Язык программирования Python

В 1989 году началось создание языка программирования Python Гвидо ван Россумом. Назван он был в честь любимого шоу создателя– «Летающий цирк Монти Пайтона» («Monty Python's Flying Circus»). Существует 3 версии Python. В настоящее время самым актуальным является Python 3.

Python – это интерпретируемый, универсальный и передовой язык программирования, который используется в следующих областях:

- анализ данных;
- написание прикладных программных систем;
- разработка графических интерфейсов;
- написание сетевого программного обеспечения;
- взаимодействие с базами данных.

К особенностям Python относится *динамическая типизация*. Типы объектов во время выполнения программы строго не определены, они могут изменяться. Так же для определения переменных не нужно объявлять их тип.

Для выделения блоков кода используется «:».

Python позволяет писать операции на одной строке с помощью разделителя «;», но делать так не рекомендуется, такой код достаточно сложно читать.

Однострочный комментарий в Python начинается с #, а многострочный выделяется с помощью “…”.

Python поддерживает функциональное программирование.

Существуют два режима запуска Python:

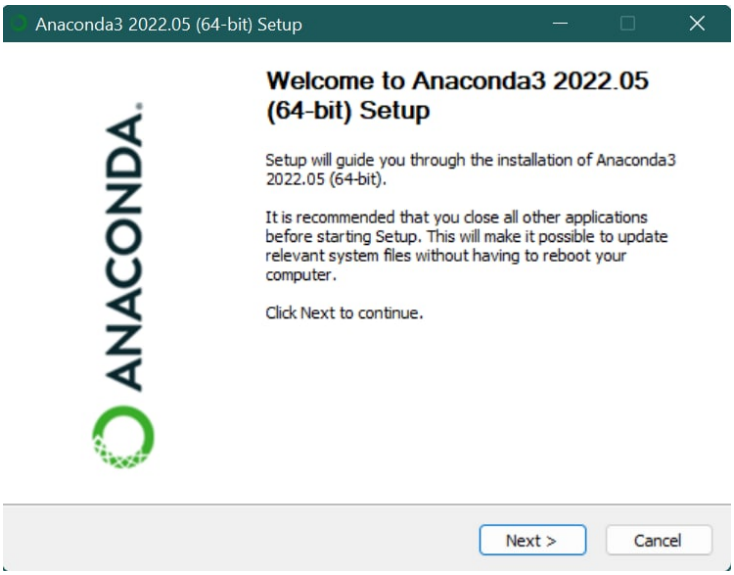
1. Стандартный режим. Необходимо сохранить файл с кодом с расширением .py, затем подать этот файл на вход интерпретатору для запуска кода.
2. Интерактивный режим. Результат работы команд виден сразу.

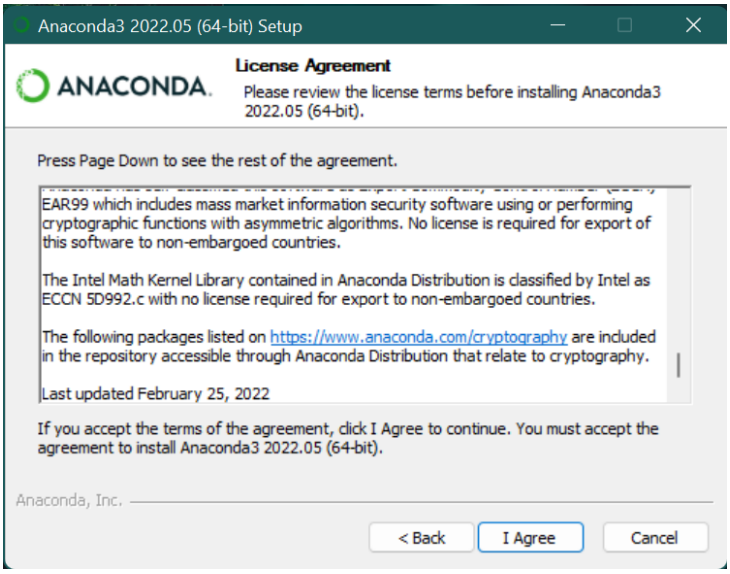
Установка Python

Один из самых простых способов установки Python – через дистрибутив Anaconda. Он содержит не только Python, но и необходимые библиотеки.

