*Python

*Python[комм 1] (МФА: [ˈpлɪθ(ə)n]; в русском языке распространено название *питóн*[13]) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

История Python

* Разработка языка Python была начата в конце 1980-х годов[20] сотрудником голландского института CWI Гвидо ван Россумом. Для распределённой ОС Amoeba требовался расширяемый скриптовый язык, и Гвидо начал писать Python на досуге, позаимствовав некоторые наработки для языка ABC (Гвидо участвовал в разработке этого языка, ориентированного на обучение программированию). В феврале 1991 года Гвидо опубликовал исходный текст в группе новостей alt.sources[21]. С самого начала Python проектировался как объектно-ориентированный язык.
* Гвидо ван Россум назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона»[51], поскольку автор был поклонником этого телешоу, как и многие другие разработчики того времени, а в самом шоу прослеживалась некая параллель с миром компьютерной техники [27].
* Наличие дружелюбного, отзывчивого сообщества пользователей считается, наряду с дизайнерской интуицией Гвидо, одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP (англ. *Python Enhancement Proposal*) — предложений по развитию Python[22].
* 3 декабря 2008 года[23], после длительного тестирования, вышла первая версия Python 3000 (или Python 3.0, также используется сокращение Py3k). В Python 3000 устранены многие недостатки архитектуры с максимально возможным (но не полным) сохранением совместимости со старыми версиями Python. На сегодня поддерживаются обе ветви развития (Python 3.x и 2.x), но поддержка Python 2.7 заканчивается в 2020 году.

*Текущая версия Python

Download the latest version for Windows

Download Python 3.11.2

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

Want to help test development versions of Python? Prereleases, Docker images

Active Python Releases

For more information visit the Python Developer's Guide.

Python version	Maintenance status	First released	End of support	Release schedule
3.11	bugfix	2022-10-24	2027-10	PEP 664
3.10	bugfix	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596
3.8	security	2019-10-14	2024-10	PEP 569
3.7	security	2018-06-27	2023-06-27	PEP 537

*Возможности Python

* Работа с xml/html файлами
* Работа с http запросами
* GUI (графический интерфейс)
* Создание веб-сценариев
* Работа с FTP
* Работа с изображениями, аудио и видео файлами
* Робототехника
* Программирование математических и научных вычислений

Таким образом, python подходит для решения львиной доли повседневных задач, будь то резервное копирование, чтение электронной почты, либо же какая-нибудь игрушка. Язык программирования Python практически ничем не ограничен, поэтому также может использоваться в крупных проектах. К примеру, python интенсивно применяется IT-гигантами, такими как, например, Google и Yandex. К тому же простота и универсальность python делают его одним из лучших языков программирования.

*Синтаксис

* Конец строки является концом инструкции (точка с запятой не требуется).
* Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым, главное, чтобы в пределах одного вложенного блока отступ был одинаков. Отступ в 1 пробел, к примеру, не лучшее решение. Используйте 4 пробела (или знак табуляции, на худой конец).
* Вложенные инструкции в Python записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, обычно с отступом под строкой основной инструкции.

Основная инструкция:

Вложенный блок инструкций

*Синтаксис

*Иногда возможно записать несколько инструкций в одной строке, разделяя их точкой с запятой:

```
a = 1; b = 2; print(a, b)
```

*Допустимо записывать одну инструкцию в нескольких строках. Достаточно ее заключить в пару круглых, квадратных или фигурных скобок:

```
if (a == 1 and b == 2 and
    c == 3 and d == 4):  # Не забываем про двоеточие
      print('spam ' * 3)
```

*Тело составной инструкции может располагаться в той же строке, что и тело основной, если тело составной инструкции не содержит составных инструкций.

```
x=10
y=4
if x > y: print(x)
```

*Переменные

*Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. Ключевых слов не так много, их легко запомнить:

```
1  False      await      else       import     pass
2  None       break      except     in         raise
3  True       class      finally    is         return
4  and        continue   for        lambda     try
5  as         def        from       nonlocal   while
6  assert     del        global     not        with
7  async      elif       if         or         yield
```

*if

*Сначала записывается часть if с условным выражением, далее могут следовать одна или более необязательных частей elif, и, наконец, необязательная часть else.

```
if test1:
    state1
elif test2:
    state2
else:
    state3
```

