


Санкт-Петербургский государственный университет

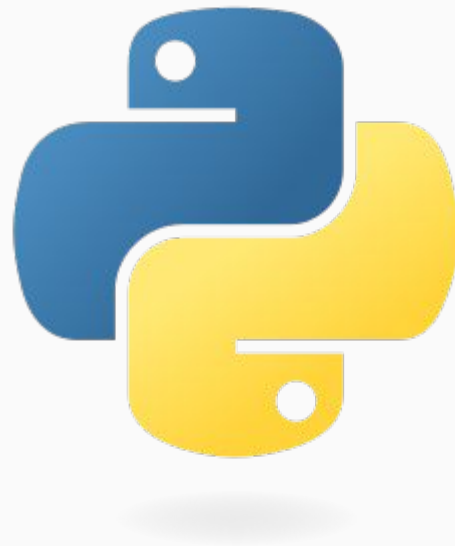
# Python

Сергей Шульман  
2017



# План лекции

- Введение
- Встроенные типы данных
  - Числа
  - Ссылочные типы данных
- Основы грамматики
  - Условные операторы и циклы
  - Функции
  - Работа с файлами



# Введение

# История Python

Разрабатывается с конца 1980-х

Опубликован в 1991 году

Существует две ветви развития:

2000 год Python 2 (2.7.14)

2008 год Python 3 (3.6.3)

Нет обратной совместимости

Есть утилита 2to3



Гвидо ван Россум

# Влияние других языков

- ABC - отступы для группировки операторов
- Modula-3 - пакеты, модули, именованные аргументы функций, исключения
- C и C++ некоторые заимствования синтаксиса
- Smalltalk - ООП
- Lisp - элементы функционального программирования
- Fortran - комплексные числа, срезы массивов (Numpy)
- Miranda - списочные выражения
- Java - логирование, unit тестирование
- Icon - генераторы

....

# Характеристики Python

- Свободный - весь код доступен
- Интерпретируемый
- Высокоуровневый
  - строгая, динамическая типизация
  - типы данных высокого уровня
  - исключения
- Мультипарадигмальный
  - итеративное программирование
  - ООП
  - функциональное программирование
- Расширяемый (API для создания модулей на C и C++)

# Особенности Python

- Медленный
- Очень плохо подходит для параллельных вычислений
- Динамическая типизация мешает читать код
- Мало ошибок видно до выполнения программы
- Плохая совместимость версий библиотек

