Кортежи

неизменяемая упорядоченная коллекция некоторых значений, функции кортежей

Кортежи (tuple)

Кортеж – тип данных, который напоминает список, но занимает меньше памяти и его нельзя изменить (относится к неизменяемым типам данных).

Кортеж – неизменяемый тип данных, который хранит в себе упорядоченную последовательность произвольных элементов.

Часто встречается при передаче и возвращении параметров в функциях, при объявлении констант.

Вопрос-ответ

? Зачем мне кортежи, если есть списки?

✔ Во-первых кортежи – это неизменяемый тип данных, соответственно нельзя изменить его элемент по индексу (используется в качестве констант или когда данные нужно защитить от изменения). Во-вторых кортежи занимают меньше места в памяти

```
>>> some_tuple = (1, 2, 3)
>>> some_tuple[1] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> some_list = [1, 2, 3]
>>> import sys
>>> sys.getsizeof(some_list)
120
>>> sys.getsizeof(some_tuple)
64
```

Создание кортежей

Кортежи можно создать двумя способами:

- С помощью ()
- С помощью функции tuple()

```
>>> tuple_1 = ()
>>> tuple_1
()
>>> tuple_2 = tuple()
>>> tuple_2
()
```

Вопрос-ответ

? Если ли генератор для кортежа?

✔ Нет, если в генераторе списка мы заменим квадратные скобки на круглые, то не получим генератор кортежа, как ожидалось, а получим генератор генератора

```
>>> is_tuple = (n for n in range(3))
>>> isinstance(is_tuple, tuple)
False
>>> type(is_tuple)
<class 'generator'>
>>> is_tuple
<generator object <genexpr> at 0x00000261D0C99970>
```

Преобразование других типов к tuple

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

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

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

Получение элемента по его индексу

Элементы в кортеже нумеруются с 0. 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_tuple = (1, 2, 3)
>>> some_tuple
(1, 2, 3)
>>> some_tuple[0]
1
>>> some_tuple[-2]
2
>>> some_tuple[2]
3
>>> some_tuple[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
```

Элементы кортежей

Элементом кортежа может быть любой объект Python, даже другая коллекция Python, например другой кортеж

```
>>> some_tuple = ((1, 2), [3, 4], {5, 6}, {7: 8, 9: 10})
>>> some_tuple
((1, 2), [3, 4], {5, 6}, {7: 8, 9: 10})
>>> some_tuple[1]
[3, 4]
>>> some_tuple[1][1]
4
>>> some_tuple[3][7]
8
```

Срезы кортежей

Элементы кортежа 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_tuple = (1, 2, 3, 4, 5)
>>> some_tuple[1:3]
(2, 3)
>>> some_tuple[3:]
(4, 5)
>>> some_tuple[:4]
(1, 2, 3, 4)
>>> some_tuple[1:4:2]
(2, 4)
>>> some_tuple[::-1]
(5, 4, 3, 2, 1)
```

Получаем длину кортежа

Для получения длины кортежа, необходимо воспользоваться встроенной функцией:

- len(кортеж)

```
>>> some_tuple = (1, 2, 3, 4, 5)
>>> len(some_tuple)
5
>>> len(())
0
```

Проверяем, есть ли элемент в кортеже

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

- элемент in кортеж

```
>>> some_tuple = (1, 2, 3)
>>> 2 in some_tuple
True
>>> 5 in some_tuple
False
```

Узнаем индекс элемента в кортеже

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

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

Генерирует исключение ValueError, если элемента не существует

```
>>> some_tuple = (1, 2, 3)
>>> some_tuple.index(2)
1
>>> some_tuple.index(2, 1, 3)
1
>>> some_tuple.index(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: tuple.index(x): x not in tuple
```

Узнаем количество элементов со значением

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

- tuple.count(значение)

```
>>> some_tuple = (1, 2, 3, 2)
>>> some_tuple.count(2)
2
>>> some_tuple.count(3)
1
>>> some_tuple.count(5)
0
```

Множества

неупорядоченная коллекция уникальных значений, генераторы множеств, функции множеств

Множества

Множество – изменяемая структура данных, которая содержит в себе уникальные значения

- Элементами множества могут быть только **hashable** объекты

Создание множества

Множество можно создать тремя способами:

- С помощью {перечисление_элементов}
- С помощью функции set()
- С помощью генератора множества, вида: {выражение if элемент in итерабельный объект} или {выражение if элемент in итерабельный объект if условие}

```
>>> set_1 = {}
>>> type(set_1)
<class 'dict'>
>>> set_1 = {1, 2, 3}
>>> set_1
{1, 2, 3}
>>> type(set_1)
<class 'set'>
>>> set_2 = set()
>>> set_2
set()
>>> set_3 = {n for n in range(3)}
>>> set_3
{0, 1, 2}
```

