

# АаDSaPз\_fZtH. Занятие 2

## Часть 1. Введение в Python

Емельянов Антон  
[login-const@mail.ru](mailto:login-const@mail.ru)

# Двумерные массивы

- Списки можно использовать для хранения таблиц (двумерных массивов с данными).

```
mas1.py x
1 a = [[1, 2, 3], [4, 5, 6]]
2 print(a[0])
3 print(a[1])
4 b = a[0]
5 print(b)
6 print(a[0][2])
7 a[0][1] = 7
8 print(a)
9 print(b)
10 b[2] = 9
11 print(a[0])
12 print(b)
13
```

```
[1, 2, 3]
[4, 5, 6]
[1, 2, 3]
3
[[1, 7, 3], [4, 5, 6]]
[1, 7, 3]
[1, 7, 9]
[1, 7, 9]
```

# Двумерные массивы

- Для обработки и вывода списка, как правило, используют два вложенных цикла.

```
mas2.py x
1  a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
2  for i in range(len(a)):
3      for j in range(len(a[i])):
4          print(a[i][j], end=' ')
5      print()
6
```

# Двумерные массивы

- Для обработки и вывода списка, как правило, используют два вложенных цикла.

```
mas2.py x
1  a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
2  for i in range(len(a)):
3      for j in range(len(a[i])):
4          print(a[i][j], end=' ')
5      print()
6
```

```
mas3.py x
1  a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
2  for row in a:
3      for elem in row:
4          print(elem, end=' ')
5      print()
6
```

# Двумерные массивы

- Создание вложенных списков. Верен ли код снизу?

```
mas4.py ×  
1 n = 3  
2 m = 4  
3 a = [[0] * m] * n  
4 a[0][0] = 5  
5 print(a[1][0])  
6
```

# Двумерные массивы

- Создание вложенных списков

```
mas5.py x
1      n = 3
2      m = 4
3      a = [0] * n
4      for idx in range(n):
5          a[idx] = [0] * m
6      a[0][0] = 5
7      print(a[1][0])
8
```

# Двумерные массивы

- Ввод массива

```
mas6.py x
1      # в первой строке ввода идёт количество строк массива
2      n = int(input())
3      a = []
4      for i in range(n):
5          row = input().split()
6          for j in range(len(row)):
7              row[j] = int(row[j])
8          a.append(row)
9
```

# Двумерные массивы

- Списки могут содержать объекты любых типов (даже самих себя).

mas7.py ×

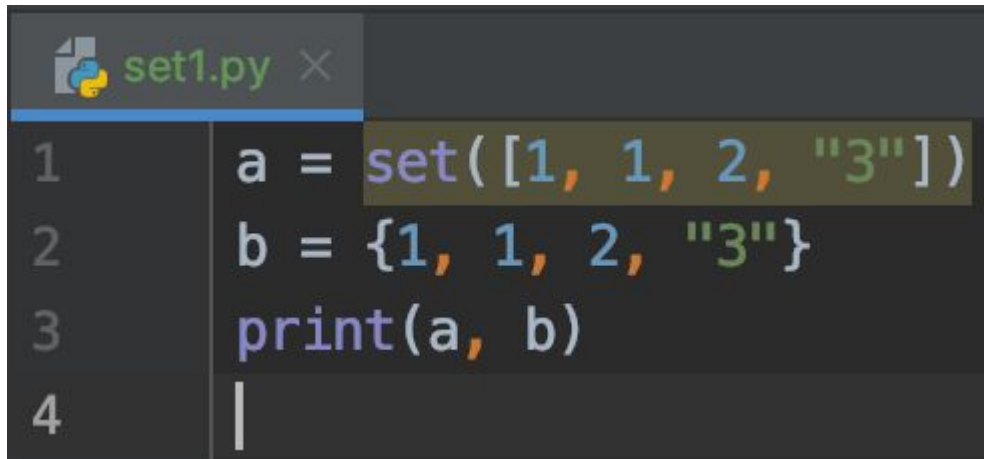
```
1 lst = ["str", 10, 11.1, [1, 2, 3, {"key": "value"}, {1, "str"}]]
2 lst.append(lst)
3 print(lst)
4
```

```
['str', 10, 11.1, [1, 2, 3, {'key': 'value'}, {1, 'str'}], [...]]
```



