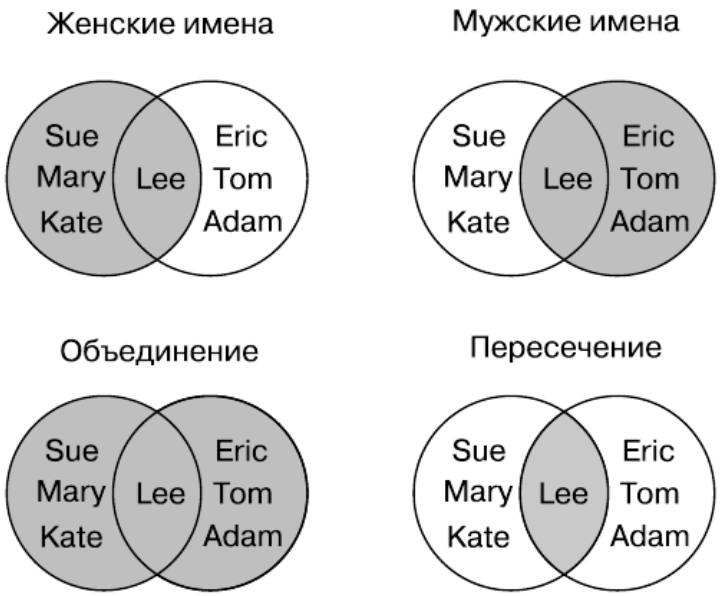

Рис. 4.1 – Базовые операции с множествами

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

Создание словаря с помощью {}

Чтобы создать словарь, вам нужно обернуть в фигурные скобки ({}) разделенные запятыми пары ключ : значение. Самым простым словарем является пустой словарь, не содержащий ни ключей, ни значений:

```
>>> empty_dict = {}
>>> empty_dict
{}
```

Создадим небольшой юмористический словарь:

```
>>> humorous_dict = {
        "Мыло-Варение" : "Особый джем из мыла",
        "Авто-Ручка" : "Ручка автомобиля",
        "МышеЛовка" : "Очень ловкая мышь",
        }
```

Ввод имени словаря в интерактивный интерпретатор выведет все его ключи и значения:

```
>>> humorous_dict
{'Мыло-Варение': 'Особый джем из мыла', 'Авто-Ручка': 'Ручка автомобиля',
 'МышеЛовка': 'Очень ловкая мышь'}
```

> ***Примечание:***
>
> *В Python допускается наличие запятой после последнего элемента списка, кортежа или словаря. Вам также не обязательно использовать выделение пробелами, как сделано в предыдущем примере, когда вы вводите ключи и значения внутри фигурных скобок. Это лишь улучшает читабельность.*

Преобразование с помощью функции dict()

Вы можете использовать функцию dict(), чтобы преобразовывать последовательности из двух значений в словари. (Вы иногда можете столкнуться с последовательностями «ключ — значение» вида «Стронций, 90, углерод, 14» или «Вавилон, 5, Глубокий космос, 9».) Первый элемент каждой последовательности применяется как ключ, а второй — как значение.

Для начала рассмотрим небольшой пример, использующий lol (список, содержащий списки, которые состоят из двух элементов):

```
>>> lol = [['a', 'b'], ['c', 'd'],['e', 'f']]
>>> dict(lol)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

> *Помните, что порядок ключей в словаре может быть произвольным, он зависит от того, как вы добавляете элементы.*

Мы могли бы использовать любую последовательность, содержащую последовательности, которые состоят из двух элементов. Рассмотрим остальные примеры.

Список, содержащий двухэлементные кортежи:

```
>>> lot = [('a', 'b'), ('c', 'd'),('e', 'f')]
>>> dict(lot)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Кортеж, включающий двухэлементные списки:

```
>>> tol = (['a', 'b'], ['c', 'd'],['e', 'f'])
>>> dict(tol)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Список, содержащий двухсимвольные строки:

