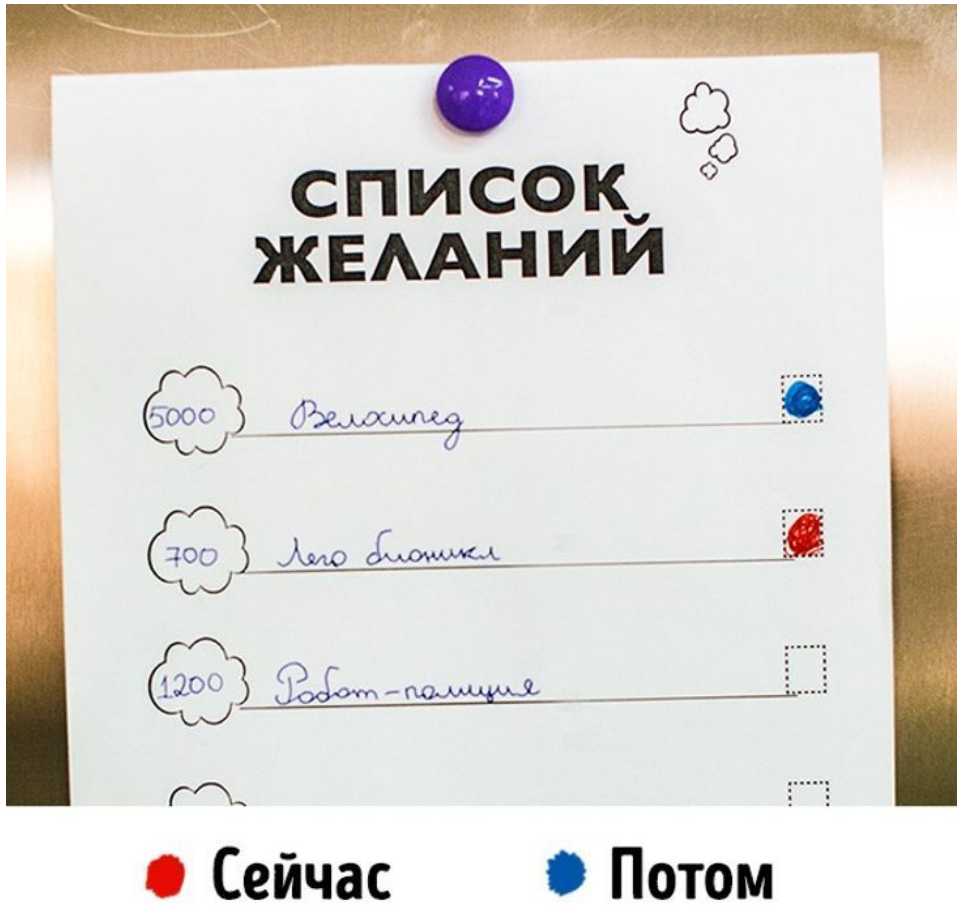
Списки

упорядоченная коллекция некоторых значений, функции списков, генераторы списков

Списки

Списки служат для того, чтобы хранить объекты в определенном порядке, особенно если порядок или содержимое могут изменяться

- **Список** – изменяемая структура данных (mutable). Можно добавлять, изменять, удалять элементы в любой удобный момент времени
- Одно и то же значение может встречаться несколько раз (в отличие от set например)



Создание списков

Списки можно создать тремя способами:

- С помощью []
- С помощью функции list()
- С помощью генератора списка, вида: [выражение if элемент in итерабельный объект] или [выражение if элемент in итерабельный объект if условие]

```
>>> list_1 = []
>>> list_1
[]
>>> list_2 = list()
>>> list_2
[]
>>> list_3 = [number for number in range(1, 4)]
>>> list_3
[1, 2, 3]
```

Преобразование других типов к list

Для преобразования других типов к list необходимо воспользоваться функцией list(объект). Объект должен быть итерируемым, например:

- Кортеж
- Строка
- Множество
- Словарь
- И т.д.

```
>>> conv_list = list(('ready', 'set', 'go'))
>>> conv_list
['ready', 'set', 'go']
>>> conv_list = list({'ready', 'set', 'go'})
>>> conv_list
['set', 'ready', 'go']
>>> conv_list = list('hello')
>>> conv_list
['h', 'e', 'l', 'l', 'o']
>>> conv_list = list({1: 'one', 2: 'two'})
>>> conv_list
[1, 2]
```