# Множества

- **Множество** — одно из ключевых понятий математики; это математический объект, сам являющийся набором, **совокупностью**, собранием каких-либо объектов, которые называются **элементами** этого множества и обладают общим для всех их характеристическим свойством.



```
set1.py ×  
1 a = set([1, 1, 2, "3"])  
2 b = {1, 1, 2, "3"}  
3 print(a, b)  
4 |
```

```
{1, 2, '3'} {1, 2, '3'} {'qwerty'}
```

# Множества

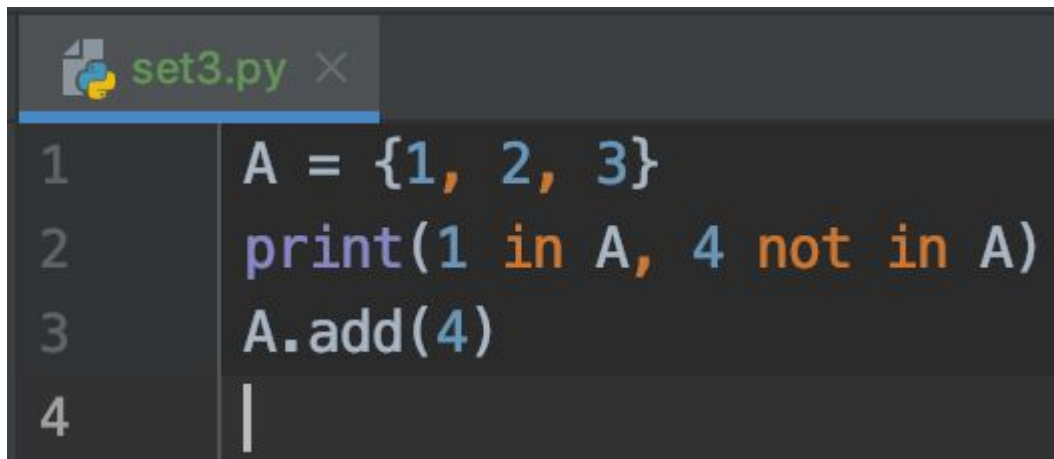
- Работать с элементами множеств можно тоже с помощью циклов

```
set2.py x
1 primes = {2, 3, 5, 7, 11}
2 for num in primes:
3     print(num, end=" ")
4
5 target = 11
6 search_result = False
7 for num in primes:
8     if target == num:
9         search_result = True
10 if search_result:
11     print(target, "in primes set")
12 else:
13     print(target, "not in primes set")
14 |
```

```
2
3
5
7
11
11 in primes set
```

# Множества

- Проверить, принадлежит ли элемент множеству можно при помощи операции `in`, возвращающей значение типа `bool`. Аналогично есть противоположная операция `not in`. Для добавления элемента в множество есть метод `add`:



```
set3.py x
1 A = {1, 2, 3}
2 print(1 in A, 4 not in A)
3 A.add(4)
4 |
```

True True

# Множества

- С множествами в питоне можно выполнять обычные для математики операции

<b><math>A \cup B</math></b> <b><code>A.union(B)</code></b>	Возвращает множество, являющееся объединением множеств <b>A</b> и <b>B</b> .
<b><math>A \cup= B</math></b> <b><code>A.update(B)</code></b>	Добавляет в множество <b>A</b> все элементы из множества <b>B</b> .
<b><math>A \cap B</math></b> <b><code>A.intersection(B)</code></b>	Возвращает множество, являющееся пересечением множеств <b>A</b> и <b>B</b> .
<b><math>A \cap= B</math></b> <b><code>A.intersection_update(B)</code></b>	Оставляет в множестве <b>A</b> только те элементы, которые есть в множестве <b>B</b> .
<b><math>A - B</math></b> <b><code>A.difference(B)</code></b>	Возвращает разность множеств <b>A</b> и <b>B</b> (элементы, входящие в <b>A</b> , но не входящие в <b>B</b> ).

# Множества

- С множествами в питоне можно выполнять обычные для математики операции над множествами.

