ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

tpu.ru

Углубленный курс информатики

Лекция 4

Структуры данных: Словари и множества

26 февраля 2022 г.

Игорь Михайлович Долганов,
к.т.н., доцент ОХИ ИШПР

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Содержание

1. Словари
   - Базовые операции со словарями
   - Изменение словарей
   - Распространенные методы словарей
   - Вложенные словари
   - Инициализация словарей
   - Генераторы словарей
   - Словарные представления
2. Множества
   - Создание объектов множеств
   - Распространенные методы множеств
   - Распространенные методы множеств
   - Генераторы множеств
   - Примеры использования множеств

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Словари

Словари (dict)

- *Словари* представляют собой наиболее гибкие структуры данных в Python. Их можно представить как некий аналог адресной книги, в которой можно найти адрес или контактную информацию о человеке, зная лишь его имя.
- Словари не сохраняют порядок следования своих элементов в отличие от списков.
- Элементы словарей хранятся и извлекаются по *ключам*, а не по индексам.
- Поиск элементов в словаре по ключу является крайне быстрой операцией, т.к. сами словари представляют собой максимально оптимизированную структуру данных.

Операция	**Интерпретация**
`d = {}`	Пустой словарь
`d = {"name": "John", "age": 35}`	Словарь из двух элементов
`d1 = "person": "name": "John", "age": 35`	Вложенный словарь
`d = dict(name="John", age=35)`	Создание словаря по ключевым словам
`d = dict([("name", "John"), ("age", 40)])`	Создание словаря по парам «ключ: значение»
`d = dict.fromkeys(["name", "age"])`	Создание словаря по списку ключей
`d["name"]`	Индексирование по ключу
`d1["person"]["age"]`	Индексирование по ключу дважды для доступа к вложенным объектам
`"age" in d`	Проверка наличия ключа

Словари

Операция	Интерпретация
`d.keys()`	Получить все ключи
`d.values()`	Получить все значения
`d.items()`	Получить все пары «ключ: значение» (в виде списка кортежей)
`d.copy()`	Копирование
`d.clear()`	Удаление всех элементов
`d.update(d2)`	Объединение по ключам
`d.get(key, default)`	Извлечение по ключу, если ключ отсутствует – возвращается стандартное значение или `None`
`d.pop(key. default)`	Удаление по ключу, если ключ отсутствует – возвращается стандартное значение или вызывается исключение (ошибка)
`d.setdefault(key, default)`	Установить по ключу, если ключ отсутствует – установить стандартное значение или `None`
`d.popitem()`	Удаление и возвращение любой пары «ключ: значение»
`len(d)`	Длина (количество сохраненных элементов)
`d[key] = 32`	Добавление ключей или изменение значений, связанных с ключами
`del d[key]`	Удаление элемента по ключу
`list(d.keys())`	Получить список ключей

Базовые операции со словарями

- Обычно сначала создается словарь при помощи литерального выражения, а затем в нем сохраняются элементы, доступ к которым в последствии производится по ключам:

```
>>> d = {"spam": 3, "ham": 2, "eggs": 4}  #Создание словаря
>>> d["spam"]  #Получение значения по ключу
3
>>> d
{"spam": 3, "ham": 2, "eggs": 4}  #Порядок может измениться

```

- Для индексации словарей применяется тот же синтаксис с квадратными скобками, что и для списков, однако в данном случае он означает доступ по ключу, а не по индексу.

Базовые операции со словарями

- Встроенная функция `len` работает со словарями также, как со списками и строками – возвращает количество элементов, хранящихся в словаре или длину списка его ключей.
- Операция проверки вхождения `in` в случае со словарями позволяет проверять существование ключей, а метод `keys` возвращает все ключи:

```
>>> len(d)                  #Количество элементов в словаре
3
>>> "eggs" in d             #Проверка вхождения
True
>>> list(d.keys())          #Создание списка ключей в словаре d
["spam", "ham", "eggs"]

```

- Вызов метода `keys` помещен внутрь функции `list`, по причине того, что метод `keys` возвращает итератор вместо физического списка. Вызов функции `list` получает все значения этого итератора и сохраняет их, хотя в ряде случаев использование списков не требуется.