# Области применения

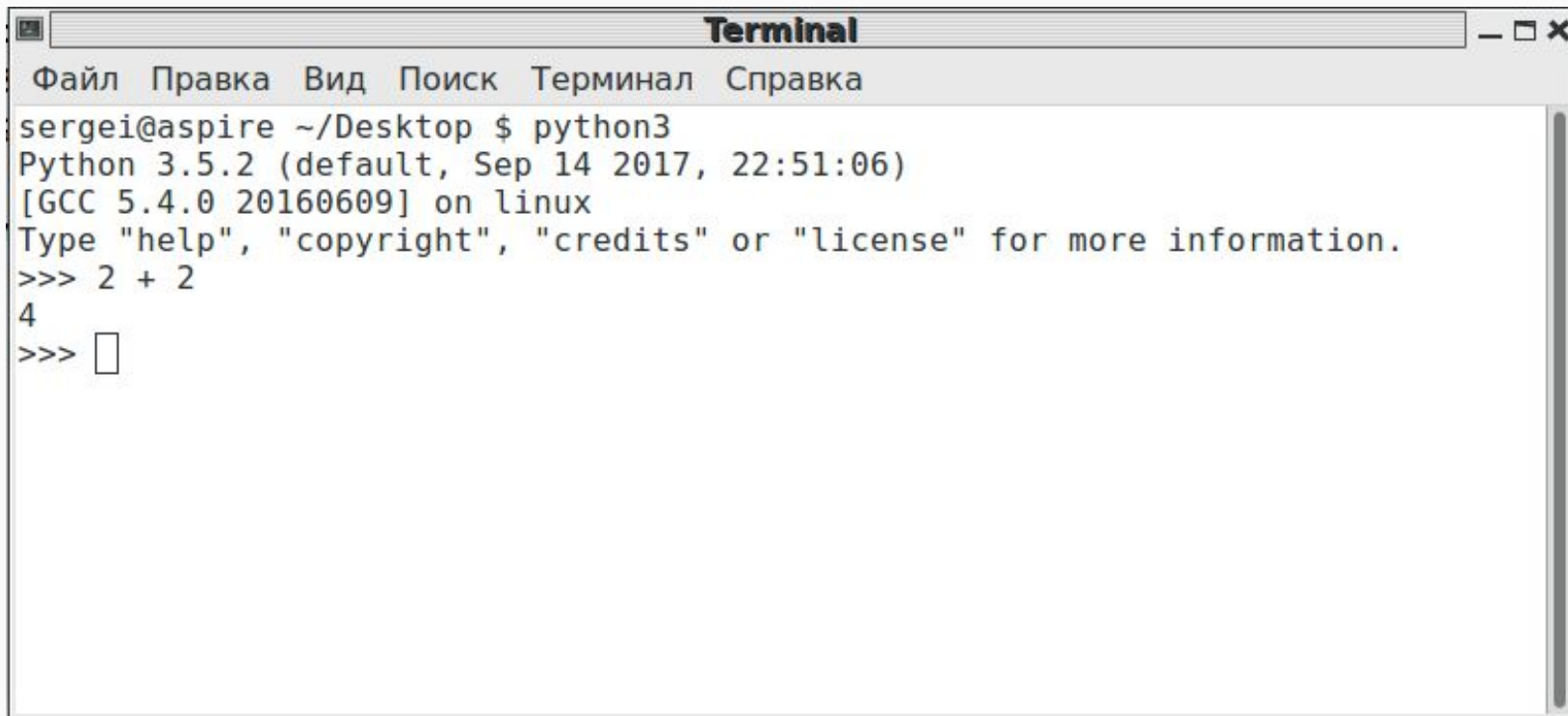
- Научные задачи
- Обработка данных
- Интернет сайты
- Проверка работоспособности алгоритма/идеи



# Способы работы с Python

- Интерактивный режим  
взаимодействие через командную строку
- Программа (Скрипт)  
текстовый файл .py с последовательностью команд,  
исполняемый интерпретатором

# Интерактивный режим

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Файл", "Правка", "Вид", "Поиск", "Терминал", and "Справка". The terminal content shows the execution of "python3" from the prompt "sergei@aspire ~/Desktop \$". It displays the Python version "Python 3.5.2 (default, Sep 14 2017, 22:51:06)", the compiler "[GCC 5.4.0 20160609] on linux", and a prompt to type "help", "copyright", "credits", or "license" for more information. The user enters ">>> 2 + 2", and the terminal outputs "4". The prompt ">>>" is followed by a cursor box.

```
Terminal
Файл Правка Вид Поиск Терминал Справка
sergei@aspire ~/Desktop $ python3
Python 3.5.2 (default, Sep 14 2017, 22:51:06)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
>>> 
```

Интерпретатор Python

# Интерактивный режим

- Jupyter (IPython) Notebook
- Online аналоги, например <https://try.jupyter.org/>

Интерактивный режим не рекомендуется использовать для чего-то ценного.

# Скрипты

- Запуск через интерпретатор  
\$ python3 script.py
- Запуск как исполняемый файл
  - Добавить в файл строку  
#!/usr/bin/env python3
  - Сделать файл исполняемым  
chmod u+x script.py
  - Запустить  
\$ ./script.py

# Online IDE

Можно пользоваться Online IDE и не ставить Python себе на компьютер.

[Trinket.io](https://trinket.io)

[Repl.it](https://repl.it)

## Python Enhancement Proposals 8

- отступ должен быть 4 пробела
- кодировка файла utf-8
- пишите комментарии по-английски
- модули и пакеты - короткие **всестрочные** имена
- классы - **СловаНачинаютсяЗаглавными**
- функции и методы - **слова\_с\_подчёркиваниями**
- не сравнивайте булевы переменные с True и False
- ....

# Используйте профессиональные инструменты

## PyCharm

- подсветка синтаксиса
- навигация
- автодополнение
- поиск ошибок

## PyCharm Edu

- рефакторинги
- отладчик
- Data View
- обучение Python

# Встроенные типы данных



# Встроенные типы данных

## Строгая динамическая типизация

- примитивные типы  
числа
- ссылочные типы  
списки, словари, строки, кортежи...

