

Коллекции Python

Последовательности

The background features abstract, overlapping geometric shapes in various shades of green, ranging from light lime to dark forest green. The shapes are primarily triangles and polygons, creating a dynamic, layered effect. The overall composition is clean and modern, with the text 'Последовательности' centered on the left side.

Последовательности

- ▶ Последовательности - это один из типов данных, поддерживающих оператор проверки на входжение (`in`), функцию определения размера (`len()`), оператор извлечения срезов (`[]`) и возможность выполнения итераций. В языке Python имеется пять встроенных типов последовательностей: `bytearray`, `bytes`, `list`, `str` и `tuple`.

Кортежи

- ▶ Кортеж (tuple) - это упорядоченная последовательность из нуля или более ссылок на объекты.
- ▶ Кортежи поддерживают тот же синтаксис получения срезов, что и строки.
- ▶ Пустой кортеж создается с помощью пары пустых круглых скобок (), а кортеж, состоящий из одного или более элементов, может быть создан с помощью запятых.

```
>>> 1,2,3,99
```

```
(1, 2, 3, 99)
```

Методы кортежей

- ▶ `t.count(x)` - возвращает количество объектов `x` в кортеже `t`;
- ▶ `t.index(x)` - возвращает индекс самого первого (слева) вхождения объекта `x` в кортеж `t` или возбуждает исключение `ValueError`, если объект `x` отсутствует в кортеже.
- ▶ Кроме того, кортежи могут использоваться с оператором `+` (конкатенации), `*` (дублирования) и `[]` (получения среза), а операторы `in` и `not in` могут применяться для проверки на вхождение.
- ▶ Кортежи могут сравниваться с помощью стандартных операторов сравнения (`<`, `<=`, `==`, `!=`, `>=`, `>`), при этом сравнение производится поэлементно (и рекурсивно, при наличии вложенных элементов, таких как кортежи в кортежах).

СПИСКИ

Список

- ▶ Список (list) - это упорядоченная последовательность из нуля или более ссылок на объекты. Списки поддерживают тот же синтаксис получения срезов, что и строки с кортежами.

Методы списков

Синтаксис	Описание
L.append(x)	Добавляет элемент x в конец списка L
L.count(x)	Возвращает число вхождений элемента x в список L
L.extend(m) L += m	Добавляет в конец списка L все элементы итерируемого объекта m; оператор += делает то же самое
L.index(x, start, end)	Возвращает индекс самого первого (слева) вхождения элемента x в список L (или в срез start:end списка L), в противном случае возбуждает исключение ValueError
L.insert(i, x)	Вставляет элемент x в список L в позицию int i

Методы списков

Синтаксис	Описание
<code>L.pop()</code>	Удаляет самый последний элемент из списка <code>L</code> и возвращает его в качестве результата
<code>L.pop(i)</code>	Удаляет из списка <code>L</code> элемент с индексом <code>int i</code> и возвращает его в качестве результата
<code>L.remove(x)</code>	Удаляет самый первый (слева) найденный элемент <code>x</code> из списка <code>L</code> или возбуждает исключение <code>ValueError</code> , если элемент <code>x</code> не будет найден
<code>L.reverse()</code>	Переставляет в памяти элементы списка в обратном порядке
<code>L.sort(...)</code>	Сортирует список в памяти. Этот метод принимает те же необязательные аргументы <code>key</code> и <code>reverse</code> , что и встроенная функция <code>sorted()</code>

Генераторы списков

- ▶ Генератор списков - это выражение и цикл с дополнительным условием, заключенное в квадратные скобки, в котором цикл используется для создания элементов списка, а условие используется для исключения нежелательных элементов.
- ▶ В простейшем виде генератор списков записывается, как показано ниже:

```
[item for item in iterable]
```

Генераторы списков

- ▶ Генераторы могут использоваться как выражения и они допускают включение условной инструкции, вследствие чего мы получаем две типичные синтаксические конструкции использования генераторов списков:

```
[expression for item in iterable]
```

```
[expression for item in iterable if condition]
```

