



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

Упаковка и распаковка в кортеж

- `t = 12345, 54321, 'hello'`

Пример упаковки в кортеж

- `x, y, z = t`

Распаковка кортежа требует, чтобы слева стояло столько переменных, сколько элементов в кортеже

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”.