```
>>> los = ['ab', 'cd', 'ef']
>>> dict(los)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Кортеж, содержащий двухсимвольные строки:

```
>>> tos = ('ab', 'cd', 'ef')
>>> dict(tos)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Добавление или изменение элемента с помощью конструкции [ключ]

Добавить элемент в словарь довольно легко. Нужно просто обратиться к элементу по его ключу и присвоить ему значение. Если ключ уже существует в словаре, имеющееся значение будет заменено новым. Если ключ новый, он и указанное значение будут добавлены в словарь. Здесь, в отличие от списков, вам не нужно волноваться о том, что Python сгенерирует исключение во время присваивания нового элемента, если вы укажете, что этот индекс находится вне существующего диапазона.

Создадим словарь, содержащий список известных студентов факультета Гриффиндор, используя их фамилии в качестве ключей, а имена — в качестве значений:

```
>>> gryffindor = {
        'Поттер' : 'Гарри',
        'Дамблдор' : 'Альбус',
        'Грейнджер' : 'Гермиона',
        'Люпин' : 'Римус',
        'Макгонагал' : 'Минерва',
        'Уизли' : 'Рон',
        }
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона',
'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Рон'}
```

Здесь не хватает Сириуса Блэка. Перед вами попытка анонимного программиста добавить его, однако он ошибся, когда вводил имя:

```
>>> gryffindor ['Блэк'] = 'Юпитер'
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона',
'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Рон', 'Блэк':
'Юпитер'}
```

А вот код другого программиста, который исправил эту ошибку:

```
>>> gryffindor ['Блэк'] = 'Сириус'
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона',
'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Рон', 'Блэк':
'Сириус'}
```

Используя один и тот же ключ ('Блэк'), мы заменили исходное значение 'Юпитер' на 'Сириус'.

Помните, что ключи в словаре должны быть уникальными. Если мы после Уизли Рона внесём Уизли Джинни, то победит последнее значение:

```
>>> gryffindor ['Уизли'] = 'Джинни'
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус'}
```

Сначала мы присвоили значение 'Рон' ключу 'Уизли', а затем заменили его на 'Джинни'.

Объединение словарей с помощью функции update()

Вы можете использовать функцию update(), чтобы скопировать ключи и значения из одного словаря в другой.

У нас есть словарь, в котором содержится перечень студентов Гриффиндора.

Кроме того, у нас есть другой словарь, содержащий ещё несколько имён, и называющийся others:

```
>>> others = {'Гриффиндор' : 'Годрик', 'Хагрид' : 'Рубеус'}
```

Теперь появляется еще один анонимный программист, который считает, что члены словаря others должны быть членами gryffindor:

```
>>> gryffindor.update(others)
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус', 'Гриффиндор': 'Годрик', 'Хагрид': 'Рубеус'}
```

Что произойдет, если во втором словаре будут находиться такие же ключи, что и в первом? Победит значение из второго словаря:

```
>>> first = {'a': 1, 'b': 2}
>>> second = {'b': 'platypus'}
>>> first.update(second)
>>> first
{'b': 'platypus', 'a': 1}
```

Удаление элементов по их ключу с помощью del

Код нашего анонимного программиста был корректным — технически. Но он не должен был его писать! Члены словаря others, несмотря на свою известность, не закончили факультет. Отменим добавление последних двух элементов:

```
>>> del gryffindor['Гриффиндор']
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус', 'Хагрид': 'Рубеус'}
>>> del gryffindor['Хагрид']
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус'}
```

Удаление всех элементов с помощью функции clear()

Чтобы удалить все ключи и значения из словаря, вам следует использовать функцию clear() или просто присвоить пустой словарь заданному имени:

```
>>> gryffindor.clear()
>>> gryffindor
{}
>>> gryffindor = {}
>>> gryffindor
{}
```

Проверяем на наличие ключа с помощью in

Если вы хотите узнать, содержится ли в словаре какой-то ключ, используйте ключевое слово in. Снова определим словарь pythons, но на этот раз опустим одного-двух участников. Затем проверим, кого мы добавили:

```
>>> gryffindor = {'Поттер': 'Гарри', 'Дамблдор': 'Альбус',
'Грейнджер': 'Гермиона'}
>>> 'Грейнджер' in gryffindor
True
>>> 'Поттер' in gryffindor
True
```

Вспомнили ли мы о Роне Уизли на этот раз?

