



# Python

## Введение



# Почему Python?

- Прост в изучении
- Большое количество модулей
- Простые конструкции
- ...

HOLLY WAR! :)

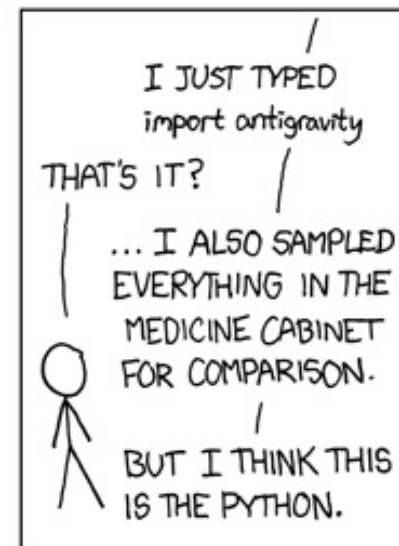
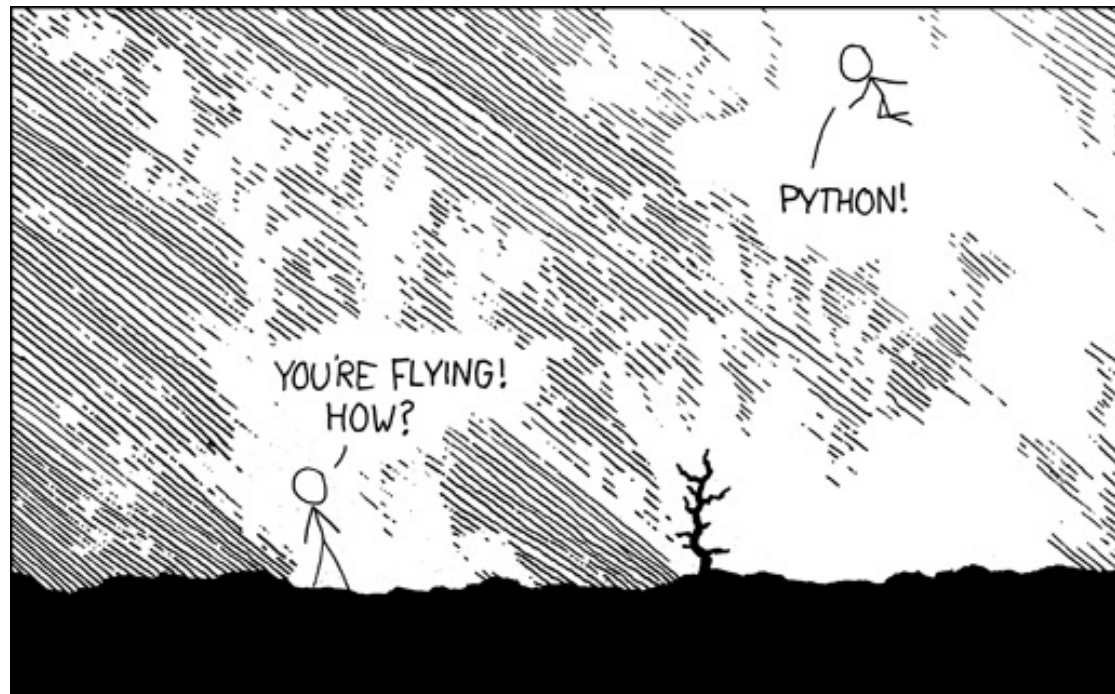


Программист на Perl смотрит на тебя как на  $\$_{[0]}$

Программист на Haskell поднимает штангу именем теории категорий



Ruby-программисты – заботливые отцы, а значит их любят девушки





# Perl vs Python

---

```
package Server;
use strict;

sub new {
    my $class = shift;
    my $self = {};
    $self->{IP} = shift;
    $self->{HOSTNAME} = shift;
    bless($self);
    return $self;
}

sub set_ip {
    my $self = shift;
    $self->{IP} = shift;
    return $self->{IP};
}

sub set_hostname {
    my $self = shift;
    $self->{HOSTNAME} = shift;
    return $self->{HOSTNAME};
}

sub ping {
    my $self = shift;
    my $external_ip = shift;
    my $self_ip = $self->{IP};
    my $self_host = $self->{HOSTNAME};
    # ...
    return 0;
}
1;
```

```
#!/usr/bin/perl

use Server;

$server = Server->new('192.168.1.15', 'grumbly');
$server->ping('192.168.1.20');
```



# Python

**Python** - высокоуровневый язык программирования общего назначения с акцентом на производительность разработчика и читаемость кода. Синтаксис ядра Python минималистичен.

