

Лекция 4. Коллекции: словари и множества

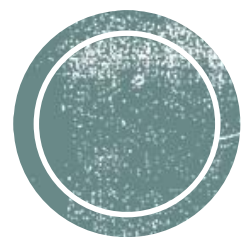
Словари, их назначение и операции над ними

Теория множеств

Множества в python, их назначение и операции над ними

Создание вложенных структур данных



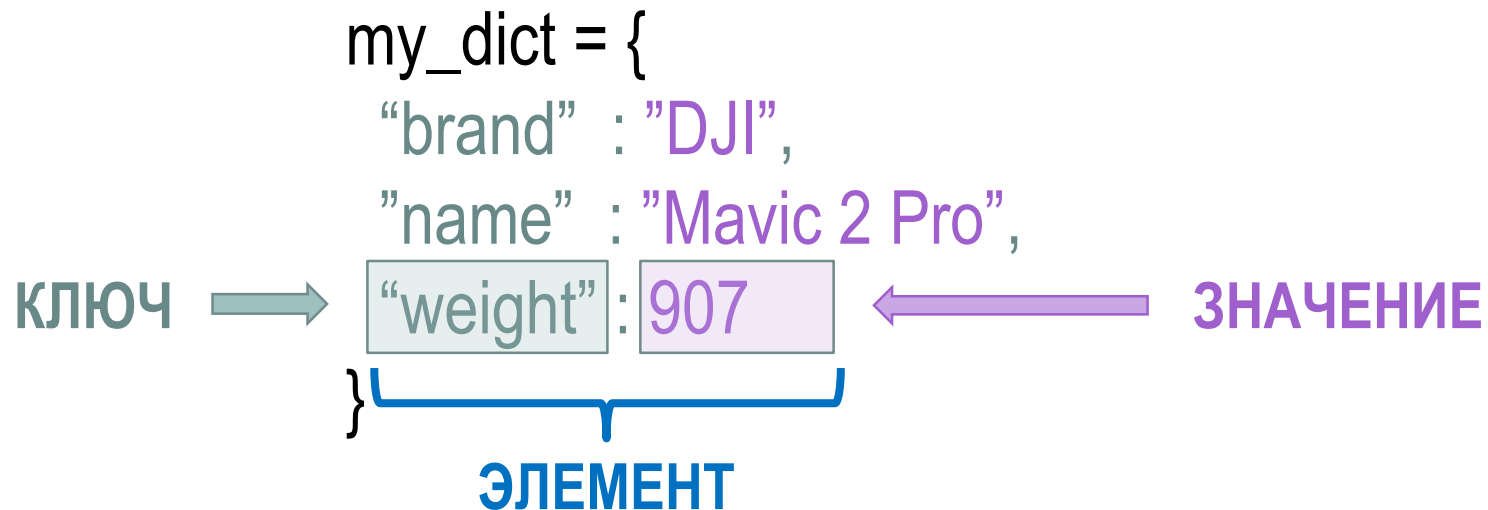


Словари

Словари и их назначение. Правила для словарей. Операции над словарями. Поиск в словаре: по ключам и по значениям

Словари

Словари (dictionaries) — неупорядоченная коллекция произвольных объектов с доступом по ключу



Правила для словарей

- ❑ Элемент словаря состоит из пары **ключ : значение**. Двоеточие между ними обязательно
- ❑ Элементы разделяются запятыми
- ❑ **Порядок** элементов в словаре **не соблюдается**. Получить значение в словаре по индексу (как в списке) – нельзя
- ❑ Доступ к значению словаря – только по ключу
- ❑ Ключи в словаре должны быть **уникальными** (не могут повторяться)
- ❑ Ключами могут быть только **неизменяемые типы** данных: числа, строки, кортежи
- ❑ Значениями могут быть любые типы данных, включая вложенные списки и словари



Создание словаря

Создание пустого словаря

```
my_dict = dict() → {}
```

```
my_dict = {} → {}
```

Создание непустого словаря

вручную:

```
my_dict = {"brand": "DJI", "name": "Mavic 2 Pro",  
"weight": 907}
```

из другого типа данных*:

```
my_dict = dict(("brand", "DJI"), ("name", "Mavic 2  
Pro"), ("weight", 907))
```

*вряд ли пригодится



Ключи словаря

Ключи словаря – неизменяемые типы данных. Если нужен ключ из нескольких элементов – берите кортеж

Пример: как оформить данные о человеке?

