Python. Введение.

Скриптовые языки программирования

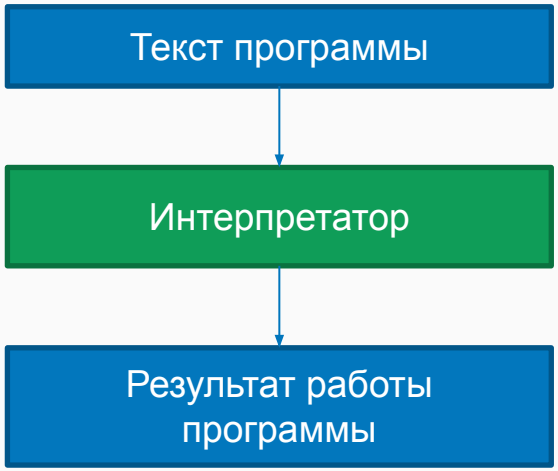
История Python

Guido van Rossum

- 1980-е – задуман Python
- 1989 – начало разработки языка
- 1991 – первая публикация кода
- 16 октября 2000 – выпущена версия 2.0 (gc, unicode)
- 3 декабря 2008 – версия 3.0

- 2010 – Python 2.7
- 2015 – Python 3.5

Приступим



Универсальнее

Быстрее

Синтаксис Python

```
code = 1
name = "Ivan Ivanov"
score = 17.26
lessons = [
    "C++",
    "Python",
    "Linux"
]
marks = {
    «Физика": 5,
    «Математика": 5,
    "Экономика": 4
}
```

- Имя переменной может состоять из букв, цифр, _.
- Двойные и одинарные кавычки равнозначны
- Все строки – unicode.
- Переменная в процессе работы программы может менять свои значения и тип.

- [PEP8] Имена переменных начинаются с маленькой буквы и формируются в snake_case.
- [PEP8] “Приватные переменные” начинаются с одного или нескольких _.

Синтаксис Python

C++

```
void foo(int x)
{
    if (x == 0) {
        bar();
        baz();
    } else {
        qux(x);
        foo(x - 1);
    }
}
```

Python

```
def foo(x):
    if x == 0:
        bar()
        baz()
    else:
        qux(x)
        foo(x - 1)
```

Синтаксис Python

```
total = item_one + \
        item_two + \
        item_three

paragraph = """Говорить по-английски
просто!
Традиционные методики в школах, ВУЗах,
на многочисленных платных курсах
практически не меняются – зубрежка,
заучивание грамматики, прослушивание
аудиоуроков.
"""

print("Hello, Python!") # комментарий
```

- Перенос statement осуществляется через обратный слеш (\).
- Multiline strings – `""" string """`
- Комментарии начинаются с #
- Многострочных комментариев нет. Вместо них используются multiline strings.

Синтаксис Python. Оператор ветвления.

```
if x >= 10:
    print("больше или равно 10")
```

```
x = 14
if x >= 10:
    print("больше или равно 10")
else:
    print("меньше 10")
```

Операторы сравнения:

==	>=
!=	<=
>	in
<	is

Любое логическое выражение имеет одно из двух значений:

- True
- False

Синтаксис Python. Оператор ветвления.

```
s = "Волшебный мир python"
if "python" in s:
    print("Что-то про питон")
elif "c++" in s:
    print("Что-то про C++")
else:
    print("Непонятно что")
```

Оператор сравнения **in** определяет вхождение левого аргумента в правый

Синтаксис Python

```
if expression:
    pass
elif expression:
    pass
else:
    pass

for i in range(100):
    print(i)

while True:
    print('hello')
    time.sleep(2)

def f(x):
    pass
```

- Группа выражений может быть объединена в блок
- Сложные выражения (напр., if, while, for, class, def) содержат заголовочную строку и блок.
- Заголовочная строка (header line) завершается двоеточием (:).
- Ключевое слово **pass** необходимо, чтобы завершить блок, в котором нет выражений.