$A -= B$ <code>A.difference_update(B)</code>	Удаляет из множества $A$ все элементы, входящие в $B$ .
$A \wedge B$ <code>A.symmetric_difference(B)</code>	Возвращает симметрическую разность множеств $A$ и $B$ (элементы, входящие в $A$ или в $B$ , но не в оба из них одновременно).
$A \wedge= B$ <code>A.symmetric_difference_update(B)</code>	Записывает в $A$ симметрическую разность множеств $A$ и $B$ .
$A \leq B$ <code>A.issubset(B)</code>	Возвращает <code>true</code> , если $A$ является подмножеством $B$ .
$A \geq B$ <code>A.issuperset(B)</code>	Возвращает <code>true</code> , если $B$ является подмножеством $A$ .

- Структура данных, позволяющая идентифицировать ее элементы не по числовому индексу, а по произвольному, называется *словарем* или *ассоциативным массивом*. Соответствующая структура данных в языке Питон называется **dict**.

```
dict1.py ×
4      # Заполним его несколькими значениями
5      Capitals['Russia'] = 'Moscow'
6      Capitals['Ukraine'] = 'Kiev'
7      Capitals['USA'] = 'Washington'
8      Countries = ['Russia', 'France', 'USA', 'Russia']
9      for country in Countries:
10         # Для каждой страны из списка проверим, есть ли она в словаре Capitals
11         if country in Capitals:
12             print('Столица страны ' + country + ': ' + Capitals[country])
13         else:
14             print('В базе нет страны с названием ' + country)
15      |
```

Словари нужно использовать в следующих случаях:

- Подсчет числа каких-то объектов. В этом случае нужно завести словарь, в котором ключами являются объекты, а значениями — их количество.
- Хранение каких-либо данных, связанных с объектом. Ключи — объекты, значения — связанные с ними данные. Например, если нужно по названию месяца определить его порядковый номер, то это можно сделать при помощи словаря `Num['January'] = 1; Num['February'] = 2; ....`
- Установка соответствия между объектами (например, “родитель—потомок”). Ключ — объект, значение — соответствующий ему объект.
- Если нужен обычный массив, но максимальное значение индекса элемента очень велико, и при этом будут использоваться не все возможные индексы (так называемый “разреженный массив”), то можно использовать ассоциативный массив для экономии памяти.

# Словари

- Пустой словарь можно создать при помощи функции `dict()` или пустой пары фигурных скобок `{}` (вот почему фигурные скобки нельзя использовать для создания пустого множества). Для создания словаря с некоторым набором начальных значений можно использовать следующие конструкции:

```
dict2.py ×  
1 Capitals = {'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}  
2 Capitals = dict(Russia='Moscow', Ukraine='Kiev', USA='Washington')  
3 Capitals = dict([("Russia", "Moscow"), ("Ukraine", "Kiev"), ("USA", "Washington")])  
4 Capitals = dict(zip(["Russia", "Ukraine", "USA"], ["Moscow", "Kiev", "Washington"]))  
5 print(Capitals)  
6
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
```



- Работа с элементами словаря

```
dict3.py x
1  A = {'ab': 'ba', 'aa': 'aa', 'bb': 'bb', 'ba': 'ab'}
2
3  key = 'ac'
4  if A.get(key) is not None:
5      del A[key]
6
7  try:
8      del A[key]
9  except KeyError:
10     print('There is no element with key "' + key + '" in dict')
11     print(A)
12
```

There is no element with key "ac" in dict  
{'ab': 'ba', 'aa': 'aa', 'bb': 'bb', 'ba': 'ab'}

- Еще один способ удалить элемент из словаря: использование метода `pop`: `A.pop(key, None)`.