# Примитивные типы данных и операции с ними

# Целые числа

Длинная арифметика идёт в комплекте

• Сложение	$2 + 2$	4	<code>+=</code>
• Вычитание	$2 - 2$	0	<code>-=</code>
• Умножение	$2 * 2$	4	<code>*=</code>
• Деление	$3 / 2$	1.5	<code>/=</code>
• Целочисленное деление	$5 // 3$	1	<code>//=</code>
• Остаток от деления	$5 \% 2$	2	<code>%=</code>
• Смена знака числа	-1	-1	
• Возведение в степень	$3^{**}3$	27	<code>**=</code>
• Модуль числа	$\text{abs}(-1)$	1	

# Побитовые операторы

&      “И”

|      “ИЛИ”

^      “исключающее ИЛИ”

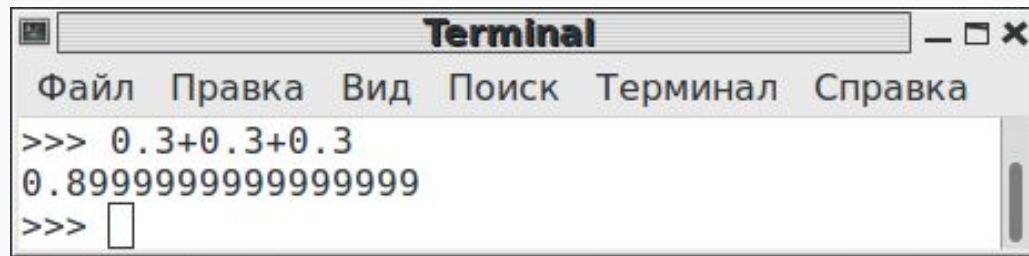
~      комплиментарный оператор  
(унарный, меняет все биты на обратные)

<<      побитовый сдвиг влево

>>      побитовый сдвиг вправо

# Числа с плавающей точкой

- 8 байтовое число с плавающей точкой
- Все операции целых чисел, кроме побитовых (включая целочисленное деление и остаток от деления)
- Целые числа могут автоматически приводиться к числам с плавающей точкой
- Арифметика не точна



The image shows a terminal window titled "Terminal" with a menu bar containing "Файл", "Правка", "Вид", "Поиск", "Терминал", and "Справка". The terminal displays the following text:

```
>>> 0.3+0.3+0.3  
0.8999999999999999  
>>> 
```

- Для точных вычислений есть классы [Decimal](#) и [Fraction](#)

# Операторы сравнения

`==`      Равно

`!=`      Неравно

`>`      Больше

`<`      Меньше

`>=`      Больше или равно

`<=`      Меньше или равно

`(<>)`      Неравно из Python 2 Deprecated)

# Логический тип

True и False

Выражения могут трактоваться как логические значения

True и False могут трактоваться как 1 и 0

Логические операторы:

- and
- or
- not

# Комплексные числа

Есть: Сложение, вычитание,  
умножение, деление,  
возведение в степень,  
real,  
imag,  
abs

Нет: целочисленного деления,  
порядка (<, >, <=, >=)

```
>>> x = 1 + 2j
>>> x
(1+2j)
>>> y = complex(3,4)
>>> y
(3+4j)
>>> x+y
(4+6j)
>>> y.real
3.0
>>> x.imag
2.0
>>> abs(x)
2.23606797749979
```



# Ссылочные типы данных

# Списки

- Состоят из элементов, которые могут иметь разные типы
- пустой список “[]”
- объединение списков “+” (“+=”)
- повторение списка “\*” (“\*=”)

```
>>> empty = []
>>> empty
[]
>>> l = [1, 2.0, True]
>>> l
[1, 2.0, True]
>>> one = empty + [1]
>>> one
[1]
>>> one * 5
[1, 1, 1, 1, 1]
```

# Списки: индексация

```
>>> intList = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> intList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> intList[0]
0
>>> intList[9]
9
>>> intList[-1]
9
>>> intList[-4]
6
>>> intList[5]
5
>>> intList[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

# Списки: срезы

