Множества и отображения

№ урока: 8 **Курс:** Python Essential

Средства обучения: PyCharm

Обзор, цель и назначение урока

После завершения урока обучающиеся будут иметь представление о множествах и отображениях в Python и основных стандартных классах множеств и отображений, их назначении и использовании.

Изучив материал данного занятия, учащийся сможет:

- Иметь представление о множествах
- Использовать классы set и frozenset
- Иметь представление об отображениях и словарях
- Использовать класс dict и другие классы из модуля collections
- Использовать представления словарей
- Создавать функции с произвольным количеством именованных параметров
- Распаковывать словари и другие отображения в именованные параметры функции

Содержание урока

1. Что такое множества?
2. Создание множеств
3. Изменение множеств
4. Удаление элементов из множества
5. Операции с множествами Python
6. Что такое отображения?

Резюме

Множества – это неупорядоченный набор элементов. Каждый элемент множества должен быть уникальным и не изменяемым.
Однако само множество изменчива. Мы можем добавлять или удалять элементы из него. Множества также могут использоваться для выполнения математических операций, таких как **объединение, пересечение, симметричная разность** и так далее.

Создание множеств
Множества создаются путем помещения всех элементов в фигурные скобки {}, разделенных запятыми, или с помощью встроенного класса set(). Обратите внимание, что порядок элементов в множествах не сохраняется.
Она может иметь любое количество элементов, и они могут быть разных типов (**целые числа, числа с плавающей запятой, кортеж, строка**). Множество **не может** иметь в качестве своих элементов изменяемые элементы, такие как списки или словари.

```
my_set = {1, 2, 3}
print(my_set)
# {1, 2, 3}

my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
# {1.0, (1, 2, 3), 'Hello'}
```

Следующий пример хорошо отображает главную характеристику множеств:

```
my_set = {1, 2, 3, 4, 3, 2}
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
print(my_set)

# множество не может хранить дубликаты (дубликаты удаляются автоматически)
# Вывод: {1, 2, 3, 4}
```

Множества можно создать с любого итерируемого объекта, например, списка:

```
my_set = set([1, 2, 3, 2])
print(my_set)

# Вывод: {1, 2, 3}
```

Мы **не можем** создать множество с изменяемых объектов:

```
my_set = {1, 2, [3, 4]}

# Traceback (most recent call last):
#   File "<string>", line 15, in <module>
#     my_set = {1, 2, [3, 4]}
# TypeError: unhashable type: 'list'
```

Создание пустое множество немного сложнее, чем пустой список. Дело в том, что синтаксис создания словаря и множества очень похож. Поэтому запись my_var = {} создаст пустой словарь, а не множество. Чтобы создать пустое множество, нужно написать: my_var = set().

Изменение множества

Множества изменяемы. Но, поскольку они неупорядоченны, индексация не имеет значения.

Мы не можем получить доступ к элементу множества или изменить его с помощью индексации или среза.

Мы можем добавить один элемент с помощью метода add() и несколько элементов с помощью метода update(). Метод update() может принимать в качестве аргумента **кортежи, списки, строки или другие наборы**. Во всех случаях дубликаты будут удалены.

```
# создание множества
my_set = {1, 3}
print(my_set)

# Такая запись не сработает
# my_set[0]

# Добавление элемента
my_set.add(2)
print(my_set)
# Вывод: {1, 2, 3}

# добавление нескольких элементов
my_set.update([2, 3, 4])
print(my_set)
# Вывод: {1, 2, 3, 4}

# добовление списка и множества одновременно
my_set.update([4, 5], {1, 6, 8})
print(my_set)
# Вывод: {1, 2, 3, 4, 5, 6, 8}
```

Удаление элементов из множества

Отдельный элемент можно удалить из набора с помощью методов discard() и remove().

Единственное различие между ними состоит в том, что функция discard() оставляет множество без изменений, если элемент отсутствует в наборе. А функция remove() вызовет ошибку, если элемент отсутствует в множестве.

