

Подпишитесь на DeepL Pro для редактирования данного документа.  
Дополнительную информацию можно найти на странице [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document).

# 📘 День 7

## Sets

Набор - это коллекция предметов. Позвольте мне вернуть вас к уроку математики в начальной или средней школе. Математическое определение множества можно применить и в Python. Множество - это коллекция неупорядоченных и неиндексированных отдельных элементов. В Python множество используется для хранения уникальных элементов, а среди множеств можно найти \_объединение\_, \_пересечение\_, \_разность\_, \_симметричную разность\_, \_подмножество\_, \_супермножество\_ и \_несовпадающее множество\_.

### Создание набора

Мы используем фигурные скобки, {} для создания множества или встроенную функцию \*set()\*.

- Создание пустого множества

``py

# синтаксис

st = {}

# или

st = set()

```

- Создание набора с начальными элементами

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

```

\*\*Пример:\*\*

``py

# синтаксис

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

```

### Получение длины набора

Для нахождения длины набора мы используем метод \*\*len()\*\*.

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

len(set)

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

len(fruits)

```

### Доступ к элементам в наборе

Для доступа к элементам мы используем циклы. Мы рассмотрим это в разделе циклов

### Проверка элемента

Чтобы проверить, существует ли элемент в списке, мы используем оператор \_in\_ membership.

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

print("Does set st contains item3? ", 'item3' in st) # Does set st contains item3? True

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

print('mango' in fruits ) # True

```

### Добавление элементов в набор

После создания набора мы не можем изменить какие-либо элементы, а также можем добавить дополнительные элементы.

- Добавьте один элемент с помощью \_add()\_

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

st.add('item5')

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

fruits.add('lime')

```

- Добавление нескольких элементов с помощью \_update()\_

Функция \*update()\* позволяет добавить несколько элементов в набор. Функция \*update()\* принимает аргумент в виде списка.

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

st.update(['item5','item6','item7'])

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

овощи = ('помидор', 'картофель', 'капуста', 'лук', 'морковь')

fruits.update(vegetables)

```

### Удаление элементов из набора

Мы можем удалить элемент из набора с помощью метода \_remove()\_. Если элемент не найден, метод \_remove()\_ выдаст ошибку, поэтому полезно проверить, существует ли элемент в данном наборе. Однако метод \_discard()\_ не вызывает никаких ошибок.

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

st.remove('item2')

```

Методы pop() удаляют случайный элемент из списка и возвращают удаленный элемент.

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

fruits.pop() # удаляет случайный элемент из набора

```

Если мы заинтересованы в удаленном предмете.

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

удаленный\_элемент = fruits.pop()

```

### Очистка элементов в наборе

Если мы хотим очистить или опустошить набор, мы используем метод \_clear\_.

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

st.clear()

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

fruits.clear()

print(fruits) # set()

```

### Удаление набора

Если мы хотим удалить сам набор, мы используем оператор \_del\_.

``py

# синтаксис

st = {'item1', 'item2', 'item3', 'item4'}

улица Дель

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

фрукты

```

### Преобразование списка в набор

Мы можем преобразовать список в набор и набор в список. При преобразовании списка в набор удаляются дубликаты, и резервируются только уникальные элементы.

``py

# синтаксис

lst = ['item1', 'item2', 'item3', 'item4', 'item1']

st = set(lst) # {'item2', 'item4', 'item1', 'item3'} - порядок случайный, потому что множества в общем случае неупорядочены

```

\*\*Пример:\*\*

``py

фрукты = ['банан', 'апельсин', 'манго', 'лимон', 'апельсин', 'банан'].

fruits = set(fruits) # {'mango', 'lemon', 'banana', 'orange'}

```

### Объединение множеств

Мы можем объединить два набора с помощью метода \_union()\_ или \_update()\_.