```
>>> intList = [0,1,2,3,4,5,6,7,8,9]
>>> intList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> intList[2:7]
[2, 3, 4, 5, 6]
>>> intList[2:7:3]
[2, 5]
>>> intList[-1:-4:-1]
[9, 8, 7]
>>> intList[100:110:-1]
[]
>>> intList[3:110:1]
[3, 4, 5, 6, 7, 8, 9]
>>> intList[3::1]
[3, 4, 5, 6, 7, 8, 9]
>>> intList[3::]
[3, 4, 5, 6, 7, 8, 9]
```

- форма записи  
list[begin : end : step]
- полуинтервал  
[begin, end)
- по умолчанию:  
begin = 0  
end - длина списка  
step = 1

# Списки: операторы и функции

Оператор членства  
in (not in)

```
>>> List = [1, 4, 2, 6, 7, 3]
```

```
>>> 2 in List
```

```
True
```

```
>>> 2 not in List
```

```
False
```

```
>>> min(List)
```

```
1
```

```
>>> max(List)
```

```
7
```

```
>>> sorted(List)
```

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

```
>>> sum(List)
```

```
23
```

Функции для списков:

- min(list)
- max(list)
- sorted(list)
- sum(list)
- ....

# Списки: методы

```
>>> List = [4,2,3,1]
>>> List.append(8)
>>> List
[4, 2, 3, 1, 8]
>>> List.insert(1,3)
>>> List
[4, 3, 2, 3, 1, 8]
>>> List.count(3)
2
>>> List.index(3)
1
```

list.append(element)  
list.count(element)

list.index(element)  
list.insert(n, element)

```
>>> List.sort()
>>> List
[1, 2, 3, 3, 4, 8]
>>> List.pop(5)
8
>>> List
[1, 2, 3, 3, 4]
>>> List.extend([7,9])
>>> List
[1, 2, 3, 3, 4, 7, 9]
```

list.sort()  
list.pop(n)  
list.extend(list2)

# Строки

Похожи на списки. Есть индексация, срезы, часть методов

```
string1 = "String"  
string2 = 'another string'  
string3 = """Very  
long  
string"""
```

```
>>> string1[-1:0:-2]  
'git'  
>>> min(string1)  
'S'  
>>> string1.index('n')  
4
```

```
>>> string = "This is a string"  
>>> string + "!"  
'This is a string!'  
>>> string * 2  
'This is a stringThis is a string'
```

# Строки: методы

```
>>> string = "This is a string"
>>> string.upper()
'THIS IS A STRING'
>>> string.lower()
'this is a string'
>>> string.startswith('This')
True
>>> string.replace('This', "It")
'It is a string'
>>> string.split()
['This', 'is', 'a', 'string']
>>> string.isdigit()
False
```



# Кортежи

Неизменяемый аналог списка

```
>>> my_tuple = (True, 7, 'string', 1.1)
>>> my_tuple[2]
'string'
>>> my_tuple[1:3:2]
(7,)
>>> my_tuple[1:4:2]
(7, 1.1)
>>> my_tuple = my_tuple*2
>>> my_tuple
(True, 7, 'string', 1.1, True, 7, 'string', 1.1)
```

# Кортежи: попытка изменить элемент

```
>>> my_tuple = (True, 7, 'string', 1.1)
```

```
>>> my_tuple[0] = 1
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

```
>>> my_list = [True, 7, 'string', 1.1]
```

```
>>> my_list[0] = 1
```

```
>>> my_list
```

```
[1, 7, 'string', 1.1]
```

Кортеж занимает меньше места в памяти, чем список

# Словари

Неотсортированная коллекция элементов, доступ по ключу.  
Ключ - должен быть не изменяем (число, строка, кортеж)

```
>>> my_dict = {}
>>> my_dict["key1"] = "value1"
>>> my_dict
{'key1': 'value1'}
>>> second_dict = {"id" : 10, 1 : False, "my_list" : [1,2,3]}
>>> second_dict
{1: False, 'my_list': [1, 2, 3], 'id': 10}
>>> second_dict.keys()
dict_keys([1, 'my_list', 'id'])
>>> second_dict.values()
dict_values([False, [1, 2, 3], 10])
```