Следующий пример иллюстрирует это:

```
# Разница между discard() и remove()
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
# Создание множества
my_set = {1, 3, 4, 5, 6}
print(my_set)

# Исключение элемента
# Вывод: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)

# Удаление элемента
# Вывод: {1, 3, 5}
my_set.remove(6)
print(my_set)

# Исключние элемента,
# которого нет в множестве
# Вывод: {1, 3, 5}
my_set.discard(2)
print(my_set)

# Удаление элемента
# которого нет в множестве
# Вывод: KeyError
my_set.remove(2)
```

Точно так же мы можем удалить и получить элемент с помощью метода pop().
Поскольку set - это **неупорядоченный** тип данных, невозможно определить, какой элемент будет "выталкиваться".
Мы также можем удалить все элементы из множества с помощью метода clear().

Операции с множествами Python
Множества могут использоваться для выполнения математических операций над наборами, таких как объединение, пересечение, разность и симметричная разность. Мы можем сделать это с помощью операторов или методов.
Рассмотрим следующие два множества для следующих операций.

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
```

Объединение множеств
Объединение A и B - это набор всех элементов из обоих множеств.
Объединение осуществляется с помощью оператора |. То же самое можно сделать с помощью метода union().

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Использвание оператора |
# Вывод: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

Попробуйте следующие примеры в оболочке Python:

```
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}

>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Пересечения множеств
Пересечение A и B - это набор элементов, общих в обоих множествах.
Пересечение выполняется с помощью оператора &. То же самое можно сделать с помощью метода crossction().

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Использование оператора &
# Вывод: {4, 5}
print(A & B)
```

Попробуйте следующие примеры в оболочке Python:

```
>>> A.intersection(B)
{4, 5}

>>> B.intersection(A)
{4, 5}
```

Разница множеств

Отличие множества B от множества A (**A - B**) - это набор элементов, которые находятся только в A, но не в B. Точно так же **B - A** - это набор элементов в B, но не в A.

Разница выполняется с помощью оператора -. То же самое можно сделать с помощью метода difference().

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Использование оператора - на A
# Вывод: {1, 2, 3}
print(A - B)
```

Попробуйте следующие примеры в оболочке Python:

```
>>> A.difference(B)
{1, 2, 3}

>>> B - A
{8, 6, 7}

>>> B.difference(A)
{8, 6, 7}
```

Семитическая разница множеств

Симметричная разность A и B - это набор элементов в A и B, но не в обоих (за исключением пересечения).

Симметричная разность выполняется с помощью оператора. То же самое можно сделать с помощью метода symmetric_difference().

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Использование опертора ^
# Вывод: {1, 2, 3, 6, 7, 8}
print(A ^ B)
```

Попробуйте следующие примеры в оболочке Python:

```
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}

>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```

Другие методы множеств

Метод	Описание
add()	Добавляет элемент в множество
clear()	Удаляет все элементы из множества

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

copy()	Возвращает копию множества
difference()	Возвращает разницу между двумя множествами
difference_update()	Уделяет все элементы, что присутствуют в другом множестве
discard()	Удаляет элемент из множества, если он является членом. (Ничего не делать, если элемент не установлен)
intersection()	Возвращает пересечение двух множеств как новое множество.
intersection_update()	Обновляет множество с пересечением себя и другого
isdisjoint()	Возвращает True, если два множества имеют нулевое пересечение
issubset()	Возвращает True, если другое множество содержит это множество
issuperset()	Возвращает True, если это множество содержит другое множество
pop()	Удаляет и возвращает произвольный элемент множества. Вызывает ошибку KeyError, если множество пусте
remove()	Удаляет элемент из множества. Если элемента нет, вызывает ошибку KeyError
symmetric_difference()	Возвращает симметричную разность двух множеств как новые множество
symmetric_difference_update()	Обновляет множество симметрической разницы между собой и другим
union()	Возвращает объединение множеств в новые множество
update()	Обновляет множество с объединением себя и других