Получение элемента по его индексу

Элементы в списке нумеруются с 0 (не с 1 как мы привыкли их нумеровать в реальной жизни). Зная индекс элемента, его можно получить с помощью []:

- Можно использовать положительные или отрицательные индексы
- Если элемента по его индексу не существует, то генерируется исключение IndexError

```
>>> some_list = [n for n in range(5)]
>>> some_list
[0, 1, 2, 3, 4]
>>> some_list[0]
0
>>> some_list[-3]
2
>>> some_list[2]
2
>>> some_list[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Элементы списков

Элементом списка может быть любой объект Python, даже другая коллекция Python, например другой список

```
>>> list_of_lists = [1, 2, [3, 4, [5, 6]]]
>>> list_of_lists
[1, 2, [3, 4, [5, 6]]]
>>> list_of_lists[0]
1
>>> list_of_lists[2]
[3, 4, [5, 6]]
>>> list_of_lists[2][2]
[5, 6]
>>> list_of_lists[2][2][1]
6
```

Срезы списков

Как и при работе со строками, элементы списка можно получать с помощью срезов (получение некоторой части):

- [:] – получаем весь список
- [start:] – от start до конца списка
- [:end] – от начала до (end – 1)
- [start:end] – от start до (end – 1)
- [start:end:step] – от start до (end – 1) из элементов, чье смещение кратно step

```
>>> some_list = [n for n in range(5)]
>>> some_list
[0, 1, 2, 3, 4]
>>> some_list[2:0]
[]
>>> some_list[2:]
[2, 3, 4]
>>> some_list[:4]
[0, 1, 2, 3]
>>> some_list[2:4]
[2, 3]
>>> some_list[1:4:2]
[1, 3]
>>> some_list[2::2]
[2, 4]
>>> some_list[::-1]
[4, 3, 2, 1, 0]
```

Получаем длину списка

Для получения длины списка, необходимо воспользоваться встроенной функцией:

- len(список)

```
>>> some_list = [1, 2, 3]
>>> len(some_list)
3
>>> len([])
0
```

Удаление элемента по значению

Для удаления некоторого конкретного элемента из списка, можно воспользоваться оператором del

```
>>> some_list = [1, 2, 3]
>>> del some_list[1]
>>> some_list
[1, 3]
```

Удаление элемента по значению

Чтобы удалить элемент из списка, если мы знаем его значение, необходимо воспользоваться функцией:

- list.remove(значение)

Удаляется первый элемент, который равен переданному значению. Если такого элемента нет, то генерируется исключение ValueError

```
1  >>> some_list = [1, 2, 3]
2  >>> some_list.remove(2)
3  >>> some_list
4  [1, 3]
```

Проверяем, есть ли элемент в списке

Для того, что проверить, есть ли элемент в списке, нужно воспользоваться оператором in:

- элемент in список

```
>>> some_list = [1, 2, 3]
>>> 2 in some_list
True
>>> 5 in some_list
False
```

Добавляем элемент в конец списка

Для того, чтобы добавить элемент в конец списка, необходимо воспользоваться функцией:

- list.append(новый_элемент)

```
1  >>> some_list = [1, 2, 3]
2  >>> some_list.append('new')
3  >>> some_list
4  [1, 2, 3, 'new']
```

Добавляем элемент на позицию

Для того, чтобы вставить элемент на конкретное место в списке, нужно воспользоваться функцией:

- list.insert(индекс, элемент)

Элемент вставляется на указанное место, а элемент, который был под этим индексом, а также последующие элементы смещаются на один

```
>>> some_list = [1, 2, 3]
>>> some_list.insert(2, 'new')
>>> some_list
[1, 2, 'new', 3]
```

Объединяем список с итерируемым объектом

Для объединение списка с каким-нибудь итерируемым объектом (например другая коллекция), необходимо воспользоваться функцией:

- list.extend(iterable)

Итерируемый объект добавляется в конец списка

```
>>> some_list = [1, 2, 3]
>>> some_list.extend(('from', 'tuple'))
>>> some_list
[1, 2, 3, 'from', 'tuple']
>>> some_list = [1, 2, 3]
>>> some_list.extend(['from', 'list'])
>>> some_list
[1, 2, 3, 'from', 'list']
>>> some_list = [1, 2, 3]
>>> some_list.extend('fr str')
>>> some_list
[1, 2, 3, 'f', 'r', ' ', 's', 't', 'r']
```

Удаление по индексу или последнего элемента

Для того чтобы удалить элемент, зная его индекс, нужно воспользоваться функцией:

- list.pop(индекс)

Для того, чтобы удалить последний элемент:

- list.pop()

Если элемента не существует, то генерируется IndexError