Преобразование других типов к set

Для преобразования других типов данных к set используется функция set(объект). Можно создать set из другой коллекции, потеряв все повторяющиеся значения

```
>>> set([1, 2, 3, 1, 2, 5, 1])
{1, 2, 3, 5}
>>> set((2, 5, 3, 2, 5))
{2, 3, 5}
>>> set('тебе привет')
{'и', 'б', 'е', 'п', 'р', 'в', 'т', ' '}
```

Получаем число элементов

Для того, чтобы узнать количество элементов в множестве, нужно воспользоваться функцией:

- len(множество)

```
1  >>> some_set = {'one', 'two' 'three'}
2  >>> len(some_set)
3  2
```

Проверяем, есть ли элемент в set

Для того, чтобы проверить, есть ли элемент в set, нужно воспользоваться оператором in

```
>>> some_set = {'one', 'two' 'three'}
>>> 'four' in some_set
False
>>> 'two' in some_set
False
```

Проверяем, есть ли у множеств общие элементы

Для того, чтобы проверить, есть ли у двух множеств общие элементы, нужно воспользоваться функцией:

- set.isdisjoint(множество)

True, есть нет общих элементов, и False, если есть

```
>>> some_set = {1, 2, 3}
>>> some_set.isdisjoint({4, 5})
True
>>> some_set.isdisjoint({3, 5})
False
```

Входит ли set в другой set

Для того, чтобы проверить, что **все** элементы одного множество принадлежат другому множеству, нужно воспользоваться функцией:

- set.issubset(множество)
- Или множество <= другое

```
>>> some_set = {1, 2}
>>> some_set.issubset({1, 2, 3, 4})
True
>>> some_set.issubset({2, 3, 4})
False
>>> some_set <= {1, 2, 3, 4}
True
>>> some_set <= {3, 4}
False
```

Включает ли set другой set

Для того, чтобы узнать, включает ли текущее множество **все** элементы другого множества, нужно воспользоваться функцией:

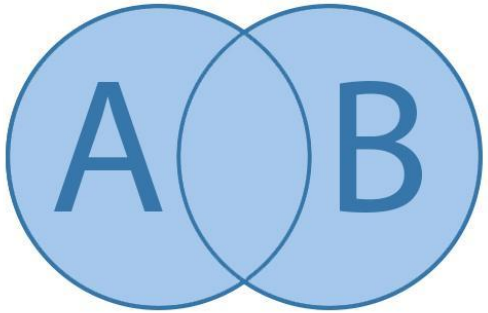
- set.issuperset(множество)
- Или множество >= другое

```
>>> some_set = {1, 2, 3, 4}
>>> some_set.issuperset({1, 3})
True
>>> some_set.issuperset({3, 5})
False
>>> some_set >= {1, 3}
True
>>> some_set >= {3, 5}
False
```

Получаем объединение множеств

Для того, чтобы получить элементы, которые есть во всех множествах, нужно воспользоваться функцией:

- set.union(множество)
- Или множество | другое

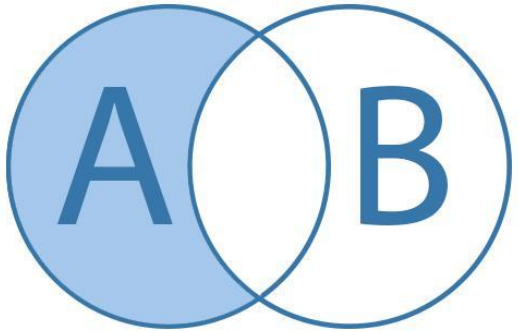



```
>>> some_set = {1, 2}
>>> some_set.union({2, 3}, {3, 4})
{1, 2, 3, 4}
>>> some_set
{1, 2}
>>> some_set | {2, 3}
{1, 2, 3}
>>> some_set
{1, 2}
```

Получаем пересечение множеств

Для того, чтобы получить элементы, которые есть в первом множестве и нет во всех других, нужно воспользоваться функцией:

- set.difference(множество)
- Или множество & другое



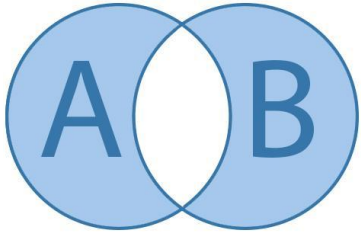

```
1  >>> some_set = {1, 2}
2  >>> some_set.difference({2, 3})
3  {1}
4  >>> some_set
5  {1, 2}
6  >>> some_set - {2, 3}
7  {1}
8  >>> some_set
9  {1, 2}
```