Синтаксис Python. Функции.

```
def f(x, y):
    z = x ** 2 + y ** 2
    return z


z = f(21, 40)
z = f(21, y=40)
z = f(x=21, y=40)
```

Синтаксис Python

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n - 1) + fib(n - 2)


n1 = fib(1)  # = 1
n10 = fib(10)  # = 55
```

- Объявление функций начинается с ключевого слова **def**.
- Т.к. объявления типов в Python нет, то и аргументы функций объявляются просто именами.
- Значение из функции возвращается с помощью **return**.
- Функция может вызывать сама себя (рекурсия).

- Вызвать функцию можно либо просто передав аргументы позиционно, либо по их именам

Типы данных Python

Python поддерживает следующие простые типы данных:

- Целочисленные (любой точности) - int
  - a = 12
  - b = 10002332
  - c = 230948329482394792834798237498324
- Числа с плавающей запятой с двойной точностью - float
  - f2 = 3123.784
  - f3 = 12e234
- Строки (юникодные) - str
  - a = “привет”
- Массив байт - bytes
  - >>> b = bytes("привет", encoding="utf-8")
  - >>> b
  - B'\xd0\xbf\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82'
- Булевый тип - bool
  - 2 объекта: True и False
- NoneType
  - Единственный объект этого типа – None

Поддерживаемые операторы

Арифметические операторы:
+ - * / % **

```
a = 12 + 3   # 15
b = a - 120   # -105
c = 12.1 * 4   # 48.4
d = 12 / 4   # 3.0
mod = 123 % 2   # 1

kb = 2 ** 10   # 1024
```

Логические операторы:
and, **or**, **not**

```
x = 14
b1 = x > 10 and x < 20   # True
b2 = x < 10 or x > 20   # False

b3 = (x % 2) == 1   # False
```

Битовые операторы:
& (И) **|** (ИЛИ) **~** (НЕ) **^** (ИСКЛ. ИЛИ)

Массивы Python

Массивы – структура данных, представляющая собой непрерывную область памяти, поддерживающая динамическое добавление и удаление элементов.

```
arr1 = []        # Объявили пустой массив
arr2 = list()    # То же самое

arr1.append(1)   # Добавили в конец 1
arr1.append(2)   # Добавили в конец 2
print(arr1)      # --> [1, 2]

len(arr1)        # Размер массива (2)
len(arr2)        # Размер массива (0)

arr2.append(3)

arr3 = arr1 + arr2   # Объединение
массивов
```

```
arr1.remove(2)  # Удаление первого
вхождения элемента со значением 2
arr1.pop(0)  # Удаление элемента с
индексом 0

print(2 in arr1)  # Проверить,
содержится ли элемент со значением 2
в массиве
```

Синтаксис Python. Оператор цикла for.

```
for n in arr1:
    n2 = n * 2
    print(n2)
```

Здесь оператор **in** используется для итерации по массиву

Синтаксис Python. Оператор цикла for.

range – функция для генерации массива заданного размера

```
print(list(range(5)))  # --> [0, 1, 2, 3, 4]

print(list(range(1)))  # --> [0]

print(list(range(0)))  # --> []
```

Массивы Python

```
# Пройтись по элементам массива (способ №1)
for el in arr1:
    print(el)  # напечатает все элементы

# Пройтись по элементам массива (способ №2)
for i in range(len(arr1)):
    print(i, arr1[i])  # напечатает все элементы и их индексы