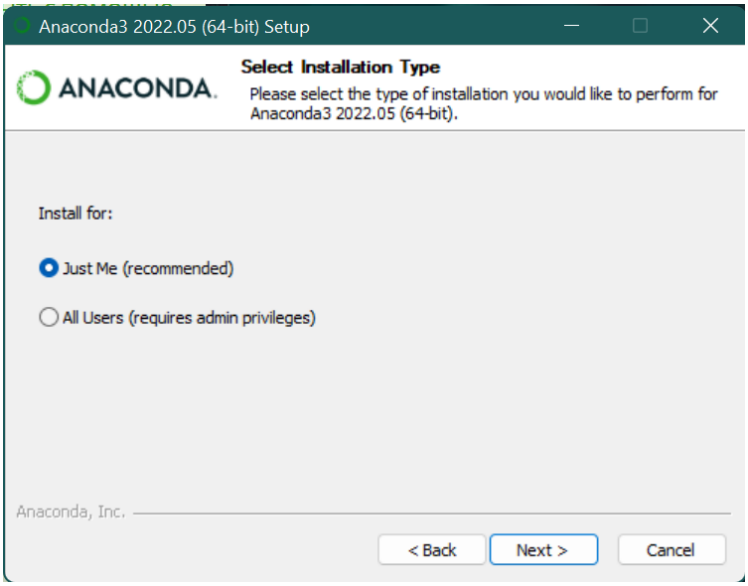
Скачать Anaconda можно тут: https://www.anaconda.com/

Далее необходимо установить Anaconda.

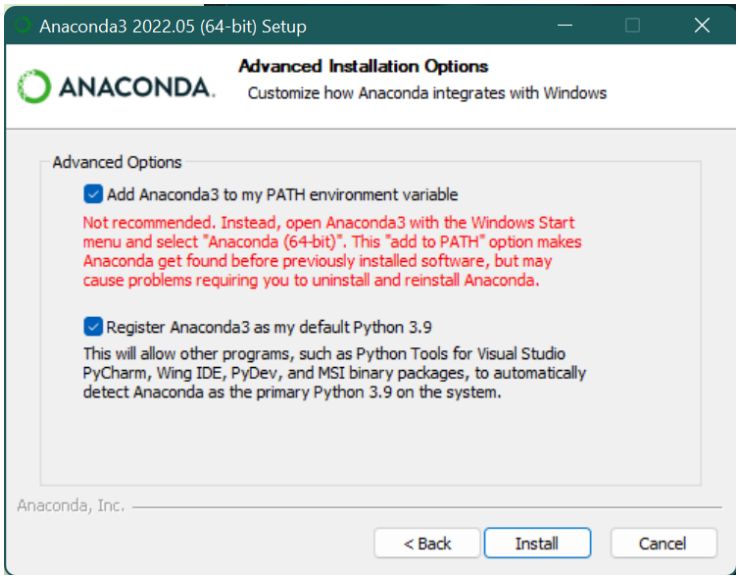


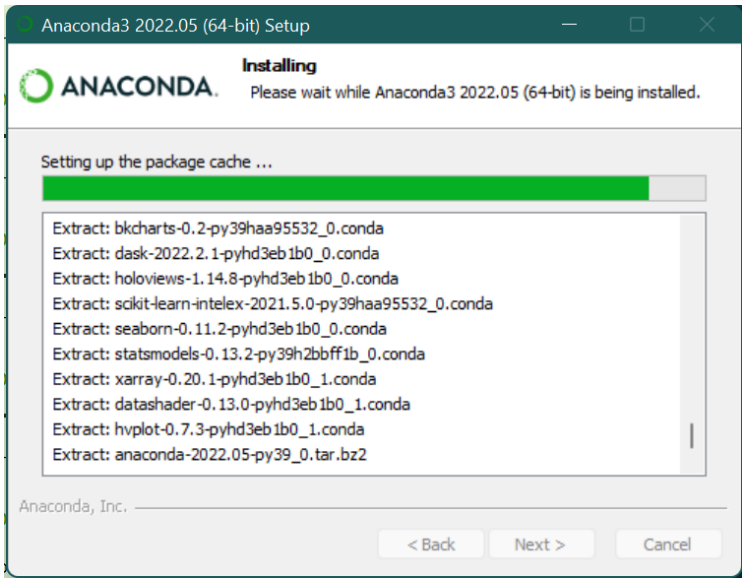


Необходимо ознакомиться с лицензией и нажать I agree.