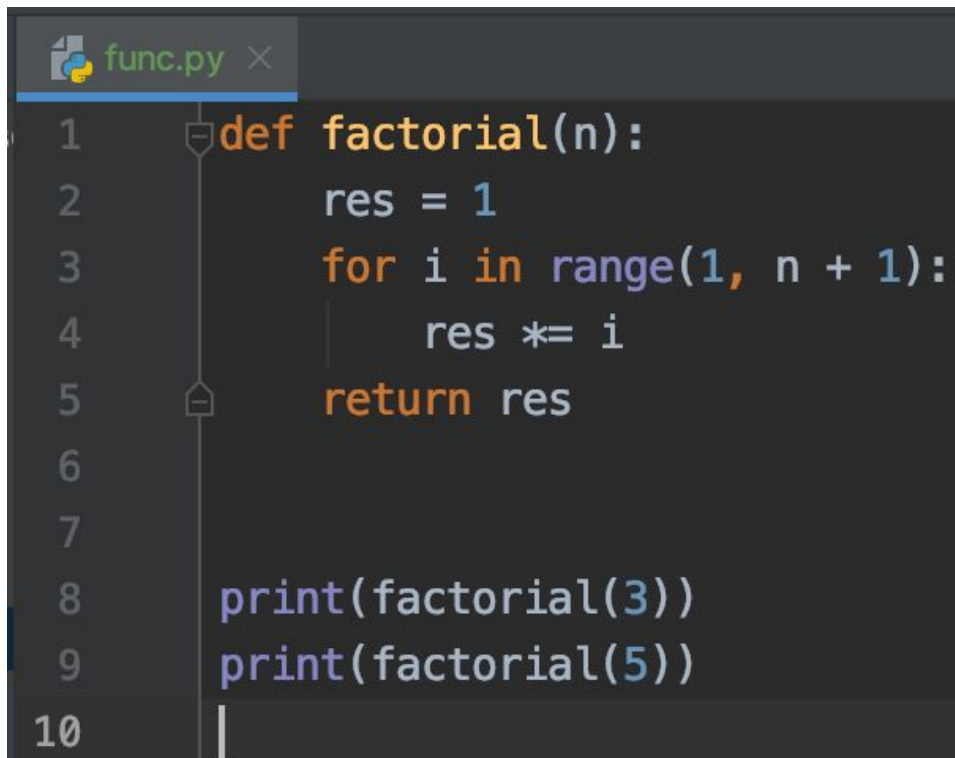
- Перебор элементов словаря

```
dict4.py ×
1  A = dict(zip('abcdef', list(range(6))))
2  for key in A:
3      print(key, A[key])
4
5  A = dict(zip('abcdef', list(range(6))))
6  for key, val in A.items():
7      print(key, val)
8
9  print(A.keys())
10 print(A.values())
```

```
dict_keys(['a', 'b', 'c', 'd', 'e', 'f'])
dict_values([0, 1, 2, 3, 4, 5])
```

# Функции

- Функции — это такие участки кода, которые изолированы от остальной программы и выполняются только тогда, когда вызываются.



```
func.py x
1  def factorial(n):
2      res = 1
3      for i in range(1, n + 1):
4          res *= i
5      return res
6
7
8  print(factorial(3))
9  print(factorial(5))
10
```

- Еще пример функции

```
func2.py x
1  def max2(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7
8  def max3(a, b, c):
9      return max2(max2(a, b), c)
10
11
12  print(max3(3, 5, 4))
13
```

- Общий интерфейс

```
func3.py x
1 def some_func(arg1, arg2, *args, kv_arg1=1, kv_arg2=2, **kwargs):
2     print(arg1, arg2, end=" ")
3     print("kv_arg1=", kv_arg1, " kv_arg2=", kv_arg2, sep="")
4     for arg in args:
5         print(arg, end=" ")
6     print()
7     for key, val in kwargs.items():
8         print(key, "=", val, end=" ", sep="")
9     print()
10
11
12 some_func(-1, -2, 0, 1, kv_arg1=2, kv_arg2=3, kv_arg3=4)
```

```
-1 -2 kv_arg1=2 kv_arg2=3
0 1
kv_arg3=4
```

# Локальные и глобальные переменные

- Внутри функции можно использовать переменные, объявленные вне этой функции.
- Но если инициализировать какую-то переменную внутри функции, использовать эту переменную вне функции не удастся.

```
glob1.py x
1  def f():
2      print(a)
3
4
5  a = 1
6  f()
```

1

```
glob2.py x
1  def f():
2      a = 1
3
4      |
5      f()
6      print(a)
```

```
Traceback (most recent call last):
  File "L2/code/glob2.py", line 6, in <module>
    print(a)
NameError: name 'a' is not defined
```