Получаем симметричное пересечение множеств

Для того, чтобы получить элементы, которые есть во всех множествах, но не являются общими нужно воспользоваться функцией:

- set.symmetric_difference(множество)
- Или множество ^ другое

```
>>> some_set = {1, 2, 3}
>>> some_set.symmetric_difference({2, 3, 4})
{1, 4}
>>> some_set
{1, 2, 3}
>>> some_set ^ {2, 3, 4}
{1, 4}
>>> some_set
{1, 2, 3}
```



Добавляем элемент в множество

Для того, чтобы добавить элемент в множество, нужно воспользоваться функцией:

- set.add(элемент)

```
1   >>> some_set = {1, 2, 3}
2   >>> some_set.add(4)
3   >>> some_set
4   {1, 2, 3, 4}
```

Удаление элементов из множества

Чтобы удалить элемент из множества, есть несколько функций:

- set.remove(элемент). Удаляет элемент. Если элемента нет, то бросает KeyError
- set.discard(элемент). Удаляет элемент, если он находится в множестве
- set.pop(). Удаляет первый элемент, но так как множество – неупорядоченная коллекция, то нельзя сказать, какой элемент будет удален

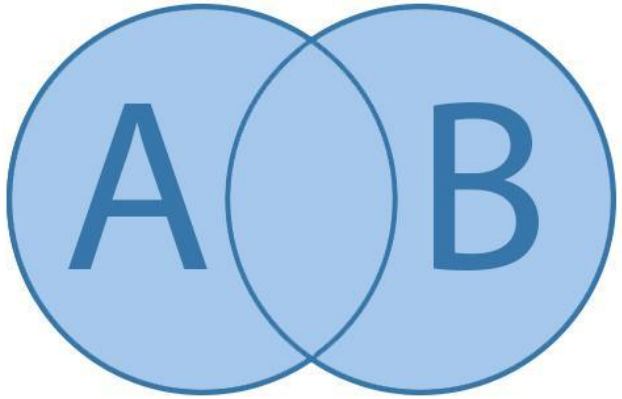
```
>>> some_set = {1, 2, 3, 4, 5, 6}
>>> some_set
{1, 2, 3, 4, 5, 6}
>>> some_set.remove(2)
>>> some_set
{1, 3, 4, 5, 6}
>>> some_set.remove(8)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 8
>>> some_set.discard(4)
>>> some_set
{1, 3, 5, 6}
>>> some_set.discard(8)
>>> some_set
{1, 3, 5, 6}
>>> some_set.pop()
1
>>> some_set.pop()
3
```

Обновление множества

Если мы хотим получить уникальные элементы из всех множеств:

- set.**update**(мн_1[, мн_2 и тд])

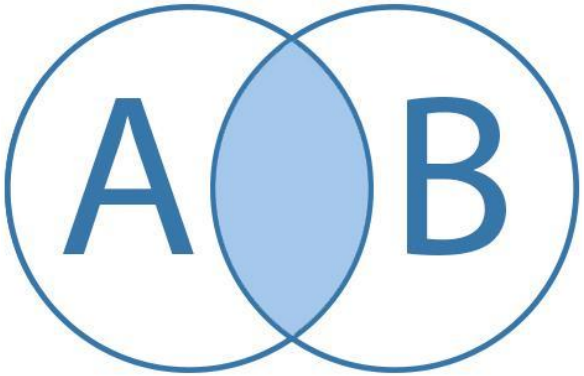
```
>>> some_set = {1, 2, 3}
>>> some_set.update({1, 4})
>>> some_set
{1, 2, 3, 4}
```



Обновление множества

Если мы хотим оставить элементы, которые есть одновременно во всех множествах, нужно воспользоваться функцией:

- set.**intersection_update**(мн_1[, мн_2 ...])

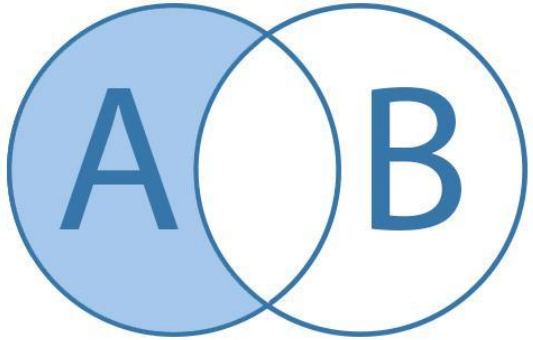



```
>>> some_set = {1, 2, 3}
>>> some_set.intersection_update({2, 3, 4})
>>> some_set
{2, 3}
```

Обновление множества

Для того, чтобы оставить только те элементы, которые есть только в первом множестве и нет в остальных, нужно воспользоваться функцией:

- set.**defference_update**(мн_1[, мн_2 и тд])



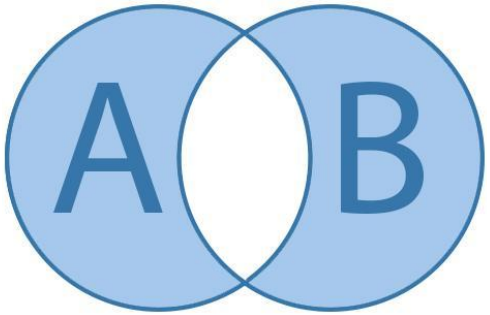

```
1  >>> some_set = {1, 2, 3}
2  >>> some_set.difference_update({2, 3, 4})
3  >>> some_set
4  {1}
```

Обновление множества

Для того, чтобы оставить элементы, которые есть во всех множествах кроме элементов, которые у них общие, нужно воспользоваться функцией:

- set.**symmetric_difference_update**(мн_1[, мн_2 и тд])

```
>>> some_set = {1, 2, 3}
>>> some_set.symmetric_difference_update({2, 3, 4})
>>> some_set
{1, 4}
```



Очистка множество

Для того, чтобы удалить все элементы из множества, нужно воспользоваться функцией:

- set.clear()

```
>>> some_set = {1, 2, 3}
>>> some_set.clear()
>>> some_set
set()
```

Копируем множество

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

- Функция set.copy()
- Функция преобразования set(множество)

```
>>> set_a = {1, 2, 3}
>>> set_b = set_a
>>> set_b is set_a
True
>>> set_b = set_a.copy()
>>> set_b == set_a
True
>>> set_b is set_a
False
>>> set_b = set(set_a)
>>> set_b == set_a
True
>>> set_b is set_a
False
```

Ваши вопросы

что необходимо прояснить в рамках данного раздела

Frozenset

неизменяемая неупорядоченная коллекция уникальных значений, функции frozenset

frozenset

frozenset – тип данных, который напоминает множество, но относится к неизменяемым типам данных.

frozenset – неизменяемый тип данных, который хранит в себе неупорядоченную последовательность произвольных элементов.

- Элементами множества могут быть только **hashable** объекты

Создание и преобразование к frozenset

frozenset можно создать с помощью функции:

- frozenset(итерируемый_объект)

```
1  frozenset({'и', 'е', 'п', 'р', 'в', 'т'})
2  >>> frozenset({1, 2, 3, 4})
3  frozenset({1, 2, 3, 4})
4  >>> frozenset((1, 2, 3, 4))
5  frozenset({1, 2, 3, 4})
6  >>> frozenset({1: 'a', 2: 'b', 3: 'c'})
7  frozenset({1, 2, 3})
```

Получаем число элементов

Для того, чтобы узнать количество элементов в frozenset, нужно воспользоваться функцией:

- len(множество)

```
1   >>> some_fs = frozenset({1, 2, 3})
2   >>> len(some_fs)
3   3
```

Проверяем, есть ли элемент в frozenset

Для того, чтобы проверить, есть ли элемент в frozenset, нужно воспользоваться оператором in

```
>>> some_fs = frozenset({1, 2, 3})
>>> 1 in some_fs
True
>>> 5 in some_fs
False
```

Проверяем, есть ли у множеств общие элементы

Для того, чтобы проверить, есть ли у двух множеств общие элементы, нужно воспользоваться функцией:

- frozenset.isdisjoint(множество)

True, есть нет общих элементов, и False, если есть

```
>>> some_fs = frozenset({1, 2, 3})
>>> some_fs.isdisjoint({4, 5})
True
>>> some_fs.isdisjoint({3, 5})
False
```

Входит ли frozenset в другое множество

Для того, чтобы проверить, что **все** элементы frozenset принадлежат другому множеству, нужно воспользоваться функцией:

- frozenset.issubset(множество)
- Или множество <= другое

```
>>> some_fs = frozenset({1, 2, 3})
>>> some_fs.issubset({1, 2, 3, 4})
True
>>> some_fs.issubset({3, 4, 5})
False
>>> some_fs <= {1, 2, 3, 4}
True
>>> some_fs <= {3, 4}
False
```

Включает ли frozenset другое множество

Для того, чтобы узнать, включает ли текущий frozenset **все** элементы другого множества, нужно воспользоваться функцией:

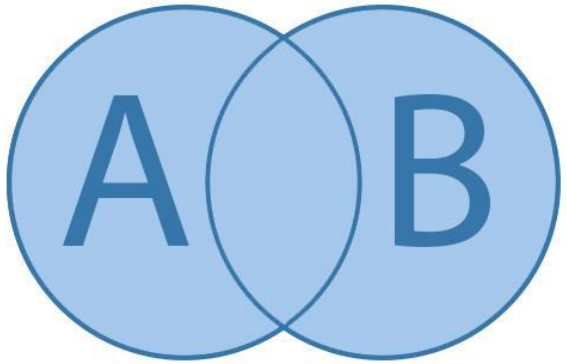
- frozenset.issuperset(множество)
- Или множество >= другое

```
>>> some_fs = frozenset({1, 2, 3})
>>> some_fs.issuperset({1, 3})
True
>>> some_fs.issuperset({3, 5})
False
>>> some_fs >= {1, 3}
True
>>> some_fs >= {3, 5}
False
```

Получаем объединение множеств

Для того, чтобы получить элементы, которые есть во всех множествах, нужно воспользоваться функцией:

- frozenset.union(множество)
- Или множество | другое

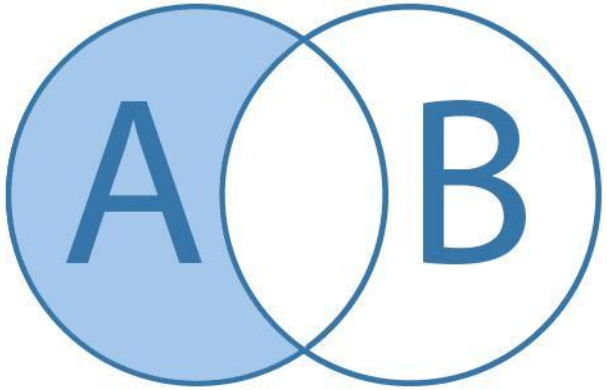



```
>>> some_fs = frozenset({1, 2})
>>> some_fs.union({2, 3}, {3, 4})
frozenset({1, 2, 3, 4})
>>> some_fs
frozenset({1, 2})
>>> some_fs | {2, 3}
frozenset({1, 2, 3})
```

Получаем пересечение множеств

Для того, чтобы получить элементы, которые есть в первом множестве и нет во всех других, нужно воспользоваться функцией:

- frozenset.difference(множество)
- Или множество & другое

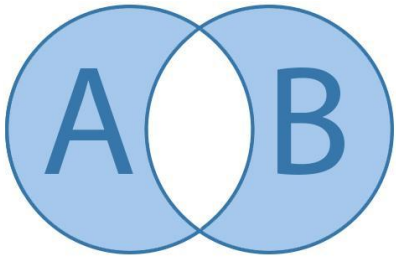



```
>>> some_fs = frozenset({1, 2, 3})
>>> some_fs.difference({2, 3})
frozenset({1})
>>> some_fs
frozenset({1, 2, 3})
>>> some_fs - {2, 3}
frozenset({1})
```

Получаем симметричное пересечение множеств

Для того, чтобы получить элементы, которые есть во всех множествах, но не являются общими нужно воспользоваться функцией:

- frozenset.symmetric_difference(множество)
- Или множество ^ другое




```
>>> some_fs = frozenset({1, 2, 3})
>>> some_fs.symmetric_difference({2, 3, 4})
frozenset({1, 4})
>>> some_fs ^ {2, 3, 4}
frozenset({1, 4})
```

Ваши вопросы

что необходимо прояснить в рамках данного раздела

Словари

неупорядоченная коллекция вида ключ-значение, генераторы словарей, функции словарей

Словарь

Словарь – изменяемая структура данных, которая содержит в себе данные в виде ключ-значение в произвольном порядке. Все ключи в словаре уникальны!

- Ключами словаря могут быть только **hashable** объекты

Создание словаря

Словарь можно создать тремя способами:

- С помощью {}
- С помощью функции dict()
- С помощью генератора словаря, вида: {выражение if элемент in итерабельный объект} или {выражение if элемент in итерабельный объект if условие}
- Функция dict.fromkeys(keys[, value])

```
>>> dict_1 = {}
>>> dict_1 = {1: 100, 2: 200}
>>> dict_2 = dict()
>>> word = 'letters'
>>> dict_3 = {let: word.count(let) for let in word}
>>> dict_3
{'l': 1, 'e': 2, 't': 2, 'r': 1, 's': 1}
>>> dict_4 = dict.fromkeys((1, 2, 3))
>>> dict_4
{1: None, 2: None, 3: None}
>>> dict_4 = dict.fromkeys((1, 2, 3), 123)
>>> dict_4
{1: 123, 2: 123, 3: 123}
```

Преобразование других типов к dict

Для преобразования других типов к dict необходимо воспользоваться функцией dict(объект). Объект должен быть итерируемым, где каждый элемент должен состоять из 2х значений, например:

- Список списков
- Список строк
- Множество кортежей

```
>>> conv_dict = dict([[1, 2], [3, 4]])
>>> conv_dict
{1: 2, 3: 4}
>>> conv_dict = dict(((1, 2), (3, 4)))
>>> conv_dict
{1: 2, 3: 4}
>>> conv_dict = dict([(1, 2), (3, 4)])
>>> conv_dict
{1: 2, 3: 4}
>>> conv_dict = dict(['ab', 'cd'])
>>> conv_dict
{'a': 'b', 'c': 'd'}
```

Создаем или изменяем элемент с помощью []

Для того, чтобы создать или изменить элемент в словаре, нужно воспользоваться следующей конструкцией:

- dict[ключ] = значение

```
>>> some_dict = {}
>>> some_dict['a'] = 'alala'
>>> some_dict['b'] = 'blabla'
>>> some_dict
{'a': 'alala', 'b': 'blabla'}
>>> some_dict['a'] = 'not alala'
>>> some_dict
{'a': 'not alala', 'b': 'blabla'}
```

Элементы словарей

Элементом словаря может быть любой объект Python, даже другая коллекция Python, например другой список, словарь.

Ключом может быть любой **hashable** элемент.

```
>>> some_dict = {1: (2, 3), 4: [5, 6], 7: {8, 9}, 8: {10: 11}}
>>> some_dict
{1: (2, 3), 4: [5, 6], 7: {8, 9}, 8: {10: 11}}
>>> some_dict = {True: 123, False: 321}
>>> some_dict
{True: 123, False: 321}
```

Получаем длину словаря

Для получения длины словаря, необходимо воспользоваться встроенной функцией:

- len(словарь)

```
1   >>> some_dict = {1: '1', 2: '2'}
2   >>> len(some_dict)
3   2
```

Получение элемента по его ключу

Элементы в словаре являются парами ключ-значение. Если мы знаем ключ, то значение можно получить 3 способами:

- Используя dict[ключ]. Если элемента не существует, то бросает KeyError
- Используя функцию dict.get(ключ[, значение по умолчанию]). Если не указано значение по умолчанию, то оно будет None

```
>>> some_dict = {1: '1', 2: '2'}
>>> some_dict[1]
'1'
>>> some_dict[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 5
>>> some_dict.get(2)
'2'
>>> some_dict.get(5)
>>> some_dict.get(5, 'Not found')
'Not found'
```

Удаляем заданный элемент

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

```
>>> some_dict = {1: '1', 2: '2'}
>>> del some_dict[2]
>>> some_dict
{1: '1'}
```

Проверяем, есть ли ключ в словаре

Для того, что проверить, есть ли ключ в словаре, нужно воспользоваться оператором in:

- ключ in словарь

```
>>> some_dict = {1: '1', 2: '2'}
>>> 5 in some_dict
False
>>> 1 in some_dict
True
```

Получаем все ключи в словаре

Для того, чтобы получить все ключи, которые есть в словаре, необходимо воспользоваться функцией:

- dict.keys()

```
1  >>> some_dict = {1: '1', 2: '2'}
2  >>> some_dict.keys()
3  dict_keys([1, 2])
```

Получаем все значения в словаре

Для того, чтобы получить все значения в словаре, нужно воспользоваться функцией:

- dict.values()

```
1  >>> some_dict = {1: '1', 2: '2'}
2  >>> some_dict.values()
3  dict_values(['1', '2'])
```

Получаем пары ключ-значение из словаря

Для того, чтобы получить все пары ключ-значение (часто используется, если мы хотим в цикле пройти по всем элементам словаря), нужно воспользоваться функцией:

- dict.items()

```
>>> some_dict = {1: '1', 2: '2'}
>>> some_dict.items()
dict_items([(1, '1'), (2, '2')])
```

Удаляем элементы в словаре

Для того, чтобы удалить элементы в словаре, можно воспользоваться следующими функциями:

- dict.pop(ключ[, default]). Если default не указан, то кидает KeyError. Возвращает значение ключа или default, если его нет
- dict.popitem() – удаляет рандомный ключ и возвращает tuple(ключ, значение). Если список пуст, то кидает KeyError

```
>>> some_dict = {1: '1', 2: '2'}
>>> some_dict.pop(23, 'Not key')
'Not key'
>>> some_dict.pop(23)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 23
>>> some_dict.pop(2)
'2'
>>> some_dict.popitem()
(1, '1')
>>> some_dict.popitem()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'popitem(): dictionary is empty'
```

Очищаем словарь

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

- dict.clear()

```
>>> some_dict = {1: '1', 2: '2'}
>>> some_dict
{1: '1', 2: '2'}
>>> some_dict.clear()
>>> some_dict
{}
```

