



Строки,

Списки,

Кортежи.

Строки (**string**)

- Строка-это последовательность букв

Для обозначения строки используются одинарные или двойные кавычки.

Для длинных строк более удобна другая запись – строка, заключенная в группы из трех одинарных или двойных кавычек.

Строки в языке Python **неизменяемы**

Управляющие последовательности

- Короткие строки могут содержать управляющие последовательности, кроме обратной косой черты ('\'), символов перехода на новую строку и кавычек, в которые строка заключена.
 - **Последовательность**
 - `\newline`
 - `\\`
 - `\”`
 - `\t`
 - `\v`
 - **Представляемый символ**
 - Игнорируется (`\newline`-символ новой строки)
 - Символ косой черты
 - Двойная кавычка
 - Символ горизонтальной табуляции
 - Символ вертикальной табуляции

Операции над строками

- Строки можно объединить (склеить) с помощью оператора **+**

Пример:

```
s = 'Hello'+ 'A'
```

Результат:

```
s = 'HelloA'
```

- Строки можно размножить с помощью оператора *****

Пример:

```
s = 'Word*3'
```

Результат:

```
s = 'WordWordWord'
```

Индексы

- Первый символ строки имеет индекс 0
- Индексы могут иметь отрицательные значения для отсчета с конца

Подстрока

- Строка-последовательность символов с произвольным доступом.

Любой символ строки может быть получен по его индексу.

Подстрока может быть определена с помощью **среза** – двух индексов, разделенных двоеточием.

Пример:

s = 'Hello'

[0:2]

Результат:

'He'

Длина строки

- Встроенная функция `len ()` возвращает длину строки

Пример:

```
s = 'Monday begins on saturday'
```

```
len (s)
```

Результат:

25

Списки

- Список является упорядоченным множеством элементов, перечисленных в квадратных скобках.
- Совсем необязательно, чтобы элементы списка были одного типа

Пример

```
s = ['hello', 100, 5]
```


Индексы

- Как и для строк, для списков нумерация индексов начинается с нуля
- При использовании отрицательных индексов отсчет ведется с конца списка

Срезы

- Указав через двоеточие два индекса, вы можете получить подмножество элементов списка, называемое “срезом”. Получаемое значение является новым списком, содержащим все элементы исходного списка в том же порядке.
- Нумерация элементов начинается с нуля

Изменение отдельных элементов списка

- В отличие от строк существует возможность изменения отдельных элементов списка

Пример:

```
a=['Alla', 100, 34]
```

```
a[1]= a[1]+19
```

Результат:

```
a=['Alla', 119, 34]
```

Длина списка

- Встроенная функция `len()` также применима к спискам, как и к строкам

Добавление элементов в список

- Метод **append** добавляет один элемент в конец списка.

Пример: `a.append('new')`

Результат: `['Alla', 119, 34, 'new']`

Метод **insert** вставляет один элемент в список.

Целочисленный аргумент является индексом первого элемента, позиция которого изменится.

Пример: `a.insert(1, 'new')`

Результат: `['Alla', 119, 'new', 34, 'new']`

Метод **extend** добавляет в конец элементы другого списка.

Пример: `a.extend(['two', 'elements'])`

Результат: `['Alla', 119, 'new', 34, 'new', 'two', 'elements']`

Изменение элементов списка

$a = [3, 8, 15, 43]$

Замена нескольких элементов:

Пример: $a[0:2] = [1, 12]$

Результат: $[1, 12, 15, 43]$

Удаление элемента:

Пример: $a[0:2] = []$

Результат: $[15, 43]$

Вставка:

Пример: $a[1:1] = ['Hello', 5]$

Результат: $[3, 'Hello', 5, 8, 15, 43]$

Копия самого себя в начале:

Пример: $a[:0]=a$

Результат: $[3, 8, 15, 43, 3, 8, 15, 43]$

Удаление элементов из списка

- Метод `remove` удаляет из списка первый элемент с указанным значением.

Пример: `a.remove('new')`

Результат: `['Alla', 119, 34, 'new', 'two', 'elements']`

Метод `remove` удаляет *только* один элемент. В данном случае строка "new" присутствует в списке дважды, но `a.remove("new")` удалит только первую.

Применение операторов к спискам

- С помощью оператора **+** можно “склеивать” списки
- Оператор ***** размножает элементы списка.

Расширенная запись списков

- Одна из самых мощных особенностей языка Python — расширенная запись списков, которая позволяет легко преобразовать один список в другой, применяя к каждому элементу функцию.

Пример:

```
li = [1, 9, 8, 4]
```

```
li =[elem*2 for elem in li]
```

Результат:

```
[2, 18, 16, 8]
```

li — список, который вы преобразуете. Python пробегает по всем элементам li, временно присваивает каждый из них переменной elem, вычисляет значение выражения elem*2 и добавляет в конец списка, который вы в результате получаете

Кортежи (**tuple**)

- **Кортеж** — это неизменяемый список.
- Кортеж определяется так же, как и список, но элементы перечисляются в **круглых** скобках вместо квадратных.
- Как и в списках, элементы в кортежах имеют определенный порядок. Точно так же нумерация элементов начинается с **нуля**.
- К кортежам, как и к спискам можно применить операцию **среза**. Обратите внимание, что срез списка — новый список, а срез кортежа — новый кортеж.