Примеры

```
>>> a = [i*10 for i in range(5)]
```

```
>>> a
```

```
[0, 10, 20, 30, 40]
```

```
>>> a = [x**2 for x in range(50) if x % 10 == 5]
```

```
>>> a
```

```
[25, 225, 625, 1225, 2025]
```

Множества



Множества

- ▶ Тип `set` - это неупорядоченная коллекция из нуля или более ссылок на объекты, указывающих на хешируемые объекты.
- ▶ Множества относятся к категории изменяемых типов, поэтому легко можно добавлять и удалять их элементы, но, так как они являются неупорядоченными коллекциями, к ним не применимо понятие индекса и не применима операция извлечения среза.
- ▶ Хешируемые объекты - это объекты, имеющие специальный метод `__hash__()`, на протяжении всего жизненного цикла объекта всегда возвращающий одно и то же значение, которые могут участвовать в операциях сравнения на равенство посредством специального метода `__eq__()`.
- ▶ Все встроенные неизменяемые типы данных, такие как `float`, `frozenset`, `int`, `str` и `tuple`, являются хешируемыми объектами и могут добавляться во множества.

Методы множеств

Синтаксис	Описание
<code>s.add(x)</code>	Добавляет элементы <code>x</code> во множество <code>s</code> , если они отсутствуют в <code>s</code>
<code>s.clear()</code>	Удаляет все элементы из множества <code>s</code>
<code>s.difference(t)</code> <code>s - t</code>	Возвращает новое множество, включающее элементы множества <code>s</code> , которые отсутствуют в множестве <code>t</code>
<code>s.discard(x)</code>	Удаляет элемент <code>x</code> из множества <code>s</code> , если он присутствует в множестве <code>s</code> ; смотрите также метод <code>set.remove()</code>
<code>s.intersection(t)</code> <code>s & t</code>	Возвращает новое множество, включающее элементы, присутствующие одновременно в множествах <code>s</code> и <code>t</code>
<code>s.isdisjoint(t)</code>	Возвращает <code>True</code> , если множества <code>s</code> и <code>t</code> не имеют общих элементов

Методы множеств

Синтаксис	Описание
<code>s.issubset(t)</code> <code>s <= t</code>	Возвращает True, если множество <code>s</code> эквивалентно множеству <code>t</code> или является его подмножеством; чтобы проверить, является ли множество <code>s</code> только подмножеством множества <code>t</code> , следует использовать проверку <code>s < t</code>
<code>s.issuperset(t)</code> <code>s >= t</code>	Возвращает True, если множество <code>s</code> эквивалентно множеству <code>t</code> или является его надмножеством; чтобы проверить, является ли множество <code>s</code> только надмножеством множества <code>t</code> , следует использовать проверку <code>s > t</code>
<code>s.pop()</code>	Возвращает и удаляет случайный элемент множества <code>s</code> или возбуждает исключение <code>KeyError</code> , если <code>s</code> - это пустое множество

Методы множеств

Синтаксис	Описание
<code>s.remove(x)</code>	Удаляет элемент <code>x</code> из множества <code>s</code> или возбуждает исключение <code>KeyError</code> , если элемент <code>x</code> отсутствует в множестве <code>s</code> ; смотрите также метод <code>set.discard()</code>
<code>s.union(t)</code> <code>s t</code>	Возвращает новое множество, включающее все элементы множества <code>s</code> и все элементы множества <code>t</code> , отсутствующие в множестве <code>s</code>
<code>s.update(t)</code> <code>s = t</code>	Добавляет во множество <code>s</code> все элементы множества <code>t</code> , отсутствующие в множестве <code>s</code>

Генераторы множеств

- ▶ Генератор множества - это выражение и цикл с необязательным условием, заключенные в фигурные скобки.
- ▶ Подобно генераторам списков, генераторы множеств поддерживают две формы записи:

```
{expression for item in iterable}
```

```
{expression for item in iterable if condition}
```

Пример

```
>>> m = {x for x in range(20) if x % 2 == 0}
```

```
>>> m
```

```
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18}
```