ФИО: Иванов Павел Андреевич

Паспорт: 1111 101202

Дата рождения: 01.01.1980

Адрес проживания: 11402, г. Москва, ул. Кетчерская, д.16



Ключи словаря (продолжение)

Простой (и совсем плохой) вариант:

```
my_dict = {"Иванов":  
           ["Павел", "Андреевич", "1111 101202", "01.01.1980", "11402, г.  
Москва, ул. Кетчерская, д.16"]}
```

Хороший вариант:

```
my_dict = {"last_name": "Иванов", "name": "Павел", "surname": "Андреевич",  
           "passport": "1111 101202", "date_of_birth": "01.01.1980", "address": "11402, г.  
Москва, ул. Кетчерская, д.16"}
```



Ключи словаря (продолжение)

Хороший вариант промежуточных данных для анализа:

```
my_dict = {("Иванов", "Павел", "Андреевич"):  
            {"passport": "1111 101202", "date_of_birth": "01.01.1980",  
             "address": "11402, г. Москва, ул. Кетчерская, д.16"}}
```

И еще один:

```
my_dict = {("Иванов", "Павел", "Андреевич"):  
            ["1111 101202", "01.01.1980", "11402, г. Москва, ул.  
            Кетчерская, д.16"]}
```



Получение данных из словаря

```
my_dict = {"brand": "DJI", "name": "Mavic 2 Pro", "weight": 907}
```

Получить **значение** по ключу:

```
my_dict["brand"] → "DJI"
```

Получить **все ключи**:

```
my_dict.keys() → ["brand", "name", "weight"]
```

Получить **все значения**:

```
my_dict.values() → ["DJI", "Mavic 2 Pro", 907]
```



Операции со словарями

dict.fromkeys(keys) dict.fromkeys(keys, value)	Создает словарь из списка ключей если задано value, то оно присваивается всем ключам. Если не задано, то присваивается None	<code>my_dict.fromkeys(["key1", "key2"]) →</code> <code>{"key1": None, "key2": None}</code> <code>my_dict.fromkeys(["key1", "key2"], 99) →</code> <code>{"key1": 99, "key2": 99}</code>
dict[key]	Возвращает значение по ключу если такого ключа нет, код упадет с ошибкой	<code>my_dict = {"key1": 1, "key2": 2}</code> <code>my_dict["key1"] → 1</code> <code>my_dict["key3"] → ошибка KeyError</code>
dict.get(key)	Безопасно возвращает значение по ключу если ключа нет, возвращает None	<code>my_dict = {"key1": 1, "key2": 2}</code> <code>my_dict.get("key1") → 1</code> <code>my_dict.get("key3") → None</code>



Операции со словарями (продолжение)

dict.setdefault(key, value)	Создает ключ и значение, если такого ключа еще нет в словаре в противном случае ничего не делает. Если не указать value, присвоит None	<pre>my_dict = {"key1":1,"key2":2} my_dict.setdefault("key3", 99) → {"key1":1,"key2":2,"key3":99}</pre>
dict.keys()	Возвращает все ключи словаря	<pre>my_dict = {"key1":1,"key2":2} my_dict.keys() → dict_keys(["key1", "key2"])</pre>
dict.values()	Возвращает все значения словаря	<pre>my_dict = {"key1":1,"key2":2} my_dict.values() → dict_values([1,2])</pre>
dict.items()	Возвращает пары ключ-значение	<pre>my_dict = {"key1":1,"key2":2} my_dict.items() → dict_items([("key1", 1), ("key2", 2)])</pre>



Операции со словарями (продолжение)

dict.pop(key)	Удаляет ключ и возвращает значение	<code>my_dict = {"key1":1,"key2":2}</code> <code>my_dict.pop("key1") → 1, {"key2":2}</code>
del dict[key]	Удаляет элемент из словаря по ключу	<code>my_dict = {"key1":1,"key2":2}</code> <code>del my_dict["key1"] → {"key2":2}</code>
dict.update(dict2)	Добавляет второй словарь в первый существующие ключи перезаписываются	<code>my_dict = {"key1":1,"key2":2}</code> <code>my_dict.update({"key1":99,"key3":3}) → {"key1":99,"key2":2,"key3":3}</code>
dict.copy()	Возвращает копию словаря создает точную копию словаря, никак не связанную с первым словарем	<code>my_dict = {"key1":1,"key2":2}</code> <code>my_dict2 = my_dict.copy() → my_dict2 = {"key1":1,"key2":2}</code>
dict.clear()	Очищает словарь	<code>my_dict = {"key1":1,"key2":2}</code> <code>my_dict.clear() → {}</code>





