Занятие 2. Словари и множества

На прошлом занятии были рассмотрены три встроенные коллекции, относящиеся к категории последовательностей, — строки, списки и кортежи. Эта глава посвящена встроенным коллекциям, которые не являются последовательностями, — словарям и множествам.
Словарь (dictionary) представляет собой неупорядоченную коллекцию для хранения пар «ключ-значение», связывающих неизменяемые ключи со значениями (по аналогии с тем, как в традиционных словарях слова связываются с определениями).
Множество (set) представляет собой неупорядоченную коллекцию уникальных неизменяемых элементов.

Словари

Словарь связывает ключи со значениями. Каждому ключу соответствует конкретное значение.

Ключи	Типы ключей	Значения	Типы значений
Названия стран	str	Коды стран в интернете	str
Целые числа	int	Римские числа	str
Штаты	str	Сельскохозяйственные продукты	список str
Пациенты в больнице	srt	Физиологические параметры	кортеж int и float
Игроки в бейсбольной команде	str	Средняя результативность	float
Единицы измерения	str	Сокращения	str
Коды складского учета	str	Запасы на складе	int

Ключи словаря должны быть неизменяемыми (например, строки, числа и кортежи) и уникальными (то есть дубликаты запрещены). Разным ключам могут соответствовать одинаковые значения — например, двум кодам складского учета могут соответствовать одинаковые размеры складских запасов.

In [1]:

```
# создание словаря
country_codes = {'Finland': 'fi', 'South Africa': 'za', 'Nepal': 'np'}
empty_dict = {} # пустой словарь
```

In [2]:

```
# проверка пустого словаря
print(len(country_codes)) #  выведет количество пар "ключ-значение"
# пустой словарь интерпретируется как False
if country_codes:
    print("Записи есть, словарь не пуст.")
else:
    print("Словарь пуст")


```

```
3
Записи есть, словарь не пуст.
```

In [3]:

```
# перебор словаря
for key, value in country_codes.items():
    print(f"key: {key} - value: {value}")
```

```
key: Finland - value: fi
key: South Africa - value: za
key: Nepal - value: np
```

In [4]:

```
roman_numerals = {'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}
roman_numerals # неправильное значение у ключа 'X'
```

Out[4]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 100}
```

In [5]:

```
# необходимо исправить ошибку выше
print(roman_numerals['X']) # к значению словаря можно обратиться по его ключу
roman_numerals['X'] = 10 # замена значения производится тоже по ключу
roman_numerals
```

```
100
```

Out[5]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10}
```

In [6]:

```
# если попробовать добавить в словарь значение по несуществующему ключу, то можно получить новую запись в словаре
roman_numerals['L'] = 50
roman_numerals
```

Out[6]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10, 'L': 50}
```

In [7]:

```
# удаление записи из словаря производится по ключу через команду del
del roman_numerals['L']
roman_numerals
```

Out[7]:

```
{'I': 1, 'II': 2, 'III': 3, 'V': 5, 'X': 10}
```

In [8]:

```
# если теперь обратиться к несуществующему ключу, получим ошибку
roman_numerals['L']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[8], line 2
      1 # если теперь обратиться к несуществующему ключу, получим ошибку
----> 2 roman_numerals['L']

KeyError: 'L'
```

Чтобы определить, содержит ли словарь заданный ключ, можно воспользоваться операторами in и not in:

In [9]:

```
'V' in roman_numerals
```

Out[9]:

```
True
```

In [10]:

```
'LI' in roman_numerals
```

Out[10]:

```
False
```

Ранее метод items словарей использовался для перебора кортежей пар «ключ : значение», содержащихся в словаре. Похожие методы keys() и values() могут использоваться для перебора только ключей или значений соответственно:

In [12]:

```
months = {'January': 1, 'February': 2, 'March': 3}
for month_name in months.keys():
    print(month_name, end=' ')
print()
for month_number in months.values():
    print(month_number, end=' ')
```

```
January February March
1 2 3
```

Словари -- неупорядоченные структуры. Чтобы обработать ключи в порядке сортировки, используйте встроенную функцию sorted:

In [13]:

```
for month_name in sorted(months.keys()):
    print(month_name, end=' ')
```

```
February January March
```

Операторы сравнения == и != могут использоваться для проверки того, имеют ли два словаря идентичное (или разное) содержимое. Проверка равенства (==) дает результат True, если оба словаря содержат одинаковые пары «ключзначение» независимо от того, в каком порядке эти пары добавлялись в каждый словарь.

Для вставки пар «ключ-значение» можно использовать метод update словарей. Начнем с создания пустого словаря country_codes:

In [14]:

```
country_codes = {}
country_codes.update({'South Africa': 'za'})
country_codes
```

Out[14]:

```
{'South Africa': 'za'}
```

Множества

Множество представляет собой неупорядоченную коллекцию уникальных значений. Множества могут содержать только неизменяемые объекты: строки, int, float и кортежи, содержащие исключительно неизменяемые элементы. Хотя множества являются итерируемыми объектами, они не относятся к последовательностям и не поддерживают индексирование и сегментацию с квадратными скобками []. Словари также не поддерживают сегментацию.

Создание множества в фигурных скобках

Следующий код создает множество строк с именем colors:

In [1]:

```
colors = {'red', 'orange', 'yellow', 'green', 'red', 'blue'}
colors
```

Out[1]:

```
{'blue', 'green', 'orange', 'red', 'yellow'}
```

Обратите внимание: повторяющаяся строка 'red' была проигнорирована (без возникновения ошибки). Одно из важных областей применения множеств — удаление дубликатов, выполняемое автоматически при создании множества. Кроме того, значения полученного множества не выводятся в порядке их перечисления во фрагменте выше. Хотя названия цветов отображаются в порядке сортировки, множества не упорядочены. Ваш код не должен зависеть от порядка элементов.

Определение длины множества

Количество элементов в множестве можно определить при помощи встроенной функции len:

In [2]:

```
len(colors)
```

Out[2]:

```
5
```

Проверка наличия значения во множестве

Чтобы проверить, содержит ли множество конкретное значение, можно воспользоваться операторами in и not in:

In [3]:

```
'red' in colors
```

Out[3]:

```
True
```

In [4]:

```
'purple' in colors
```

Out[4]:

```
False
```

Перебор элементов множества

Множества являются итерируемыми объектами. Для перебора их элементов можно воспользоваться циклом for:

In [5]:

```
# поскольку множества не упорядочены, порядок перебора роли не играет
for color in colors:
    print(color.upper(), end=' ')
```

```
ORANGE GREEN RED YELLOW BLUE
```

Создание множества встроенной функцией set

Множество также можно создать на базе другой коллекции значений, используя встроенную функцию set, — здесь мы создаем список, содержащий несколько дубликатов целочисленных значений, и передаем этот список в аргументе set:

In [6]:

```
numbers = list(range(10)) + list(range(5))
numbers
```

Out[6]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4]
```

In [7]:

```
# трансформируем список во множество, удалив тем самым дубликаты
set(numbers)
```

Out[7]:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Сравнение множеств

Для сравнения множеств могут использоваться различные операторы и методы. Следующие множества содержат одинаковые значения, поэтому == возвращает True, а != возвращает False.

In [8]:

```
{1, 3, 5} == {3, 5, 1}
```

Out[8]:

```
True
```

In [9]:

```
{1, 3, 5} != {3, 5, 1}
```

Out[9]:

```
False
```

Оператор < проверяет, является ли множество в левой части подмножества строгим подмножеством множества в правой части, то есть все элементы левого операнда присутствуют в правом операнде и множества не равны:

In [10]:

```
{1, 3, 5} < {3, 5, 1}
```

Out[10]:

```
False
```

In [12]:

```
{1, 3, 5} == {3, 5, 1}
```

Out[12]:

```
True
```

In [11]:

```
{1, 3, 5} < {7, 3, 5, 1}
```

Out[11]:

```
True
```

Оператор <= проверяет, является ли множество в левой части нестрогим подмножеством множества в правой части, то есть все элементы левого операнда присутствуют в правом операнде и эти множества могут быть равны:

In [13]:

```
1 {1, 3, 5} <= {3, 5, 1}
```

Out[13]:

```
True
```

Для проверки нестрогого подмножества также можно воспользоваться методом issubset:

In [14]:

```
1 {1, 3, 5}.issubset({3, 5, 1})
```

Out[14]:

```
True
```

In [15]:

```
1 {1, 3, 5}.issubset({3, 5, 1})
```

Out[15]:

```
True
```

Математические операции с множествами

В этом разделе представлены основные математические операции множеств |, &, - и ^, а также соответствующие методы.