Изменение словарей

- Наряду со списками, словари являются *изменяемыми* объектами.
- Для изменения или создания элемента необходимо просто выполнить присваивание по ключу.
- Оператор `del` также определен для словарей: он производит удаление элемента с ключом, указанным в качестве индекса.

```
>>> d = {"spam": 3, "ham": 2, "eggs": 4}
>>> d["ham"] = ["grill", "bake", "fry"]    #Замена значения на список
>>> d
{"spam": 3, "ham": ["grill", "bake", "fry"], "eggs": 4}
>>> del d["spam"]  #Удаление элемента
>>> d
{"ham": ["grill", "bake", "fry"], "eggs": 4}
>>> d["brunch"] = "Bacon"  #Добавление нового элемента
>>> d
{"ham": ["grill", "bake", "fry"], "eggs": 4, "brunch": "Bacon"}
```

- Присваивание по *новому* ключу словаря приводит к созданию нового элемента.

Распространенные методы словарей

- Методы `values` и `items` возвращают, соответственно, значения словаря и список кортежей с парами «ключ: значение».
- Совместно с методом `keys` их удобно использовать в циклах, которые нужны для прохода по элементам.
- Как и метод `keys`, методы `values` и `items` возвращают итераторы, поэтому их вызовы помещены в функцию `list`:

```
>>> d = {"spam": 4, "ham": 1, "eggs": 2}

>>> list(d.values())
[4, 1, 2]

>>> list(d.items())
[("spam", 4), ("ham", 1), ("eggs", 2)]
```

Распространенные методы словарей

- Во многих случаях невозможно предугадать содержимое словаря до запуска программы или при написании кода.
- Извлечение по несуществующему ключу интерпретируется как ошибка, однако метод `get` возвращает в таких случаях стандартное значение – `None` или переданное ему значение.

```
>>> d.get("spam")
4

>>> d.get("banana")

>>> print(d.get("banana"))
None

>>> d.get("banana", "Empty")
"Empty"

```

Распространенные методы словарей

- Метод `update` *объединяет* ключи и значения одного словаря с ключами и значениями другого словаря, переписывая значения одинаковых ключей при возникновении конфликтов:

```
>>> d = {"buritto": 2, "burger": 3}

>>> d1 = {"pizza": 2, "nachos": 5}

>>> d.update(d1)

>>> d
{"buritto": 2, "burger": 3, "pizza": 2, "nachos": 5}

```

Распространенные методы словарей

- Метод `pop` удаляет ключ из словаря и возвращает ассоциированное с ним значение.
- Во многом этот метод похож на одноименный для списков, однако вместо необязательного индекса он принимает ключ:

```
>>> d
{"buritto": 2, "burger": 3, "pizza": 2, "nachos": 5}

>>> d.pop("burger")
3

>>> d.pop("pizza")
2

>>> d
{"buritto": 2, "nachos": 5}
```

Вложенные словари

- Встроенные типы Python позволяют с легкостью представлять *структурированную* информацию, особенно, когда они вложены друг в друга.

```
>>> data = {"name": "John",
...         "jobs": ["developer", "professor"],
...         "web": "www.john.org/john",
...         "home": {"state": "Overworked", "zip": 13546}}
```

- Для доступа к компонентам вложенных объектов нужно просто составить цепочку операций индексирования:

```
>>> data["name"]
"John"
>>> data["jobs"]
["developer", "professor"]
>>> data["jobs"][1]
"professor"
>>> data["home"]["state"]
"Overworked"
```

Инициализация словарей

В качестве иллюстрации стандартного способа инициализации словаря, рассмотрим объединение его ключей и значений при помощи функции `zip` с последующей передачей результата вызову `dict`:

```
>>> list(zip(["a", "b", "c"], [1, 2, 3])) #Упаковка ключей и значений
[("a", 1), ("b", 2), ("c", 3)]

>>> d = dict(zip(["a", "b", "c"], [1, 2, 3]))

>>> d
{"a": 1, "b": 2, "c": 3}
```