# Функции

- Изменение локальных и глобальных переменных

```
glob3.py x
1 def f():
2     a = 1
3     print(a)
4
5
6 a = 0
7 f()
8 print(a)
9
```

```
1
0
```

```
glob4.py x
1 def f():
2     print(a)
3     a = 0
4
5
6 a = 1
7 f()
8
```

```
Traceback (most recent call last):
  File "L2/code/glob4.py", line 7, in <module>
    f()
  File "L2/code/glob4.py", line 2, in f
    print(a)
UnboundLocalError: local variable 'a' referenced before assignment
```

# Функции

- Чтобы функция могла изменить значение глобальной переменной, необходимо объявить эту переменную внутри функции, как глобальную, при помощи ключевого слова **global**:

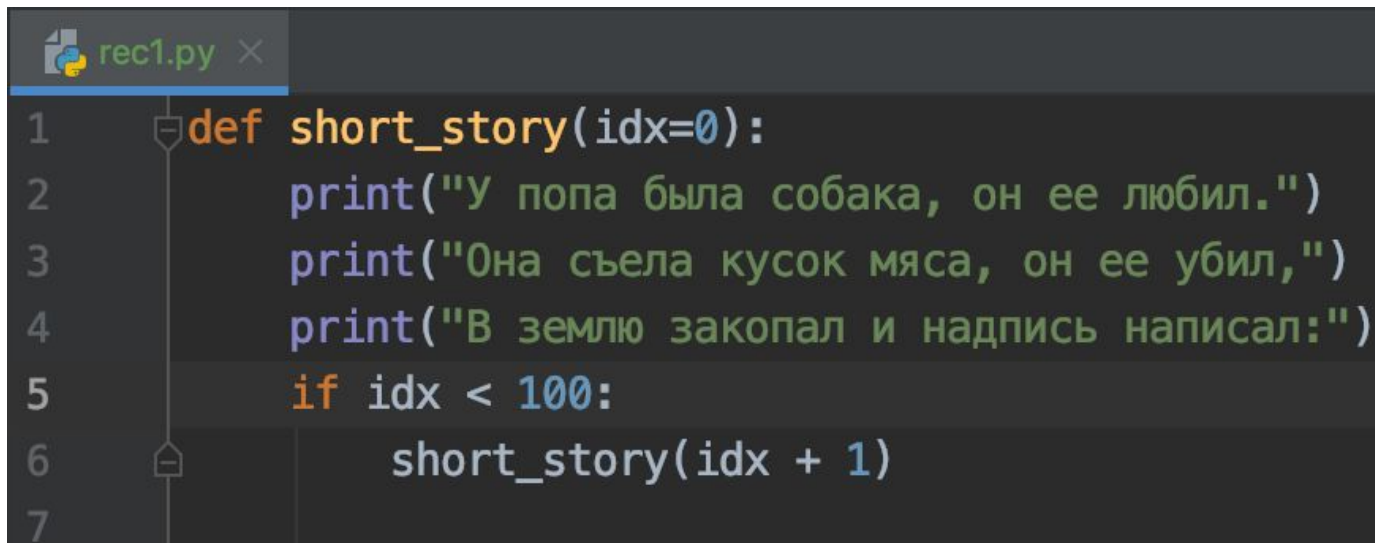
```
global5.py x
1  def f():
2      global a
3      a = 1
4      print(a)
5
6
7  a = 0
8  f()
9  print(a)
10 |
```

```
1
1
```