Что такое отображения?
Отображение - это контейнер с неупорядоченной коллекцией пар элементов "ключ-значение". В разных языках синонимом отображений являются термины словарь, хеш-таблица или ассоциативный массив. Отображения в Python представлены единственным типом dict (**словарь**), в котором в качестве ключа может выступать любой **хэшируемый** объект, а в качестве значения - произвольный объект.
Создать словарь можно несколькими способами:

```
# Все эти примеры создают одинаковые словари
a = dict(one=1, two=2, three=3)
b = {'one': 1, 'two': 2, 'three': 3}
c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
d = dict([('two', 2), ('one', 1), ('three', 3)])
e = dict({'three': 3, 'one': 1, 'two': 2})

print(a == b == c == d == e)

print(a)

# Использование включений словарей (аналогично списковым включениям)
print({string: string.upper() for string in ('one', 'two', 'three')})
```

Те, кто имел опыт программирования на Си-подобных языках, могут подумать, что в этой строке ошибка (потому что в них операции сравнения связывались бы слева направо):

```
print(a == b == c == d == e)
```

Однако в Python такое сравнение абсолютно корректно и действительно проверяет все значения на равенство друг другу. Все последовательные операции сравнения и проверки равенства объединяются при помощи операции and.
В функции можно передавать произвольное количество позиционных аргументов, которые сохраняются в **кортеже**. Так же можно передавать произвольное количество именованных аргументов, которые сохраняются в **словаре**. Для этого перед именем данного словаря в списке формальных параметров ставится два символа **. Если используются оба способа передачи произвольного количества аргументов, параметр в форме **kwargs в сигнатуре функции должен идти после параметра в форме *args.

```
def function(**kwargs):
    print(kwargs)
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
function(arg1='value1', arg2='value2')

# Аналогично можно и распаковывать любые отображения
# в именованные параметры при вызове функции.

options = {
    'sep': ', ',
    'end': ';\n'
}

print('value1', 'value2', **options)
```