```
>>> 'Уизли' in gryffindor
False
```

Получение элемента словаря с помощью конструкции [ключ]

Этот вариант использования словаря - самый распространенный. Вы указываете словарь и ключ, чтобы получить соответствующее значение:

```
>>> gryffindor['Грейнджер']
'Гермиона'
```

Если ключа в словаре нет, будет сгенерировано исключение:

```
>>> gryffindor['Блэк']
Traceback (most recent call last):
  File "<pyshell#271>", line 1, in <module>
    gryffindor['Блэк']
KeyError: 'Блэк'
```

Есть два хороших способа избежать возникновения этого исключения. Первый из них — проверить, имеется ли заданный ключ, с помощью ключевого слова in, что вы уже видели в предыдущем разделе:

```
>>> 'Блэк' in gryffindor
False
```

Второй способ — использовать специальную функцию словаря get(). Вы указываете словарь, ключ и опциональное значение. Если ключ существует, вы получите связанное с ним значение:

```
>>> gryffindor.get('Поттер')
'Гарри'
```

Если такого ключа нет, получите опциональное значение, если указывали его:

```
>>> gryffindor.get('Малфой', 'Не учился на факультете')
'Не учился на факультете'
```

В противном случае будет возвращен объект None (интерактивный интерпретатор не выведет ничего):

```
>>> gryffindor.get('Малфой')
```

Получение всех ключей с помощью функции keys()

Вы можете использовать функцию keys(), чтобы получить все ключи словаря. Для следующих примеров мы берем другой словарь:

```
>>> signals = {'Зеленый' : 'Вперёд!', 'Желтый' : 'Беги быстрее',
'Красный' : 'Стоп!'}
>>> signals.keys()
dict_keys(['Зеленый', 'Желтый', 'Красный'])
```

> *Примечание:*
>
> *В Python 2 функция keys() возвращает простой список. В Python 3 эта функция возвращает dict_keys() — итерабельное представление ключей. Это удобно для крупных словарей, поскольку не требует времени и памяти для создания и сохранения списка, которым вы, возможно, даже не воспользуетесь. Но зачастую вам нужен именно список. В Python 3 надо вызвать функцию list(), чтобы преобразовать dict_keys в список:*
>
> *>>>list(signals.keys())*
>
> *['Зеленый', 'Желтый, 'Красный']*
>
> *В Python 3 вам также понадобится использовать функцию list(), чтобы преобразовать результат работы функций values() и items() в обычные списки. Я пользуюсь этой функцией в своих примерах.*

Получение всех значений с помощью функции values()

Чтобы получить все значения словаря, используйте функцию values():

```
>>> list(signals.values())
['Вперёд!', 'Беги быстрее', 'Стоп!']
```

Получение всех пар «ключ — значение» с помощью функции items()

Когда вам нужно получить все пары «ключ — значение» из словаря, используйте функцию items():

```
>>> list(signals.items())
[('Зеленый', 'Вперёд!'), ('Желтый', 'Беги быстрее'), ('Красный',
'Стоп!')]
```

Каждая пара будет возвращена как кортеж вроде ('Зеленый', 'Вперёд!').

Присваиваем значения с помощью оператора =, копируем их с помощью функции copy()

Как и в случае со списками, если вам нужно внести в словарь изменение, оно отразится для всех имен, которые ссылаются на него.