# Рекурсия

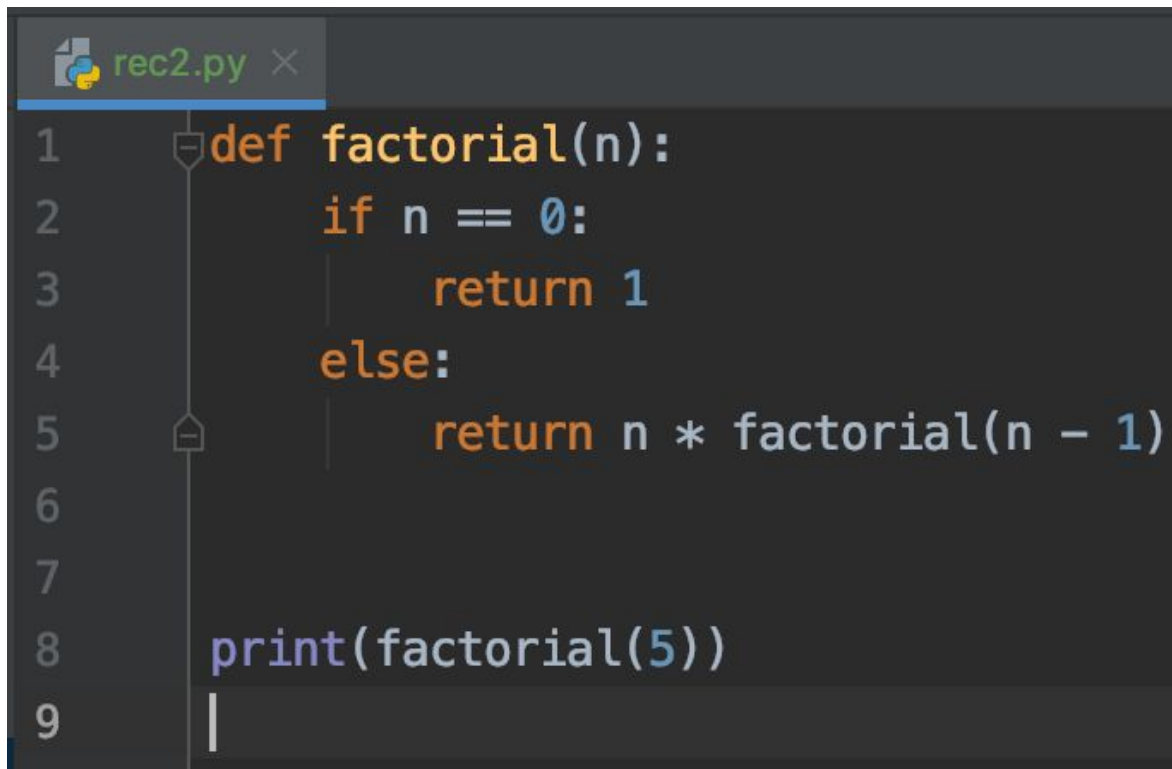
- **Рекурсия** — определение, описание, изображение какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» используется в различных специальных областях знаний — от лингвистики до логики, но наиболее широкое применение находит в математике и информатике.
- Или функция вызывает сама себя.



```
rec1.py x
1 def short_story(idx=0):
2     print("У попа была собака, он ее любил.")
3     print("Она съела кусок мяса, он ее убил,")
4     print("В землю закопал и надпись написал:")
5     if idx < 100:
6         short_story(idx + 1)
7
```

# Рекурсия

- Рекурсивный факториал



The image shows a code editor window with a tab labeled 'rec2.py'. The code is written in Python and defines a recursive function 'factorial' and calls it. The code is as follows:

```
1 def factorial(n):  
2     if n == 0:  
3         return 1  
4     else:  
5         return n * factorial(n - 1)  
6  
7  
8 print(factorial(5))  
9 |
```

# Импорт

- Для использования этих функций в начале программы необходимо подключить математическую библиотеку, что делается командой `import module_name`



```
imp1.py x
1  from rec2 import factorial as fact
2  import rec2
3
4
5  print(fact(4))
6  print(rec2.factorial(2))
7
```

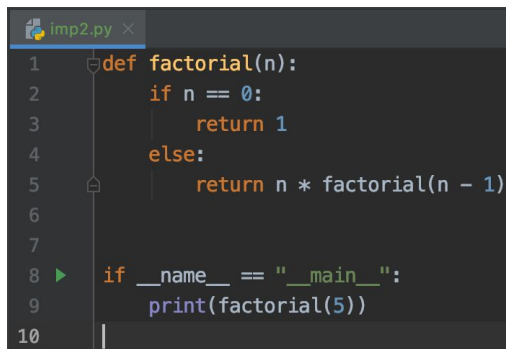
120

24

2

# Импорт

- `if __name__ == "__main__":`: ее основное назначение — разделение кода, который будет выполняться при вызове кода как модуля (при импортировании его в другой скрипт) — и при запуске самого модуля, как отдельного файла.