# Пройтись по элементам массива (способ №3)
for i, el in enumerate(arr1):
    print(i, el)  # напечатает все элементы и их индексы
```

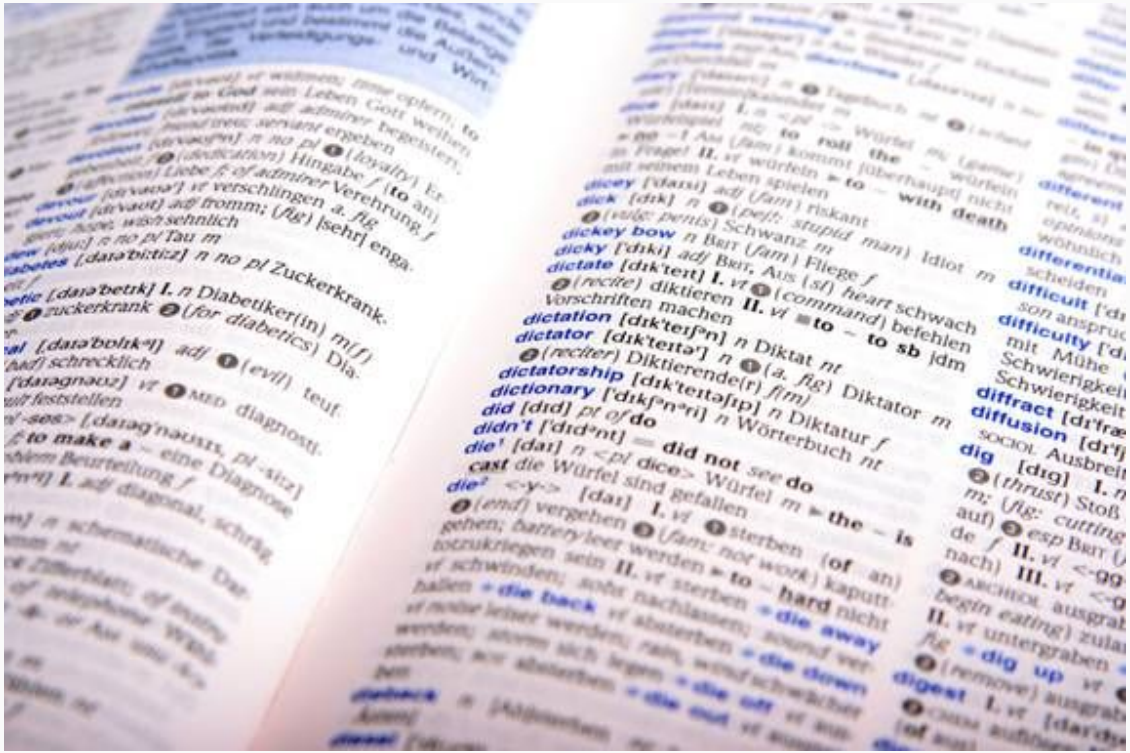
Кортежи Python

Кортежи – неизменяемые массивы. Нельзя ни добавить, ни удалить элементы из кортежа.

```
t1 = ()        # Объявили пустой кортеж
t2 = tuple()    # То же самое

t1 = (1, 2, 3)

len(t1)        # Размер кортежа (3)
len(t2)        # Размер кортежа (0)

t3 = t1 + t2   # Объединение кортежей

t4 = ("ninja",)  # Кортеж из одного элемента
```

Словари Python

Словарь - структура данных, отображающая одни объекты ***(ключи)*** в другие ***(значения)***



Словари Python

```
d1 = {
    'doctor': 'Gregory House',
    'pilot': 'Anakin Skywalker',
    'wizard': 'Gandalf The White'
}

print(d1['doctor'])
print(d1['pilot'])
print(d1['president'])  # --> KeyError
print(d1.get('president'))  # --> None

d1['president'] = 'Bill Gates'
print(d1['president'])

del d1['doctor']  # Удаление элемента
```

```
print(len(d1))  # Число ключей в словаре

print(d1.keys())  # --> ['president', 'wizard', 'pilot']

print(d1.values())  # --> ['Bill Gates', 'Gandalf The White', 'Anakin Skywalker']
```

Словари Python

```
# Пройтись по словарю (Способ №1)
for key in d1:
    print(key, d1[key])

# Пройтись по словарю (Способ №2)
for key in d1.keys():
    print(key, d1[key])

# Пройтись по словарю (Способ №3)
for key, value in d1.items():
    print(key, value)
```

Множества Python

Множество – структура данных, содержащая в себе неповторяющиеся элементы

```
s1 = set()  # Создание пустого множества
s2 = { 101, 1220, 231 }

s3 = set([1, 2, 3, 1, 2, 1, 4])  # == {1, 2, 3, 4}
s4 = { 2, 3, 6, 7 }

s3.add(5)  # Добавить элемент в множество
s3.remove(5)  # Удалить элемент из множества