```
>>> signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Сто
п!'}
>>> save_signals = signals
>>> signals['Голубой'] = 'Конфуз'
>>> save_signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Сто
п!', 'Голубой': 'Конфуз'}
```

Чтобы скопировать ключи и значения из одного словаря в другой и избежать этого, вы можете воспользоваться функцией copy():

```
>>> signals = {'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Крас
ный': 'Стоп!'}
>>> original_signals = signals.copy()
>>> signals['Голубой'] = 'Неожиданно'
>>> signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Стоп!',
'Голубой': 'Неожиданно'}
>>> original_signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Стоп!'}
```

Множества

Создание множества с помощью функции set()

Чтобы создать множество, вам следует использовать функцию set() или разместить в фигурных скобках одно или несколько разделенных запятыми значений, как показано здесь:

```
>>> empty_set = set()
>>> empty_set
set()
>>> even_numbers = {0,2,4,6,8}
>>> even_numbers
{0, 2, 4, 6, 8}
>>> odd_numbers = {1,3,5,7,9}
>>> odd_numbers
{1, 3, 5, 7, 9}
```

Как и в случае со словарем, порядок ключей в множестве не имеет значения.

Примечание:

> Поскольку пустые квадратные скобки [] создают пустой список, вы могли бы рассчитывать на то, что пустые фигурные скобки {} создают пустое множество. Вместо этого пустые фигурные скобки создают пустой словарь. Именно поэтому интерпретатор выводит пустое множество как set() вместо {}. Почему так происходит? Словари появились в Python раньше и успели захватить фигурные скобки в свое распоряжение.

Преобразование других типов данных с помощью функции set()

Вы можете создать множество из списка, строки, кортежа или словаря, потеряв все повторяющиеся значения.

Для начала взглянем на строку, которая содержит более чем одно включение некоторых букв:

```
>>> set('letters')
{'e', 't', 's', 'l', 'r'}
```

Обратите внимание на то, что множество содержит только одно включение букв «e» или «t», несмотря на то, что в слове letters по два включения каждой из них.

Создадим множество из списка:

```
>>> set(['Железный человек', 'Капитан Америка', 'Соколиный глаз', 'Халк'])
{'Халк', 'Капитан Америка', 'Железный человек', 'Соколиный глаз'}
```

А теперь из кортежа:

```
>>> set(('Флеш', 'Аквамен', 'Бетмен', 'Зелёная стрела'))
{'Зелёная стрела', 'Аквамен', 'Флеш', 'Бетмен'}
```

Когда вы передаете функции set() словарь, она возвращает только ключи:

```
>>> set({'Яблоко' : 'Зелёное', 'Апельсин' : 'Оранжевый', 'Грейпфрут' :
'Оранжевый'})
{'Яблоко', 'Грейпфрут', 'Апельсин'}
```

Проверяем на наличие значения с помощью ключевого слова in

Такое использование множеств самое распространенное. Мы создадим словарь, который называется recipes. Каждый ключ будет названием коктейля, а соответствующие значения — множествами ингредиентов:

```
recipes = {
    'Борщ' : {'Капуста', 'Картошка', 'Помидоры'},
    'Салат Весенний' : {'Помидоры', 'Огурцы', 'Сметана'},
    'Луковый суп' : {'Бульон', 'Лук'},
    'Омлет' : {'Яйца', 'Помидоры'},
    'Пельмени' : {'Фарш', 'Мука'},
    'Смузи' : {'Банан', 'Клубника', 'Йогурт'}
    }
```

Несмотря на то что и словарь, и множества окружены фигурными скобками ({ и }), множество — это всего лишь последовательность значений, а словарь — это набор пар «ключ — значение».

Какой из рецептов содержит помидоры? (Обратите внимание на то, что для выполнения этих проверок я заранее демонстрирую использование ключевых слов for, if, and и or, которые будут рассмотрены только в следующей работе.)