Генераторы словарей

- Генераторы словарей выполняют подразумеваемый цикл, накапливая на каждом шаге результаты «ключ: значение» и используя их для заполнения нового словаря:

```
>>> d = {k: v for k, v in zip(["a", "b", "c"], [1, 2, 3])}
>>> d
{"a": 1, "b": 2, "c": 3}
```

- Допускается использовать генераторы словарей для отображения одиночного потока значений на словари, а ключи могут вычисляться при помощи выражений, как и значения:

```
>>> d = {x: x ** 2 for x in [1, 2, 3, 4, 5]}
>>> d
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> d = {c: c * 5 for c in "hello"}
>>> d
{"h": "hhhhh", "e": "eeeee", "l": "lllll", "o": "ooooo"}
>>> fruits = {fruit.upper(): fruit + "!"
              for fruit in ["banana", "orange", "apple"]}
>>> fruits
{"BANANA": "banana!", "ORANGE": "orange!", "APPLE": "apple!"}
```

Генераторы словарей

Генераторы словарей также удобно использовать при инициализации словарей из списка ключей, во многом подобно тому, как это можно реализовать при помощи метода `fromkeys`:

```
>>> d = dict.fromkeys(["a", "b", "c", "d"], -1) #Инициализация
                                                #словаря из ключей и значения -1
>>> d
{"a": -1, "b": -1, "c": -1, "d": -1}
>>> d = {key: -1 for key in ["a", "b", "c", "d"]}
>>> d
{"a": -1, "b": -1, "c": -1, "d": -1}
>>> d = dict.fromkeys("hello")
>>> d
{"h": None, "e": None, "l": None, "o": None} #Другой итерируемый
                                              #объект и значение по умолчанию
>>> d = {key: None for key in "hello"}
>>> d
{"h": None, "e": None, "l": None, "o": None}
```

Словарные представления

- Методы словарей `keys`, `values` и `items` возвращают *объекты представлений*.
- Объекты представлений являются *итерируемыми* объектами, т.е. возвращают результирующие элементы по одному за раз, а не генерируют весь список в памяти.

```
>>> d = dict(x=1, y=2, z=3)
>>> d
{"x": 1, "y": 2, "z": 3}
>>> k = d.keys()
>>> k
dict_keys(["x", "y", "z"])
>>> list(k)
["x", "y", "z"]
>>> v = d.values()
>>> v
dict_values([1, 2, 3])
>>> list(v)
[1, 2, 3]
>>> d.items()
dict_items([("x", 1), ("y", 2), ("z", 3)])
>>> list(d.items())
[("x", 1), ("y", 2), ("z", 3)]
```

Словарные представления

- Сами словари в Python имеют встроенные итераторы, которые возвращают последовательность ключей, поэтому в большинстве случаев нет необходимости вызова метода `keys`:

```
>>> d = dict.fromkeys(["a", "b", "c"], -1) #Инициализация
                                           #словаря из ключей и значения -1
>>> for key in d:
...     print(key, end=" ")
...
a b c

```

Словарные представления

- Словарные представления отражают изменения, вносимые в словарь после их создания:

```
>>> d
{"x": 1, "y": 2, "z": 3}

>>> k = d.keys()
>>> k
dict_keys(["x", "y", "z"])

>>> v = d.values()
>>> v
dict_values([1, 2, 3])

>>> del d["x"]
>>> d
{"y": 2, "z": 3}

>>> k
dict_keys(["y", "z"])

>>> v
dict_values([2, 3])

```

Замечания по использованию словарей

- Словари являются отображениями и не поддерживают операции, специфичные для последовательностей.
- Среди элементов словарей отсутствует понятие порядка, поэтому конкатенация и срезы для них неприменимы.
- Присваивание по новым индексам добавляет элементы в словарь.
- Ключи могут создаваться при написании словарного литерала или при присваивании значений новым ключам.
- Ключи не обязательно должны быть строковыми объектами. Например, в качестве ключей можно использовать целые числа, кортежи также могут быть ключами словаря.
- Изменяемые объекты: списки, множества и другие словари не могут быть ключами, но могут использоваться как значения.