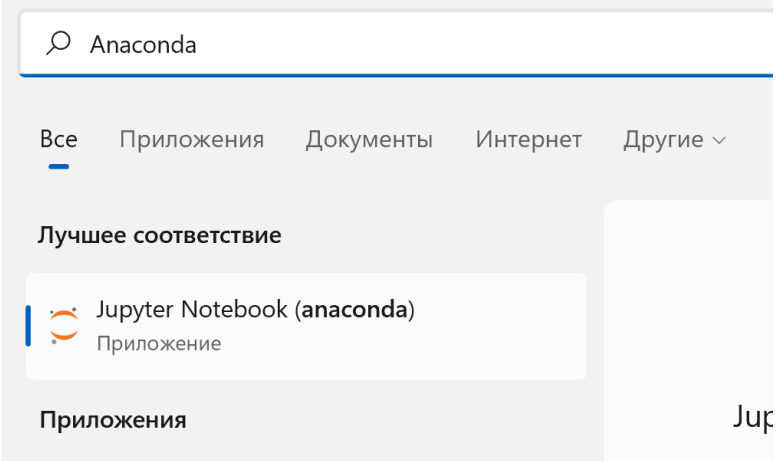


Далее предлагается внести изменения в окружение для того, чтобы Python мог по умолчанию запускаться из Anaconda. Выбрана первая галочка для того, чтобы в пункте PATH был прописан путь до расположения Anaconda. В дальнейшем это позволит писать команды в консоли Windows (cmd) команды для установки/удаления/обновления библиотек Python, например pip install/conda install

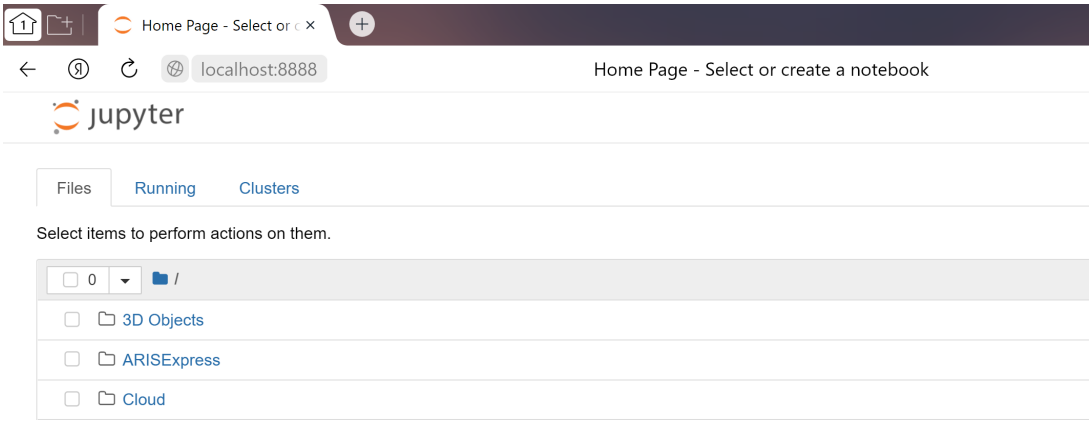




Проверим, работает ли Jupyter Notebook.

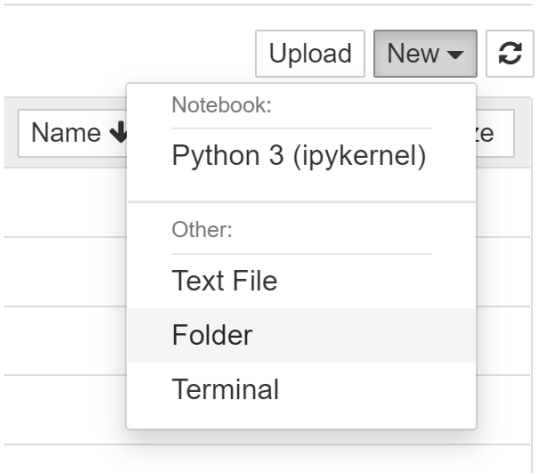


Должна открыться новая вкладка в браузере.



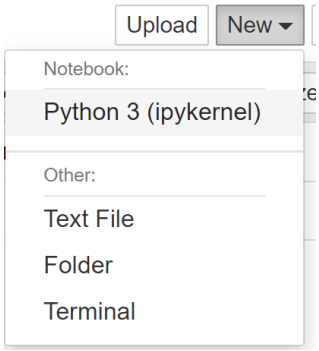
Работа с Jupyter Notebook

Создадим новую папку.