```
>>> for name, components in recipes.items():
        if 'Помидоры' in components:
            print(name)


Борщ
Салат Весенний
Омлет
```

Мы хотим съесть блюдо с помидорами, но не переносим лактозу, а на клубнику у нас аллергия:

```
>>> for name, components in recipes.items():
        if 'Помидоры' in components and not ('Сметана' in components
or 'Клубника' in components):
            print(name)


Борщ
Омлет
```

Перепишем этот пример чуть более сжато в следующем разделе.

Комбинации и операторы

Что, если вам нужно проверить наличие сразу нескольких значений множества?

Предположим, вы хотите найти любой рецепт, содержащий много крахмала (как в картофеле или бананах). Для этого мы используем оператор пересечения множеств (&):

```
>>> for name, components in recipes.items():
        if components & {'Картошка', 'Банан'}:
            print(name)


Борщ
Смузи
```

Результатом работы оператора & является множество, содержащее все элементы, которые находятся в обоих сравниваемых списках. Если ни один из заданных ингредиентов не содержится в предлагаемых рецептах, оператор & вернет пустое множество. Этот результат можно считать равным False.

Теперь перепишем пример из предыдущего раздела, в котором мы хотели помидоров, не смешанных со сметаной или клубникой:

```
>>> for name, components in recipes.items():
    if 'Помидоры' in components and not components &{'Сметана', 'Клубника'}:
        print(name)


Борщ
Омлет
```

Сохраним множества ингредиентов для двух рецептов в переменных, чтобы нам не пришлось набирать много текста в дальнейших примерах:

```
>>> Salad = recipes['Салат Весенний']
>>> Borsh = recipes['Борщ']
```

В следующих примерах демонстрируется использование операторов множеств. В одних из них демонстрируется применение особой пунктуации, в других — особых функций, в третьих — и того и другого. Мы будем использовать тестовые множества a (содержит элементы 1 и 2) и b (содержит элементы 2 и 3)

```
>>> a = {1, 2}
>>> b = {2, 3}
```

Пересечение множеств (члены обоих множеств) можно получить с помощью особого пунктуационного символа & или функции множества intersection(), как показано здесь

```
>>> a & b
{2}
>>> a.intersection(b)
{2}
```

В этом фрагменте используются сохраненные нами переменные:

```
>>> Salad & Borsh
{'Помидоры'}
```

В этом примере мы получаем объединение (члены обоих множеств), используя оператор | или функцию множества union():

```
>>> a | b
{1, 2, 3}
>>> a.union(b)
{1, 2, 3}
```

Обеденная версия:

```
>>> Salad | Borsh
{'Сметана', 'Огурцы', 'Капуста', 'Картошка', 'Помидоры'}
```

Разность множеств (члены только первого множества, но не второго) можно получить с помощью символа – или функции difference():

```
>>> a - b
{1}
>>> a.difference(b)
{1}
>>> Borsh - Salad
{'Капуста', 'Картошка'}
```

Самыми распространенными операциями с множествами являются объединение, пересечение и разность. Для полноты картины я включил в этот раздел и остальные операции, но вы, возможно, никогда не будете их использовать.

Для выполнения исключающего ИЛИ (элементы или первого, или второго множества, но не общие) используйте оператор ^ или функцию symmetric_difference():

```
>>> a ^ b
{1, 3}
>>> a.symmetric_difference(b)
{1, 3}
>>> Borsh ^ Salad
{'Сметана', 'Капуста', 'Огурцы', 'Картошка'}
```

В последнем примере показано, в чём разница между борщём и салатом ☺.

Вы можете проверить, является ли одно множество подмножеством другого (все члены первого множества являются членами второго), с помощью оператора <= или функции issubset():

```
>>> a = a.union(b)
>>> a
{1, 2, 3}
>>> a <= b
False
>>> b <= a
True
```

```
>>> a <= b
False
>>> a.issubset(b)
False
```

Является ли любое множество подмножеством самого себя? Ага.