Множества

Теория множеств. Множества в python, их назначение и операции над ними

Множество

Множество — это неупорядоченная совокупность произвольных уникальных элементов. Множества позволяют очень быстро (за секунды или доли секунды) сравнивать между собой огромные наборы элементов

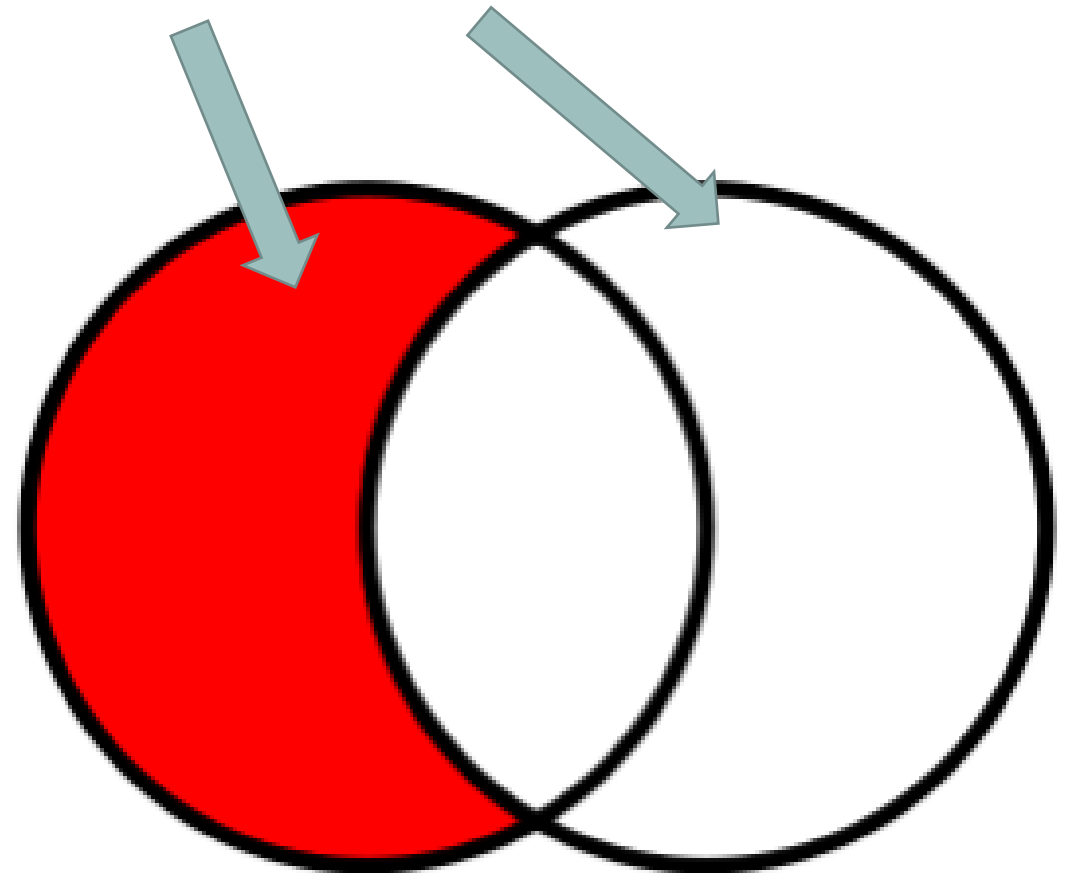
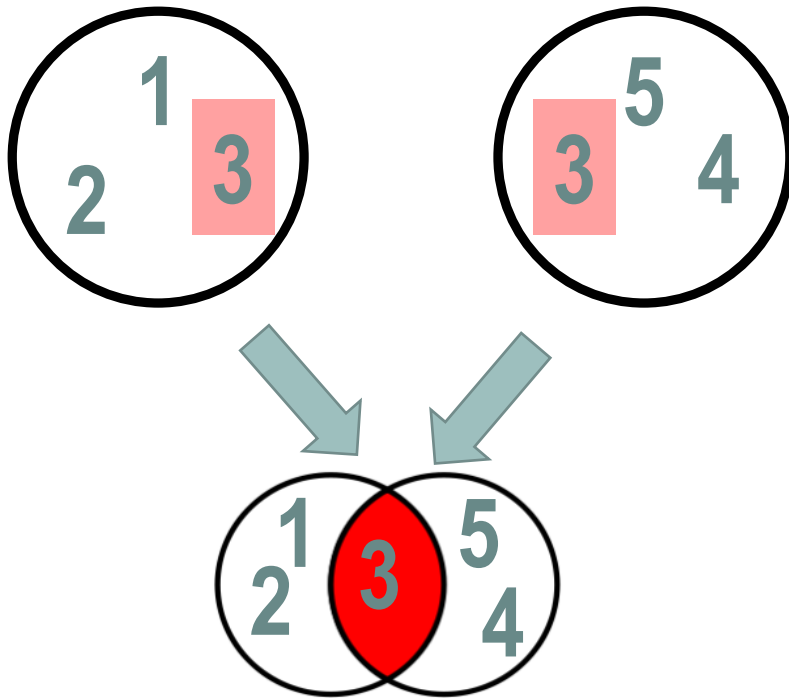
Правила для множеств:

- ❑ Элементы разделяются запятыми
- ❑ **Порядок** элементов в множестве **не соблюдается**. Получить значение по индексу (как в списке) — нельзя...
- ❑ ... но множество можно преобразовать в список и тогда - можно
- ❑ Элементы в множестве должны быть **уникальными** (не могут повторяться)
- ❑ Элементы множества реализуют **теорию множеств**

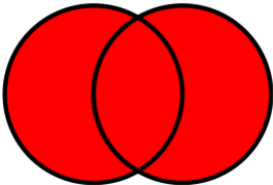
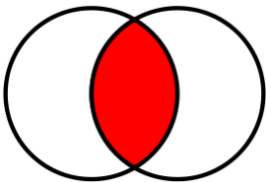
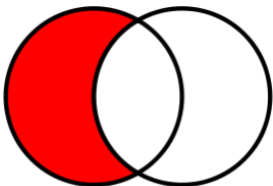
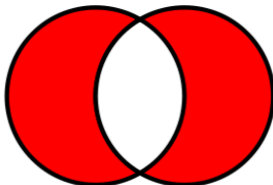


Теория множеств

КРУГИ ЭЙЛЕРА (ДИАГРАММЫ ВЕННА)



Теория множеств (продолжение)

Объединение(union) Логическое “ИЛИ” (or)		$\text{set_a} = \{1, 2, 3\}$ $\text{set_b} = \{3, 4, 5\}$ $\text{set_c} = \text{set_a} \mid \text{set_b} \rightarrow \{1, 2, 3, 4, 5\}$
Пересечение(intersection) Логическое “И” (and)		$\text{set_c} = \text{set_a} \& \text{set_b} \rightarrow \{3\}$
Разность (difference)		$\text{set_c} = \text{set_a} - \text{set_b} \rightarrow \{1, 2\}$
Симметричная разность (symmetric difference)		$\text{set_c} = \text{set_a} \wedge \text{set_b} \rightarrow \{1, 2, 4, 5\}$



Визуализация операций над множествами

Перед вами классическая иллюстрация **невозможности** получить все и сразу, выраженная через **круги Эйлера** 😊

Например, **быструю** и **качественную** работу может сделать только профессионал – но профессионал стоит **дорого**.

Специалист **высокого класса**, пообещавший вам сделать работу **со скидкой**, будет делать ее в последнюю очередь – а значит, **долго**.

Быстро и **дешево** можно сделать только в спешке и экономя на всем подряд – то есть не очень хорошо или вовсе **криво**.



Создание множества

Создание пустого множества

```
my_set = set() → {}
```

Создание непустого множества

вручную:

```
my_set = {"Mavic 2 Pro", "Phantom  
4", "Mini 2"}
```

Создание непустого множества

из списка:

```
my_set = set(["Mavic 2 Pro", "Phantom 4", "Mini 2"])  
→ {"Mavic 2 Pro", "Phantom 4", "Mini 2"}
```

из словаря:

```
my_set = set({'brand': 'DJI', 'name': 'Mavic 2 Pro',  
'weight': 907}) → {"brand", "name", "weight"}
```

из строки:

```
my_set({"Мама"}) → {"М", "а", "м"}
```



Операции с множествами

len(set)	Подсчитывает количество элементов в множестве	<code>len({1, 2}) → 2</code>
X in set	Проверяет, есть ли элемент X в множестве	<code>1 in {1, 2} → True</code> <code>1 in {"1", "2"} → False</code>
set.copy()	Создает копию множества чтобы не испортить исходное множество	<code>set1 = {1, 2}</code> <code>set2 = set1.copy() → set2 = {1, 2}, но set1 != set2</code>
set.add(X)	Добавляет элемент X в множество если этого элемента в множестве еще нет	<code>my_set = {1, 2}</code> <code>my_set.add(1) → {1, 2}</code> <code>my_set.add(3) → {1, 2, 3}</code>