Примеры:

```
if 1:
    print('true')
else:
    print('false')
```

```
print('Введите число:')
a = int(input())
if a < -5:
    print('Low')
elif -5 <= a <= 5:
    print('Mid')
else:
    print('High')
```

Проверка истинности

- Любое число, не равное 0, или непустой объект - истина.
- Числа, равные 0, пустые объекты и значение None - ложь
- Операции сравнения применяются к структурам данных рекурсивно
- Операции сравнения возвращают True или False
- Логические операторы and и or возвращают истинный ложный объект-операнд:

```
X and Y
X or Y
```

стина, если оба значения X и Y истинны.

- истина, если хотя бы одно из значений X или Y

```
not X
```

но.

- истина, если X ложно.

*Трехместное выражение if/else

```
if X:
    A = Y
else:
    A = Z

#заменяется на

A = Y if X else Z
```

* В данной инструкции интерпретатор выполнит выражение Y, если X истинно, в противном случае выполнится выражение Z.

*Цикл while

While - один из самых универсальных циклов в Python, поэтому довольно медленный.

Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
i = 5
while i < 15:
        print(i)
        i = i + 2
```

```
5
7
9
11
13
>>>
```

*Цикл for

*Цикл for уже чуточку сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла while. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
for i in 'hello world':
    print(i * 2, end='')
```

```
====== RESTART: D:\Users\
hheellllloo  wwoorrlldd
>>>
```

*Оператор continue

*Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла (for или while)

```
for i in 'hello world':
    if i == 'o':
        continue
    print(i * 2, end='')
```

```
======= RESTART: D:\U
hheellll  wwrrlldd
>>>
```

*Оператор break

*Оператор break досрочно прерывает цикл.

```
for i in 'hello world':
    if i == 'o':
        break
    print(i * 2, end='')
```

```
hheellll
>>>
```

*else в цикле

*Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
for i in 'hello world':
    if i == 'a':
        break
else:
    print('Буквы a в строке нет')
```

```
Буквы a в строке нет
>>>
```

Целые числа (int)

Числа в Python 3 ничем не отличаются от обычных чисел. Они поддерживают набор самых обычных математических операций:

x + y	Сложение
x - y	Вычитание
x * y	Умножение
x / y	Деление
x // y	Получение целой части от деления
x % y	Остаток от деления
-x	Смена знака числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x ** y	Возведение в степень
pow(x, y[, z])	x^y по модулю (если модуль задан)

Системы счисления

- **int**([object], [основание системы счисления]) - преобразование к целому числу в десятичной системе счисления. По умолчанию система счисления десятичная, но можно задать любое основание от 2 до 36 включительно.
- **bin**(x) - преобразование целого числа в двоичную строку.
- **hex**(x) - преобразование целого числа в шестнадцатеричную строку.
- **oct**(x) - преобразование целого числа в восьмеричную строку.

Вещественные числа (float)

* Вещественные числа поддерживают те же операции, что и целые. Однако (из-за представления чисел в компьютере) вещественные числа неточны, и это может привести к ошибкам:

```
>>> 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1
0.9999999999999999
```

* Для высокой точности используют другие объекты (например Decimal и Fraction)).
* Также вещественные числа не поддерживают длинную арифметику:

```
>>> a=3**1000
>>> a+0.1
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    a+0.1
OverflowError: int too large to convert to float
```

Дополнительные методы

- **loat.as_integer_ratio**() - пара целых чисел, чьё отношение равно этому числу.
- **float.is_integer**() - является ли значение целым числом.
- **float.hex**() - переводит float в hex (шестнадцатеричную систему счисления).
- classmethod **float.fromhex**(s) - float из шестнадцатеричной строки.
- Модуль math предоставляе ые математические функции.

```
>>> import math
>>> math.pi
3.141592653589793
>>>
```

- Модуль random реализует гене и функции случайного выбора.

