

## Базовые типы

*integer, float, boolean, string*

```
int 783 0 -192
float 9.23 0.0 -1.7e-6
bool True False
str "One\nTwo" 'I\'m'
```

↑  
неизменяемая, упорядоченная последовательность символов

перевод строки  
многострочные  
экранирована  
символ табуляции

## Контейнерные типы

- упорядоченная последовательность, быстрый доступ по индексу
- как упорядоченная последовательность символов
- порядок заранее неизвестен, быстрый доступ по ключу, ключи = базовые типы или кортежи

```
list [1,5,9] ["x",11,8.9] ["word"] []
tuple (1,5,9) 11,"y",7.4 ("word",) ()
dict {"key":"value"} {}
set {"key1","key2"} {1,9,3,0} set()
```

словарь  
соответствие между ключами и значениями

## Имена

для переменных, функций, модулей, классов...

**a..zA..Z\_** потом **a..zA..Z\_0..9**

- нелатинские буквы разрешены, но избегайте их
- ключевые слова языка запрещены
- маленькие/БОЛЬШИЕ буквы отличаются

```
@ a toto x7 y_max BigOne
@ 8y and
```

## Присвоение переменным

```
x = 1.2+8+sin(0)
y,z,r = 9.2,-7.6,"bad"
```

↑  
значение или вычисляемое выражение  
имя переменной (идентификатор)

имена переменных  
контейнер с несколькими значениями (здесь кортеж)

**x+=3** ← добавление  
вычитание → **x-=2**

**x=None** «неопределённая» константа

## Преобразования

**type(выражение)**

```
int("15") можно указать целое основание системы исчисления вторым параметром
int(15.56) отбросить дробную часть (для округления делайте round(15.56))
float("-11.24e8")
str(78.3) и для буквального преобразования → repr("Text")
см. форматирование строк на другой стороне для более тонкого контроля
bool → используйте сравнения (==, !=, <, >, ...), дающие логический результат
list("abc") использует каждый элемент последовательности → ['a','b','c']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) использует каждый элемент последовательности → {'one','two'}
": ".join(['toto','12','pswd']) → 'toto:12:pswd'
соединяющая строка последовательность строк
"words with spaces".split() → ['words','with','spaces']
"1,4,8,2".split(",") → ['1','4','8','2']
строка-разделитель
```

## Доступ к элементам последовательностей

для списков, кортежей, строк...

отрицательный индекс	-6	-5	-4	-3	-2	-1
положительный индекс	0	1	2	3	4	5

```
lst=[11, 67, "abc", 3.14, 42, 1968]
```

положительный срез 0 1 2 3 4 5 6  
отрицательный срез -6 -5 -4 -3 -2 -1

```
lst[: -1] → [11, 67, "abc", 3.14, 42]
lst[1: -1] → [67, "abc", 3.14, 42]
lst[: : 2] → [11, "abc", 42]
lst[: :] → [11, 67, "abc", 3.14, 42, 1968]
```

срез без указания границ → с начала до конца

## Доступ к элементам последовательностей

```
len(lst) → 6
```

доступ к отдельным элементам через [индекс]

```
lst[1] → 67 lst[0] → 11 первый
lst[-2] → 42 lst[-1] → 1968 последний
```

доступ к подпоследовательности [начало среза : конец среза : шаг]

```
lst[1:3] → [67, "abc"]
lst[-3:-1] → [3.14, 42]
lst[:3] → [11, 67, "abc"]
lst[4:] → [42, 1968]
```

Для изменяемых последовательностей, полезно удаление **del lst[3:5]** и изменение с помощью присвоения **lst[1:4]=['hop', 9]**

## Булева логика

Сравнения: < > <= >= == !=  
≤ ≥ = ≠

```
a and b логическое и оба верны одновременно
a or b логическое или верно хотя бы одно
not a логическое нет
True константа «истина»
False константа «ложь»
```

## Блоки инструкций

```
родительская инструкция :
→ блок инструкций 1...
:
родительская инструкция :
→ блок инструкций 2...
:
след. инструкция после блока 1
```

## Условный оператор

выражения в блоке выполняется только если условие истинно

```
if логическое выражение :
→ блок выражений
```

может сопровождаться несколькими elif, elif, ..., но только одним окончательным else. Пример:

```
if x==42:
# блок выполнится, если x==42 истинно
print("real truth")
elif x>0:
# иначе блок, если лог. выражение x > 0 истинно
print("be positive")
elif bFinished:
# иначе блок, если лог. перем. bFinished истинна
print("how, finished")
else:
# иначе блок для всех остальных случаев
print("when it's not")
```

## Математика

числа с плавающей точкой... приближенные значения!