s3 & s4  # Пересечение: {2, 3}
s3 | s4  # Объединение: {1, 2, 3, 4, 6, 7}
s3 ^ s4  # XOR: {1, 4, 6, 7}
s1 - s2  # Разность: {1, 4}
```

Типы данных Python. Резюме.

Python поддерживает следующие сложные типы данных:

- Массивы
  - a = [1, 2, 3]
  - b = [“hi”, “hello”, “good morning”]
  - c = [12, “soon”, 42, [1, 2, 3]]
- Кортежы (неизменяемые массивы)
  - a = (1, 2, 3)
  - b = (“hi”, “hello”, “good morning”)
  - c = (12, “soon”, 42, [1, 2, 3])
  - d = () # пустой кортеж
  - e = (12,) # кортеж из одного элемента (внимание на запятую)
- Словари
  - d = { ‘a’: 1, ‘b’: 2, ‘c’: 3 }
- Множества
  - s1 = { ‘a’, ‘b’, ‘c’ }
  - s2 = set([‘a’, ‘b’, ‘c’, ‘a’, ‘d’]) # == {‘a’, ‘b’, ‘c’, ‘d’}

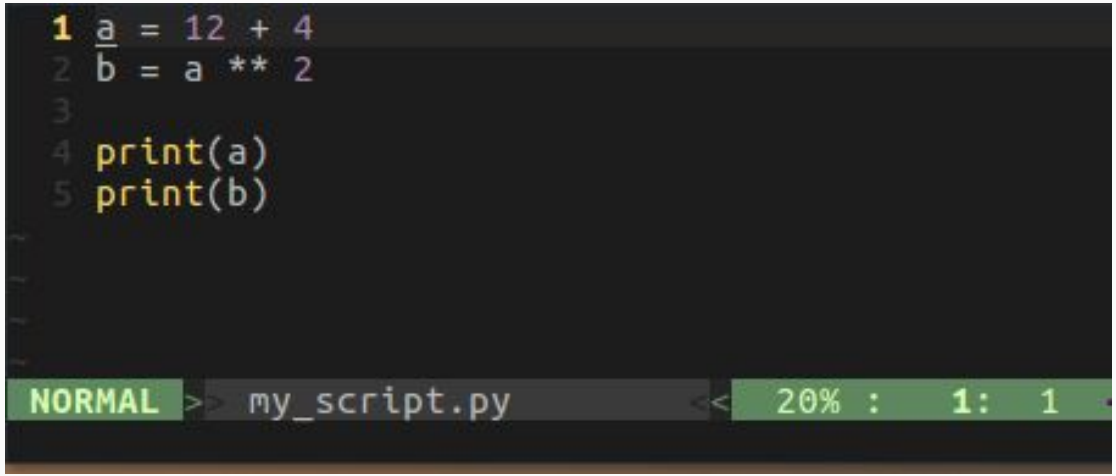
Как это использовать?

- Вариант1. Запустить python (python3) в интерактивном режиме

```
$ python3
Python 3.5.2 (default, Jul  5 2016, 12:43:10)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 12 + 4
>>> a
16
>>> _
```

Как это использовать?

- Вариант 2. Запустить скрипт с написанным заранее кодом из файла с расширением .py (например, my_script.py)




```
$ python3 my_script.py
16
256
```

list/dict comprehensions

List comprehensions

Создать массив из квадратов последовательных чисел

```
arr = []
for x in range(10):
    arr.append(x * x)
```

List comprehensions

Создать массив из квадратов последовательных чисел

```
arr = [x * x for x in range(10)]
```

```
>>> arr = [x * x for x in range(10)]
>>> arr
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List comprehensions

Создать массив из квадратов последовательных четных чисел

```
arr = [x * x for x in range(10) if x % 2 == 0]
```

```
>>> arr = [x * x for x in range(10) if x % 2 == 0]
>>> arr
[0, 4, 16, 36, 64]
```

Dict comprehensions

Создать отображение чисел в их квадраты

```
d = {}
for x in range(10):
    d[x] = x * x
```

Dict comprehensions

Создать отображение чисел в их квадраты

```
d = {x: x*x for x in range(10)}
```

```
>>> d = {x: x*x for x in range(10)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Dict comprehensions

Создать отображение простых чисел в их квадраты

```
d = {x : x*x for x in range(10) if is_prime(x)}
```

```
>>> def is_prime(x):
...     return all(x % i != 0 for i in range(2, x))
...
>>> d = {x : x*x for x in range(10) if is_prime(x)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 5: 25, 7: 49}
```

“Функциональные” функции (функции высших порядков)

Map, reduce и все такое

range(*start*, *stop*[, *step*]) - Возвращает последовательность чисел
class **list**([*iterable*]) - преобразует любую коллекцию/итератор в список

reversed(*seq*) - Переворачивает коллекцию
filter(*function*, *iterable*) - Фильтрует коллекцию, используя функцию
map(*function*, *iterable*) - функция для преобразования коллекции

sorted(*iterable[, key][, reverse]*) - Возвращает коллекцию в отсортированном виде

zip(**iterables*) - связывает коллекции между собой поэлементно

range