- Союз

Этот метод возвращает новый набор

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item5', 'item6', 'item7', 'item8'}

st3 = st1.union(st2)

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

овощи = { 'помидор', 'картофель', 'капуста', 'лук', 'морковь'}

print(fruits.union(vegetables)) # { {'лимон', 'морковь', 'помидор', 'банан', 'манго', 'апельсин', 'капуста', 'картофель', 'лук'}

```

- Обновление

Этот метод вставляет множество в заданное множество

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item5', 'item6', 'item7', 'item8'}

st1.update(st2) # содержимое st2 добавляется в st1

```

\*\*Пример:\*\*

``py

фрукты = {'банан', 'апельсин', 'манго', 'лимон'}

овощи = { 'помидор', 'картофель', 'капуста', 'лук', 'морковь'}

fruits.update(vegetables)

print(fruits) # { {'lemon', 'carrot', 'tomato', 'banana', 'mango', 'orange', 'cabbage', 'potato', 'onion'}

```

### Поиск элементов пересечения

Intersection возвращает набор элементов, которые находятся в обоих наборах. Смотрите пример

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item3', 'item2'}

st1.intersection(st2) # {'item3', 'item2'}

```

\*\*Пример:\*\*

``py

целые\_числа = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

четные\_числа = {0, 2, 4, 6, 8, 10}

whole\_numbers.intersection(even\_numbers) # {0, 2, 4, 6, 8, 10}

python = {'p', 'y', 't', 'h', 'o', 'n'}

dragon = {'d', 'r', 'a', 'g', 'o', 'n'}

python.intersection(dragon) # {'o', 'n'}

```

### Проверка подмножества и супермножества

Множество может быть подмножеством или супермножеством других множеств:

- Подмножество: \_issubset()\_

- Суперсет: \_issuperset\_

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item2', 'item3'}

st2.issubset(st1) # True

st1.issuperset(st2) # True

```

\*\*Пример:\*\*

``py

целые\_числа = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

четные\_числа = {0, 2, 4, 6, 8, 10}

whole\_numbers.issubset(even\_numbers) # Ложь, потому что это супермножество

whole\_numbers.issuperset(even\_numbers) # True

python = {'p', 'y', 't', 'h', 'o', 'n'}

dragon = {'d', 'r', 'a', 'g', 'o', 'n'}

python.issubset(dragon) # Ложь

```

### Проверка разницы между двумя наборами

Он возвращает разницу между двумя множествами.

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item2', 'item3'}

st2.difference(st1) # set()

st1.difference(st2) # {'item1', 'item4'} => st1\st2

```

\*\*Пример:\*\*

``py

целые\_числа = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

четные\_числа = {0, 2, 4, 6, 8, 10}

целые\_числа.разность(четные\_числа) # {1, 3, 5, 7, 9}

python = {'p', 'y', 't', 'o','n'}

dragon = {'d', 'r', 'a', 'g', 'o', 'n'}

python.difference(dragon) # {'p', 'y', 't'} - результат неупорядочен (характерно для множеств)

dragon.difference(python) # {'d', 'r', 'a', 'g'}

```

### Нахождение симметричной разницы между двумя множествами

Он возвращает симметричную разность между двумя множествами. Это означает, что возвращается множество, которое содержит все элементы из обоих множеств, за исключением элементов, которые присутствуют в обоих множествах, математически: (A\B) ∪ (B\A)

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item2', 'item3'}

# это значит (A\B)∪(B\A)

st2.symmetric\_difference(st1) # {'item1', 'item4'}

```

\*\*Пример:\*\*

``py

целые\_числа = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

some\_numbers = {1, 2, 3, 4, 5}

whole\_numbers.symmetric\_difference(some\_numbers) # {0, 6, 7, 8, 9, 10}

python = {'p', 'y', 't', 'h', 'o', 'n'}

dragon = {'d', 'r', 'a', 'g', 'o', 'n'}

python.symmetric\_difference(dragon) # {'r', 't', 'p', 'y', 'g', 'a', 'd', 'h'}

```

### Объединение множеств

Если два множества не имеют общего элемента или элементов, мы называем их несовпадающими множествами. Мы можем проверить, являются ли два множества совместными или несовместными, используя метод \_isdisjoint()\_.

``py

# синтаксис

st1 = {'item1', 'item2', 'item3', 'item4'}

st2 = {'item2', 'item3'}

st2.isdisjoint(st1) # Ложь