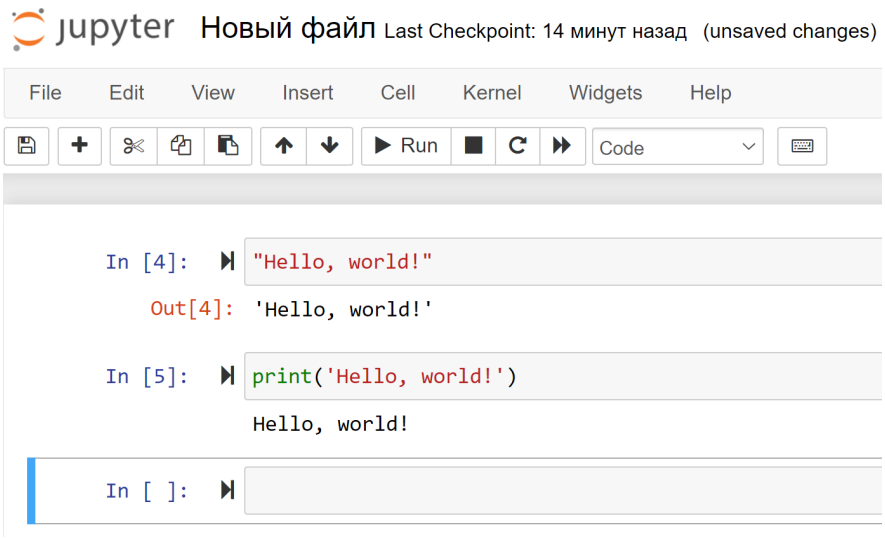


Для того, чтобы переименовать папку необходимо выбрать ее и в левом верхнем углу выбрать «Rename».

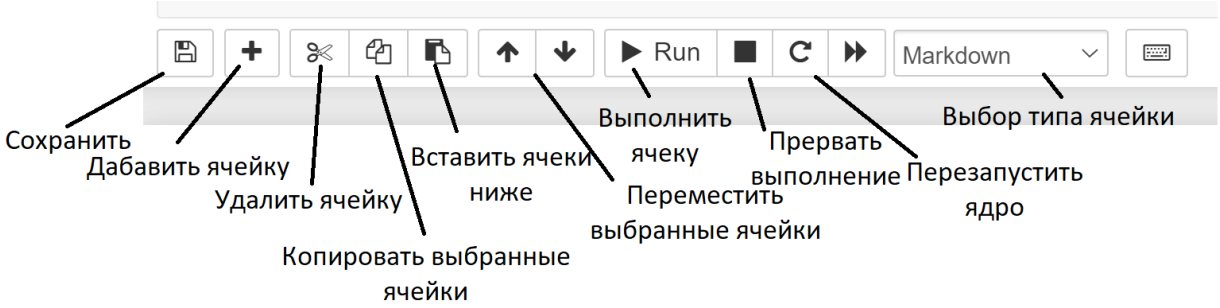
Создадим новый файл IPython. Файл с расширением ipynb.



Выведем предложение «Hello, world!»

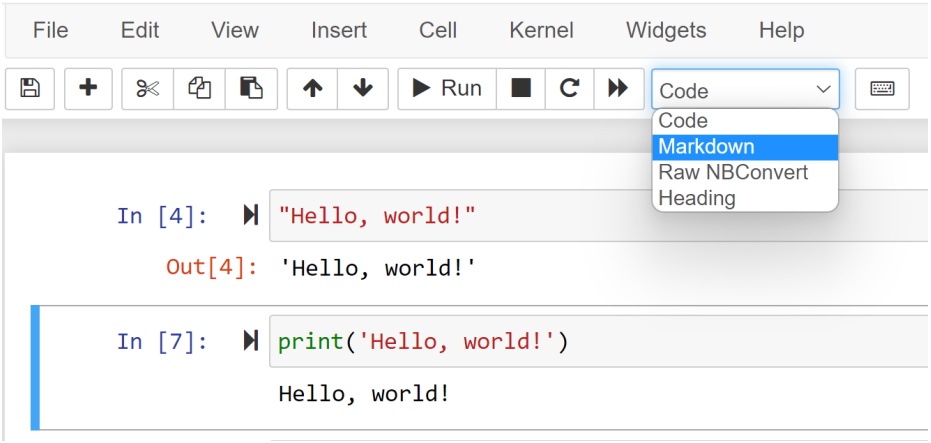


Для запуска кода в ячейке необходимо использовать комбинацию клавиш «Shift+Enter».