```
>>> range(5)
range(0, 5)
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1, 4))
[1, 2, 3]
>>> list(range(1, 7, 2))
[1, 3, 5]
```

reversed

Возвращает итератор (будет дальше)

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> reversed(arr)
<list_reverseiterator object at 0x7f75ffbab860>
>>>
>>> list(reversed(arr))
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

filter

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]

def is_even(x):
    return x % 2 == 0

filter(is_even, arr)
```

```
>>> filter(is_even, arr)
<filter object at 0x7f76018bca58>
>>>
>>> list(filter(is_even, arr))
[2, 4, 6, 8]
```

filter

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]

def is_even(x):
    return x % 2 == 0

filter(is_even, arr)
```

```
>>> filter(is_even, arr)
<filter object at 0x7f76018bca58>
>>>
>>> list(filter(is_even, arr))
[2, 4, 6, 8]
```

Данная функция должна вернуть True, если элемент, переданный в неё, должен остаться.

map

```
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
def square(x):
    return x ** 2

map(square, arr)
```

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
>>> def square(x):
...     return x ** 2
...
>>> map(square, arr)
<map object at 0x7f76018bcb00>
>>>
>>> list(map(square, arr))
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Данная функция должна вернуть преобразованный элемент.

sorted

Тут все совсем просто

```
>>> arr
[8, 3, 1, 6, 5, 7, 2, 4, 9]
>>> sorted(arr)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

sorted

Как перемешать массив?

```
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> import random
>>> random.shuffle(arr)
>>> arr
[8, 3, 1, 6, 5, 7, 2, 4, 9]
>>> sorted(arr)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> _
```

zip

Пусть для простоты есть 2 списка:

```
>>> X = [1, 2, 3, 4, 5]
>>> Y = [1, 4, 9, 16, 25]
```

Задача: пройтись по обоим массивам одновременно

zip

```
>>> X = [1, 2, 3, 4, 5]
>>> Y = [1, 4, 9, 16, 25]
```

Наивный подход:

```
assert len(X) == len(Y)
for i in range(len(X)):
    x = X[i]
    y = Y[i]
    print('Point: ({}, {})'.format(x, y))
```

```
Point: (1, 1)
Point: (2, 4)
Point: (3, 9)
Point: (4, 16)
Point: (5, 25)
```

zip

```
>>> X = [1, 2, 3, 4, 5]
>>> Y = [1, 4, 9, 16, 25]
```

Python-way:

```
for x, y in zip(X, Y):
    print('Point: ({}, {})'.format(x, y))
```

```
Point: (1, 1)
Point: (2, 4)
Point: (3, 9)
Point: (4, 16)
Point: (5, 25)
```

zip

```
>>> X = [1, 2, 3, 4, 5]
>>> Y = [1, 4, 9, 16, 25]
```

Что же делает zip?

```
>>> list(zip(X, Y))
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

Zip просто соединяет попарно элементы из переданных ему коллекций

zip

Что если колекций больше 2х?

```
>>> X = [1, 2, 3, 4, 5]
>>> Y = [1, 4, 9, 16, 25]
>>> Z = ['a', 'b', 'c', 'd']
>>>
>>> list(zip(X, Y, Z))
[(1, 1, 'a'), (2, 4, 'b'), (3, 9, 'c'), (4, 16, 'd')]
```

Все то же самое :)

Лямбда-функции

Lambda

```
def pow2(x):
    return x * x

y = 2
pow2(y) # 4

y = [2, 3, 4]
list(map(pow2, y))
# [4, 9, 16]
```

```
def get_key(x):
    return x[1]

y = {
    "a": 5,
    "b": 3,
    "c": 4
}
sorted(y.items(), key=get_key)
# b c a
```

Хочется как-то покороче...

Lambda

C#:

```
(input parameters) => expression
(int x, string s) => s.Length > x
```

Python

```
lambda input_parameters: expression
lambda x, y: x + y
```

Результат выражения будет возвращен вызывающему лямбду коду

Примеры lambda

```
# Инкремент числа
increment = lambda x: x + 1
print(increment(2)) # 3


# Возведение в квадрат
(lambda x: x * x)(5)
```

Примеры lambda

```
# Фильтрация массива
y = [25, 10, 4, 20, 50, 8]

y = list(filter(lambda x: x > 10, y)) # [25, 20, 50]
```

Примеры lambda

```
y = 'Подсчет длины слов в предложении'


list(map(lambda w: len(w), y.split()))

list(map(len, y.split()))

# [7, 5, 4, 1, 11]
```