

Введение в питон



Guido van Rossum (Гвидо ван Россум)
автор языка Python и
«великодушный пожизненный диктатор»
в 2018 ушел в отпуск, в 2020 году в отставку
сейчас работает в Microsoft

- Актуальная версия python-3: <https://www.python.org>

Запуск и выполнение программ на Python

Питон как калькулятор

- «запустив» `python3` попадаем в интерактивную оболочку пригодную как для простых вычислений так и для программ
- настройку интерактивной оболочки делают через питоновскую же программу: `export PYTHONSTARTUP=~/.pythonrc`

```
# .pythonrc - Python start up file  
import math
```

Выполнение программы

- программа на питоне это текстовый файл (`.py`) в кодировке UTF8

```
# hello.py  
print('Hello World!', "Привет мир!")
```

- Выполнение: `% python3 hello.py`

Переменные

- в питоне переменная всегда **указывает, ссылается** на какой то объект
- объекты имеют свой определенный тип: целое число, текст, список ...
- 👉 переменные могут менять указание на объект другого типа:
динамическая типизация

```
x = 1.2345      # floating point number
i = 1; j = 2    # two integer variables on one line
s = 'hello'     # string: текст в одинарных или двойных кавычках
l = [1,2,3]     # list, список заключенный в []
x = 'text'      # теперь x это string
```

- 👉 # – начало однострочного комментария, а многострочных нет
- 👉 ; – точка с запятой необязательна, но может использоваться для разделения нескольких операторов на одной линии

Числа

Целые: положительные, отрицательные, ноль 📌 размер неограничен

```
i = 10_000_000    # '_' используется для улучшения читаемости
print(i,type(i))  # 10000000, <class 'int'>
b = 0b1100_0011   # 195 бинарный формат
e = 0o12;h= 0xAF   # e=10 восьмеричное, h=175 шестнадцатеричное
```

С плавающей точкой: binary64 в IEEE-754 стандарте (double в C)

```
f = 1.2345_6789   # число с плавающей точкой: float
print(f,type(f))  # 1.23456789 <class 'float'>
d = 1e-10          # 0.0000000001 научная нотация
```

Комплексные: числа с действительной и мнимой float частью

```
c = 1+1j          # комплексное число, перед j должно быть число
print(c,type(c))  # (1+1j) <class 'complex'>
1+j;1+1*j       # SyntaxError: нет числа; умножение не нужно
```

Простейшие операции

- арифметические: `+` `-` `*` `/` `%`, `//` – целое деление, `**` – возведение в степень; некоторые из них работают и со «сложными» типами:

```
s='hi'*3 # s = 'hihihi'
```

- гибридные операции: `+=` `*=` `...`

```
s+='ha' # s = 'hihihiha'
```

- операции сравнения: `==` `!=` `>=` `>` `<=` `<`

```
1 == 2 # False
```

```
2 >= 2 # True
```

True and False are boolean constants

```
1 < x < 2 # chained inequalities are allowed in Python!
```

- логические операции: `and`, `or`, `not`:

```
x == 2 or y == 5
```

```
x > 1 and x < 2 # the same as 1 < x < 2
```

```
not( x!=1 or y!=1 ) # brackets are required
```



операции `++` и `--` отсутствуют! Используйте `x+=1`

Управление потоком вычислений

• условный оператор if

```
if 1 < x < 2:      # braces are optional
    print(x-1)
elif 2 < x < 3:
    print(x-2)
else:
    print('x=',x)
```

• цикл while [else] (else необязательная часть)

```
i=1; s=0
while i<10:
    s+=i; i+=1
else: executed when the condition is false
    s/=i
print(s)  # 4.5
```

● цикл `for ... in [else]`

- используется для перебора элементов некоторой «коллекции»

```
lst=[1,2,3]                # list
for l in lst:               # l will run through 1,2,3
    print('l=',l,end=', ') # l= 1, l= 2, l= 3,
else:
    print('end of for')    # end
```

- необязательный блок `else` содержит код, который будет выполнен после нормального завершения цикла

👉 «новый `for`» в C++11 позаимствован из питона

● операторы `break` и `continue`

- работают в циклах так же как в C
- блок `else` не будет выполнен, если цикл остановлен оператором `break`