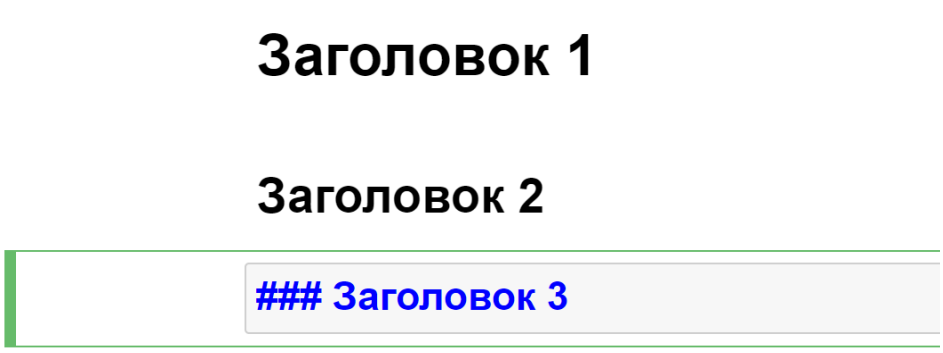
Для того, чтобы выделить несколько ячеек, необходимо с зажатым «Shift» их выбрать.

Существует два вида ячеек в Jupyter Notebook: ячейка для выполнения кода (Code) и ячейка для написания текста (Markdown). Тип ячейки всегда можно изменить, например:

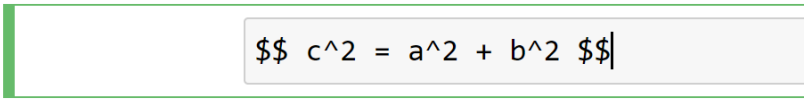


Результат:

Для того, чтобы сделать заголовок в текстовой ячейке, необходимо использовать #. Чем больше #, тем меньше будет уровень заголовка. Пример:



В ячейках для текста также можно вставлять формулы с помощью знака $$, например:



$$c^2 = a^2 + b^2$$

Типы переменных Python

В Python 3 есть следующие базовые типы переменных: bool, str, float, int, complex.

Целочисленные переменные (int) и основные операции с ними:

```
In [2]: a = 5

In [3]: print(type(a))
        <class 'int'>

In [4]: b = 2

In [5]: A = a + b
        B = a * b
        C = a / b
        D = a // b # деление с округлением до целой части
        E = a ** b # a в степени b

In [6]: print(A,B,C,D,E)
        7 10 2.5 2 25

In [7]: -a / b
Out[7]: -2.5

In [8]: -a//b
Out[8]: -3

In [9]: -(a//b)
Out[9]: -2
```

При целочисленном делении отрицательного целого числа -5 на 2 получаем -3, идет округление в меньшую сторону. Чтобы получить такой же результат, как и в случае с положительными числами, необходимо сначала поделить положительные числа, а затем добавить минус (In [9]).

Комплексные переменные (complex):

In [10]:
```
c = 5 + 9j
```

In [11]:
```
print(type(c))
```

```
<class 'complex'>
```

In [12]:
```
x = complex(3,2)
print(x)
```

```
(3+2j)
```

In [13]:
```
print(c+x)
print(c-x)
print(c*x)
print(c/x)
```

```
(8+11j)
(2+7j)
(-3+37j)
(2.5384615384615388+1.307692307692308j)
```

In [14]:
```
print(x.conjugate()) #сопряженное число
```

```
(3-2j)
```

In [15]:
```
print(x.imag) #мнимая часть
print(x.real) #действительная часть
```

```
2.0
3.0
```

Вещественные числа (float):

In [16]:
```
a = 5.0
b = 3.2
print(type(b))
```

```
<class 'float'>
```

In [17]:
```
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a//b)
```

```
8.2
1.7999999999999998
16.0
1.5625
1.0
```

In [18]:
```
d = int(a)
print(d)
```

```
5
```

Логический тип (bool):

Логический тип (True или False) можно привести к целочисленному типу.