Рассмотрим некоторые операции над словарями:

```
"""Обзор операций со словарями"""

phonebook = {
    'Jack': '032-846',
    'Guido': '917-333',
    'Mario': '120-422',
    'Mary': '890-532',  # последняя запятая игнорируется
}

# len(d) – количество элементов.
print(len(phonebook), 'entries found')

print()

# d[key] – получение значения с ключом key. Если такой ключ не существует
# и отображение реализует специальный метод __missing__(self, key), то он
# вызывается. Если ключ не существует и метод __missing__ не определён,
# выбрасывается исключение KeyError.
try:
    print('Mary:', phonebook['Mary'])
    print('Lumberjack:', phonebook['Lumberjack'])
except KeyError as e:
    print('No entry for', *e.args)

print()

# d[key] = value – изменить значение или создать новую пару ключ-значение, если
# ключ не существует.
phonebook['Lumberjack'] = '000-777'

# key in d, key not in d – проверка наличия ключа в отображении.
for person in ('Guido', 'Mary', 'Ahmed'):
    if person in phonebook:
        print(person, 'is in the phonebook')
    else:
        print('No entry found for', person)

print()

# iter(d) – то же самое, что iter(d.keys()).
print('People in the phonebook:')
for person in phonebook:
    print(person)

print()

# copy() – создать неполную копию словаря.
phonebook_copy = phonebook.copy()
print('Phonebook:', phonebook)
print('Phonebook copy:', phonebook_copy)

print()

# clear() – удалить все элементы словаря.
phonebook_copy.clear()
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
print('Phonebook:', phonebook)
print('Phonebook copy:', phonebook_copy)

print()

# (метод класса) dict.fromkeys(sequence[, value]) – создаёт новый словарь с
# ключами из последовательности sequence и заданным значением (по умолчанию –
# None).
numbers_dict = dict.fromkeys(range(3), 42)
print(numbers_dict)

print()

# d.get(key[, default]) – безопасное получение значения по ключу (никогда не
# выбрасывает KeyError).  Если ключ не найден, возвращается значение default
# (по-умолчанию – None).
for key in range(5):
    print('{}:'.format(key), numbers_dict.get(key, 0))

print()

# d.items() – в Python 3 возвращает объект представления словаря,
# соответствующий парам (двухэлементным кортежам) вида (ключ, значение).  В
# Python 2 возвращает соответствующий список, а метод iteritems() возвращает
# итератор.  Аналогичный метод в Python 2.7 – viewitems().
print('Items:', phonebook.items())

# d.keys() – в Python 3 возвращает объект представления словаря,
# соответствующий ключам словаря.  В Python 2 возвращает соответствующий
# список, а метод iterkeys() возвращает итератор.  Аналогичный метод в Python
# 2.7 – viewkeys().
print('Keys:', phonebook.keys())

# d.values() – в Python 3 возвращает объект представления словаря,
# соответствующий значениям.  В Python 2 возвращает соответствующий список, а
# метод itervalues() возвращает итератор.  Аналогичный метод в Python 2.7 –
# viewvalues().
print('Values:', phonebook.values())

print()

# d.pop(key[, default]) – если ключ key существует, удаляет элемент из словаря
# и возвращает его значение.  Если ключ не существует и задано значение
# default, возвращается данное значение, иначе выбрасывается исключение
# KeyError.
number = phonebook.pop('Lumberjack')
print('Deleted Lumberjack (was ' + number + ')')
print(phonebook)

print()

# d.popitem() – удаляет произвольную пару ключ-значение и возвращает её.  Если
# словарь пустой, возникает исключение KeyError.  Метод полезен для алгоритмов,
# которые обходят словарь, удаляя уже обработанные значения (например,
# определённые алгоритмы, связанные с теорией графов).
person = phonebook.popitem()
print('Popped {} (phone: {})'.format(*person))

print()

# d.setdefault(key[, default]) – если ключ key существует, возвращает
# соответствующее значение.  Иначе создаёт элемент с ключом key и значением
# default.  default по умолчанию равен None.
for person in ('Jack', 'Liz'):
    phone = phonebook.setdefault(person, '000-000')
    print('{}: {}'.format(person, phone))

print(phonebook)
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
print()

# d.update(mapping) – принимает либо другой словарь или отображение, либо
# итерабельный объект, состоящий из итерабельных объектов – пар ключ-значение,
# либо именованные аргументы.  Добавляет соответствующие элементы в словарь,
# перезаписывая элементы с существующими ключами.
phonebook.update({'Alex': '832-438', 'Alice': '231-987'})
phonebook.update([('Joe', '217-531'), ('James', '783-428')])
phonebook.update(Carl='783-923', Victoria='386-486')
print(phonebook)
```

Дополнительное задание

Задание
Создайте словарь с ключами-строками и значениями-числами. Создайте функцию, которая принимает произвольное количество именованных параметров. Вызовите её с созданным словарём и явно указывая параметры.

Самостоятельная деятельность учащегося

Задание 1
Даны две строки. Выведите на экран символы, которые есть в обоих строках.

Задание 2
Создайте программу, которая эмулирует работу сервиса по сокращению ссылок. Должна быть реализована возможность ввода изначальной ссылки и короткого названия и получения изначальной ссылки по её названию.

Задание 3
Ознакомьтесь при помощи документации с классами OrderedDict, defaultdict и ChainMap модуля collections.

Рекомендуемые ресурсы

Документация Python
https://docs.python.org/3/tutorial/datastructures.html#sets
https://docs.python.org/3/tutorial/datastructures.html#dictionaries
https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset
https://docs.python.org/3/library/stdtypes.html#mapping-types-dict
https://docs.python.org/3/library/stdtypes.html#dictionary-view-objects
https://docs.python.org/3/library/collections.html

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Множество
https://ru.wikipedia.org/wiki/Множество_(тип_данных)
https://en.wikipedia.org/wiki/Map_(mathematics)
https://ru.wikipedia.org/wiki/Ассоциативный_массив
https://ru.wikipedia.org/wiki/Мультимножество

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com