Блоки выполнения

- ✓ Питон использует отступы для группировки операторов в блоки
- ✓ Отступы состоят из пробелов и знаков табуляции, которые превращаются в пробелы
- ✓ Величина отступов **должна быть одинаковой**, выполнение остановится если будет ошибка в отступах
- 👉 **Никогда не смешивайте пробелы и знаки табуляции в одной программе**

Нравится вам это или нет

... Python isn't going to stop using it

Список (List)

- ✓ список хранит последовательность данных, возможно разного типа
- ✓ обращение по индексу

```
lst=['start',1,2.34,0]    # list for example
print(lst[0])             # start
print(lst[1])             # 1
print(lst[-1])            # 0 (the last element)
print(lst[4])           # IndexError: list index out of range
lst.append("end")         # add to the end of the list
print(lst[4])             # end
print(lst[-1])            # end
```

- 📌 отрицательные индексы отсчитываются от конца списка:
-1 – последний элемент

● под-списки, срезы, slices

👉 границы задаются в полуоткрытом промежутке $bg:end \equiv [bg, end)$

```
l=[1,3,5,7,9]          # list for example
print(l[2:4])           # [5, 7]
print(l[:2])            # [1, 3] starting from the first element
print(l[3:])            # [7, 9] ending with the last element
print(l[3:-1])          # [7]
```

● добавление/удаление элементов

```
l=[]                    # [] empty list
l.append(1)             # [1] add to the end
l+= [3,5,7]             # [1, 3, 5, 7] add to the end
l.insert(2, 'A')        # [1, 3, 'A', 5, 7], insert before index
l.pop()                 # return 7 and delete the last: [1, 3, 'A', 5]
l.remove('A')           # [1, 3, 5] delete the first occurrence of 'A'
del l[:2]               # [5] delete slice
```

● операции со списками

```
la=[2,4,6,8]           # lists for example
lb=[3,5,7]             #
lc=la+lb               # lc = [2, 4, 6, 8, 3, 5, 7]
lb*=2                  # lb = [3, 5, 7, 3, 5, 7]
l=[0.]*1000            # лист на 1000 элементов из нулей
```

● множество функций работы со списком

```
l=[1,3,9,17,13]        # list for example
print(len(l))          # 5   - number of elements
print(max(l))          # 17  - maximal element
print(min(l))          # 1   - minimal element
print(l.index(9))       # 2   - index in list
l.reverse()            # [13, 17, 9, 3, 1]
l.sort()               # [1, 3, 9, 13, 17]
L=l.copy()             # new list: L=[1,3,9,13,17]
```

Генераторы списков (list comprehensions)

Конструкций имеющие следующий вид:

- 1 `[expression for item in object]`
- 2 `[expression for item in object if condition]`

```
l=list(range(1,5))    # list for examples: [1, 2, 3, 4]
```

```
dl=[2*v for v in l]           # doubling all elements: [2, 4, 6, 8]  
dlo=[2*v for v in l if v%2 == 1] # odd elements only: [2, 6]
```

```
x=[0,1]; y=[-1,1]  
xy=[[a,b] for a in x for b in y]  
# nested list: [[0, -1], [0, 1], [1, -1], [1, 1]]
```

Кортеж (Tuple)

- ✓ последовательность объектов, как в списке, но после создания изменить `tuple` нельзя: `immutable`
- ✓ легко распаковывается в «набор переменных»

```
tpl=()                # empty tuple
tpl='start',          # one item: comma is required
tpl='start',1,2.34,0  # 4 items: ('start', 1, 2.34, 0)
print(tpl[0],tpl[-1]) # start 0
tpl[1]=-2           # TypeError exeption
s,a,b,c = tpl         # unpacking: b is 2.34 for example
_,a,b,_ = tpl         # распаковка: _ для «неинтересных» элементов
x,y = y,x             # swap two variables
```

 скобки необязательны: `(a,b)` то же самое как `a,b`

Стринг (String)

- ✓ последовательность символов UTF8: можно использовать как одинарные так и двойные кавычки 'text' или "text"
- ✓ многострочный текст можно задать с помощью тройных кавычек:
''' ... ''' или """ ... """
- ✓ **immutable** – после создания изменить нельзя
- ✓ индексирование, суб-стринги (slicing), множество встроенных функций...

```
s="Привет Мир!"          # двойные кавычки часть текста
print(s)                 # "Привет Мир!"
print(s[8:-2])           # Мир
s[7]='_-'                # TypeError exeption
s=s[:7]+'_'+s[8:]        # "Привет_Мир!" => сделали новый стринг
len(s)                   # 13
print(11, str(12))       # 11 12 - кавычки не печатаются
```