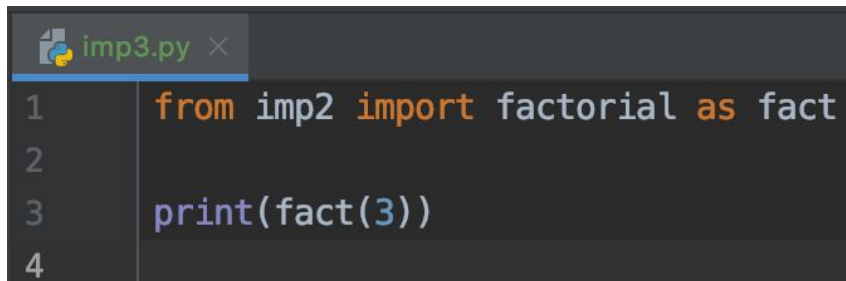
```
imp2.py x
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7
8 if __name__ == "__main__":
9     print(factorial(5))
10
```



```
python3 imp2.py
120
```

# Импорт


- `if __name__ == "__main__":`: ее основное назначение — разделение кода, который будет выполняться при вызове кода как модуля (при импортировании его в другой скрипт) — и при запуске самого модуля, как отдельного файла.



```
1 from imp2 import factorial as fact
2
3 print(fact(3))
4
```



```
$ python3 imp3.py
6
```



# АаDSaPз\_fZtH. Занятие 2

## Часть 2. Сортировки начало

Емельянов Антон  
[login-const@mail.ru](mailto:login-const@mail.ru)

# Определение сортировки

- Сортировка (англ. *sorting* — классификация, упорядочение) — последовательное расположение или разбиение на группы чего-либо в зависимости от выбранного критерия.
- Поле, по которому производится сортировка, называется **ключом (key)**, а остальные поля — дополнительные данные.
- Алгоритм сортирует ключи, но вместе с ними перемещаются дополнительные данные без изменения.
- Будем рассматривать алгоритмы сортировки ключей и считать, что данные перемещаются за  $O(1)$ .

# Глупая сортировка

- Просматриваем массив слева-направо и по пути сравниваем соседей. Если мы встретим пару взаимно неотсортированных элементов, то меняем их местами и возвращаемся на круги своя, то бишь в самое начало. Снова проходим-проверяем массив, если встретили снова «неправильную» пару соседних элементов, то меняем местами и опять начинаем всё сызнова. Продолжаем до тех пор пока массив потихоньку-полегоньку не отсортируется.
- Сложность  $O(N^3)$

```
sort1.py x
1  def stupid_sort(data):
2      i, size = 1, len(data)
3      while i < size:
4          if data[i - 1] > data[i]:
5              data[i - 1], data[i] = data[i], data[i - 1]
6              i = 1
7          else:
8              i += 1
9      return data
10
```



# Сортировка пузырьком

- обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. И тд.

```
sort2.py x
1  def bubble(data):
2      size = len(data)
3      for i in range(size - 1):
4          for j in range(size - i - 1):
5              if data[j] > data[j + 1]:
6                  data[j], data[j + 1] = data[j + 1], data[j]
7
```

# Сортировка пузырьком

- обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. И тд.
- Сложность  $O(N^2)$

```
sort2.py x
1  def bubble(data):
2      size = len(data)
3      for i in range(size - 1):
4          for j in range(size - i - 1):
5              if data[j] > data[j + 1]:
6                  data[j], data[j + 1] = data[j + 1], data[j]
7
```

## Домашнее задание

- Прорешать задачи тут  
[http://pythontutor.ru/lessons/inout and arithmetic operations/](http://pythontutor.ru/lessons/inout_and_arithmetic_operations/)
  - Пункты: 3, 5, 8, 9, 10, 11
- Доказать оценку “глупой сортировки”

# Источники

- [https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE#cite\\_note-1](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE#cite_note-1)
- <http://pythontutor.ru/>
- <https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F>
- [https://rtfm.co.ua/python-zachem-nuzhen-if-\\_\\_name\\_\\_-\\_\\_main\\_\\_/](https://rtfm.co.ua/python-zachem-nuzhen-if-__name__-__main__/)
- <https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0>
- <https://habr.com/ru/post/204600/>
- <http://algotlab.valemak.com/stupid>
- <https://younglinux.info/algorithm/bubble>
- [https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\\_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC)
- <https://e-maxx.ru/bookz/files/cormen.pdf>