```
>>> import random
>>> random.random()
0.2195087320037108
```

*Комплексные числа (complex)

*В Python встроены комплексные числа:

```
>>> x=complex(1,2)
>>> print(x)
(1+2j)
>>> y=complex(3,4)
>>> print(y)
(3+4j)
>>> z=x+y
>>> print(z)
(4+6j)
>>>
```

Также для работы с комплексными числами используется также модуль cmath.

*Строки

*Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

*Литералы строк - строки в апострофах и в кавычках

```
>>> s='spam"s'
>>> print(s)
spam"s
>>> s="spam's"
>>> print(s)
spam's
```

*Строки в апострофах и в кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в литералы строк символы кавычек или апострофов, не используя экранирование.

*Экранированные последовательности - служебные символы

*Экранированные последовательности позволяют вставить символы, которые сложно ввести с клавиатуры.

*"Сырые" строки - подавляют экранирование

Экранированная последовательность	**Назначение**
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhhh	16-битовый символ Юникода в 16-ричном представлении
\Uhhhh…	32-битовый символ Юникода в 32-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

«Сырые» строки

- Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

```
>>> S = r'C:\newt.txt'
>>> print(S)
C:\newt.txt
```

- Но, несмотря на назначение, "сырая" строка не может заканчиваться символом обратного слэша. Пути решения:

```
>>> S = r'\n\n\\'[:-1]
>>> print(S)
\n\n\
>>> S = r'\n\n' + '\\'
>>> print(S)
\n\n\
```

*Строки в тройных апострофах или кавычках

*Главное достоинство строк в тройных кавычках в том, что их можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

```
>>> c = '''это очень большая
строка, многострочный
блок текста'''
>>> c
'это очень большая\nстрока, многострочный\nблок текста'
>>> print(c)
это очень большая
строка, многострочный
блок текста
```

*Базовые операции

*Конкатенация (сложение)

*Дублирование строки

*Длина строки (функция len)

```
>>> s1='spam'
>>> s2="eggs"
>>> print(s1+s2)
spameggs
>>> print("spam"*5)
spamspamspamspamspam
>>> print("spam "*5)
spam spam spam spam spam
>>> len("spam")
4
```

*Доступ по и

```
>>> print(s)
spam
>>> s[0]
's'
>>> s[2]
'a'
>>> s[-2]
'a'
```

*Базовые операции

*Извлечение среза

Оператор извлечения среза: [X:Y]. X – начало среза, а Y – окончание;

символ с номером Y в срез не входит. По умолчанию первый индекс равен 0, а второй - длине строки.

```
>>> s="spameggs"
>>> s[3:5]
'me'
>>> s[2:-2]
'ameg'
>>> s[:6]
'spameg'
>>> s[1:]
'pameggs'
>>> s[:]
'spameggs'
```

Методы строк

- Список методов можно найти в документации
- При вызове методов необходимо помнить, что строки в Python относятся к категории неизменяемых последовательностей, то есть все функции и методы могут лишь создавать новую строку.

```
>>> s='spam'
>>> s[1]='b'
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in <module>
    s[1]='b'
TypeError: 'str' object does not support item assignment
>>> s=s[0]+'b'+s[2:]
>>> s
'sbam'
```

- Поэтому все строковые методы возвращают новую строку, которую потом следует присвоить переменной.

Метод format

- Иногда (а точнее, довольно часто) возникают ситуации, когда нужно сделать строку, подставив в неё некоторые данные, полученные в процессе выполнения программы (пользовательский ввод, данные из файлов и т. д.). Подстановку данных можно сделать с помощью форматирования строк. Форматирование можно сделать с помощью оператора %, либо с помощью метода format.

```
>>> 'Hello, {}!'.format('Vasya')
'Hello, Vasya!'
```

- Если для подстановки требуется только один аргумент, то значение - сам аргумент.

*Метод format

```
>>> '{0}, {1}, {2}'.format('a','b','c')
'a, b, c'
>>> '{}, {}, {}'.format('a','b','c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a','b','c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}, {1}, {0}'.format('abra','cad')
'abra, cad, abra'
>>> '{0}{1}{0}'.format('abra','cad')
'abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37,24N',longitude='-115,81W')
'Coordinates: 37,24N, -115,81W'
```

*Метод format. Синтаксис

```
поле замены     ::=  "{" [имя поля] ["!" преобразование] [":" спецификация] "}"
имя поля        ::=  arg_name ("." имя атрибута | "[" индекс "]")*
преобразование  ::=  "r" (внутреннее представление) | "s" (человеческое представление)
спецификация    ::=  см. ниже
```