Операции с кортежами

- **Нельзя** добавлять элементы в кортеж
- **Нельзя** удалять элементы из кортежа
- **Нельзя** искать элементы в кортеже с помощью `index`
- Однако, **можно** с помощью `in`
- При совершении операций с кортежем (например `+=`) создается **новый** кортеж

Пустые и одноэлементные кортежи

- **Пустой** кортеж создается с помощью пустой пары скобок
- Кортеж с **одним** элементом создается с помощью значения и следующей за ним запятой, просто значения недостаточно

Связь кортежа и списка

- Кортеж может быть преобразован в список и наоборот. Встроенная функция **tuple** воспринимает список в качестве аргумента и возвращает кортеж с теми же самыми элементами, и функция **list** воспринимает кортеж в качестве аргумента и возвращает список. В результате **tuple** “замораживает” список, а **list** его “размораживает”.



Кортеж (tuple)

Кортежи в Python - упорядоченные **неизменяемые** совокупности объектов произвольных типов, заключенные в круглые скобки

Например:

(23, 656, -20, 67, -45)	<i># кортеж целых чисел</i>
(4.15, 5.93, 6.45, 9.3, 10.0, 11.6)	<i># кортеж из дробных чисел</i>
("Katy", "Sergei", "Oleg", "Dasha")	<i># кортеж из строк</i>
("Москва", "Титова", 12, 148.4)	<i># смешанный кортеж</i>
([0, 0, 0], [0, 0, 1], (0, 1, 0), 100)	<i># кортеж, состоящий из списков, кортежей и числа</i>

Кортеж, по сути - **неизменяемый** список.



Особенности кортежа

- Кортеж защищен от изменений, как намеренных (что плохо), так и случайных (что хорошо). То есть «защита от дурака».

- Имеет меньший размер:

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

- Кортежи работают быстрее, чем списки.

- Кортежи можно использовать в качестве ключей словаря

Кортежи и списки



Основное отличие между кортежами и списками состоит в том, что кортежи не могут быть изменены. На практике это означает, что у них нет методов, которые бы позволили их изменить. У **СПИСКОВ** есть такие методы, как `append()`, `extend()`, `insert()`, `remove()`, и `pop()`.

У кортежей **ни одного** из этих методов нет!

Способы создания кортежей



1. Пустой кортеж:

```
>>> a = ()
```

или

```
>>> b = tuple()
```

2. Одноэлементный кортеж:

```
>>> a = (5, )
```

```
>>> print(a)
```

```
(5, )
```



Способы создания кортежей

3. Произвольный кортеж можно создать простым перечислением элементов:

```
>>> a = (1, 2, 3, 4, 5)
```

```
>>> print(a)
```

```
(1, 2, 3, 4, 5)
```

Или

```
>>> a = tuple((1, 2, 3, 4)) # скобки!
```

```
>>> print(a)
```

```
(1, 2, 3, 4)
```

Или

```
>>> a = ('hello, world!')
```

```
>>> a
```

```
('h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!')
```



Доступ к элементам кортежа

Осуществляется через индекс:

```
>>> a = (1, 2, 3, 4, 5)
```

```
>>> print(a[0])
```

1

```
>>> print(a[1:3])
```

(2, 3)

```
>>> a[1] = 3
```

TypeError: 'tuple' object does not support item assignment

Удаление кортежей



```
>>> a = (1, 2, 3, 4, 5)
```

```
>>> del a[0]
```

TypeError: 'tuple' object doesn't support item deletion

```
>>> del a
```

```
>>> print(a)
```

NameError: name 'a' is not defined

Кортежи. Выводы.



1. **Вы не можете** добавить элементы к кортежу. Кортежи не имеют методов `append()` или `extend()`.
2. **Вы не можете** удалять элементы из кортежа. Кортежи не имеют методов `remove()` или `pop()`.
3. **Вы можете** искать элементы в кортежи, поскольку это не изменяет кортеж.
4. **Вы также можете** использовать оператор `in`, чтобы проверить существует ли элемент в кортеже.



Кортеж \leftrightarrow Список

Функция **tuple()** принимает список и возвращает кортеж из всех его элементов:

```
a = [1, 2, 3, 4, 5]    # наш список
b = tuple(a)          # преобразование в кортеж
print(b)              # вывод кортежа (1, 2, 3, 4, 5)
```

Функция **list()** принимает кортеж и возвращает список:

```
a = (1, 2, 3, 4, 5, [1, 2]) # наш кортеж
b = list(a)                 # преобразование в список
print(b)                    # вывод списка [1, 2, 3, 4, 5, [1, 2]]
```

*Иначе, **tuple()** замораживает список,
а **list()** размораживает кортеж.*



Базовые операторы кортежей

<i>Выражение</i>	<i>Результат</i>	<i>Название</i>
<code>len((1, 2, 3))</code>	3	длина
<code>(1, 2, 3) + (4, 5, 6)</code>	<code>(1, 2, 3, 4, 5, 6)</code>	конкатенация
<code>('Hi!') * 4</code>	<code>('Hi!', 'Hi!', 'Hi!', 'Hi!')</code>	повторение
<code>3 in (1, 2, 3)</code>	True	вхождение
<pre>for x in (1, 2, 3): print(x)</pre>	1 2 3	итерация
<pre>a = ('end', 'End', 'END') b = a[2] c = a[-2] g = a[1:]</pre>	END End <code>('End', 'END')</code>	срезы

Методы кортежей



1. **cmp(tuple1, tuple2)** - сравнение элементов двух кортежей;
2. **len(tuple)** - количество элементов в кортеже;
3. **max(tuple)** - получить наибольший элемент кортежа;
4. **min(tuple)** - получить наименьший элемент кортежа;
5. **sorted(tuple)** – отсортировать кортеж;
6. **tuple.index()** – получить индекс указанного элемента кортежа;
7. **tuple.count()** - получить количество вхождений в кортеж указанного элемента