Обновляем словарь элементами из другого

Если мы хотим обновить словарь элементами из другого словаря, нужно воспользоваться функцией:

- dict.update(словарь)

```
>>> some_dict = {1: '1', 2: '2'}
>>> some_dict.update({3: '3'})
>>> some_dict
{1: '1', 2: '2', 3: '3'}
```

Копируем словарь

Словарь – 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 dict.copy()
- Функция преобразования dict(словарь)

```
>>> dict_a = {1: [2, 3], 4: [5, 6]}
>>> dict_b = dict_a
>>> dict_b == dict_a
True
>>> dict_b is dict_a
True
>>> dict_b = dict_a.copy()
>>> dict_b == dict_a
True
>>> dict_b is dict_a
False
>>> dict_b[1] is dict_a[1]
True
>>> dict_b = dict(dict_a)
>>> dict_b == dict_a
True
>>> dict_b is dict_a
False
>>> dict_b[1] is dict_a[1]
True
```

Ваши вопросы

что необходимо прояснить в рамках данного раздела

Модуль collections

некоторые дополнительные полезные коллекции

Модуль collections

Модуль collections предоставляет коллекции альтернативные стандартным list, set, dict и tuple с расширенным функционалом

```
1  >>> import collections
2
3  >>> from collections import namedtuple
```

namedtuple

Класс namedtuple позволяет создавать тип данных, который ведет себя как кортеж, но каждому элементу присваивается имя, по которому можно получить к нему доступ

```
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(x=1, y=2)
>>> p
Point(x=1, y=2)
>>> p.x
1
>>> p[0]
1
```

deque

collections.deque([iterable, [maxlen]]) - создаёт очередь из итерируемого объекта с максимальной длиной maxlen.

Очереди очень похожи на списки, за исключением того, что добавлять и удалять элементы можно либо справа, либо слева

```
>>> from collections import deque
>>> d = deque([1, 2, 3])
>>> d
deque([1, 2, 3])
>>> d.append(4)
>>> d
deque([1, 2, 3, 4])
>>> d.appendleft(0)
>>> d
deque([0, 1, 2, 3, 4])
>>> d.pop()
4
>>> d.popleft()
0
>>> d.extend([5, 6])
>>> d
deque([1, 2, 3, 5, 6])
>>> d.extendleft([-1])
>>> d
deque([-1, 1, 2, 3, 5, 6])
>>> d.rotate(2)
>>> d
deque([5, 6, -1, 1, 2, 3])
>>> d.reverse()
>>> d
deque([3, 2, 1, -1, 6, 5])
>>> d.insert(2, 10)
>>> d
deque([3, 2, 10, 1, -1, 6, 5])
>>> d.remove(10)
>>> d
deque([3, 2, 1, -1, 6, 5])
```

deque

Методы, определённые в deque:

- append(x) - добавляет x в конец
- appendleft(x) - добавляет x в начало
- clear() - очищает очередь
- count(x) - количество элементов, равных x
- extend(iterable) - добавляет в конец все элементы iterable
- extendleft(iterable) - добавляет в начало все элементы iterable (начиная с последнего элемента iterable)
- pop() - удаляет и возвращает последний элемент очереди
- popleft() - удаляет и возвращает первый элемент очереди
- remove(value) - удаляет первое вхождение value
- reverse() - разворачивает очередь
- rotate(n) - последовательно переносит n элементов из начала в конец (если n отрицательно, то с конца в начало)

ChainMap

Класс ChainMap создает словареподобный объект, который предоставляет один интерфейс к множеству словарей

```
>>> from collections import ChainMap
>>> dict1 = {"a": 1, "b": 2}
>>> dict2 = {"c": 3, "d": 4}
>>> c_map = ChainMap(dict1, dict2)
>>> c_map["a"]
1
>>> c_map["c"]
3
>>>
```

Counter

Класс Counter позволяет создавать словареподобный объект для подсчета hashable объектов. Ключи – объекты, значения - счетчик

```
>>> from collections import Counter
>>> c = Counter()
>>> c = Counter("Python is awesome")
>>> sorted(c.elements())
[' ', ' ', 'P', 'a', 'e', 'e', 'h', 'i', 'm', 'n',
'o', 'o', 's', 's', 't', 'w', 'y']
>>> c["o"]
2
>>> c["P"]
1
>>> c.most_common()
[('o', 2), (' ', 2), ('s', 2), ('e', 2), ('P', 1), (
'y', 1), ('t', 1), ('h', 1), ('n', 1), ('i', 1), (
'a', 1), ('w', 1), ('m', 1)]
>>> c.most_common(3)
[('o', 2), (' ', 2), ('s', 2)]
>>> c.subtract("It's true")
>>> c
Counter({'o': 2, 'P': 1, 'y': 1, 'h': 1, 'n': 1, ' '
: 1, 'i': 1, 's': 1, 'a': 1, 'w': 1, 'e': 1, 'm': 1
, 't': -1, 'I': -1, "'": -1, 'r': -1, 'u': -1})
>>>
```