```
>>> "Units destroyed: {players[0]}".format(players = [1, 2, 3])
'Units destroyed: 1'
>>> "Units destroyed: {players[0]!r}".format(players = ['1', '2', '3'])
"Units destroyed: '1'"
```

*Спецификация в формате

```
спецификация ::=  [[fill]align][sign][#][0][width][,][.precision][type]
заполнитель  ::=  символ кроме '{' или '}'
выравнивание ::=  "<" | ">" | "=" | "^"
знак         ::=  "+" | "-" | " "
ширина       ::=  integer
точность     ::=  integer
тип          ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" |
                  "n" | "o" | "s" | "x" | "X" | "%"
```

*Выравнивание в спецификации

Флаг	Значение
'<'	Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию).
'>'	выравнивание объекта по правому краю.
'='	Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами.
'^'	Выравнивание по центру.

*Знак в спецификации

*Опция "знак" используется только для чисел и может принимать следующие значения:

Флаг	**Значение**
'+'	Знак должен быть использован для всех чисел.
'-'	'-' для отрицательных, ничего для положительных.
'Пробел'	'-' для отрицательных, пробел для положительных.

Тип	Значение
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).
'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).
's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

*Тип в специ-кации

*Списки (массивы)

*Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться).

*Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией **list**

```
>>> list('список')
['с', 'п', 'и', 'с', 'о', 'к']
```

*Списки

* Список можно создать и при помощи литерала

```
>>> s = []  # Пустой список
>>> l = ['s', 'p', ['isok'], 2]
>>> s
[]
>>> l
['s', 'p', ['isok'], 2]
```

* Список может содержать любое количество любых объектов (в том числе и вложенные списки), или не содержать ничего

*Списки

*И еще один способ создать список - это **генераторы списков**. Генератор списков - способ построить новый список, применяя выражение к каждому элементу последовательности. Генераторы списков очень похожи на цикл for.

```
>>> c = [c * 3 for c in 'list']
>>> c
['lll', 'iii', 'sss', 'ttt']
```

*Списки

*Возможна и более сложная конструкция генератора списков:

```
>>> c = [c * 3 for c in 'list' if c != 'i']
>>> c
['lll', 'sss', 'ttt']
>>> c = [c + d for c in 'list' if c != 'i' for d in 'spam' if d != 'a']
>>> c
['ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
```

*Но в сложных случаях лучше пользоваться обычным циклом for для генерации списков.

*Методы списков

*Нужно отметить, что методы списков, в отличие от строковых методов, изменяют сам список, а потому результат выполнения не нужно записывать в эту переменную.

```
>>> l = [7, 2, 3, 5, 1]
>>> l.sort()
>>> l
[1, 2, 3, 5, 7]
>>> l = l.sort()
>>> print(l)
None
```

*Кортежи (tuple)

*Кортеж, по сути - неизменяемый список.

*Защита от дурака. То есть кортеж защищен от изменений, как намеренных (что плохо), так и случайных (что хорошо).

*Меньший размер

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a.__sizeof__()
36
>>> b.__sizeof__()
44
```

*Кортежи (tuple)

*Создаем пустой кортеж

```
>>> a = tuple() # С помощью встроенной функции tuple()
>>> a
()
>>> a = () # С помощью литерала кортежа
>>> a
()
```

*Создаем кортеж из одного элемента.

```
>>> a = ('s')
>>> a
's'
```

*Получился список. Кортеж создается так

```
>>> a = ('s',)
>>> a
('s',)
```

*Все дело - в запятой. Сами по себе скобки ничего не значат, точнее, значат то, что внутри них находится одна инструкция, которая может быть отделена пробелами, переносом строк и прочим мусором.

