

integer, float, boolean, string	Базовые типы		Контейнерные типы
int 783 0 -192			list [1, 5, 9] ["x", 11, 8.9] ["word"] []
float 9.23 0.0 -1.7e-6			tuple (1, 5, 9) 11, "y", 7.4 ("word",) ()
bool True False	10 ⁶		неизменяемые str как упорядоченная последовательность символов
str "One\nTwo" 'I'm'	перевод строки многострочные неизменяемая, упорядоченная последовательность символов	'\n' экранирована символ табуляции	выражение с одними запятыми dict {"key": "value"} {} словарь {1: "one", 3: "three", 2: "two", 3.14: "π"} соответствие между ключами и значениями set {"key1", "key2"} {1, 9, 3, 0} set()

Имена	Преобразования
для переменных, функций, модулей, классов...	type (выражение)
a.zA.z потом a.zA.z_0..9	можно указать целое основание системы исчисления вторым параметром
□ нелатинские буквы разрешены, но избегайте их	int("15")
□ ключевые слова языка запрещены	int(15.56) отбросить дробную часть (для округления делайте round(15.56))
□ маленькие/БОЛЬШИЕ буквы отличаются	float("-11.24e8")
② a toto x7 y_max BigOne	str(78.3) и для буквального преобразования → repr("Text")
③ By and	см. форматирование строк на другой стороне для более тонкого контроля
Присвоение переменным	bool
x = 1.2+8+sin(0)	→ используйте сравнения (==, !=, <, >, ...), дающие логический результат
значение или вычислимое выражение	
имя переменной (идентификатор)	
y, z, r = 9.2, -7.6, "bad"	list("abc") → использует каждый элемент последовательности → ['a', 'b', 'c']
имена переменных	dict([(3, "three"), (1, "one")]) → {1: 'one', 3: 'three'}
контейнер с несколькими значениями (здесь кортеж)	set(["one", "two"]) → использует каждый элемент последовательности → {'one', 'two'}
x+=3 ← добавление	":".join(['toto', '12', 'pswd']) → 'toto:12:pswd'
x-=2 ← вычитание	соединяющая строка → последовательность строк
x=None «неопределенная» константа	"words with spaces".split() → ['words', 'with', 'spaces']
	"1,4,8,2".split(",") → ['1', '4', '8', '2']
	строка-разделитель

для списков, кортежей, строк...							Доступ к элементам последовательностей
отрицательный индекс	-6	-5	-4	-3	-2	-1	len(lst) → 6
положительный индекс	0	1	2	3	4	5	доступ к отдельным элементам через [индекс]
lst=[11, 67, "abc", 3.14, 42, 1968]	1	2	3	4	5	6	lst[1] → 67 lst[0] → 11 первый
положительный срез	0	1	2	3	4	5	lst[-2] → 42 lst[-1] → 1968 последний
отрицательный срез	-6	-5	-4	-3	-2	-1	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]
							срез без указания границ → с начала до конца

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

Булева логика	Блоки инструкций	Условный оператор
Сравнения: < > <= >= == != ≤ ≥ = ≠	родительская инструкция: блок инструкций 1... ⋮ родительская инструкция: блок инструкций 2... ⋮ след. инструкция после блока 1	выражения в блоке выполняется только если условие истинно
a and b логическое и оба верны одновременно	отступы!	if логическое выражение: └ блок выражений
a or b логическое или верно хотя бы одно		может сопровождаться несколькими elif, elif, ..., но только одним окончательным else. Пример:
not a логическое нет		if x==42: # блок выполнится, если x==42 истинно print("real truth")
True константа «истина»		elif x>0: # иначе блок, если лог. выражение x > 0 истинно print("be positive")
False константа «ложь»		elif bFinished: # иначе блок, если лог. перем. bFinished истинна print("how, finished")
числа с плавающей точкой... приближенные значения!	математика	else: # иначе блок для всех остальных случаев print("when it's not")
Операторы: + - * / // % ** × ÷ ↑ ↑ a ^b деление без остатка остаток	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. доку)	
(1+5.3)*2→12.6 abs(-3.2) →3.2 round(3.57, 1) →3.6		

Цикл с условием

блок инструкций выполняется до тех пор, пока условие истинно

while логическое выражение:

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

условие с хотя бы одним изменяющимся значением

while i <= 100: (здесь **i**)

выражения вычисляются пока $i \leq 100$

s = s + i2**

i = i + 1 изменяет переменную цикла

print("sum:", s) вычисленный результат цикла

остерегайтесь бесконечных циклов!

Управление циклом

break немедленный выход

continue следующая итерация

Цикл перебора

блок инструкций выполняется для всех элементов контейнера или итератора

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: Посчитать число букв **e** в строке

if c == "e":

cnt = cnt + 1

print("found", cnt, "'e'")

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 большие 15, запомнить потерянные значения

Пройти одновременно по индексам и значениям:

for idx, val in enumerate(lst):

Печать / Ввод

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

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).

Операции с контейнерами

len(c) → количество элементов

min(c) max(c) sum(c) Прим.: для словарей и множеств эти операции работают с **ключами**.

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

изменяют первоначальный список

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 списками

добавить элемент в конец

вставить значение по индексу

удалить первое вхождение **val**

удалить значение по индексу и вернуть его

сортировать/обратить список по месту

Генераторы последовательностей int

часто используются в циклах **for** по умолчанию 0 не включается

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

range(5) → 0 1 2 3 4

range(3, 8) → 3 4 5 6 7

range(2, 12, 3) → 2 5 8 11

range возвращает «генератор», чтобы увидеть значения, преобразуйте его в последовательность, например:

print(list(range(4)))

Определение функций

имя функций (идентификатор) именованные параметры

def fctname(p_x, p_y, p_z):

^{"""documentation""""}

 # инструкции, вычисление результата

return res → результат вызова. если нет возврата значения, по умолчанию вернёт **None**

параметры и весь этот блок существуют только во время вызова функции («черная коробка»)

Вызов функций

r = fctname(3, i+2, 2*i)

один аргумент каждому параметру

получить результат (если нужен)

Файлы

Сохранение и считывание файлов с диска

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

файловая переменная

имя файла на диске

режим работы

кодировка

символов в текстовых файлах: utf8 ascii cp1251 ...

□ 'r' read

□ 'w' write

□ 'a' append...

см. функции в модулях **os** и **os.path**

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

Форматирование строк

форматные директивы

значения для форматирования

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

"{селектор:формат!преобразование}"

□ Селекторы:

- 2
- x
- 0.nom
- 4[key]
- 0[2]

Примеры

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

□ Формат:

заполнение выравнивание знак минус/плюс точность~максимальная ширина

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

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

float: e or E экспоненциальная запись, f or e фикср. точка, g or G наиболее подходящая из e или F, % перевод долей в %

строки: s ...

□ Преобразование: s (читаемый текст) или x (в виде литерала)