Объединение

Объединением двух множеств называется множество, состоящее из всех уникальных элементов обоих множеств. Для вычисления объединения можно воспользоваться оператором | или методом union типа множества:

In [16]:

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

Out[16]:

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

In [17]:

```
1 {1, 3, 5}.union([20, 20, 3, 40, 40])
```

Out[17]:

```
{1, 3, 5, 20, 40}
```

Оба операнда бинарных операторов множеств (таких как |) должны быть множествами. Соответствующие методы множеств могут получать в аргументе любой итерируемый объект — например, мы передавали список. Если математический метод множества получает аргумент — итерируемый объект, который не является множеством, то он сначала преобразует итерируемый объект в множество, а затем применяет математическую операцию. Еще раз: хотя в строковых представлениях новых множеств значения следуют по возрастанию, ваш код не должен зависеть от этого порядка.

Пересечение

Пересечением двух множеств является множество, состоящее из всех уникальных элементов, входящих в оба множества. Пересечение можно вычислить оператором & или методом intersection типа множества:

In [18]:

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

Out[18]:

```
{3}
```

In [19]:

```
1 {1, 3, 5}.intersection([1, 2, 2, 3, 3, 4, 4])
```

Out[19]:

```
{1, 3}
```

Разность

Разностью двух множеств является множество, состоящее из элементов левого операнда, не входящих в правый операнд. Разность можно вычислить оператором или методом difference типа множества:

In [20]:

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

Out[20]:

```
{1, 5}
```

In [21]:

```
{1, 3, 5}.difference({2, 3, 4})
```

Out[21]:

```
{1, 5}
```

Симметрическая разность

Симметрической разностью двух множеств является множество, состоящее из элементов каждого множества, не входящих в другое множество. Симметрическая разность вычисляется оператором ^ или методом symmetric_difference типа множества:

In [22]:

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

Out[22]:

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

In [23]:

```
{1, 3, 5, 7}.symmetric_difference([2, 2, 3, 3, 4, 4])
```

Out[23]:

```
{1, 2, 4, 5, 7}
```

Генераторы множеств

Трансформации множеств, как и трансформации словарей, определяются в фигурных скобках. Создадим новое множество, которое содержит только уникальные четные значения из списка numbers:

In [24]:

```
numbers = [1, 2, 2, 3, 4, 5, 6, 6, 7, 8, 9, 10, 10]
evens = {item for item in numbers if item % 2 == 0}
evens
```

Out[24]:

```
{2, 4, 6, 8, 10}
```

Задачи на закрпление

Битовое множество

Байт представляет собой последовательность из 8 бит. Можно представить себе реализацию небольшой структуры данных с использованием одного байта. Набор будет содержать не более 8 элементов из цифр от 0 до 7. Значение каждого бита в байте будет указывать, включен ли индекс этого бита в набор.

Рассмотрим следующий байт, где индекс каждого бита отмечен ниже.

Байт: 0 1 1 0 0 1 0 1
Индекс: 0 1 2 3 4 5 6 7

Этот байт будет представлять набор {1, 2, 5, 7}. Сходным образом,

10101010 ==> {0, 2, 4, 6}

11100000 ==> {0, 1, 2}

Ваша задача — написать функцию byte_to_set, которая принимает один байт (целое число от 0 до 255) и возвращает соответствующий набор.

In [ ]:

```

```

Число-дублетон

Мы будем называть натуральное число «дублетоном», если оно содержит ровно две различные цифры. Например, 23, 35, 100, 12121 являются числами-дублетонами, а 123 и 9980 — нет.

По заданному натуральному числу n (от 1 до 1 000 000) необходимо найти следующий дублетон. Если n само по себе является дублетоном, верните следующий элемент, больший, чем n.

Примеры:
doubleton(120) == 121
doubleton(1234) == 1311
doubleton(10) == 12

In [ ]:

```
1
```

Email-адреса

Данные об email-адресах студентов хранятся в словаре:

In [25]:

```
emails = {
    'mgu.edu': ['andrei_serov', 'alexander_pushkin', 'elena_belova', 'kirill_stepanov'],
    'gmail.com': ['alena.semyonova', 'ivan.polekhin', 'marina_abrabova'],
    'msu.edu': ['sergei.zharkov', 'julia_lyubimova', 'vitaliy.smirnoff'],
    'yandex.ru': ['ekaterina_ivanova', 'glebova_nastya'],
    'harvard.edu': ['john.doe', 'mark.zuckerberg', 'helen_hunt'],
    'mail.ru': ['roman.kolosov', 'ilya_gromov', 'masha.yashkina']
}
```

Нужно дополнить код таким образом, чтобы он вывел все адреса в алфавитном порядке и в формате имя_пользователя@домен.

In [ ]:

```
1
```