```
>>> some_list = [1, 2, 3]
>>> some_list.pop(1)
2
>>> some_list
[1, 3]
>>> some_list.pop()
3
>>> some_list
[1]
>>> some_list.pop(55)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop index out of range
```

Узнаем индекс элемента в списке

Для того, чтобы узнать индекс элемента в списке по его значению, нужно воспользоваться функцией (можно вести поиск от индекса start до индекса end):

- list.index(значение[, start [, end]])

Генерирует исключение ValueError, если элемента не существует

```
>>> some_list = [1, 2, 3]
>>> some_list.index(2)
1
>>> some_list.index(2, 1, 3)
1
```

Узнаем количество элементов со значением

Для того, чтобы узнать, сколько элементов в списке с некоторым значение, нужно воспользоваться функцией:

- list.count(значение)

```
>>> some_list = [1, 2, 3]
>>> some_list.count(2)
1
>>> some_list.count(5)
0
>>> some_list.append(2)
>>> some_list.count(2)
2
```

Узнаем количество элементов со значением

Для того, чтобы узнать, сколько элементов в списке с некоторым значение, нужно воспользоваться функцией:

- list.count(значение)

```
>>> some_list = [1, 2, 3]
>>> some_list.count(2)
1
>>> some_list.count(5)
0
>>> some_list.append(2)
>>> some_list.count(2)
2
```

Сортируем элементы списка

Часто нужно изменять порядок элементов по их значению. Для этого в Python есть 2 функции:

- Функция списка list.sort([key][, reverse]), которая сортирует текущий список
- Общая функция sorted(список[, key][, reverse]), которая возвращает отсортированную копию списка

Функции в Python реализуют алгоритм Tim Sort, основанный на сортировке слиянием и сортировке вставкой (https://ru.wikipedia.org/wiki/Timsort)

```
>>> some_list = [1, 3, 2, 5]
>>> sorted(some_list)
[1, 2, 3, 5]
>>> some_list
[1, 3, 2, 5]
>>> sorted(some_list, reverse=True)
[5, 3, 2, 1]
>>> some_list.sort()
>>> some_list
[1, 2, 3, 5]
>>> some_list.sort(reverse=True)
>>> some_list
[5, 3, 2, 1]
```

Разворачиваем список

Для того, чтобы изменить порядок элементов на обратный, нужно воспользоваться функцией:

- list.reverse()

```
>>> some_list = [1, 3, 2, 5]
>>> some_list.reverse()
>>> some_list
[5, 2, 3, 1]
```

Очищаем список

Для того, чтобы удалить все элементы из списка, можно воспользоваться методом:

- list.clear()

```
>>> some_list = [1, 3, 2, 5]
>>> some_list.clear()
>>> some_list
[]
```

Копируем список

Список – изменяемый тип данных. Он передается по ссылке. Для того, чтобы скопировать (поверхностная копия, а не глубокая) список в переменную, а не только скопировать ссылку на него, есть следующие методы:

- Функция list.copy()
- Функция преобразования list(список)
- Срез list[:]

```
>>> list_a = [1, 2, 3]
>>> list_b = list_a
>>> list_b == list_a
True
>>> list_b is list_a
True
>>> list_b = list_a.copy()
>>> list_b == list_a
True
>>> list_b is list_a
False
>>> list_b = list(list_a)
>>> list_b == list_a
True
>>> list_b is list_a
False
>>> list_b = list_a[:]
>>> list_b == list_a
True
>>> list_b is list_a
False
```

Вопрос-ответ

? Что такое поверхностная и глубокая копия?

✔ Поверхностная копия копирует объект, но если в нем есть некоторые данные с изменяемым типом, то она старается сохранить ссылку на них в копии, что позволяет экономить место. Глубокая копия копирует без сохранения ссылок

```
>>> list_b = [1, 2]
>>> list_a = [0, 1, list_b]
>>> list_c = list_a.copy()
>>> list_a == list_c
True
>>> list_a is list_c
False
>>> import copy
>>> list_c = copy.copy(list_a)
>>> list_a == list_c
True
>>> list_a is list_c
False
>>> list_a[2] is list_c[2]
True
>>> list_c = copy.deepcopy(list_a)
>>> list_a == list_c
True
>>> list_a is list_c
False
>>> list_a[2] is list_c[2]
False
```