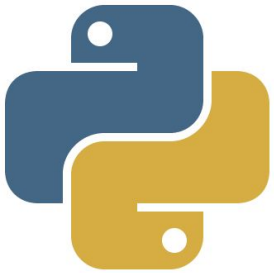
Входной файл:

Alex: 55
Bob: 32
Nick: 88

Вывод:

Nick: 88

1.10. Модули. Пакеты



Модули

- По мере возрастания программы у Вас наверняка появится необходимость разбить ее на несколько файлов, чтобы их было легче поддерживать.
- Может возникнуть необходимость в многократном использовании написанных функций в нескольких программах, не копируя их определения в каждую из программ.

Модули

- Модуль — это файл, содержащий определения и операторы **Python**.
- Именем файла является имя модуля с расширением **.py**.
- Для импортирования модуля используется команда **import**.

start.py

```
def hello():
   return 'Hello world!'
```

main.py

```
import start
print(start.hello())
```

Стандартные модули

- За один раз можно импортировать сразу несколько модулей, для этого их нужно перечислить через запятую после слова **import.**

```
import math, datetime
print(math.factorial(5))
#120
print(datetime.datetime.now())
#2018-04-16 17:05:00.000000
```

Импорт модулей

- При импорте модуля с помощью ключевого слова **as** можно модуль его переименовать.
- Для импорта отдельных функций используется конструкция **from** <модуль> **import** <функция>.
- Аргумент * позволяет импортировать все функции модуля.

```
from math as m
print(m.pi)
```

```
from math import pi
print(pi)
```

```
from math import *
print(pi)
```

Пакеты

- Пакет в **Python** – это каталог, включающий в себя другие каталоги и модули. Пакеты позволяют работать с модулями в виде иерархии через указание уровня вложенности (через точку).
- Для импортирования пакетов используется тот же синтаксис, что и для работы с модулями.

```
TCP/
    _init_.py
    main.py
    Server/
        _init_.py
        tcp.py
        server.py lib.py
    Client/
        _init_.py
        tcp.py
        client.py
        lib.py
```

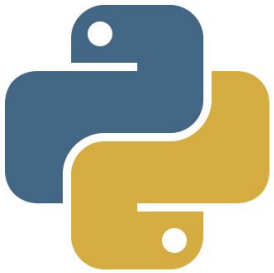
Библиотека requests

- Библиотека requests позволяет выполнять HTTP-запросы и получать информацию об их выполнении

r.status_code	Код статуса ответа от сервера
r.headers	Заголовки страницы
r.text	Текст страницы

```
import requests
r=requests.get('https://perm.hse.ru')
print(r.text)
```

1.11. Кортежи. Множества. Словари



Кортежи

- Кортеж **tuple** представляет последовательность неизменяемых элементов.
- Для создания кортежа необходимо перечислить все его элементы через запятую в скобках (можно и без них) или в конструкторе **tuple()**.
- Аналогичный функционал, что и у списков за исключением добавления, изменения и удаления элементов.
- Кортежи можно рассматривать как списки доступные только для чтения.

```
#nums = tuple(<массив>)
nums = 1, 3, 2, 1
nums = (1, 3, 2, 1)
nums = tuple([1, 3, 2, 1])
print(nums)
#(1, 3, 2, 1)
```

Множества

- Множество **set** представляет последовательность уникальных элементов.
- Для создания множества необходимо перечислить все его элементы через запятую в фигурных скобках **{}** или в конструкторе **set()**.

```
#nums = set(<массив>)
nums = {1, 3, 2, 1}
nums = set([1, 3, 2, 1])
print(nums)
#{1, 2, 3}
```

Множества

- Для добавления одиночного элемента вызывается метод **add()**.
- Для удаления одного элемента вызывается метод **remove().**
- Для перебора элементов можно использовать цикл **for**.
- Проверка наличия элемента в множестве с помощью **in.**

```
nums = set([1, 3, 2, 1])
nums.add(7)
nums.remove(3)
print(1 in nums)
#True
```

Операции со множествами

Операции со множествами

`A | B`	`A.union(B)`	Возвращает множество, являющееся объединением множеств `A` и `B`.
`A |= B`	`A.update(B)`	Добавляет в множество `A` все элементы из множества `B`.
`A & B`	`A.intersection(B)`	Возвращает множество, являющееся пересечением множеств `A` и `B`.
`A &= B`	`A.intersection_update(B)`	Оставляет в множестве `A` только те элементы, которые есть в множестве `B`.
`A - B`	`A.difference(B)`	Возвращает разность множеств `A` и `B` (элементы, входящие в `A`, но не входящие в `B`).
`A -= B`	`A.difference_update(B)`	Удаляет из множества `A` все элементы, входящие в `B`.
`A <= B`	`A.issubset(B)`	Возвращает `True`, если `A` является подмножеством `B`.