*Кортежи (tuple)

* Создать кортеж из итерируемого объекта можно с помощью все той же пресловутой функции tuple()

```
>>> a = tuple('hello, world!')
>>> a
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```

* В качестве операций над кортежами применимы все операции над списками, не изменяющие список (сложение, умножение на число, методы index() и count() и некоторые другие операции). Можно также по-разному менять элементы местами и так далее.

* Например, гордость программистов на python - поменять местами значения двух переменных:

```
>>> a, b = b, a
>>>
```

*Словари

* **Словари в Python** - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

* Создать словарь можно несколькими способами. Во-первых, с помощью литерала:

```
>>> d={}
>>> d
{}
>>> d = {'dict': 1, 'dictionary': 2}
>>> d
{'dict': 1, 'dictionary': 2}
```

* Во-вторых, с помощью функции **dict**:

```
>>> d = dict(short='dict', long='dictionary')
>>> d
{'short': 'dict', 'long': 'dictionary'}
>>> d = dict([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

*Словари

*В-третьих, с помощью метода fromkeys:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d
{'a': None, 'b': None}
>>> d = dict.fromkeys(['a', 'b'], 100)
>>> d
{'a': 100, 'b': 100}
```

*В-четвертых, с помощью генераторов словарей, которые очень похожи на генераторы списков.

```
>>> d = {a: a ** 2 for a in range(7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

*Методы словарей

*Попробуем добавить записей в словарь и извлечь значения ключей:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d[1]
2
>>> d[4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d['1']
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    d['1']
KeyError: '1'
```

*Присвоение по новому ключу расширяет словарь, присвоение по существующему ключу перезаписывает его, а попытка извлечения несуществующего ключа порождает исключение.

*Методы словарей можно посмотреть в документации

Множества (set и frozenset)

* Множество в python - "контейнер", содержащий **не повторяющиеся** элементы в случайном порядке.

```
>>> a = set()
>>> a
set()
>>> a = set('hello')
>>> a
{'h', 'e', 'o', 'l'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'a', 'b', 'c', 'd'}
>>> a = {i ** 2 for i in range(10)} # генератор множеств
>>> a
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
>>> a = {}  # А так нельзя!
>>> type(a)
<class 'dict'>
```

* Как видно из примера, множества имеет тот же литерал, что и словарь, но пустое множество с помощью литерала создать нельзя.

Множества (set и frozenset)

* Множества удобно использовать для удаления повторяющихся элементов:

```
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

* С множествами можно выполнять множество операций: находить объединение, пересечение...
* Список методов множеств можно найти в документации

Множества (set и frozenset)

* Единственное отличие set от frozenset заключается в том, что set - изменяемый тип данных, а frozenset - нет. Примерно похожая ситуация с списками и кортежами.

```
>>> a = set('qwerty')
>>> b = frozenset('qwerty')
>>> a == b
True
>>> type(a - b)
<class 'set'>
>>> type(a | b)
<class 'set'>
>>> a.add(1)
>>> a
{'t', 1, 'q', 'w', 'e', 'y', 'r'}
>>> b.add(1)
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    b.add(1)
AttributeError: 'frozenset' object has no attribute 'add'
```

*Индексы

*Как и во многих других языках, нумерация элементов начинается с нуля. При попытке доступа к несуществующему индексу возникает исключение IndexError.

*Взять элемент по индексу можно у разных типов: список, строка, кортеж.

*В Python также поддерживаются отрицательные индексы, при этом нумерация идёт с конца

```
>>> a = [1, 3, 8, 7]
>>> a[0]
1
>>> a[-1]
7
>>> a[-5]
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    a[-5]
IndexError: list index out of range
```

Срезы

- В Python, кроме индексов, существуют ещё и **срезы**.
- item[START:STOP:STEP] - берёт срез от номера START, до STOP (не включая его), с шагом STEP. По умолчанию START = 0, STOP = длине объекта, STEP = 1. Соответственно, какие-нибудь (а возможно, и все) параметры могут быть опущены.
- Также с помощью срезов можно не только извлекать элементы, но и добавлять и удалять элементы (разумеется, только для изменяемых последовательностей).

```
>>> a = [1, 3, 8, 7]
>>> a[:]
[1, 3, 8, 7]
>>> a[1:]
[3, 8, 7]
>>> a[:3]
[1, 3, 8]
>>> a[::2]
[1, 8]
```

```
>>> a[::-1]
[7, 8, 3, 1]
>>> a[:-2]
[1, 3]
>>> a[-2::-1]
[8, 3, 1]
>>> a[1:4:-1]
[]
```