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Множества

Множества (`set`)

- *Множество* представляет собой неупорядоченную коллекцию *уникальных* элементов, являющихся неизменяемыми объектами.
- Элемент встречается во множестве только один раз, в независимости от того, сколько раз он был добавлен.
- Поскольку множества представляют собой коллекции других элементов, они разделяют некоторые свойства и поведение с такими типами, как *списки* и *словари*. Например, множества являются *итерируемыми объектами*, их можно увеличивать и уменьшать по требованию, в них можно добавлять объекты других типов.
- Множества не сохраняют порядок следования элементов и не отображают ключи на значения.
- Множества являются *изменяемыми объектами* и не могут быть вложенны в другие множества.

Создание объектов множеств

- Для создания объекта множества можно вызвать функцию `set` и передать ей в качестве аргумента любой тип последовательности или другой итерируемый объект:

```
>>> x = set("hello")
>>> y = set("python")
```

- В качестве результата будет получен объект множества, содержащий все элементы из переданного объекта(порядок элементов может варьироваться):

```
>>> x
{"l", "h", "e", "o"}
```

- Для создания объектов множеств можно также использовать форму литералов множеств, применяя фигурные скобки {}. Следующие операторы эквивалентны:

```
>>> set([10, 20, 30, 40, 50])
{40, 10, 50, 20, 30}
>>> {10, 20, 30, 40, 50}
{40, 10, 50, 20, 30}
```

Создание объектов множеств

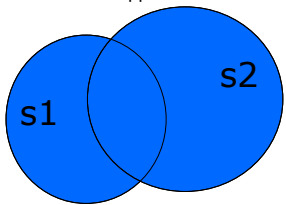
- Множества по сути похожи на *словари без значений*, т.к. элементы множеств не сохраняют порядок и в целом ведут себя похоже на ключи словаря.
- Обратите внимание на то, что пустые фигурные скобки {} – это операция, создающая пустой словарь.
- Пустые множества должны создаваться при помощи встроенной функции `set`:

```
>>> s = {10, 20, 30, 40}
>>> s - {10, 20, 30, 40}   #Пустые множества выводятся по-другому
set()
>>> type({})               #{} - это пустой словарь
<class "dict">
>>> s = set()              #Создание пустого множества
>>> s.add(100)
>>> s
{100}
```

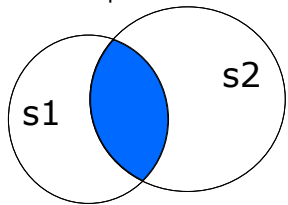
Операции над множествами

Созданные объекты множеств поддерживают распространенные математические операции, характерные для множеств:

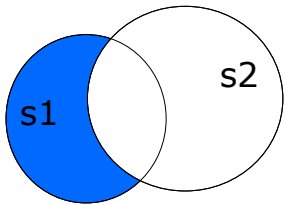
Объединение



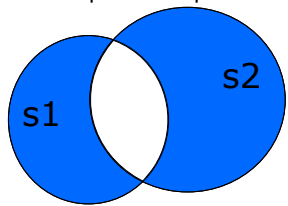
Пересечение



Разность



Симметрическая разность



Операции над множествами

Операторы *выражений*, определенные для объектов множеств:

```
>>> x = set("hello")

>>> y = set("python")

>>> x - y  #Разность
{"l", "e"}

>>> x | y  #Объединение
{"l", "h", "p", "y", "t", "n", "e", "o"}

>>> x & y  #Пересечение
{"h", "o"}

>>> x ^ y  #Симметрическая разность
{"l", "n", "p", "y", "t", "e"}

>>> x > y, x < y  #Надмножество, подмножество
(False, False)

