Структуры данных

Очень часто в программировании необходимо хранить набор значений одного или различных типов, с возможностью обращаться к этим значениям. Для этого используются коллекции.

Коллекции поддерживают функцию len() возвращающую количество элементов:

>> len([1, 2, 3])

3

и оператор in (True – если элемент содержится в коллекции, иначе – False):

>> 1 in [1, 2, 3]

True

>> 5 in [1, 2, 3]

False

Пустая коллекция интерпретируется как ложное условие (False), в то время как коллекция, состоящая хотя бы из одного элемента, интерпретируется как истинное (True):

>> bool([])

False

>> bool([1, 2, 3])

True

# Последовательности

Последовательности – это коллекции, к которой можно обращаться по индексу, т. е. у каждого элемента коллекции есть свой порядковый номер - индекс. В Python коллекции могут содержать элементы различных типов данных (int, bool, string и т. д.), в том числе другие коллекции.

Обращение по индексу производится с помощью оператора фигурные скобки ([]). В языке Python первый элемент в последовательности будет иметь индекс 0.

>> l = [1, 2, 3]

>> l[0]

1

>> l[1]

2

Ранее рассмотренные нами строки тоже являются последовательностями, элементами которых являются символы. Операторы, которые мы определили для строк (операторы сравнения (<, <=, ==, !=, >, >=) и +, \*) справедливы для всех последовательностей.

Мы рассмотрим два фундаментальных типа последовательностей – кортежи и списки.

## КОРТЕЖИ

Кортеж – это неизменяемая последовательность элементов. Когда мы рассматривали строки, мы уже использовали понятие «неизменяемый», это означает, что после того, как мы создали объект, его уже невозможно изменить (т. е. вставить, изменить или удалить какой-либо элемент).

Для создания кортежа можно использовать конструктор tuple() или перечислить элементы через запятую в круглых скобках:

# Создание пустого кортежа

t1 = tuple()  
t2 = ()

# Создание кортежа c элементами

t1 = tuple('мама', 1, True)  
t2 = ('мама', 1, True)

Если в кортеже 1 элемент, то при создании кортежа после него необходимо указать запятую:

# Создание кортежа из одного элемента

t = ('м',)

## СПИски

Список – это изменяемая последовательность элементов.

Для создания листа можно использовать конструктор list() или перечислить элементы через запятую в квадратных скобках:

# Создание пустого списка

l1 = list()  
l2 = []

# Создание списка c элементами

l1 = list('мама', 1, True)  
l2 = ['мама', 1, True]

Так как списки изменяемы, то для них определены следующие методы, непосредственно их изменяющие:

* l.append(x) – добавляет элемент x в конец списка;
* l.insert(i, x) – вставляет элемент x на позицию i;
* l.remove(x) – удаляет первый элемент из списка, имеющий значение x;
* l.pop(i) – удаляет i-ый элемент списка и возвращает его;
* l.index(i) – возвращает i-ый элемент списка;
* l.sort() – сортирует список.

Преобразовать одну последовательность к другой очень просто. Например, если мы хотим получить отсортированный кортеж, можно просто преобразовать его в список, с помощью конструктора list(), затем отсортировать с помощью метода sort(), и преобразовать обратно в кортеж, с помощью tuple():

>> t = (2, 1, 3)

>> t

(2, 1, 3)

>> l = list(t)

>> l

[2, 1, 3]

>> l.sort()

>> l

[1, 2, 3]

>> t = tuple(l)

>> t

(1, 2, 3)

# Словари и множества

## Словари

Словарь – неупорядоченная коллекция пар «ключ-значение». Понятие «неупорядоченность» означает, что к элементу в коллекции нельзя обратиться по индексу, т. к. элементы не расположены по порядку, один за другим. Доступ к значениям в словаре производится по «ключу». Это сравнимо с поиском по толковому словарю, где «ключ» — это слово, а «значение» — определение этого слова.

В качестве ключа можно использовать любой изученный нами тип данных, в качестве значения – что угодно, даже другой словарь.

Для создания словаря можно использовать конструктор dict() или перечислить пары «ключ-значение» через запятую в фигурных скобках:

# Создание пустого словаря

d = dict()  
d = {}

# Различные способы создания одного и того же словаря

d = dict({'name': 'Вася', 'age': 12})  
d = dict(name='Вася', age=12)  
d = dict([('name', 'Вася'), ('age', 12)])  
d = {'name': 'Вася', 'age': 12}

Для доступа к элементам используются фигурные скобки, в которых указывается ключ:

>> d = {'м': 1}

>> d['м']

1

Однако, если обратиться к несуществующему ключу (например, d['а']), вызовется исключение KeyError и работы программы завершится. Для безопасного доступа к словарю используется функция get():

>> d.get('а', 2)

2

Ключи в словаре являются уникальными, т. е. по одному и тому же ключу не могут храниться разные значения, поэтому если в словарь добавляется пара «ключ-значение» с ключом, который уже существует, то в результате происходит замена старого значения новым:

>> d = {'м':1}

>> d['м']

1

>> d['м'] = 2

>> d['м']

2

Некоторые методы словарей:

* d.get(key[, default]) – возвращает значение ключа или default если ключа нет в словаре;
* d.items() – возвращает пары «ключ-значение» словаря;
* d.keys() – возвращает ключи словаря;
* d.values() – возвращает значения словаря;
* d.pop(key[, default]) – удаляет ключ и возвращает значение или default.

## множества

Множество – это неупорядоченная коллекция элементов. Множество хранит в себе элементы также, как и математическое множество. Все элементы множества должны быть уникальны.

Для создания множества можно использовать конструктор set() или перечислить элементы через запятую в круглых фигурных скобках:

# Создание пустого множества

s1 = set()

# Создание множества c элементами

s1 = set('мама') # {'а', 'м'}  
s2 = {'м', 'а', 'м', 'а'} # {'а', 'м'}

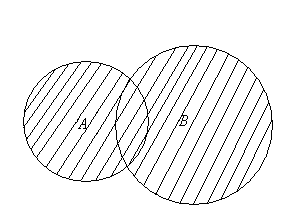
Как показано в последнем примере, мы можем преобразовать в множество последовательность. Таким образом, множество может использоваться для контроля уникальности элементов в последовательности. Множество можно привести обратно к последовательности с помощью конструкторов.

Множества поддерживают стандартные математические операции на множествами:

* Объединение:

>> set('мама') | set('папа')

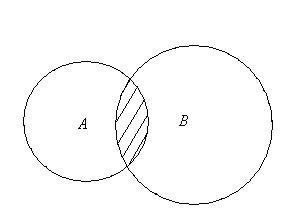
{'а', 'п', 'м'}

;

* Пересечение:

>> set('мама') & set('папа')

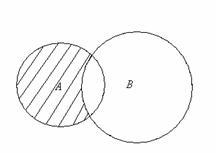
{'а'}

;

* Разность:

>> set('мама') - set('папа')

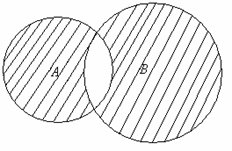
{'м'}

;

* Исключающее ИЛИ

>> set('мама') ^ set('папа')

{'п', 'м'}

.

Некоторые методы множеств:

* s.add(x) - добавляет элемент в множество;
* s.remove(x) - удаляет элемент из множества.