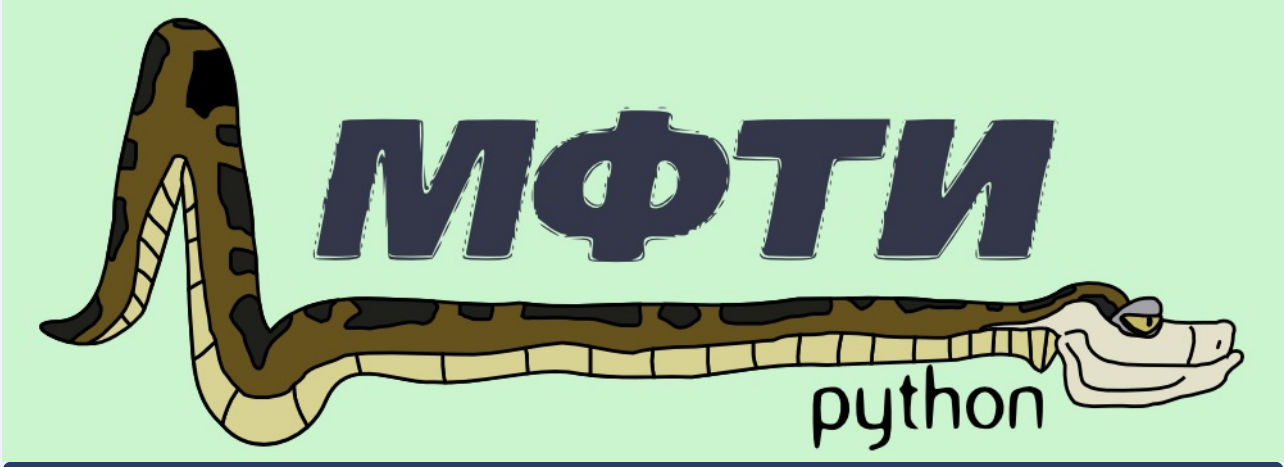




# Множества и словари в Python

Содержание



## Множества Python

Множество (`set`) - встроенная структура данных языка Python, имеющая следующие свойства:

- **множество -- это коллекция**
  множество содержит элементы

- **множество неупорядоченно**
  Множество не записывает (не хранит) *позиции* или *порядок добавления* его элементов. Таким образом, множество не имеет свойств последовательности (например, массива): у элементов множества нет индексов, невозможно взять срез множества...

- **элементы множества уникальны**
  Множество не может содержать два одинаковых элемента.

- **элементы множества - хешируемые объекты (`hashable objects`)**

  В Python множество `set` реализовано с использованием хеш-таблицы. Это приводит к тому, что элементы множества должны быть неизменяемыми объектами. Например, элементом множества может быть строка, число, кортеж `tuple`, но не может быть список `list`, другое множество `set`...

Эти свойства множеств часто используются, чтобы проверять вхождение элементов, удаление дубликатов из последовательностей, а также для математических операций пересечения, объединения, разности...

## Создание и изменение множества

Запустите в терминале Python в интерпретируемом режиме и проработайте примеры ниже.

Пустое множество создаётся с помощью функции `set`

```
>>> A = set()
>>> type(A)
<class 'set'>
>>> len(A)
0
>>> A
set()
```

Обратите внимание, что размер множества множества можно получить с помощью функции `len`.

Добавим несколько элементов

```
>>> A.add(1)
>>> A
{1}
>>> A.add(2)
>>> A
{1, 2}
>>> A.add(2)
>>> A
{1, 2}
```

Заметьте, что повторное добавление не имеет никакого эффекта на множество.

Также, из вывода видно, что литералом множества являются фигурные скобки `{}`, в которых через запятую указаны элементы. Так, ещё один способ создать **непустое** множество - воспользоваться литералом

```
>>> B = {1, 2}
>>> B
{1, 2}
```

При попытке добавления изменяемого объекта возникнет ошибка

```
>>> B.add([3,4,5])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Здесь произошла попытка добавить массив в множество `B`.

У операции добавления `set.add` существует обратная - операция удаления `set.remove`

```
>>> B
{1, 2}
>>> B.remove(1)
>>> B
{2}
>>> B.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

При попытке удаления элемента, не входящего в множество, возникает ошибка `KeyError`.