```

Операции над множествами

- Для проверки вхождения элемента во множество используется операция `in`:

```
>>> "h" in x  #Проверка вхождения во множество
True

>>> 2 in [1, 2, 3]  #Работает и с другими последовательностями
True

```

- Множества, как и все итерируемые объекты, поддерживают функцию `len` и циклы `for`.
- Однако в силу того, что они являются неупорядоченными объектами, операции индексации и срезов не поддерживаются:

```
>>> for item in set("hello"):
...     print(item * 5)
...
lllll
hhhhh
eeeee
ooooo

```

Распространенные методы множеств

- Метод `add` добавляет один элемент.
- Метод `update` производит объединение множеств на месте.
- Метод `remove` удаляет элемент по значению.

```
>>> x
{"l", "h", "e", "o"}
>>> y
{"h", "p", "y", "t", "n", "o"}
>>> z = x.intersection(y)            #Эквивалентно x & y
>>> z
{"h", "o"}
>>> z.add("SPAM!")                   #Добавление одного элемента
>>> z
{"h", "SPAM!", "o"}
>>> z.update(set(["ham", "eggs"]))   #Объединение на месте
>>> z
{"h", "SPAM!", "o", "eggs", "ham"}
>>> z.remove("o")                    #Удаление одного элемента
>>> z
{"h", "SPAM!", "eggs", "ham"}

```

Распространенные методы множеств

В то время как показанные выше выражения с множествами требуют двух множеств в качестве операндов, их аналоги, реализованные в виде методов, во многих случаях принимают любые итерируемые объекты:

```
>>> s = set([10, 20, 30])
>>> s | set([30, 40])
{20, 40, 10, 30}

>>> s | [30, 40]  #Выражения требуют множеств
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for |: "set" and "list"

>>> s.union([30, 40])  #А методы разрешают использовать любой
                       #итерируемый объект
{40, 10, 20, 30}

>>> s.intersection((10, 30, 50))
{10, 30}

>>> s.issubset(range(100))
True
```

Операции и методы множеств

Операции, характерные для множеств и соответствующие методы:

Метод / операция	Описание
`s1.union(s2)` `s1 | s2`	Возвращает ммножество, являющееся объединением множеств s1 и s2
`s1.update(s2)` `s1 |= s2`	Добавляет в множество s1 все элементы из множества s2
`s1.itersection(s2)` `s1 & s2`	Возвращает множество, являющееся пересечением множеств s1 и s2
`s1.itersection_update(s2)` `s1 &= s2`	Оставляет в множестве s1 только те элементы, которые есть в множестве s2
`s1.difference(s2)` `s1 - s2`	Возвращает разность множеств s1 и s2 (элементы, входящие в s1, но не входящие в s2)
`s1.difference_update(s2)` `s1 -= s2`	Удаляет из множества s1 все элементы, входящие в s2
`s1.symmetric_difference(s2)` `s1 ^ s2`	Возвращает симметрическую разность множеств s1 и s2 (элементы, входящие в s1 или в s2, но не в оба из них одновременно)
`s1.symmetric_difference_update(s2)` `s1 ^= s2`	Записывает в s1 симметрическую разность множеств s1 и s2
`s1.issubset(s2)` `s1 <= s2`	Возвращает `True`, если s1 является подмножеством s2
`s1.issuperset(s2)` `s1 >= s2`	Возвращает `True`, если s2 является подмножеством s1

Генераторы множеств

- Выражение генератора множеств по форме похоже на выражение генератора списков, однако записывается в фигурных, а не квадратных скобках.
- Генераторы множеств запускают цикл и на каждой его итерации накапливают результат выражения. Результатом является новый объект множества.

```
>>> {x ** 2 for x in [10, 20, 30, 40, 50]}
{1600, 900, 2500, 100, 400}
```