Операции с множествами (продолжение)

set.remove(X)	Удаляет элемент X из множества если этого элемента в множестве нет, код «упадет» с ошибкой <code>KeyError</code>	<code>my_set = {1, 2}</code> <code>my_set.remove(1) → {2}</code> <code>my_set.remove(3) → ошибка KeyError</code>
set.discard(X)	Безопасно удаляет элемент X из множества если элемента нет, ничего не происходит	<code>my_set = {1, 2}</code> <code>my_set.discard(1) → {2}</code> <code>my_set.discard(3) → {2}</code>
set.clear()	Очищает содержимое множества	<code>my_set = {1, 2}</code> <code>my_set.clear() → {}</code>



Операции с множествами (сравнение)

set1 == set2	Сравнение двух множеств если множества полностью совпадают по составу элементов, возвращает True, иначе - False	set1 = {1, 2} set1 == {1, 2, 3} → False set1 == {1, 2} → True
set1.issubset(set2)	Является ли set1 подмножеством set2? set1 либо входит в set2, либо они идентичны (то есть выполняется ==)	set1 = {1, 2} set1.issubset({1, 2, 3}) → True set1.issubset({1, 2}) → True
set1.issuperset(set2)	Является ли set2 подмножеством set1? set1 либо включает в себя set2, либо они идентичны (то есть выполняется ==)	set1 = {1, 2} set1.issuperset({1, 2, 3}) → False set1.issuperset({1}) → True



Операции с множествами (объединение)

set1.union(set2, ...) или set1 set2 ..	Объединение нескольких множеств объединяет set1 и любое количество множеств в скобках. Также можно использовать операцию . В результате создается новое множество	set1 = {1, 2} set1.union({4, 5}, {3}) → {1,2,3,4,5} set1 {4, 5} {3} → {1,2,3,4,5}
set1.update(set2, ...) или set1 = set2 ..	Объединение множеств в set1 записывает объединение сразу в set1	set1 = {1, 2} set1.update({4, 5}, {3}) → set1 = {1,2,3,4,5} или set1 = {4, 5} {3} → set1 = {1,2,3,4,5}



Операции с множествами (пересечение)

set1.intersection(set2, ...) или set1 & set2 & ..	Пересечение нескольких множеств делает пересечение set1 и любого количества множеств в скобках. Также можно использовать операцию & . В результате создается новое множество, содержащее элементы, которые совпали во всех без исключения множествах. Если таких совпадений нет, создается пустое множество {}	set1 = {1, 2} set1.intersection({1, 5}, {3}) → {} set1.intersection({1, 5}, {1, 3}) → {1} set1 & {1, 5} & {1, 3} → {1}
set1.intersection_update(set2, ...) или set1 &= set2 & ..	Пересечение множеств в set1 записывает пересечение сразу в set1	set1 = {1, 2} set1.intersection_update({1, 5}, {1, 3}) → set1 = {1} или set1 &= {1, 5} & {1, 3} → set1 = {1}



Операции с множествами (разность)

set1.difference(set2, ..) или set1 - set2 - ...	Разность множеств вычитает из set1 все элементы, присутствующие в множествах set2, .. Создает новое множество	set1 = {1, 2, 3, 4, 5} set1.difference({1, 5}, {3}) → {2, 4} {6, 7} - set1 → {6, 7}
set1.difference_update (set2, ...) или set1 -= set2 - ...	Разность множеств в set1 записывает разность сразу в set1	set1 = {1, 2, 3, 4, 5} set1.difference_update({1, 5}, {3}) → set1 = {2, 4} или set1 -= {1, 5} - {3} → {2, 4}



Операции с множествами (симметрическая разность)

set1.symmetric_difference (set2, ..) или set1 ^ set2 ^ ...	Симметрическая разность множеств оставляет в set1 и остальных множествах set2, .. только те элементы, которые уникальны. Создает новое множество	set1 = {1, 2} set1.symmetric_difference({3, 2}) → {1, 3}
set1. symmetric_difference_update (set2, ...) или set1 ^= set2 ^ ...	Симметрическая разность множеств в set1 записывает симметрическую разность сразу в set1	set1 = {1, 2} set1.symmetric_difference_update({3, 2}) → set1 = {1, 3} или set1 ^= {3, 2} → set1 = {1, 3}



Статичное множество

Статичное множество (`frozenset`) — это такое же множество, в котором нельзя изменять элементы. Все остальные операции над ними работают как в обычных множествах.

Статичные множества — это то же самое, что кортежи для списков