**Python** поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений.



# Perl vs Python

```
#!/usr/bin/env python

class Server(object):
    def __init__(self, ip, hostname):
        self.ip = ip
        self.hostname = hostname
    def set_ip(self, ip):
        self.ip = ip
    def set_hostname(self, hostname):
        self.hostname = hostname
    def ping(self, ip_addr):
        print "Pinging %s from %s (%s)" % (ip_addr, self.ip, self.hostname)

if __name__ == '__main__':
    server = Server('192.168.1.20', 'bumbly')
    server.ping('192.168.1.15')
```



# Python - Дзен

- `>>> import this`
- The Zen of Python, by Tim Peters
  
- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *\*right\** now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!






# Как выполнять

- CPython
- Jython - Интерпретатор Python, реализованный на Java. Позволяет компилировать программы на Python в байт-код Java.
- .....



# Запуск

- python
- `#!/usr/bin/env python`
- ...
- IPython



# Python 2.x vs 3.x

- В третьей версии улучшена стандартная библиотека и добавлены новые функции.
- Много библиотек не стабильно работают на версии 3.
- Работать полностью в третьей ветке тяжело.



# Типы данных

- Int
- long int
- float
- complex

```
>> 4j + 2 + 3j
```

```
(2+7j)
```

```
>> complex (2,7)
```

```
(2+7j)
```

```
>> (2+7j ). real + (2+7j ). imag
```

```
9.0
```

```
>> (2+7j ). conjugate ()
```

```
(2-7j)
```



# Переменные


```
>>> x = 2
```

```
>>> x
```

```
2
```

```
>>> print x
```

```
2
```



## Получение данных от пользователя

```
>>> x = raw_input("Hello: ")
```

```
Hello: aaaa
```

```
>>> x
```

```
'aaaa'
```

```
>>> x = int(raw_input("Hello: "))
```

```
Hello: 123
```

```
>>> x
```

```
123
```



# УСЛОВИЯ

```
>>> if x < 0:  
...     x = 0  
...     print 'Negative'  
... elif x == 0:  
...     print 'Zero'  
... else:  
...     print 'Positive'
```

Сравнение: == != >= <=  
x = a if (condition) else b



# While

- `>>> #comment`
- `... a, b = 0, 1 # множественное присваивание`
- `>>> while b < 100:`
- `... print b`
- `... a, b = b, a+b`
- `...`
- `1`
- `1`
- `2`
- `3`
- `5`
- `8`
- `13`
- `21`
- `34`
- `55`
- `89`





# While

```
>>> #comment
... a, b = 0, 1
>>> while b < 100:
...     print b,
...     a, b = b, a+b
...
1 1 2 3 5 8 13 21 34 55 89
```



# Комментарии

# - однострочный комментарий

""" – многострочный комментарий. Доступен через `__doc__` или `help(...)`

```
>>> def sum(a, b):  
...     """  
...     Returns sum of a and b  
...     """  
...     return a+b  
>>> print sum.__doc__
```

Returns sum of a and b

```
>>> help(sum)
```



# Списки

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a[2] = a[2] + 23
>>> a
['spam', 'eggs', 123, 1234]
Присваивание срезу:
>>> a[0:2] = [1, 12] # замена
>>> a
[1, 12, 123, 1234]
>>> a[0:2] = [] # удаление
>>> a
[123, 1234]
>>> a[1:1] = ['bletch', 'xyzzy'] # вставка
>>> a
[123, 'bletch', 'xyzzy', 1234]
```



# Списки

Четыре способа добавить элементы в список.

```
>>> a_list = ['a']
>>> a_list = a_list + [2.0, 3]
>>> a_list
['a', 2.0, 3]
>>> a_list.append(True)
>>> a_list
['a', 2.0, 3, True]
>>> a_list.extend(['four', 'Ω'])
>>> a_list
['a', 2.0, 3, True, 'four', 'Ω']
>>> a_list.insert(0, 'Ω')
>>> a_list
['Ω', 'a', 2.0, 3, True, 'four', 'Ω']
```



# СПИСКИ

Удаление элементов из списка:.

```
>>> a_list = ['a', 'b', 'new', 'mpilgrim', 'new']
```

```
>>> del a_list[1]
```

```
>>> a_list
```

```
>>> ['a', 'new', 'mpilgrim', 'new']
```

```
>>> a_list.remove('new')
```

```
>>> a_list
```

```
['a', 'mpilgrim', 'new']
```

```
>>> a_list.pop()
```

```
'new'
```

```
>>> a_list
```

```
['a', 'mpilgrim']
```

```
>>> a_list.pop(1)
```

```
'mpilgrim'
```



# СПИСКИ

Поиск

```
>>> a_list = ['a', 'b', 'new', 'mpilgrim', 'new']
```

```
>>> a_list.count('new')
```

```
2
```

```
>>> 'new' in a_list
```

```
True
```

```
>>> 'c' in a_list
```

```
False
```

```
>>> a_list.index('mpilgrim')
```

```
3
```

```
>>> a_list.index('new')
```

```
2
```

```
>>> a_list.index('c')
```

```
Traceback (innermost last):
```

```
File "<interactive input>", line 1, in ?
```

```
ValueError: list.index(x): x not in list
```



# Кортежи

Значения менять нельзя!