- Генераторы множеств могут также выполнять проход по объектам других типов, таких, как строки:

```
>>> {x for x in "hello"}
{"l", "h", "e", "o"}

>>> {c * 5 for c in "SPAM!"}
{"!!!!!", "SSSSS", "AAAAA", "MMMMM", "PPPPP"}
```

Примеры использования множеств

- Поскольку элементы во множестве сохраняются только однократно, множества могут быть использованы для *фильтрации дубликатов* в коллекциях.
- Коллекцию лишь нужно преобразовать во множество и затем выполнить обратное преобразование (если в этом есть необходимость):

```
>>> a = [1, 2, 2, 3, 5, 4, 1, 1, 2, 5, 4]

>>> set(a)
{1, 2, 3, 4, 5}

>>> a = list(set(a))

>>> a
[1, 2, 3, 4, 5]

>>> list(set(["hh", "ee", "ll", "ll", "oo"])) #Порядок
                                              #не сохраняется
["oo", "hh", "ll", "ee"]
```

Примеры использования множеств

- Множества могут также быть использованы при нахождении *различий* в списках, строках и прочих итерируемых объектах, однако снова нужно помнить о том, что исходный порядок следования элементов в сравниваемых объектах может быть изменен:

```
>>> a = [1, 2, 3, 4, 5, 7]

>>> a2 = [1, 2, 4, 5, 6]

>>> set(a) - set(a2)  #Различия в списках
{3, 7}

>>> s1 = "spam"

>>> s2 = "ham"

>>> set(s1) - set(s2)  #Различия в строках
{"s", "p"}

>>> set("tomato") - set(["p", "o", "t", "a", "t", "o"])  #Объекты
                                                         #разных типов
{"m"}
```

Примеры использования множеств

- Множества также можно применить для выполнения проверок на *равенство, нейтральное к порядку*.
- Два множества равны только в том случае, когда каждый элемент одного множества содержится в другом, иначе говоря, одно множество является подмножеством другого.
- К примеру, такой прием можно использовать для сравнения выводов программ, которые должны работать одинаковым образом, но могут генерировать результаты в разном порядке:

```
>>> a1, a2 = [1, 2, 3, 5, 4, 6], [2, 5, 3, 4, 1, 6]

>>> a1 == a2  #Порядок следования имеет значение
False

>>> set(a1) == set(a2)  #Проверка, нейтральная к порядку элементов
True

>>> "hello" == "olleh", set("hello") == set("olleh")
(False, True)
```

Примеры использования множеств

- Множества удобны при работе с крупными наборами данных, например, результатами запросов к базе данных.
- *Пересечение* двух множеств содержит объекты, общие для обоих множеств, а *объединение* – все элементы в том и другом множестве.
- Пример использования множеств со списками сотрудников вымышленной компании:

```
>>> engineers = {"Petr", "Ivan", "Fedor", "Anna", "Victoriya"}
>>> managers = {"Petr", "Anna", "Boris"}
>>> "Fedor" in engineers    #Является ли сотрудник инженером?
True
>>> engineers & managers    #Кто одновременно инженер и менеджер?
{"Anna", "Petr"}
>>> engineers | managers    #Все сотрудники из двух категорий
{"Anna", "Boris", "Fedor", "Petr", "Ivan", "Victoriya"}
>>> engineers - managers    #Инженеры, не являющиеся менеджерами
{"Ivan", "Fedor", "Victoriya"}
>>> engineers > managers    #Все ли инженеры - менеджеры?
False
>>> {"Fedor", "Anna"} < engineers #Оба ли сотрудника инженеры?
True
>>> engineers | managers > managers #Все сотрудники - надмножество менеджеров?
True
>>> managers ^ engineers       #Кто находится только в одной категории?
{"Boris", "Fedor", "Ivan", "Victoriya"}
```

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Контакты

Игорь Михайлович Долганов,
к.т.н., доцент ОХИ ИШПР

Учебный корпус №2, ауд. 136
dolganovim@tpu.ru
+7-960-978-43-07

Благодарю за внимание!