Операторы: + - \* / // % \*\*  
× ÷ ↑ ↑ a<sup>b</sup>  
деление без остатка остаток

```
(1+5.3)*2 → 12.6
abs(-3.2) → 3.2
round(3.57, 1) → 3.6
```

углы в радианах

```
from math import sin, pi...
sin(pi/4) → 0.707...
cos(2*pi/3) → -0.4999...
acos(0.5) → 1.0471...
sqrt(81) → 9.0
log(e**2) → 2.0 и т.д. (см. доки)
```

блок инструкций выполняется  
до тех пор, пока условие истинно

## Цикл с условием

**while** логическое выражение:

→ блок инструкций

**s = 0**  
**i = 1** } инициализации **перед** циклом

условие с хотя бы одним изменяющимся значением  
(здесь i)

```
while i <= 100:
    # выражения вычисляются пока i ≤ 100
    s = s + i**2
    i = i + 1
```

**print("sum:", s)** } вычисленный результат цикла  
! остерегайтесь бесконечных циклов!

$$s = \sum_{i=1}^{100} i^2$$

блок инструкций выполняется  
для всех элементов контейнера или итератора

## Цикл перебора

**for** переменная **in** последовательность:  
→ блок инструкций

Проход по элементам последовательности

**s = "Some text"** } инициализации **перед** циклом  
**cnt = 0**  
переменная цикла, значение управляется циклом **for**

```
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found", cnt, "e")
```

Посчитать число  
букв **e** в строке

цикл по dict/set = цикл по последовательности ключей

используйте срезы для проходов по подпоследовательностям

Проход по индексам последовательности

□ можно присваивать элемент по индексу

□ доступ к соседним элементам

```
lst = [11, 18, 9, 12, 23, 4, 17]
```

```
lost = []
```

```
for idx in range(len(lst)):
```

```
    val = lst[idx]
```

```
    if val > 15:
```

```
        lost.append(val)
```

```
    lst[idx] = 15
```

```
print("modif:", lst, "-lost:", lost)
```

Ограничить значения  
больше 15, запомнить  
потерянные значения

Пройти одновременно по индексам и значениям:

```
for idx, val in enumerate(lst):
```

**print("v=", 3, "cm :", x, ", ", y+4)**

## Печать / Ввод

элементы для отображения: литералы, переменные, выражения  
настройки **print**:

□ **sep=" "** (разделитель аргументов, по умолч. пробел)

□ **end="\n"** (конец печати, по умолч. перевод строки)

□ **file=f** (печать в файл, по умолч. стандартный вывод)

```
s = input("Instructions: ")
```

**input** всегда возвращает строку, преобразуйте её к нужному  
типу сами (см. «Преобразования» на другой стороне).

**len(c)** → количество элементов

**min(c)** **max(c)** **sum(c)**

**sorted(c)** → отсортированная копия

**val in c** → boolean, membership operator **in** (absence **not in**)

**enumerate(c)** → итератор по парам (индекс, значение)

Только для последовательностей (lists, tuples, strings):

**reversed(c)** → reverse iterator

**c\*5** → повторить **c+c2** → соединить

**c.index(val)** → позиция

**c.count(val)** → подсчёт вхождений

## Операции с контейнерами

Прим.: для словарей и множеств эти  
операции работают с ключами.

! изменяют первоначальный список

## Операции со списками

```
lst.append(item)
```

добавить элемент в конец

```
lst.extend(seq)
```

добавить последовательность в конец

```
lst.insert(idx, val)
```

вставить значение по индексу

```
lst.remove(val)
```

удалить первое вхождение val

```
lst.pop(idx)
```

удалить значение по индексу и вернуть его

```
lst.sort() lst.reverse()
```

сортировать/обратить список по месту

## Операции со словарями

```
d[key]=value d.clear()
```

```
d[key]→value del d[key]
```

```
d.update(d2)
```

Обновить/добавить пары

```
d.keys()
```

просмотр ключей,

```
d.values()
```

значений и пар

```
d.items()
```

```
d.pop(key)
```

## Операции с множествами

Операторы:

| → объединение (вертикальная черта)

& → пересечение

- ^ → разность/симметричная разн.

< <= > >= → отношения включения

```
s.update(s2)
```

```
s.add(key) s.remove(key)
```

```
s.discard(key)
```

Сохранение и считывание файлов с диска

## Файлы

```
f = open("fil.txt", "w", encoding="utf8")
```

файловая  
переменная  
для операций

имя файла  
на диске  
(+путь...)

режим работы  
□ 'r' read  
□ 'w' write  
□ 'a' append...

кодировка  
символов в тек-  
стовых файлах:  
utf8 ascii  
cp1251 ...

запись

```
f.write("hello")
```

! текстовый файл → чтение/запись  
только строк, преобразуйте  
требуемые типы

```
f.close()
```

! не забывайте закрывать после использования

Автоматическое закрытие: **with open(...) as f:**

очень часто: цикл по строкам (каждая до '\n') текстового файла

```
for line in f:
```

! # блок кода для обработки строки

пустая строка при конце файла

```
s = f.read(4)
```

прочитать следующую  
строку

```
s = f.readline()
```

если количество  
символов не указано,  
прочитает весь файл

## Форматирование строк

форматные директивы

значения для форматирования

```
"model {} {} {}".format(x, y, r) → str
```

Селекторы:

```
2
```

```
x
```

```
0.nom
```

```
4[key]
```

```
0[2]
```

Формат:

заполнение выравнивание знак минимирина .точность-максимирина тип

```
<>^= +- пробел 0 в начале для заполнения 0
```

целые: **b** бинарный, **c** символ, **d** десятичн. (по умолч.), **o** 8-ричн, **x** или **X** 16-ричн.

float: **e** or **E** экспоненциальная запись, **f** or **F** фиксир. точка,

**g** or **G** наиболее подходящая из **e** или **F**, **%** перевод долей в %

строки: **s** ...

! Преобразование: **s** (читаемый текст) или **r** (в виде литерала)

Примеры:

```
"{:+2.3f}".format(45.7273)
→ '+45.727'
"{1:>10s}".format(8, "toto")
→ '      toto'
"{!r}".format("I'm")
→ "'I\'m'"
→ "'I\'m'"
```