**Оглавление**

[**TODO** 2](#_Toc98163082)

[**Конспекты** 3](#_Toc98163083)

[**Иммутабельность переменных** 3](#_Toc98163084)

[**Списковое включение/представление списков (*list comprehension*)** 3](#_Toc98163085)

[**Контекстный менеджер** 3](#_Toc98163086)

[**Отличия кортежа от списка** 3](#_Toc98163087)

[**Функция *map*** 3](#_Toc98163088)

[**Функция *filter*** 4](#_Toc98163089)

[**Отличие *type()* от *isinstance()*** 4](#_Toc98163090)

**TODO:**

1. docker
2. poetry
3. django + debug toolbar
4. parser
5. ооп
6. **прогуглить вопросы и задачи на собесах**
7. итераторы и их отличия от генераторов

**Конспекты**

## **Иммутабельность переменных**

Тип переменной *a* **иммутабелен**, если после перезаписи значения переменной изменился ее идентификатор (*id(a)*). Изменение идентификатора указывает на то, что после перезаписи значения создался новый объект, следовательно, старый объект нельзя было изменить.

## **Списковое включение/представление списков (*list comprehension*)**

***List comprehensions*** – короткий способ генерирования списков.

**Пример**

squares = [i \* i for i in range(10) if i % 2 == 0]

Представление списков состоит из трех элементов:

1. *expression* – выражение, возвращающее значение. В приведенном выше примере выражение *i \* i* выражением.
2. *member* – является объектом или значением в списке или итерируемым объектом (*iterable*). В приведенном выше примере значением элемента является *i*.
3. *iterable* – объект, возвращающий элементы. В примере выше этим объектом является *range*.

## **Контекстный менеджер**

<https://devpractice.ru/python-lesson-21-context-manager/>

## **Отличия кортежа от списка**

Элементы списка можно редактировать, а кортежа – нет.

## **Функция *map***

**Функция** *map* применяет функцию к каждому элементу последовательности и **возвращает итератор** с результатами.

**Пример**

list\_of\_str = ['1', '2', '5', '10']

list = list(map(int, list\_of\_str))

print(list)

> [1, 2, 5, 10]

Вместо *map* можно использовать *list comprehension*, но следует учитывать, что *map* **возвращает итератор**, а *list comprehension* – **список**. Если нужно сгенерировать большое количество элементов, то использование *map* выглядит уместнее в целях экономии памяти.

## **Функция *filter***

Функция *filter* применяет функцию ко всем элементам последовательности и **возвращает итератор** с теми объектами, для которых функция вернула *True*. Например, вернуть только те строки, в которых находятся числа:

list\_of\_strings = ['one', 'two', 'list', '', 'dict', '100', '1', '50']

list(filter(str.isdigit, list\_of\_strings))

> ['100', '1', '50']

Из списка чисел оставить только нечетные:

list(filter(lambda x: x % 2 == 1, [10, 111, 102, 213, 314, 515]))

> [111, 213, 515]

## **Отличие *type()* от *isinstance()***

В отличие от *type*, функция *isinstance* **возвращает** не тип данных аргумента, а **булево значение**, говорящее о том, принадлежит объект к определенному классу или нет:

num = 4.44

print(isinstance(num, float))

> True

Также отличие от *type* состоит в том, что *isinstance* **"знает" о наследовании**. Функция воспринимает объект производного класса, как объект базового. Поэтому нельзя использовать *type* для проверок наследующихся типов

class BaseExample:

pass

class DerivedExample(BaseExample):

pass

test = DerivedExample()

print(isinstance(test, BaseExample))

> True

## **Итераторы**

## **Генераторы**