```

\*\*Пример:\*\*

``py

четные\_числа = {0, 2, 4 , 6, 8}

четные\_числа = {1, 3, 5, 7, 9}

even\_numbers.isdisjoint(odd\_numbers) # True, потому что нет общего элемента

python = {'p', 'y', 't', 'h', 'o', 'n'}

dragon = {'d', 'r', 'a', 'g', 'o', 'n'}

python.isdisjoint(dragon) # Ложь, есть общие элементы {'o', 'n'}

```

🌕 Вы - восходящая звезда. Вы только что завершили 7-й день испытаний и продвинулись на 7 шагов вперед на пути к величию. Теперь сделайте несколько упражнений для вашего мозга и мышц.

## 💻 Упражнения: День 7

``py

# наборы

it\_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}

A = {19, 22, 24, 20, 25, 26}

B = {19, 22, 20, 25, 26, 24, 28, 27}

возраст = [22, 19, 24, 25, 26, 24, 25, 24]

```

### Упражнения: Уровень 1

1. Найдите длину множества it\_companies

2. Добавьте 'Twitter' в it\_companies

3. Вставьте сразу несколько ИТ-компаний в набор it\_companies

4. Удалите одну из компаний из набора it\_companies

5. В чем разница между "удалить" и "выбросить

### Упражнения: Уровень 2

1. Соедините A и B

1. Найдите A пересечение B

1. Является ли A подмножеством B

1. Являются ли множества A и B непересекающимися

1. Соедините A с B и B с A

1. Какова симметричная разность между A и B

1. Полностью удалите наборы

### Упражнения: Уровень 3

1. Преобразуйте возраст в множество и сравните длину списка и множества, какая из них больше?

1. Объясните разницу между следующими типами данных: строка, список, кортеж и набор

2. \_Я учитель и люблю вдохновлять и учить людей.\_ Сколько уникальных слов было использовано в предложении? Используйте методы разделения и набора, чтобы получить уникальные слова.

# 📘 День 8

## Словари

Словарь - это коллекция неупорядоченных, изменяемых (mutable) данных парного типа (ключ: значение).

### Создание словаря

Для создания словаря мы используем фигурные скобки, {} или встроенную функцию \*dict()\*.

``py

# синтаксис

empty\_dict = {}

# Словарь со значениями данных

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

```

\*\*Пример:\*\*

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

```

Приведенный выше словарь показывает, что значением может быть любой тип данных: строка, булево, список, кортеж, набор или словарь.

### Длина словаря

Он проверяет количество пар 'ключ: значение' в словаре.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

print(len(dct)) # 4

```

\*\*Пример:\*\*

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

print(len(person)) # 7

```

### Доступ к элементам словаря

Мы можем получить доступ к элементам словаря, обратившись к имени его ключа.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

print(dct['key1']) # value1

print(dct['key4']) # value4

```

\*\*Пример:\*\*

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

print(person['first\_name']) # Asabeneh

print(person['country']) # Финляндия

print(person['skills']) # ['JavaScript', 'React', 'Node', 'MongoDB', 'Python']

print(person['skills'][0]) # JavaScript

print(person['address']['street']) # Космическая улица

print(person['city']) # Ошибка

```

Доступ к элементу по имени ключа приводит к ошибке, если ключ не существует. Чтобы избежать этой ошибки, сначала нужно проверить, существует ли ключ, или можно воспользоваться методом \_get\_. Метод get возвращает None, который является типом данных объекта NoneType, если ключ не существует.