```
>>> a <= a
True
>>> a.issubset(a)
True
```

Для того чтобы одно подмножество стало подмножеством второго, второе множество должно содержать все члены первого и несколько других. Определяется это с помощью оператора <:

```
>>> a < b
False
>>> a < a
False
>>> b < a
True
>>> Borsh < Salad
False
```

Множество множествпротивоположно подмножеству (все члены второго множества являются также членами первого). Для определения этого используется оператор >= или функция issuperset():

```
>>> a >= b
True
>>> a.issuperset(b)
True
```

Любое множество является множеством множеств самого себя:

```
>>> a >= a
True
>>> a.issuperset(a)
True
```

И наконец, вы можете найти собственное множество множеств (первое множество содержит все члены второго и несколько других) с помощью оператора >:

```
>>> b > a
False
>>> a > b
True
>>> Borsh > Salad
False
```

Множество не может быть собственным множеством множеств самого себя:

```
>>> a > a
False
```

Сравнение структур данных

Напомню, список создается с помощью квадратных скобок ([]), кортеж — с помощью запятых, а словарь — с помощью фигурных скобок ({}). Во всех случаях вы получаете доступ к отдельному элементу с помощью квадратных скобок:

```
>>> frend_list = ['Чендлер', 'Росс', 'Джо']
>>> frend_tuple = 'Чендлер', 'Росс', 'Джо'
>>> frend_dict = {'Бинк' : 'Чендлер', 'Геллер' : 'Росс', 'Трибиани' : 'Джо'}
>>> frend_list[2]
'Джо'
>>> frend_tuple[2]
'Джо'
>>> frend_dict['Трибиани']
'Джо'
```

Для списка и кортежа значение, находящееся в квадратных скобках, является целочисленным смещением. Для словаря же оно является ключом. Для всех троих результатом будет значение.

Создание крупных структур данных

Ранее мы работали с простыми булевыми значениями, числами и строками. Теперь же мы работаем со списками, кортежами, множествами и словарями. Вы можете объединить эти встроенные структуры данных в собственные структуры, более крупные и сложные. Начнем с трех разных списков:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> pythons = ['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin']
>>> stooges = ['Moe', 'Curly', 'Larry']
```

Мы можем создать кортеж, который содержит в качестве элементов каждый из этих списков:

```
>>> tuple_of_lists = friends, physicists, gryffindor
>>> tuple_of_lists
(['Чендлер', 'Росс', 'Джо'], ['Леонард', 'Шелдон', '
Радж', 'Говард'], ['Гарри', 'Рон', 'Гермиона'])
```

Можем также создать список, который содержит три списка:

```
>>> list_of_lists = [friends, physicists, gryffindor]
>>> list_of_lists
[['Чендлер', 'Росс', 'Джо'], ['Леонард', 'Шелдон', 'Радж',
'Говард'], ['Гарри', 'Рон', 'Гермиона']]
```

Наконец, создадим словарь из списков. В этом примере используем название группы комиков в качестве ключа, а список ее членов — в качестве значения:

```
>>> dict_of_lists = {'Друзья' : friends, 'Физики' :
physicists, 'Гриффиндорцы' : gryffindor}
>>> dict_of_lists
{'Друзья': ['Чендлер', 'Росс', 'Джо'], 'Физики': ['Л
еонард', 'Шелдон', 'Радж', 'Говард'], 'Гриффиндорцы'
: ['Гарри', 'Рон', 'Гермиона']}
```

Вас ограничивают только сами типы данных. Например, ключи словаря должны быть неизменяемыми, поэтому список, словарь или множество не могут быть ключом для другого словаря. Но кортеж может быть ключом. Например, вы можете создать алфавитный указатель достопримечательностей, основываясь на GPS-координатах (широте, долготе и высоте):

```
>>> houses = {
        (44.79, -93.14, 285): 'My House',
        (38.89, -77.03, 13): 'The White House'
        }
```

Упражнения

Вы познакомились с более сложными структурами данных: списками, кортежами, словарями и множествами. Используя их и типы данных, описанные в предыдущих работах (числа и строки), вы можете представить множество элементов реального мира.