Словари

- Словарь **dict** — это неупорядоченное множество пар ключ: значение.
- Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется словарем.
- Объявляется в фигурных скобках **{}** или в конструкторе **dict()**.

```
capitals = dict()
capitals['Russia'] = 'Moscow'
capitals = {'Russia': 'Moscow'}
```

Операции со словарями

- **items()** — возвращает список ключей и значений.
- **keys()** — возвращает список ключей.
- **values()** — возвращает список значений
- **pop()** — извлекает значение по ключу с последующим удалением.
- **update()** — объединяет два словаря.

```
d = {}
for key, value in d.items():
    print(key, value)
```

Комплексные словари

- Кроме простейших объектов чисел и строк словари также могут хранить и более сложные объекты - списки, кортежи или другие словари.

```
users = {
   "Tom": {
     "phone": "+971478745",
     "email": "tom12@gmail.com"
     }
}
```

Задачи

1. Отсортировать список автобусов по номеру маршрута, по госномеру.
2. Определить сколько автобусов на каждом маршруте.

```
buses = [{
        'number': 'А568ОР159',
        'route': '13',
        'description': 'пл. Дружбы - Нагорный'},
    {
        'number': 'А356ВК59',
        'route': '1т',
        'description': 'Автовокзал - Большое Савино'
    }]
```

Задачи

3. Дан текст. Выведите все слова, встречающиеся в тексте и укажите сколько раз оно встретилось в тексте.
4. Дан текст в файле. Выведете ТОП-3 самых популярных словосочетаний в тексте.
5. Вам дан словарь, состоящий из пар слов. Каждое слово является синонимом к парному ему слову. В последней строке записано одно слово, необходимо определить все его синоним.

Задачи

4. Для каждого файла известно, с какими действиями можно к нему обращаться: запись W, чтение R, запуск X. В первой строке содержится число N — количество файлов содержащихся в данной файловой системе. В следующих N строчках содержатся имена файлов и допустимых с ними операций, разделенные пробелами. Далее указано число M — количество запросов к файлам. В последних M строках указан запрос вида Операция Файл. К одному и тому же файлу может быть применено любое количество запросов. Вам требуется восстановить контроль над правами доступа к файлам (ваша программа для каждого запроса должна будет возвращать OK если над файлом выполняется допустимая операция, или же Access denied, если операция недопустима.

```
4
helloworld.exe R X
pinglog W R
nya R
goodluck X W R
2
read nya
write helloworld.exe
```

Задачи

5. Выборы. Перечислено кол-во проголосовавших жителей каждого города за различных кандидатов президентов. Необходимо рассчитать кто сколько наберет процентов. Сообщить итоги выборов (кандидата победителя или переход к второму туру).
6. Даны ударения слов с помощью верхнего регистра (plAy), затем вводится предложение. Необходимо расставить в нем ударения.
7. Роберт играет в компьютерные игры. На протяжении *N* часов компьютер каждый час фиксирует информацию о том, играл ли Роберт в компьютер. Необходимо сказать в каком периоде *K* часов Роберт играл большего всего.

2

Perm 3

Ivanov 40

Petrov 11

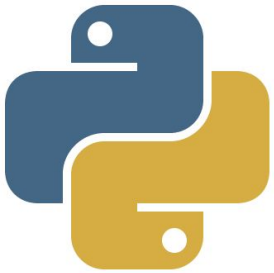
Sidorov 9

Moscow 2

Ivanov 115

Pavlov 50

1.12. Тестирование



Модульные тесты

- **unittest** – это библиотека для тестирования, входящий в стандартную библиотеку языка Python.
- Вначале теста формируются входные данные к тестируемой функции.
- Каждый тест заканчивается сравнением ожидаемого и полученного результата.

```
import unittest
from math import factorial as fac
class FactTest(unittest.TestCase):
    def test_small(self):
     input_data = 3
        expected = 6
        actual = fac(input_data)
        self.assertEqual(actual, expected)
```

Ожидаемые и полученные значения

Модуль **unittest** предоставляет множество функций для самых различных проверок:

- assertEqual(a, b) — a == b
- assertNotEqual(a, b) — a != b
- assertTrue(x) — bool(x) is True
- assertFalse(x) — bool(x) is False
- assertIs(a, b) — a is b
- assertIsNot(a, b) — a is not b
- assertIsNone(x) — x is None
- assertIsNotNone(x) — x is not None
- assertIn(a, b) — a in b
- assertNotIn(a, b) — a not in b
- assertMultiLineEqual(a, b) — строки (strings)
- assertSequenceEqual(a, b) — последовательности (sequences)

Пример тестов для программы calc

calc.py

```
def add(a, b):
   return a + b
def sub(a, b):
   return a-b
def mul(a, b):
   return a * b
def div(a, b):
   return a / b
```

tests.py

```
import unittest
import calc
class CalcTest(unittest.TestCase):
   def test_add(self):
     self.assertEqual(calc.add(1, 2), 3)
   def test_sub(self):
   self.assertEqual(calc.sub(4, 2), 2)
```

Параметризированные тесты

- Если у тестов разница лишь во входных и выходных параметрах, то их преобразовывают в параметризированные тесты.

```
import unittest
from math import factorial as fac
class FactTest(unittest.TestCase):
    def test_numbers(self):
        input_data = [3, 5, 10]
        expected_data = [6, 120, 3628800]
        for data, expected in zip(input_data, expected_data):
            with self.subTest():
                actual = fac(data)
                self.assertEqual(actual, expected)
```

Негативные тесты

- Негативным называют тестирование, в рамках которого применяются сценарии, которые соответствуют внештатному поведению тестируемой системы.

```
import unittest
import calc
class CalcTest(unittest.TestCase):
    def test_div_error(self):
        self.assertRaises(ZeroDivisionError, calc.div, 1, 0)
```

Ссылки

- http://pythontutor.ru/
- http://rus-linux.net/MyLDP/BOOKS/python.pdf
- https://www.coursera.org/learn/diving-in-python
- http://catbo.net/c/translated/Python-enru/Python.zip/tutorial/index.html
- https://younglinux.info/python.php
- https://devpractice.ru/
- https://www.yuripetrov.ru/edu/python/ch_intro.html