In [19]:
```
a = True
b = False
print(type(a))
```

```
<class 'bool'>
```

In [20]:
```
print(int(a))
print(int(b))
```

```
1
0
```

In [21]:
```
print(a+a)
print(a+b)
print(a-b)
print(b-a)
```

```
2
1
1
-1
```

In [22]:
```
#операции с логическим типом
print(True and False)
print(True and True)
print(False and False)

print(True or False)
print(True or True)
print(False or False)
```

In [22]:
```
#операции с логическим типом
print(True and False)
print(True and True)
print(False and False)

print(True or False)
print(True or True)
print(False or False)
```

```
False
True
False
True
True
False
```

Проверка числа, подающегося на вход:

```
In [31]: inp = int(input('Введите целое число: '))
         print( inp <= 10 and inp >= 3)

         Введите целое число: 5
         True
```

```
In [32]: inp = int(input('Введите целое число: '))
         print( inp <= 10 and inp >= 3)

         Введите целое число: 11
         False
```

Строковый тип (str):

В Python строка – это последовательность символов, фиксация которой начинается с 0. Строки являются неизменяемыми.

```
In [23]: a = "Hello"
         b = 'Python'
         print(type(a))

         <class 'str'>
```

```
In [24]: print(a + b)

         HelloPython
```

```
In [25]: print(a*3)

         HelloHelloHello
```

```
In [26]: print(a.upper()) # верхний регистр
         print(b.lower()) # нижний регистр

         HELLO
         python
```

```
In [27]: print(a[2])
         print(a[1:3])
         print(a[:3]+'000'+a[3:]) #срез

         l
         el
         Hel000lo
```

```
In [28]: print(len(a)) #длина строки

         5
```

```
In [29]: print(bool(a)) # пуста строка или нет
         print(bool(''))

         True
         False
```

```
In [30]: help(str)

         Return a formatted version of the string as desc
         |
         |  __ge__(self, value, /)
         |      Return self>=value.
         |
         |  __getattribute__(self, name, /)
         |      Return getattr(self, name).
         |
         |  __getitem__(self, key, /)
         |      Return self[key].
         |
         |  __getnewargs__(...)
         |
         |  __gt__(self, value, /)
         |      Return self>value.
         |
         |  __hash__(self, /)
         |      Return hash(self).
         |
         |  __iter__(self, /)
```

```
In [31]: a.__sizeof__() #строка в памяти занимает 54 байта

Out[31]: 54
```

\n – символ перевода на новую строку.

```
print("Светает. Радуется солнце,\nСтолбы, деревья, небо, лед\nИ даже иней на оконце.\nСтудент угрюм. Экзамен ждет.\n")
```

```
Светает. Радуется солнце,
Столбы, деревья, небо, лед
И даже иней на оконце.
Студент угрюм. Экзамен ждет.
```

Python поддерживает форматированный вывод строк с помощью оператора форматирования %. Например:

```
print("У Миши %d яблок, а у Маши %d яблок" %(5,6))
```

```
У Миши 5 яблок, а у Маши 6 яблок
```

Условный оператор (if, else, elif):

Классическая задача – проверка числа на четность. Для этого используется знак %, который позволяет посчитать остаток от деления, например:

```
In [33]: print(13%2)
1

In [34]: print(10%2)
0
```

Пример реализации условия в Python. В начале пишется ключевое слово if, затем блок кода, который должен выполниться при выполненном условии. Затем, если условие не выполняется, но есть еще одно условие, то пишется ключевое слово elif, после чего снова идет блок кода. Если же ни одно из условий не выполняется, то пишется ключевое слово else и соответствующий ему блок кода. В Python блоки кода выделяются отступом.

```
if Условие 1:
        Блок 1
elif Условие 2:
        Блок 2
else:
        Блок 3
```

Пример:

```
In [33]: inp = int(input('Введите целое число: '))
if inp % 2 == 0:
    print('Число четное')
else:
    print('Число нечетное')

Введите целое число: 11
Число нечетное

In [34]: inp = int(input('Введите целое число: '))
if inp % 2 == 0:
    print('Число четное')
else:
    print('Число нечетное')

Введите целое число: 4
Число четное
```

Структуры данных Python

В Python все структуры данных являются объектами. У них есть методы и поля. Структуры данных Python: список (list), кортеж (tuple), множество (set), словарь (dict).