Чтобы проверить вхождение элемента в множество использовать оператор `in`

```
>>> B = {1,2}
>>> B
{1, 2}
>>> 3 in B
False
```

Кроме того, существует метод `set.discard`, который удаляет элемент из множества, только в том случае, если этот элемент присутствовал в нём.

# Математические операции

Множества Python поддерживают привычные математические операции

# Проверки

Одинаковые множества

```
>>> A = {1, 2, 3}
>>> B = {1, 2, 3}
>>> A == B
True
>>> B.add(4)
>>> A
{1, 2, 3}
>>> B
{1, 2, 3, 4}
>>> A == B
False
```

Проверка на нестрогое подмножество `set.issubset`

```
>>> A
{1, 2, 3}
>>> B
{1, 2, 3, 4}
>>> A.issubset(B)
True
>>> B.issubset(A)
False
>>> A.issubset(A)
True
```

Проверка на нестрогое надмножество `set.issuperset`

```
>>> A
{1, 2, 3}
>>> B
{1, 2, 3, 4}
>>> A.issuperset(B)
False
>>> B.issuperset(A)
True
>>> B.issuperset(B)
True
```

Операции получения новых множеств

```
>>> A = {1, 2, 4}
>>> B = {1, 2, 3}
>>> A.union(B)  # union - объединение множеств
{1, 2, 3, 4}
>>> A.intersection(B)  # intersection - пересечение
{1, 2}
>>> A.difference(B)  # difference - разность множеств
{4}
>>> B.difference(A)
{3}
>>> A.symmetric_difference(B)  # symmetric_difference - симметрическая разность
{3, 4}
>>> B.symmetric_difference(A)
{3, 4}
```

# Сводная таблица по множествам (`cheatsheet`)

Обозначения

- `elem` - Python-объект
- `A` - множество `set`
- `B, C,..`

    1. В случае использования в *методах* `A.method_name(B, C,..)`: `B, C,..` являются любыми итерируемыми объектами. Методы допускают такие аргументы, например, `{-1}.union(range(2)) == {-1, 0, 1}` вернёт `True`.

    2. В случае использования с *операторами*, например, `A > B` или `A & B & C & ...` : `B, C,..` являются множествами. Дело в том, что эти операторы *определены* для операндов типа `set` (и также `frozenset`, о которых речь позже).

| Операция | Синтаксис | Тип результата |
|---|---|---|
| Вхождение элемента | `elem in A` | `bool` |
| Равенство | `A == B` | `bool` |
| Является нестрогим подмножеством | `A.issubset(B)` или `A <= B` | `bool` |
| Является строгим подмножеством | `A < B` | `bool` |
| Является нестрогим надмножеством | `A.issuperset(B)` или `A >= B` | `bool` |
| Являляется строгим надмножеством | `A > B` | `bool` |
| Объединение множеств | `A.union(B, C,..)` | `set` |
| | `A \| B \| C \| ...` | `set` |
| Пересечение множеств | `A.intersection(B, C,..)` | `set` |
| | `A & B & C & ...` | `set` |
| Разность множеств | `A.difference(B, C,..)` | `set` |
| | `A - B - C - ...` | `set` |
| Симметрическая разность множеств | `A.symmetric_difference(B, C,..)` | `set` |
| | `A ^ B ^ C ^ ...` | `set` |

Кроме того, у операций, порождающих новые множества, существует `inplace` варианты. Для методов это те же названия, только с префиксом `_update`, а для соответствующих операторов добавляется знак равенства `=`. Ниже показан вариант для операции разности множеств

```
>>> A = {1, 2, 3, 4}
>>> B = {2, 4}
>>> A.difference_update(B)
>>> A
{1, 3}
>>> A = {1, 2, 3, 4}
>>> B = {2, 4}
>>> A -= B
>>> A
{1, 3}
```

## Неизменяемые множества

В Python существует неизменяемая версия множества - `frozenset`. Этот тип объектов поддерживает все операции обычного множества `set`, за исключением тех, которые его меняют.

