Множества, словари

Рассмотрим ещё одну коллекцию Python — множество (set). Чтобы задать множество, можно перечислить его элементы внутри фигурных скобок. Например, создадим множество гласных букв русского алфавита:

```
vowels = {"а", "е", "ё", "и", "о", "у", "ы", "э", "ю", "я"}
```

Для создания пустого множества следует использовать функцию set() без аргументов, а для определения количества элементов используется уже известная нам функция len():

```
empty_set = set()print(f"Длина пустого множества равна {len(empty_set)}.")
```

Вывод программы:

```
Длина пустого множества равна 0.
```

Множество можно получить из других коллекций, применив к ним функцию set(). Например, создадим множество из строки:

```
word = "коллекция"
letters = set(word)print(letters)
```

Выполнив программу два раза, получим следующий вывод программы:

```
{'е', 'о', 'и', 'я', 'к', 'л', 'ц'}
{'л', 'к', 'и', 'ц', 'я', 'е', 'о'}
```

Обратите внимание: порядок вывода элементов множества при выполнении примера может меняться произвольно из-за свойства неупорядоченности множества. Так, элементы множества не имеют индексов, и можно только проверить принадлежность элемента множеству.

Другое свойство множества — уникальность его элементов: они не имеют дублей.

В итоге в примере мы получили множество уникальных букв слова, потеряв при этом порядок.

Проверить, принадлежит ли значение множеству, можно с помощью оператора in.

Узнаем, принадлежит ли введённая буква русского алфавита к гласным:

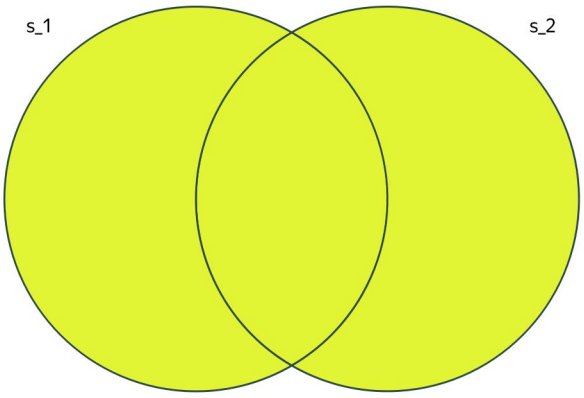
```
vowels = {"а", "е", "ё", "и", "о", "у", "ы", "э", "ю", "я"}
letter = input("Введите букву русского алфавита: ")if letter.lower() in vowels:
  print("Гласная буква")else:
  print("Согласная буква")
```

По элементам множества можно пройти в цикле:

```
vowels = {"а", "е", "ё", "и", "о", "у", "ы", "э", "ю", "я"}for letter in vowels:
  print(letter)
```

Выполнив пример несколько раз, мы снова видим разную последовательность вывода букв.

Множества в Python позволяют выполнять следующие операции:

- Объединение множеств. Возвращает множество, состоящее из элементов всех объединяемых множеств. Обозначается union() или |. Графическое представление операции:
- 

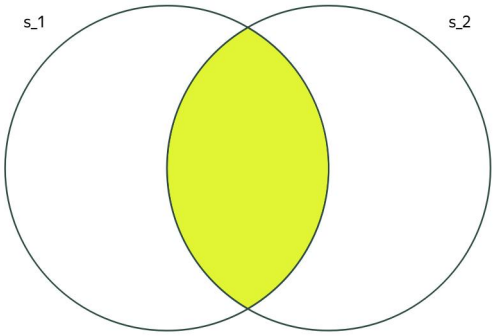
- Пример:

```
s_1 = {1, 2, 3}
s_2 = {3, 4, 5}
s_union = s_1 | s_2# s_union = s_1.union(s_2)print(s_union)
```

Вывод программы:

```
{1, 2, 3, 4, 5}
```

Пересечение множеств. Возвращает множество, состоящее из общих элементов пересекаемых множеств. Обозначается intersection или &. Графическое представление операции:



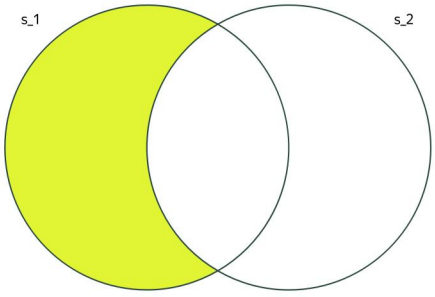
Пример:

```
s_1 = {1, 2, 3}
s_2 = {3, 4, 5}
s_intersection = s_1 & s_2# s_intersection = s_1.intersection(s_2)print(s_intersection)
```

Вывод программы:

```
{3}
```

Разность множеств. Возвращает множество, где указаны элементы из первого множества, которых нет во втором — вычитаемом — множестве. Обозначается difference или -. Графическое представление операции:



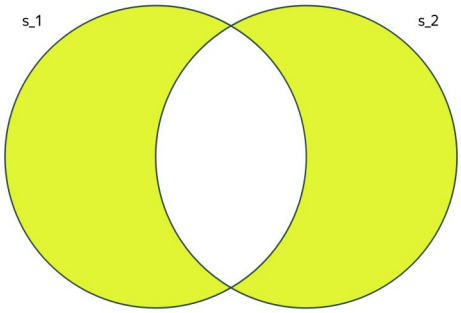
Пример:

```
s_1 = {1, 2, 3}
s_2 = {3, 4, 5}
s_dif = s_1 - s_2# s_dif = s_1.difference(s_2)print(s_dif)
```

Вывод программы:

{1, 2}

Симметричная разность множеств. Возвращает множество, состоящее из элементов, встречающихся в первом или втором множестве, но не в обоих сразу.
Обозначается symmetric_difference или ^.



Пример:

```
s_1 = {1, 2, 3}
s_2 = {3, 4, 5}
s_sym_dif = s_1 ^ s_2# s_sym_dif = s_1.symmetric_difference(s_2)print(s_sym_dif)
```

Вывод программы:

{1, 2, 4, 5}

В качестве примера использования множеств и их операций определим, какие гласные буквы встречаются в слове «коллекция»:

```
vowels = {"а", "е", "ё", "и", "о", "у", "ы", "э", "ю", "я"}
letters = set("коллекция")print(", ".join(letters & vowels))
```

Вывод программы:
о, я, и
Для множеств Python доступны следующие операции сравнения:

- Совпадение двух множеств. Обозначается ==. Пример:

```
s_1 = {1, 2, 3}
s_2 = {3, 1, 2}print(s_1 == s_2)
```

Вывод программы:
True

- Подмножество. Все элементы первого множества есть во втором. Обозначается <=.

Пример:

```
s_1 = {1, 2, 3}
s_2 = {1, 2, 3, 4}print(s_1 <= s_2)
```

Вывод программы:
True

- Надмножество. Первое множество содержит все элементы второго. Обозначается >=.

Пример:

```
s_1 = {1, 2, 3}
s_2 = {1, 2, 3, 4}print(s_2 >= s_1)
```

Вывод программы:
True

Множество является изменяемой коллекцией. Методы, изменяющие исходное множество, перечислены в следующей таблице.

Метод	Описание	Пример	Вывод
set.add(e)	Добавить элемент во множество	s = set() s.add(1) print(s)	{1}
set.remove(e)	Удалить элемент множества. Возвращает исключение KeyError, если элемент не принадлежит множеству	s = {1, 2, 3} s.remove(2) print(s)	{1, 3}
set.discard(e)	Удалить элемент, если он принадлежит множеству	s = {1, 2, 3} s.discard(2) print(s)	{1, 3}
set.pop()	Вернуть и удалить произвольный элемент множества	s = {1, 2, 3} x = s.pop() print(x) print(s)	2 {1, 3}
set.clear()	Очистить множество, удалив все его элементы	s = {1, 2, 3} s.clear() print(s)	set()