# Словари: использование

```
>>> second_dict = {"id" : 10, 1 : False, "my_list" : [1,2,3]}
>>> second_dict["id"]
10
>>> second_dict["id"] = True
>>> second_dict["id"]
True
>>> second_dict["my_list"] += [4]
>>> second_dict["my_list"]
[1, 2, 3, 4]
>>> second_dict[15]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 15
```

# Сет

Неупорядоченная изменяемая коллекция с уникальными элементами

```
>>> my_set = set()
>>> my_set
set()
>>> my_set = {"string", 1}
>>> my_set
{1, 'string'}
>>> my_set.add(2)
>>> my_set
{1, 2, 'string'}
>>> my_set.add(1)
>>> my_set
{1, 2, 'string'}
```

```
>>> second_set = {1}
>>> my_set.difference(second_set)
{2, 'string'}
>>> my_set.intersection(second_set)
{1}
>>> my_set.pop()
1
>>> my_set
{2, 'string'}
>>> my_set.remove('string')
>>> my_set
{2}
```

# Преобразование типов данных

```
>>> int(1.9)
```

```
1
```

```
>>> int("10", 16)
```

```
16
```

```
>>> float("10.1")
```

```
10.1
```

```
>>> complex(1, 10)
```

```
(1+10j)
```

```
>>> str(1)
```

```
'1'
```

```
>>> tuple('hello')
```

```
('h', 'e', 'l', 'l', 'o')
```

```
>>> tuple([1,2,3])
```

```
(1, 2, 3)
```

```
>>> list((1,2,3))
```

```
[1, 2, 3]
```

```
>>> dict([(1,2), (3,4)])
```

```
{1: 2, 3: 4}
```

```
>>> set([1,2,3,2])
```

```
{1, 2, 3}
```

# Основы грамматики

# Комментарии и кодировки

```
# line comment  
a = 1 # comment
```

В python 3 используется utf-8 для строк по умолчанию

Можно явно указывать кодировку в начале файла

```
# coding=utf-8  
# coding: utf8  
# -*- coding: utf-8 -*-
```



# ВВОД-ВЫВОД

```
a = input()
b = input("Please, type b: ")
print(a)
print(1,2,3,4)
print(1,2,3,4, sep="!")
print(1,2,3,4, sep="!", end="\nend\n")
```

У функции print() есть  
именованные параметры:

- sep - разделитель (пробел)
- end - последний символ  
(перенос строки)

```
$ python3 inoutoutput.py
1
Please, type b: 2
1
1 2 3 4
1!2!3!4
1!2!3!4
end
```

# Вывод в Python 2

В Python 2 `print` используется другой синтаксис.  
Можно писать без скобок:

```
print "The answer is", 2*2  
print 2  
print 2,  
print 3
```

```
The answer is 4  
2  
2 3
```

# Условные операторы и циклы

# Условный оператор

```
a = int(input())  
if a > 0:  
    print ("a is positive")  
elif a < 0:  
    print ("a is negative")  
else:  
    print ("a is zero")
```

elif и else могут  
отсутствовать

Тернарный оператор

```
val = float(input())  
abs_val = val if val >= 0 else -val
```

# Цикл while

```
# while expression:
#     statement(s)
# else:
#     statement(s)

count = 0
while (count < 9):
    print(count, ' less than ', 9)
    count += 1
else:
    print(count, ' is not less than ', 9)
```

---

```
$ python3 example.py
0 less than 9
1 less than 9
2 less than 9
3 less than 9
4 less than 9
5 less than 9
6 less than 9
7 less than 9
8 less than 9
9 is not less than 9
```

# Цикл for

```
my_list = [0, 1, 2, 4]
for i in my_list:
    print(i, end=' ')
print()
```

```
my_tuple = ('a', 'b', 'c', 'd')
for i in my_tuple:
    print(i, end=' ')
print()
```

```
my_set = {True, False}
for i in my_set:
    print(i, end=' ')
print()
```

Цикл for  
перебирает  
элементы  
итерируемого  
объекта

```
$ python3 for.py
0 1 2 4
a b c d
False True
```

# Функция range()

Получаем неизменяемую последовательность чисел.