Список (list).

Список – это изменяемая структура данных. Может хранить в себе элементы разных типов. Обращение к элементу списка происходит с помощью индекса, начиная с 0. Чтобы обратиться к элементу с конца списка, то используется -1, -2 и т.д. Объявление списка происходит с помощью [] или list(). Пример:

```
L1 = [1,'Hello',3,'World',[1,1,1,1,1]] #элементами списка также могут быть другие списки
L2 = [6,7,8,9,10]
```

```
print(type(L2))
```

```
<class 'list'>
```

```
print(L1)
```

```
[1, 'Hello', 3, 'World', [1, 1, 1, 1, 1]]
```

```
print(L1[0:3])
print(L1[-1])
print(L1[-3:-1])
```

```
[1, 'Hello', 3]
[1, 1, 1, 1, 1]
[3, 'World']
```

```
print(len(L2)) #количество элементов в списке
```

```
5
```

```
L3 = L2+L2
print(L3)
```

```
[6, 7, 8, 9, 10, 6, 7, 8, 9, 10]
```

```
L2.append('Новое значение') #добавление элемента в конец списка
L2.insert(1,'New') #вставка элемента в определенный индекс списка
```

```
print(L2)
```

```
[6, 'New', 7, 8, 9, 10, 'Новое значение']
```

```
L2.remove(6) #удаления первой 6, начиная обход списка с 0 индекса
print(L2)
```

```
['New', 7, 8, 9, 10, 'Новое значение']
```

```
S = [4,5,8,2,1,3,4,9]
```

```
S.sort() #сортировка списка в порядке позрастания
print(S)
```

```
[1, 2, 3, 4, 4, 5, 8, 9]
```

```
S.sort(reverse = True) #сортировка списка в порядке убывания
print(S)
```

```
[9, 8, 5, 4, 4, 3, 2, 1]
```

```
print(S.count(4)) #количество 4-рок в списке
```

```
2
```

```
d = S.pop(0) #удаление и возвращение переменной по индексу
print(d)
print(S)
```

```
9
[8, 5, 4, 4, 3, 2, 1]
```

```
L1.reverse() #перевернуть список
print(L1)
```

```
[[1, 1, 1, 1, 1], 'World', 3, 'Hello', 1]
```

```
print(S)
```

```
[8, 5, 4, 4, 3, 2, 1]
```

```
del(S[0]) #удаление элемента списка
```

```
S
```

```
]: [5, 4, 4, 3, 2, 1]
```

```
print(max(S))
print(min(S))
```

```
5
1
```

Кортеж (tuple).

Кортеж – неизменяемая структура данных. Аналогично списку, может хранить элементы разных типов. Кортеж объявляется через () или tuple().

Кортежи работают быстрее списков за счет своей неизменяемости. Они хранятся в памяти особым образом, поэтому операции с элементами кортежа выполняются быстрее. Кортежи используются, когда есть требования к безопасности.

```
In [61]: t1 = tuple('Hello')
         t2 = (4,6,7,1,2)
         t3 = 'c',

In [62]: print(t1)
         print(t3)
         print(type(t3))

         ('H', 'e', 'l', 'l', 'o')
         ('c',)
         <class 'tuple'>

In [63]: print(t2[1])
         print(t2[1:4])

         6
         (6, 7, 1)

In [64]: print(4 in t2) #есть ли 4 в кортеже t2
         print(11 in t2) #есть ли 11 в кортеже t2

         True
         False

In [65]: ts = sorted(t2) #сортировка кортежа

In [66]: ts

    Out[66]: [1, 2, 4, 6, 7]

In [67]: #удалить элемент из кортежа нельзя, так как он неизменяем
         #но можно удалить весь картеж
         del(ts)

In [68]: ts

         ---------------------------------------------------------------------------
         NameError                                 Traceback (most recent call last)
         ~\AppData\Local\Temp/ipykernel_2688/968267673.py in <module>
         ----> 1 ts

         NameError: name 'ts' is not defined

In [69]: # конкатенация кортежей, аналогично спискам
         print(t1 + t2)

         ('H', 'e', 'l', 'l', 'o', 4, 6, 7, 1, 2)
```

```
In [70]: print(t1*3)
('H', 'e', 'l', 'l', 'o', 'H', 'e', 'l', 'l', 'o', 'H', 'e', 'l', 'l', 'o')

In [71]: print(t1.index('H')) #индекс элемента
0

In [72]: s = ''.join(t1) #из кортежа в строку
         print(s, type(s))
Hello <class 'str'>

In [73]: s = list(t1) #из кортежа в список
         print(s, type(s))
['H', 'e', 'l', 'l', 'o'] <class 'list'>
```

Множество (set).