Чтобы перейти к ещё одной коллекции, рассмотрим пример: пусть в программе нужно хранить информацию о странах и их столицах. Эту задачу можно решить и с помощью известных нам коллекций. К примеру, списком, в котором элементами будут кортежи, содержащие пары значений — страну и её столицу:

```
countries_and_capitals = [("Россия", "Москва"), ("США", "Вашингтон"), ("Франция", "Париж")]
```

Представим, что в такой программе нам необходимо вывести столицу для какой-то страны, например для Франции. Тогда нам придётся пройти в цикле по списку кортежей, сравнивая нулевой элемент каждого кортежа со строкой «Франция». Когда такой кортеж будет найден, мы выведем первый элемент этого кортежа, содержащий строку с названием столицы:

```
countries_and_capitals = [("Россия", "Москва"), ("США", "Вашингтон"), ("Франция", "Париж")]for country in countries_and_capitals:
  if country[0] == "Франция":
    print(country[1])
    break
```

Было бы неплохо, если бы мы могли взять из коллекции значение (название столицы в нашем примере) по названию страны, то есть не по числовому индексу, а по строке. Такая коллекция есть в Python. Она называется «словарь» (dict). Словарь похож на список, но вместо индексов элементов в словаре используются ключи, а по ключам в словаре хранятся значения. Благодаря словарям нашу программу можно переписать, используя в качестве ключей словаря названия стран, а в качестве значений по ключам — названия столиц этих стран:

```
countries_and_capitals = {"Россия": "Москва",
                "США": "Вашингтон",
                "Франция": "Париж"}print(countries_and_capitals["Франция"])
```

Как видно из программы, для создания словаря можно перечислить в фигурных скобках пары: ключ и значение. Ключ и значение отделяются двоеточием и пробелами, а пары перечисляются через запятую.
В качестве ключей можно использовать значения неизменяемого типа: числа, строки, кортежи. Значения по ключу могут быть любого типа данных.
Чтобы взять значение по ключу, необходимо указать ключ в квадратных скобках после имени словаря.
Если же нужно добавить новый ключ в словарь, то его указывают после имени словаря в левой части операции присваивания, а значение, которое будет храниться по этому ключу, — в правой части:

```
countries_and_capitals = {"Россия": "Москва",
                          "США": "Вашингтон",
                          "Франция": "Париж"}
countries_and_capitals["Сербия"] = "Белград"print(countries_and_capitals)
```

Обратите внимание, при записи значения по уже существующему ключу он создаётся заново с новым значением, а прошлый стирается:

```
d = {"key": "old_value"}
d["key"] = "new_value"print(d["key"])
```

Вывод программы:

```
new_value
```

При попытке взять значение по несуществующему ключу происходит исключение KeyError:

```
countries_and_capitals = {"Россия": "Москва",
                          "США": "Вашингтон",
                          "Франция": "Париж"}print(countries_and_capitals["Сербия"])
```

Вывод программы:

```
KeyError: 'Сербия'
```

Для проверки существования ключа в словаре следует использовать уже известный нам оператор in:

```
countries_and_capitals = {"Россия": "Москва",
                          "США": "Вашингтон",
                          "Франция": "Париж"}if "Сербия" in countries_and_capitals:
  print(countries_and_capitals["Сербия"])else:
  print("Страна пока не добавлена в словарь")
```

По ключам в словаре можно пройти в цикле for:

```
countries_and_capitals = {"Россия": "Москва",
                          "США": "Вашингтон",
                          "Франция": "Париж"}for country in countries_and_capitals:
  print(f"У страны {country} столица — {countries_and_capitals[country]}.")
```

Вывод программы:

```
У страны Россия столица — Москва.
У страны США столица — Вашингтон.
У страны Франция столица — Париж.
```