Разбор текста, parsing

```
txt='      [1.23, 4, 5, 6.78]' # text for example
txt=txt.lstrip(' [') # strip away the brackets and spaces to the left
txt=txt.rstrip('] ') # ... to the right: '1.23, 4, 5, 6.78'
lst=txt.split(',') # split text into parts using comma as separator
                    # lst = ['1.23', ' 4', ' 5', ' 6.78']
nums=[float(n) for n in lst] # convert from string to floating point
                    # nums = [1.23, 4.0, 5.0, 6.78]
```

в одну строку

```
nums=[float(n) for n in txt.strip(' [] ').split(',')]  
print(nums) # [1.23, 4.0, 5.0, 6.78]
```

join() – функция обратная к split()

```
s=', '.join([str(n) for n in nums]) # convert to string  
print(s) # 1.23,4.0,5.0,6.78 note: no quotes
```

«Красивое» форматирование: `str.format()`

- Фигурные скобки, называемые полями формата, заменяются объектами, переданными в `str.format()`

```
s='heads {} and legs {}'          # {} - format fields
sf=s.format(1,2)                  # 'heads 1 and legs 2'
```

- Внутри фигурных скобок могут стоять номера позиционных или имена именованных аргументов ➡ про аргументы функции в следующей лекции

```
s='heads {h} and legs {l}'        # h,k - keys
sf=s.format(l=1,h=2)              # 'heads 2 and legs 1'
```

- Можно использовать спецификаторы формата

```
s='animal {a:s} has {l:d} legs'   # s,d - format specifiers
sf=s.format(a='sheep',l=4)        # 'animal sheep has 4 legs'
import math
s='π= {v:f} or {v:.10f} or {v:6.2f}' # floating point specifiers
sf=s.format(v=math.pi)           # π= 3.141593 or 3.1415926536 or 3.14
```


f-string или интерполяция (python \geq 3.6)

- перед кавычками ставится **f** и внутри в фигурных скобках выражение для печати возможно со спецификатором формата

```
sfp = f'pi/2 = {math.pi/2}'          # 'pi/2 = 1.5707963267948966'  
sfe = f'e^2 = {math.e**2 : 9.6f}'    # 'e^2 = 7.389056'
```

Старый способ форматирования: не рекомендуется

- оператор **%** для **string**

```
osf = 'pi= %.6f' % math.pi # pi= 3.141593
```

 спецификаторы формата во всех трех способах «такие же» как в C


Вывод на экран, print()

Функция print()

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- Каждый `object` превращается в текст вызовом `str()`
- Объекты разделяются пробелом или `sep`
- Печать завершается символом новой строки `\n` или `end`
- Для форматирования используйте `str.format()` или `f-string`

```
print(1,2,3)                # 1 2 3
print(1,2,3,sep=':')        # 1:2:3
print(1,2,3,end=' ');print(4) # 1 2 3 4
```

 Более подробно про аргументы функции в следующей лекции

Ввод с клавиатуры

Функция `input(prompt)`

```
s = input('enter an integer number: ')
enter an integer number: -23
type(s)      # <class 'str'>
print(s)     # -23
```

- 1 если `prompt` задан, то он печатается в стандартный вывод
- 2 читает одну линию и возвращает `string` объект, `'\n'` исключается

Множество (Set)

- ✓ Неупорядоченный набор **без повторяющихся элементов**
- ✓ Имеются математические операции: объединение (**union**), пересечение (**intersection**), разность (**difference**) и симметричная разность

```
A = {1, 3}           # создается set {1,3}
A.add(2)             # 1,2,3
A.add(2)             # 1,2,3 все элементы уникальны
B = set()            # пустой set, а {} это пустой dictionary
B.update([6,5,4,3,4,5]) # {3, 4, 5, 6} обновление из list, tuple...
for i in B: print(i)  # цикл по set: 3,4,5,6
len(B)               # 4 количество элементов
```

frozenset: замороженное, (не может быть изменено) множество

✓ `FB = frozenset(B)` # FB такое же как B, но immutable

two sets for the examples

A = {1, 2, 3, 4, 5}; B = {4, 5, 6, 7, 8}

операции с множествами

```
U = A.union(B)           # {1, 2, 3, 4, 5, 6, 7, 8} union of two sets
U = A | B                 # the same with operator '|'
I = A.intersection(B)    # {4, 5} intersection of two sets
I = A & B                 # the same this operator '&'
D = B.difference(A)       # {8, 6, 7} only in B but not in A
D = A - B                 # {1, 2, 3} only in A but not in B
SD = A.symmetric_difference(B) # in A and B but not in both
SD = A ^ B                # {1, 2, 3, 6, 7, 8}
```