1. Создайте список years_list, содержащий год, в который вы родились, и каждый последующий год вплоть до вашего пятого дня рождения. Например, если вы родились в 1980 году, список будет выглядеть так: years_list = [1980, 1981, 1982, 1983, 1984, 1985].
2. В какой из годов, содержащихся в списке years_list, был ваш третий день рождения? Помните, в первый год вам было 0 лет.
3. В какой из годов, перечисленных в списке years_list, вам было больше всего лет?
4. Создайте список things, содержащий три элемента: "mozzarella", "cinderella", "salmonella".
5. Напишите с большой буквы тот элемент списка things, который относится к человеку, а затем выведите список. Изменился ли элемент списка?
6. Переведите сырный элемент списка things в верхний регистр целиком и выведите список.
7. Удалите болезнь из списка things, получите Нобелевскую премию и затем выведите список на экран.
8. Создайте список, который называется surprise и содержит элементы 'Groucho', 'Chico' и 'Harpo'.
9. Напишите последний элемент списка surprise со строчной буквы, затем обратите его и напишите с прописной буквы.
10. Создайте англо-французский словарь, который называется e2f, и выведите его на экран. Вот ваши первые слова: dog/chien, cat/chat и walrus/morse.
11. Используя словарь e2f, выведите французский вариант слова walrus.
12. Создайте французско-английский словарь f2e на основе словаря e2f. Используйте метод items.
13. Используя словарь f2e, выведите английский вариант слова chien.
14. Создайте и выведите на экран множество английских слов из ключей словаря e2f.

15. Создайте многоуровневый словарь life. Используйте следующие строки для ключей верхнего уровня: 'животные', 'растения' и 'другое'. Сделайте так, чтобы ключ 'животные' ссылался на другой словарь, имеющий ключи 'котики', 'осьминоги' и 'эму'. Сделайте так, чтобы ключ 'котики' ссылался на список строк со значениями 'Оппенгеймер', 'Эйнштейн' и 'Лапусик'. Остальные ключи должны ссылаться на пустые словари.
16. Выведите на экран высокоуровневые ключи словаря life.
17. Выведите на экран ключи life['животные'].
18. Выведите значения life['животные']['котики'].

Контрольные вопросы

1. В чем отличие словарей от других структур данных в Python?
2. Как создать словарь?
3. Как создать словарь из кортежа? Из списка? Из списка двухсимвольных строк?
4. Как добавить данные в словарь?
5. Какие требования предъявляются значению ключа в словаре?
6. Что произойдет, если новый ключ совпадет с уже имеющимся?
7. Как объединить два словаря?
8. Как удалить элементы из словаря?
9. Как очистить словарь целиком?
10. Как получить элемент словаря по ключу?
11. Какая функция ищет в словаре заданный элемент и возвращает заданное значение, если элемент не найден?
12. Как получить ключи словаря?
13. Как получить значения словаря?
14. Как скопировать значения из одного словаря в другой и избежать их связывания?
15. В чём особенность множеств?
16. Как создать пустое множество?
17. Что такое пересечение множеств? Как его найти? Какой будет получен тип данных?
18. Что такое объединение множеств? Как его найти?
19. Что такое разность множеств? Как ее получить?
20. Что такое «исключающее ИЛИ»? Как его получить?
21. Как проверить, является ли одно множество подмножеством другого?
22. Как создать кортеж из списков?
23. Как создать список списков?
24. Как создать словарь из списков?
25. Какие структуры можно использовать в качестве ключа словаря?

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.