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

print(person.get('first\_name')) # Асабенех

print(person.get('country')) # Финляндия

print(person.get('skills')) #['HTML','CSS','JavaScript','React','Node','MongoDB','Python']

print(person.get('city')) # Нет

```

### Добавление элементов в словарь

Мы можем добавить новые пары ключей и значений в словарь

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

dct['key5'] = 'value5'

```

\*\*Пример:\*\*

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

person['job\_title'] = 'Instructor'

person['skills'].append('HTML')

print(person)

```

### Изменение элементов в словаре

Мы можем изменять элементы в словаре

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

dct['key1'] = 'value-one'

```

\*\*Пример:\*\*

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

person['first\_name'] = 'Eyob'

person['age'] = 252

```

### Проверка ключей в словаре

Мы используем оператор \_in\_, чтобы проверить, существует ли ключ в словаре

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

print('key2' in dct) # True

print('key5' в dct) # Ложь

```

### Удаление пар ключей и значений из словаря

- \_pop(key)\_: удаляет элемент с указанным именем ключа:

- \_popitem()\_: удаляет последний элемент

- \_del\_: удаляет элемент с указанным именем ключа

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

dct.pop('key1') # удаляет элемент key1

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

dct.popitem() # удаляет последний элемент

del dct['key2'] # удаляет элемент key2

```

\*\*Пример:\*\*

``py

человек = {

'first\_name':'Asabeneh',

'last\_name':'Yetayeh',

'возраст':250,

'страна': 'Финляндия',

'is\_marred':True,

'skills':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],

'адрес':{

'улица': 'Космическая улица',

'zipcode':'02210'

}

}

person.pop('first\_name') # Удаляет элемент firstname

person.popitem() # Удаляет элемент адреса

del person['is\_married'] # Удаляет элемент is\_married

```

### Изменение словаря в список элементов

Метод \_items()\_ изменяет словарь на список кортежей.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

print(dct.items()) # dict\_items([('key1', 'value1'), ('key2', 'value2'), ('key3', 'value3'), ('key4', 'value4')])

```

### Очистка словаря

Если нам не нужны элементы в словаре, мы можем очистить их с помощью метода \_clear()\_.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

print(dct.clear()) # Нет

```

### Удаление словаря

Если мы не используем словарь, мы можем удалить его полностью

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

дель дкт

```

### Копирование словаря

Мы можем скопировать словарь, используя метод \_copy()\_. Используя копирование, мы можем избежать мутации исходного словаря.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

dct\_copy = dct.copy() # { {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

```

### Получение ключей словаря в виде списка

Метод \_keys()\_ выдает нам все ключи словаря в виде списка.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

ключи = dct.keys()

print(keys) # dict\_keys(['key1', 'key2', 'key3', 'key4'])

```

### Получение значений словаря в виде списка

Метод \_values\_ предоставляет нам все значения словаря в виде списка.

``py

# синтаксис

dct = {'key1':'value1', 'key2':'value2', 'key3':'value3', 'key4':'value4'}

значения = dct.values()

print(values) # dict\_values(['value1', 'value2', 'value3', 'value4'])

```

🌕 Вы поразительны. Теперь вы заряжены силой словарей. Вы только что завершили 8-й день испытаний, и вы на 8 шагов впереди на пути к величию. Теперь сделайте несколько упражнений для вашего мозга и мышц.

## 💻 Упражнения: День 8

1. Создайте пустой словарь под названием dog

2. Добавьте кличку, цвет, породу, лапы, возраст в словарь собаки

3. Создайте словарь студентов и добавьте имя\_имя\_фамилия, фамилия\_фамилия, пол, возраст, семейное положение, навыки, страну, город и адрес в качестве ключей для словаря.

4. Получите длину словаря студента

5. Получите значение skills и проверьте тип данных, это должен быть список

6. Измените значения навыков, добавив один или два навыка

7. Получить ключи словаря в виде списка

8. Получить значения словаря в виде списка

9. Измените словарь на список кортежей с помощью метода \_items()\_.

10. Удалите один из элементов словаря

11. Удалить один из словарей