```
>>> t = 12345, 54321, 'hello!'
```

```
>>> t[0]
```

```
12345
```

```
>>> t
```

```
(12345, 54321, 'hello!')
```

```
>>> u = t, (1, 2, 3, 4, 5) # могут быть вложенными
```

```
>>> u
```

```
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
>>> empty = ()
```



# for, range

```
>>> list = ['It', 'is an', 'interesting', 'lecture']
```

```
>>> for x in list:
```

```
... print x,
```

```
....
```

```
It is an interesting lecture
```

```
для удобства
```

```
>>> range(10)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>> range(5, 10) # диапазон
```

```
[5, 6, 7, 8, 9]
```

```
>>> range(0, 10, 3) # задаем шаг
```

```
[0, 3, 6, 9]
```





# Словари

```
>>> dict = {} # пустой словарь
```

```
>>> circus = {"lion" : 4, "hippo" : 1, "giraffe" : 2}
```

```
>>> circus["hippo"]
```

```
1
```

```
>>> circus["snake"] = 7 # добавление ключа
```

```
>>> circus["lion"] = 5 # изменение ключа
```

```
>>> circus
```

```
{'hippo': 1, 'lion': 5, 'giraffe': 2, 'snake': 7}
```



# Словари

```
>>> len(circus)#количество элементов в словаре
5
>>> circus["cat"] = "yes, please!"#разные типы значений
>>> circus[42] = "number"# разные типы ключей
>>> del circus["hippo"]# удаление ключа
>>> circus
{42: 'number', 'cat': 'yes, please!', 'lion': 5, 'giraffe': 2, 'snake': 7}
>>> circus.keys()# возвращает список ключей
[42, 'cat', 'lion', 'giraffe', 'snake']
>>> circus.values()# возвращает список значений
['number', 'yes, please!', 5, 2, 7]
>>> 'dog' in circus#проверяет наличие ключа в словаре
False
```

**При обращении по несуществующему ключу –исключение KeyError**



# Исключения

```
>>> print circus['dog']  
Traceback(most recent call last):  
  File "<pyshell#1>", line 1, in <module>  
    print circus['dog']  
KeyError: 'dog'
```

Можно поймать это исключение:

try:

```
    print circus['dog']
```

except KeyError:

```
    print "No such key in the dictionary"
```



# ФУНКЦИИ

Передача аргументов в функцию -по ссылке.

```
def dostuff(mylist) :
```

```
    """ Appends [1, 2, 3] to the list. """
```

```
    mylist.append([1, 2, 3])
```

```
    mylist= ['Katya', 'Kolya', 'Vitya']
```

```
    mylist.append('Sveta')
```

```
    print mylist
```

```
a = [4, 5, 'f']
```

```
dostuff(a)# результат: ['Katya', 'Kolya', 'Vitya', 'Sveta']
```

```
print a# результат: [4, 5, 'f', [1, 2, 3]]
```



# Функции

НО!

```
def square(n) :
```

```
    n *= n
```

```
a = 3
```

```
square(a)
```

```
print a
```

Результат выполнения:

3

Потому что число –простой тип, оно передается по значению.



# Переменные

Переменные внутри функции –локальные. Поиск переменных: сперва среди локальных, потом среди глобальных, потом среди встроенных.

```
n = 10
```

```
def printn():
```

```
    print n# переменная nвидна внутри функции
```

```
def changeandprintn()
```

```
    n = 2# теперь n –это локальная переменная
```

```
    print n
```

```
printn()
```

```
changeandprintn()
```

```
print n
```

Результат выполнения:

```
10
```

```
2
```

```
10
```



# Переменные

```
n = 10
def reallychangeandprintn()
    global n# переменная n –глобальная
    n = 2
    print n
```

```
reallychangeandprintn()
print n
```

Результат:

2

2

**global** a, b, c-указывает, что идентификаторы a, b, c в текущем блоке ссылаются на глобальные переменные



# ФУНКЦИИ

Как передать в функцию произвольное число аргументов:

`f([formal_args,] *tuple)`: tuple—кортеж, содержащий аргументы, не входящие в список формальных параметров

```
def mean(*args):
```

```
    sum = 0.
```

```
    for a in args:
```

```
        sum += a
```

```
    return sum/len(args)
```

```
print mean(1, 2, 3, 4, 5)# результат: 3
```

```
print mean(40, 3)# результат: 21.5
```





# Задание

0) <http://www.pythonchallenge.com/>

- 1) Посчитать количество существительных в романе «Война и мир». Смотреть на **Py morphology**
- 2) Найдите все составные числа меньше  $N$ , которые представимы в виде произведения двух простых чисел.
- 3) Написать функцию, вычисляющую произведение двух матриц (матрица – список списков). Также написать функцию для вывода такой матрицы в красивом виде.