генераторы множеств (set comprehensions)

👉 аналогичны генераторам списков, но заключены в фигурные скобки

```
A = {x for x in 'This is a set comprehension' if x not in 'abracadabra'}
print(A) # {' ', 'p', 'h', 'T', 'i', 'm', 'o', 'n', 'e', 's', 't'}
```

Ассоциативный массив (Dictionary)

- ✓ Ассоциативный массив задает соответствие между множеством ключей `keys` и множеством значений `values`
- ✓ содержит пары (k, v) где k ключ, а v значение
- ✓ ключ `key` уникален: может встречаться в словаре только один раз

Example: name of coordinate → value

```
pp={}                                # empty dictionary
pp={'x':1.2,'y':-4.3,'z':456}        # 3 entries
pp['r']=sqrt(sum([v**2 for v in pp])) # add one more entry
pp['theta']=acos(pp['z']/pp['r'])      # add one more entry
pp['phi']=atan2(pp['y'],pp['x'])       # add one more entry
rho=pp['rho']                      # KeyError!
rho=pp['r']                          # access
del pp['x']                           # remove key-value pair
```

● функция `get(key)`

```
print(pp.get('y'))           # -4.3
print(pp.get('rho'))         # None функция "ничего не вернула"
if pp.get('rho') == None:
    print('no rho')          # no rho
```

● в словаре цикл `for` идет по всем ключам

```
for k in pp:
    print(k, '->', pp[k])
```

● генераторы словарей (dict comprehensions)

👉 аналогичны генераторам множеств, но в левой части перед `for` стоит пара **ключ : значение** разделенные двоеточием

```
C = {x : int(x,16) for x in 'ABCDEF'}
print(C) # {'A': 10, 'B': 11, 'C': 12, 'D': 13, 'E': 14, 'F': 15}
```

Оператор in

• val in «collection»:

- ✓ для `list`, `tuple`, `string`, `set` проверяет наличие указанного значения в наборе
- ✓ для словаря проверяет наличие указанного ключа

Пример: list

```
l=['white','black','red','blue']  
'red' in l           # True  
'Red' in l           # False  
'Red' not in l       # True
```

Пример: dictionary

```
d={1:'o', 2:'t', 10:'x'}  
1 in d               # True  
'o' in d             # False  
10 not in d          # False
```

👉 `val not in obj` \equiv `not(val in obj)`

Дополнительные слайды

Функции

- для задания функции используется оператор `def`

```
def fact(k):    # fact - name of function, k - argument
    ret = 1
    if k > 2:
        for i in range(2,k+1):
            ret *= i
    return ret
```

```
i=30
print(str(i)+"!= ",fact(i)) # 30!= 265252859812191058636308480000000
```



Более подробно о функциях в следующей лекции

Некоторые встроенные функции

- `int()` преобразует `float` или `string` в `int`
`int(' 123_000 ')` `# 123000`
- `float()` преобразует `int` или `string` в `float`
`float(' 123.456 ')` `# 123.456`
- `complex()` преобразует `string` или два числа в `complex`
`complex('1+5j')` `# (1+5j)`
`complex(5,1)` `# (5+1j)`
- `str()` преобразует «все что угодно» в `string`:
`str(math.pi)` `# '3.141592653589793'`
- `pow(base,exp[,mod])` возведение в степень: `base**exp` для двух аргументов и `base**exp % mod` для трех целых аргументов
`pow(11,12345,5)` `# 1`
- `abs()` абсолютное значение числа
`abs(1+1j)` `# 1.4142135623730951`

функция `range(start, stop, step)`

- `range(i, j)` возвращает «список» с элементами: $[i, i + 1, \dots, j - 1]$
- `range(j)` то же самое, что `range(0, j)`;
- `range(i, j, k)`, `k` – шаг приращения

👉 часто используется в `for` после `in`

```
for l in range(1,4):    # l will run through 1,2,3
    print(l,end=', ')   # 1, 2, 3,
```