Неизменяемые множества являются хешируемыми объектами, поэтому они могут быть элементами множества `set`. Так можно реализовать, например, множество множеств, где множество `set` состоит из множеств типа `frozenset`.

Для создания `frozenset` используется функция `frozenset(iterable)`, в качестве аргумента принимающая итерирумый объект.

```
>>> FS = frozenset({1, 2, 3})
>>> FS
frozenset({1, 2, 3})
>>> A = {1, 2, 4}
>>> FS & A
frozenset({1, 2})
>>> A & FS
{1, 2}
```

В этом примере показано создание `frozenset` из обычного множества `{1, 2, 3}`. Обратите внимание на тип возвращаемого объекта для операции пересечения `&`. Возвращаемый объект имеет тип, соответствующий типу **первого** аргумента. Такое же поведение будет и с другими операциями над множествами.

## Словари Python

Словарь (dictionary) в Python -- это ассоциативный массив, реализовать который вы пробовали на прошлом занятии. Ассоциативный массив это структура данных, содержащая пары вида `ключ:значение`. Ключи в ассоциативном массиве уникальны.

В Python есть встроенный ассоциативный массив - `dict`. Его реализация основана на хеш-таблицах. Поэтому

- `ключом` может быть только хешируемый объект
- `значением` может быть любой объект

## Создание и изменение словаря

Пустой словарь можно создать двумя способами:

```
>>> d1 = dict()
>>> d2 = {}
>>> d1
{}
>>> d2
{}
>>> type(d1)
<class 'dict'>
>>> type(d2)
<class 'dict'>
```

Добавить элемент в словарь можно с помощью квадратных скобок:

```
>>> domains = {}
>>> domains['ru'] = 'Russia'
>>> domains['com'] = 'commercial'
>>> domains['org'] = 'organizations'
>>> domains
{'ru': 'Russia', 'com': 'commercial', 'org': 'organizations'}
```

Из этого примера видно, что литералом словаря являются квадратные скобки, в которых через запятую перечислены пары в формате `ключ:значение`. Например, словарь `domains` можно было создать так
`domains = {'ru': 'Russia', 'com': 'commercial', 'org': 'organizations'}` .

Доступ к элементу осуществляется по ключу:

```
>>> domains['com']
'commercial'
>>> domains['de']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'de'
```

Удалить элемент можно с помощью оператора `del`. Если ключа в словаре нет, произойдет ошибка `KeyError`

```
>>> domains
{'ru': 'Russia', 'com': 'commercial', 'org': 'organizations'}
>>> del domains['de']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'de'
>>> del domains['ru']
>>> domains
{'com': 'commercial', 'org': 'organizations'}
```

Кроме того, для добавления, получения и удаления элементов есть методы `dict.setdefault`, `dict.get`, `dict.pop`, которые задействует дополнительный аргумент на случай, если ключа в словаре нет

```
>>> d1 = {}
>>> d1.setdefault('a', 10)
10
>>> d1.setdefault('b', 20)
20
>>> d1
{'a': 10, 'b': 20}
>>> d1.setdefault('c')
>>> d1
{'a': 10, 'b': 20, 'c': None}
>>> d1.setdefault('a', 123)
10
>>> d1
{'a': 10, 'b': 20, 'c': None}
>>> d1.get('a')
10
>>> d1.get('d')  # вернул None
>>> d1.get('d', 'NoKey')
'NoKey'
>>> d1.pop('d')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'd'
>>> d1.pop('d', 255)
255
>>> d1
{'a': 10, 'b': 20, 'c': None}
>>> d1.pop('a', 255)
10
>>> d1
{'b': 20, 'c': None}
```

## Примечание о числовых ключах

Ключом может являться и число: `int` или `float`. Однако при работе со словарями в Python помните, что два ключа разные, если для них верно `k1 != k2 # True`.

Вот пример:

```
>>> d = {0: 10}
>>> d
{0: 10}
>>> d[0] = 22
>>> d
{0: 22}
>>> d[0.0] = 33
>>> d
{0: 33}
>>> 0.0 != 0
False
```

Поэтому при возможности избегайте в качестве ключей `float`-объектов.