`range(stop)`

`range(start, stop[, step])`

по умолчанию `start = 0`, `step = 1`

`stop` не входит в результат.

```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> list(range(10, 1, -1))  
[10, 9, 8, 7, 6, 5, 4, 3, 2]
```

# Цикл for с range()

```
print(end='\t')
for j in range(6):
    print(j, end='\t')
print()
for i in range(4,7):
    print(i, end='\t')
    for j in range(6):
        print(i*j, end='\t')
    else:
        print()
```

range() часто используется  
в цикле for

	0	1	2	3	4	5
4	0	4	8	12	16	20
5	0	5	10	15	20	25
6	0	6	12	18	24	30

В Питоне 2 вместо range() использовалась xrange(),  
а range() возвращала список



# Break и continue

break - выход из внутреннего цикла

continue - переход к следующей итерации цикла

```
for i in range(10):  
    if i > 3 and i < 6:  
        continue  
    print(i)  
    if i > 7:  
        break
```

```
$ python3 break.py  
0  
1  
2  
3  
6  
7  
8
```

# Генераторы списков

```
list1 = []  
for i in range(10):  
    list1 += [i**2]
```

```
list2 = [i**2 for i in range(10)]  
my_set = {i**2 for i in range(10)}
```

```
print(list1)  
print(list2)  
print(my_set)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

# Функции

# Грамматика функции

```
def <имя функции> (<аргументы функции>):  
    <тело функции>
```

```
def foo():  
    pass
```

pass - ключевое слово, используемое там,  
где грамматика требует statement, но нам  
не нужно выполнение каких-либо действий

# Пример функций без аргументов

```
def foo():  
    print("Foo")  
    return 1
```

```
def bar():  
    print("Bar")
```

```
print(foo())  
print(bar())
```

Функция возвращает:

- какое-то значение
- None

```
$ python3 foo.py  
Foo  
1  
Bar  
None
```

# None

Ничего. Объект без поведения.

```
var = None
if var is None:
    print("It is None")
```

# Пример функции с двумя аргументами

```
def print_a_b(a, b):  
    print(a, b)
```

```
print_a_b(1, 2)  
print_a_b(3, b=4)  
print_a_b(b=6, a=5)  
lst = [7, 8]  
print_a_b(*lst)  
dct = {'a' : 9, 'b' : 10}  
print_a_b(**dct)
```

Аргументы в функцию  
можно передавать  
разными способами

```
$ python3 args.py  
1 2  
3 4  
5 6  
7 8  
9 10
```

# Аргументы функций

- Позиционные
- Позиционные со значениями по умолчанию
- Позиционные с произвольным количеством
- Именованные
- Именованные со значениями по умолчанию
- Именованные с произвольным количеством



# Позиционные аргументы

```
def foo(arg):  
    print("foo arg:", arg)
```

```
def bar(arg1, arg2, arg3, *other_args):  
    print("arg1 is", arg1)  
    print("arg2 is", arg2)  
    print("arg3 is", arg3)  
    print("other_args:", other_args)
```

```
foo("string")  
bar(1.2, True, [1,2,3], {True, False}, 4, "Hello!")
```

# Позиционные аргументы

```
$ python3 args.py  
foo arg: string  
arg1 is 1.2  
arg2 is True  
arg3 is [1, 2, 3]  
other_args: ({False, True}, 4, 'Hello!')
```

- Различаются по порядку
- Перед именем произвольного количества аргументов ставится \*
- Произвольное количество аргументов рассматривается как кортеж

# Позиционные аргументы

```
def bar(arg1, arg2 = 1, *args):  
    print("arg1 is", arg1)  
    print("arg2 is", arg2)  
    print("other_args:", args)
```

```
bar(1)
```

```
bar(1, 2)
```

```
bar(1, 2, 3)
```

```
bar(1, 2, 3, 4)
```

```
# неправильно
```

```
# bar(1, arg2 = 2, 3, 4)
```

```
# bar(1, 3, 4, arg2 = 2)
```

```
$ python3 args.py
```

```
arg1 is 1
```

```
arg2 is 1
```

```
other_args: ()
```

```
arg1 is 1
```

```
arg2 is 2
```

```
other_args: ()
```

```
arg1 is 1
```

```
arg2 is 2
```

```
other_args: (3,)
```

```
arg1 is 1
```

```
arg2 is 2
```

```
other_args: (3, 4)
```

# Именованные аргументы

Могут идти только после произвольного количества позиционных аргументов `*args`