Множество – изменяемая структура данных, которая имеет требование к уникальности элементов. Множество удобно использовать для удаления повторяющихся элементов.

```
In [74]: S = set('123455')

In [75]: print(S)
{'2', '1', '5', '3', '4'}

In [76]: S.add(10) #добавление элемента
         print(S)
{'2', '1', '5', 10, '3', '4'}

In [77]: S.remove(10) #удаление элемента
         print(S)
{'2', '1', '5', '3', '4'}

In [78]: S.clear() #очистка множества

In [79]: print(S)
set()
```

Словарь (dict).

Словарь – ассоциативный массив, который хранит пары ключ:значение. Объявляется через {} при указании {ключ1:значение1, ключ2:значение2}. Ключ – уникальный и неизменяемый, поэтому tuple может использоваться в качестве ключа.

```
In [80]: T1 = {'ключ1':1,'ключ2':2}

In [81]: print(T1, type(T1))

         {'ключ1': 1, 'ключ2': 2} <class 'dict'>

In [82]: print(len(T1))

         2

In [83]: print(T1.keys()) #ключи словаря
         print(T1.values()) #значения словаря
         print(T1.items()) #пары ключ:значение словаря

         dict_keys(['ключ1', 'ключ2'])
         dict_values([1, 2])
         dict_items([('ключ1', 1), ('ключ2', 2)])

In [84]: up = {'ключ3':3,'ключ1':10}

In [85]: T1.update(up) #обновление словаря элементами из другого словаря

In [86]: # если ключ повторяется, то его значение обновляется, если ключ новый,
         # то он добавляется к словарю
         print(T1)

         {'ключ1': 10, 'ключ2': 2, 'ключ3': 3}
```

In [87]:
```
print(T1.get('ключ1')) # получить значение по ключу
```

```
10
```

In [88]:
```
print(T1.pop('ключ3')) #удаление и возврат знаения по ключу
```

```
3
```

In [89]:
```
print(T1)
```

```
{'ключ1': 10, 'ключ2': 2}
```

In [90]:
```
print(T1.popitem()) #возврат и удаление последней пары в словаре
print('Словарь после удаления последнего элемента: ',T1)
```

```
('ключ2', 2)
Словарь после удаления последнего элемента:  {'ключ1': 10}
```

In [91]:
```
T2 = T1 #это неглубокая копия словаря, это ссылка на ту же ячейку памяти (или .copy())
        #если изменить словарь T1, то словарь T2 тоже изменится
        #чтобы сделать отдельный словарь, необходимо использовать .copy()
T3 = T1.copy()
```

In [92]:
```
T1.update(up) #обновление словаря элементами из другого словаря
T1.popitem()
print(T2)
print(T3)
```

```
{'ключ1': 10}
{'ключ1': 10}
```

```
1 import copy #импорт библиотеки
2 T4 = copy.deepcopy(T1) #глубокая копия словаря
```

```
1 T1.clear() #очистка словаря T1
2 print(T1)
3 print(T4) #словарь T4 не изменился, он не зависит от T1
```

```
{}
{'ключ1': 10}
```

Практическая работа

1. Установить Python.

2. Написать программу, которая вычисляет площадь фигуры, параметры которой подаются на вход. Фигуры, которые подаются на вход: треугольник, прямоугольник, круг. Результатом работы является словарь, где ключ – это название фигуры, а значение – это площадь.

3. Написать программу, которая на вход получает два числа и операцию, которую к ним нужно применить. Должны быть реализованы следующие операции: +, -, /, //, abs – модуль, pow или ** – возведение в степень. Результатом работы программы является одно число.

4. Написать программу, вычисляющую площадь треугольника по переданным длинам трёх его сторон по формуле Герона:

$$\sqrt{p(p-a)(p-b)(p-c)},$$

где $p = \frac{a+b+c}{2}$.

На вход программе подаются целые числа, выводом программы должно являться вещественное число, соответствующее площади треугольника.

5. Оформить отчет о проделанной работе. В отчете должен быть приведен код и результат его выполнения.