## Использование DictView: циклы и множественные операции

Если попробовать пройтись в цикле по словарю, то это будет проход по `ключам`

```
>>> d = {'a': 10, 'c': 30, 'b': 20}
>>> for k in d:
...     print(k)
...
a
c
b
```

Зачастую необходимо пройтись в цикле по `ключам`, `значениям` или парам `ключ:значение`, содержащиеся в словаре. Для этого существуют методы `dict.keys()`, `dict.values()`, `dict.items()`. Они возвращают специальные `DictView` объекты, которые можно использовать в циклах:

```
>>> d = {'a': 10, 'c': 30, 'b': 20}
>>> for k in d.keys():
...     print(k)
...
a
c
b
>>> for v in d.values():
...     print(v)
...
10
30
20
>>> for k, v in d.items():
...     print(k, v)
...
a 10
c 30
b 20
```

Объекты `DictView`, содержащие только ключи, ведут себя подобно множествам. Кроме того, если `DictView` объекты для значений или пар содержат **неизменяемые** объекты, тогда они тоже ведут себя подобно множествам. Это означает, что привычные для множеств операции пересечения, вхождения и другие также работают с `DictView`.

```
>>> d
{'a': 10, 'c': 30, 'b': 20}
>>> dkeys = d.keys()
>>> 'abc' in dkeys
False
>>> 'c' in dkeys
True
>>> {'a', 'b', 'c'} == dkeys
True
>>> dkeys & {'b', 'c', 'd'}
{'b', 'c'}
```

## Словарь с упорядоченными ключами `OrderedDict`

Если внимательно просмотреть примеры на циклы выше, то видно, что порядок итерирования в циклах совпадает с порядком *добавления* элементов в словарь.

Однако, такое поведение у стандартных словарей `dict` гарантируется, начиная с версии 3.7 (лабораторные примеры были сделаны из-под версии 3.7.4). Узнать свою версию Python можно, например, из терминала `python3 --version` или зайдя в интерпретируемый режим (версия будет написана сверху).

Если для вашей программы важно упорядочивание элементов, но вы не знаете, какой версии интерпретатор будет исполнять ваш скрипт, то вам нужно воспользоваться упорядоченной версией словарей `OrderedDict`.

Она находится в библиотеке `collections`.

Упорядоченный словарь поддерживает все операции, что и обычный словарь.

```
>>> import collections
>>> od = collections.OrderedDict()
>>> od
OrderedDict()
>>> od['a'] = 10
>>> od['c'] = 30
>>> od['b'] = 20
>>> od
OrderedDict([('a', 10), ('c', 30), ('b', 20)])
```

# Задачи

Задачи отранжированы в порядке сложности.

## Студенты

Вам даны три списка (`list` или `set`) студентов.

1. `learners_french` - изучающие французский язык
2. `pianists` - владеющие игрой на фортепиано
3. `swimmers` - занимающиеся плаванием

Создайте программу, вычисляющую список пловцов-пианистов, не изучающих французский.

## Уникальные числа

Вам даны два списка (`list`) чисел. Подсчитайте количество уникальных в каждом по отдельности, и количество уникальных среди обоих списков.

## Уже встречался!

Создайте программу, которая считывает поток чисел с клавиатуры (ввели число, нажали `Enter`, вводят следующее...). При этом при введении нового числа программа должна сообщать, встречалось ли оно раньше.

## Голосование

У каждого первокрусника спросили: "Какой ваш любимый фильм?". Ответы записали. Необходимо вывести рейтинг фильмов с количеством голосов за каждый в порядке убывания.

## Частотный анализ слов в тексте

Напишите программу, подсчитывающую количество каждого слова в тексте. В тексте присутствует пунктуация. Слова с заглавной и строчной буквы не различать. Итоговый список слов и их количества в тексте выводить в порядке убывания по количеству.

## Генеалогическое древо

Заведите словарь `d`, в котором ключ - имя человека, а значение - список родителей. Напишите функции, которые по имени человека `name` сообщают о его родственниках.

Вот список функций для реализации

- `parents(d, name)` - возвращает список родителей
- `grandparents(d, name)` - возвращает список бабушек и дедушек
- `sibling(d, name)` - возвращает список родных братьев и сестёр
- `children(d, name)` - возвращает список детей
- `grandchildren(d, name)` - возвращает список внуков и внучек