```
def bar(*args, named1, named2 = "a"):
    print("args:", args)
    print("named1 is", named1)
    print("named2 is", named2)

bar(1, 2, named1 = 1)
bar(1, 2, named1 = 1, named2 = "z")
```

\$ python3 args.py  
args: (1, 2)  
named1 is 1  
named2 is a  
args: (1, 2)  
named1 is 1  
named2 is z

# Именованные аргументы

```
def bar(*args, **kwargs):  
    print("args:", args)  
    print("kwargs:", kwargs, "\n")
```

```
bar(1, 2, 3)  
bar(1, 2, a = 3)  
bar(a = "argument")  
bar(1, 2, a = "s", b = [1,2])
```

```
$ python3 args.py  
args: (1, 2, 3)  
kwargs: {}
```

```
args: (1, 2)  
kwargs: {'a': 3}
```

```
args: ()  
kwargs: {'a': 'argument'}
```

```
args: (1, 2)  
kwargs: {'b': [1, 2], 'a': 's'}
```

- Перед именем произвольного количества аргументов ставится \*\*
- Произвольное число аргументов - словарь

# Пространства имён

```
a = 'a'
def foo():
    b = 'b'

    def foobar():
        a = 'aa'

    def bar():
        c = 'c'
        b = 'bb'
        print(a, b, c)

    bar()

foo()
```

Каждая функция задаёт своё пространство. Поиск осуществляется сначала в своём пространстве, потом в его родителях.

```
$ python3 names.py
a bb c
```

# Изменение параметров функции

```
def foo(l1, l2, a):  
    l1 += [4, 5]  
    l1[0] = 10  
    l2 = [True, False]  
    a = 2
```

```
list1 = [1, 2, 3]  
list2 = ['a', 'b', 'c']  
a = 1+1j
```

```
foo(list1, list2, a)  
print(list1, list2, a, sep='\n')
```

Передача  
параметров:

- по ссылке
- по значению

```
$ python3 test.py  
[10, 2, 3, 4, 5]  
['a', 'b', 'c']  
(1+1j)
```

# Возвращение нескольких значений

Функция может возвращать несколько значений. Они автоматически запаковываются в кортеж и распаковываются из него.

```
def foo():  
    return 1, 2, 3
```

```
a, b, c = foo()  
print(a, b, c)  
print(foo())
```

```
$ python3 return.py  
1 2 3  
(1, 2, 3)
```



# Работа с файлами

# Открытие и закрытие файлов

```
my_file = open("filename.txt", 'r')  
my_file.close()
```

```
with open("filename.txt", 'r') as f:  
    pass
```

Файл нужно открыть перед началом работы.  
После выполнения чтения/записи файл нужно закрыть.  
Операция with open() as выполняет закрытие автоматически.

# Режимы доступа

- **r** Только для чтения. Указатель в начале
- **r+** Для чтения и записи. Указатель в начале
- **w** Только для записи. Указатель в начале
- **w+** Для чтения и записи. Указатель в начале
- **a** Для добавления в файл. Указатель в конце
- **a+** Для добавления и чтения. Указатель в конце

**w**, **w+**, **a**, **a+** создают файл с именем имя\_файла, если он не существует.

Может быть добавлено **b** для чтения в двоичном формате

# Атрибуты файлового объекта

```
with open("filename.txt", 'wb') as file:  
    print (file.closed)  
    print (file.mode)  
    print (file.name)
```

---

```
$ python3 file.py  
False  
wb  
filename.txt
```

- открыт ли файл
- режим доступа
- имя файла

# Методы чтения и записи

**f.read(n)** – прочитать n байт (n можно опустить, тогда чтение до конца файла)

**f.readline(size=-1)** – прочитать строку (не более size байт)

**f.readlines()** – прочитать все строки (список строк)

**f.seek(n)** – перейти на позицию n-го байта

**f.truncate(n)** – обрезать файл до размера n байт

**f.write(s)** – записать строку s в файл

**f.writelines(l)** – записать список строк в файл

....

# Построчное чтение

```
with open("filename.txt", 'r') as file:  
    for line in file:  
        print(line)
```

Файл - итерируемый объект!

Вопросы?

# Через неделю

- Классы и объекты
- Исключения
- Пакеты
- Библиотеки
- Где можно учиться



Спасибо за внимание