Вспомним, что значением по ключу в словаре может быть значение любого типа. Рассмотрим пример, в котором значением по ключу будет список.
Пусть с клавиатуры вводятся названия стран, каждая с новой строки. При вводе возможны повторы стран. Сигналом окончания ввода будет строка «СТОП». Необходимо вывести, в каких строках (начиная с 0) встречалась каждая из стран. Для решения задачи будем использовать словарь, в котором по ключам — названиям стран будем хранить список номеров строк, в которых эти страны встречались.

```
# создаём пустой словарь
countries = dict()# вводим первую строку до цикла (можно заменить, использовав оператор-морж)
country = input()# создаём счётчик номеров строк
str_number = 0# продолжаем цикл, пока не введена строка «СТОП»while country != "СТОП":
```

```
    # если введённой страны нет в словаре, создаём ключ и записываем по ключу список из одного номера строки
    if country not in countries:
        countries[country] = [str_number]
    # иначе добавляем в список по ключу новое значение номера строки
    else:
        countries[country].append(str_number)
    # увеличиваем счётчик
    str_number += 1
    # вводим следующую строку
    country = input()# выводим название страны и полученные списки с новой строкиfor country in countries:
    print(f"{country}: {countries[country]}")
```

Пример ввода:

США
США
Россия
Россия
Россия
Франция
Сербия
СТОП

Вывод программы:

США: [0, 1]
Россия: [2, 3, 4]
Франция: [5]
Сербия: [6]

Основные операции для словарей перечислены в следующей таблице.

Операция	Описание	Пример	Вывод
len(d)	Возвращает количество ключей в словаре	d = {"a": 1, "b": 2, "c": 3} print(len(d))	3
del d[key]	Удалить ключ из словаря. Если ключа нет, то вызывается исключение KeyError	d = {"a": 1, "b": 2, "c": 3} del d["b"] print(d)	{"a": 1, "c": 3}
dict.clear()	Удалить все ключи и значения в словаре	d = {"a": 1, "b": 2, "c": 3} d.clear() print(d)	{}
dict.copy()	Возвращает копию словаря	d = {"a": 1, "b": 2, "c": 3} d_new = d.copy() print(d_new)	{"a": 1, "b": 2, "c": 3}
dict.get(key, default)	Возвращает значение по ключу key. Если ключа нет, то возвращает значение default	d = {"a": 1, "b": 2, "c": 3} print(d.get("e", "Ключа нет в словаре"))	Ключа нет в словаре
dict.items()	Возвращает итерируемый объект,	d = {"a": 1, "b": 2, "c":	a 1

Операция	Описание	Пример	Вывод
	состоящий из кортежей (ключ, значение) словаря	3} for key, value in d.items(): print(key, value)	b 2 c 3
dict.keys()	Возвращает итерируемый объект, состоящий из ключей словаря	d = {"a": 1, "b": 2, "c": 3} for key in d.keys(): print(key)	a b c
d.pop(key, default)	Возвращает значение по ключу key и удаляет его из словаря. Если ключа нет, то возвращает default	d = {"a": 1, "b": 2, "c": 3} x = d.pop("a") print(x)	1
dict.values()	Возвращает итерируемый объект, состоящий из значений словаря	d = {"a": 1, "b": 2, "c": 3} for value in d.values(): print(value)	1 2 3

Перепишем код примера про страны с использованием метода get():

```
# создаём пустой словарь
countries = dict()# вводим первую строку до цикла (можно заменить, использовав оператор-морж)
country = input()# создаём счётчик номеров строк
str_number = 0# продолжаем цикл, пока не введена строка «СТОП»while country != "СТОП":
  # Если страна country есть среди ключей, то get() возвращает список,
  # хранящийся по этому ключу, иначе get() возвращает пустой список.
  # Добавляем в список значение str_number.
  countries[country] = countries.get(country, []) + [str_number]
  # увеличиваем счётчик
  str_number += 1
  # вводим следующую строку
  country = input()# выводим название страны и полученные списки с новой строкиfor country in countries:
  print(f"{country}: {countries[country]}")
```

Метод get() позволил заменить четыре строчки программы одной. Проверьте самостоятельно, что вывод программы не изменился для прежних входных данных.