OrderedDict

Класс OrderedDict подобен стандартному dict за исключением того, что он запоминает порядок элементов

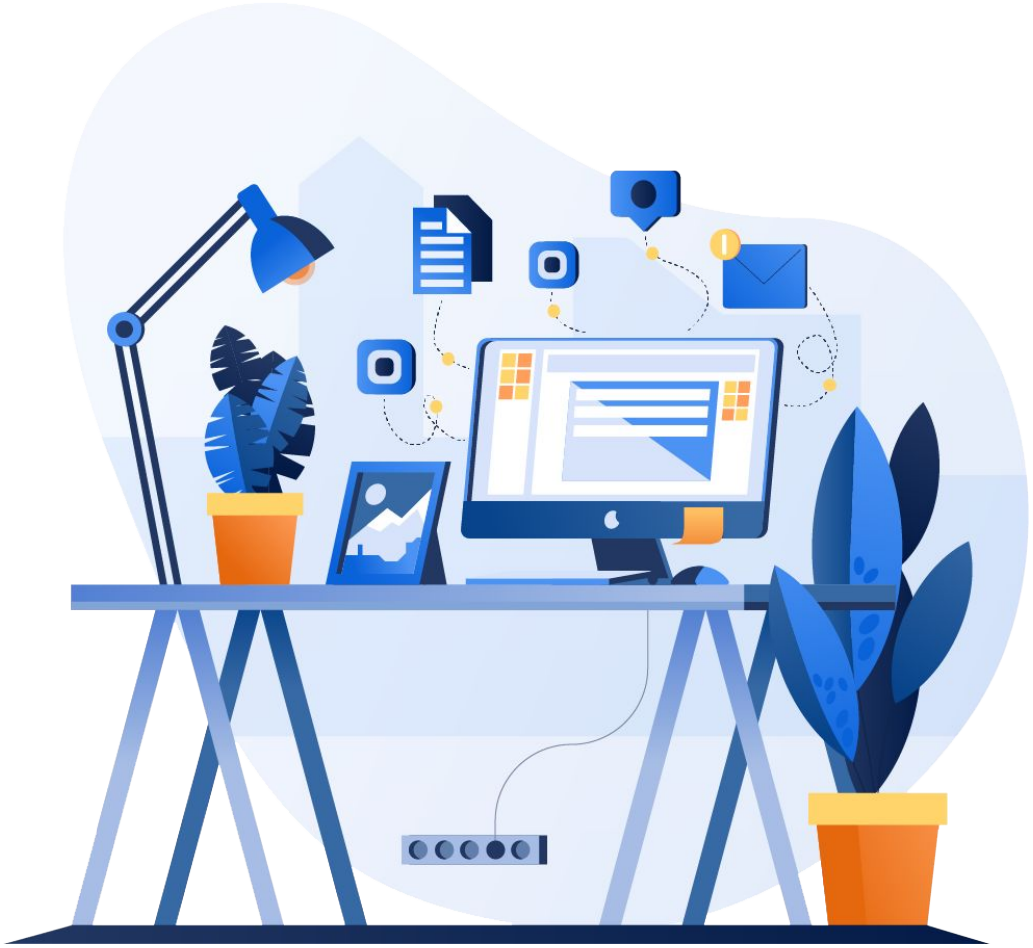
```
>>> from collections import OrderedDict
>>> od = OrderedDict(a=1, b=2, c=3)
>>> od
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
>>> od.popitem()
('c', 3)
>>> od
OrderedDict([('a', 1), ('b', 2)])
>>> od.move_to_end("a")
>>> od
OrderedDict([('b', 2), ('a', 1)])
>>>
```

Ваши вопросы

что необходимо прояснить в рамках данного раздела

Полезные ссылки

- Python Big O Time Complexity: https://wiki.python.org/moin/TimeComplexity
- Функция sorted: https://pythonz.net/references/named/sorted/
- Стандартные типы данных: https://docs.python.org/3/library/stdtypes.html
- Модуль collections: https://docs.python.org/3/library/collections.html



Спасибо за внимание!

вопросы во вне учебное время можно задать по контактам с 9:00 до 21:00:

t.me/ediboba

+375(29)339-25-87 (МТС)

rineisky@gmail.com