Генераторы списков

Методы заполнения массивов, использующие цикл, работают в большинстве языков программирования. Однако в Python есть особые возможности, которые позволяют решать многие задачи, используя более компактную запись.

Например, две операции – создание и заполнение массива – можно объединить в одну с помощью **генератора списков** – выражения, напоминающего цикл. Например, вот синтаксис будет у генератора, который заполняет список квадратами чисел от 1 до 10:

```
arr = [i*i for i in range(1, 11)]
```

Генератор позволяет заменить цикл при считывании данных с клавиатуры или использовании функции `map`. Вот такой генератор заменит конструкцию `list(map(int, input().split()))`:

```
arr = [int(x) for x in input().split()]
```

Если элементы списка задаются по одному в строке, то их можно считать в список следующим генератором:

```
arr = [int(input()) for i in range(n)]
```

1 Синтаксис генератора списков

Основной синтаксис генератора списков выглядит следующим образом:

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

Рассмотрим каждый элемент этого выражения по отдельности:

1. `expression` – выражение, которое будет вычислено для каждого элемента списка
2. `item` – переменная, которая будет использоваться для итерации по элементам `iterable`

3. `iterable` – итерируемый объект (например, список, кортеж, множество, словарь, строка, range и т.д.)
4. `condition` – необязательное условие. Выполнение выражения будет происходить только если условие истинно.

Для каждого элемента `iterable`, для которого выполнено условие `condition` (если оно указано), генератор подставляет его в выражение `expression` в виде `item`; вычисляя это выражение для каждого `item`, генератор составляет из них список.

Рассмотрим работу генератора списка на примере. Пусть у нас есть `b` – список натуральных чисел. Мы хотим создать новый список из двузначных чисел исходного списка. Это можно сделать следующим образом:

```
b = [8, 93, 56, 72, 14, 1001, 4]
a = [x for x in b if 9 < x < 100]
print(a)
```

```
[93, 56, 72, 14]
```

2 enumerate

Иногда для решения задач нужно хранить не только элементы списка, но и их индексы. Для этой цели можно использовать специальный объект `enumerate`, который принимает на вход обязательный параметр – последовательность, которую он нумерует, и параметр старта (по умолчанию он равен 0). Возвращаемый объект этой функции – список кортежей из двух значений – номера и элемента последовательности:

```
st = 'banana apple pear'
arr = [x for x in enumerate(st.split(), start = 1)]
print(arr)
```

```
[(1, 'banana'), (2, 'apple'), (3, 'pear')]
```

3 Модуль random

Для тестирования решения задач часто приходится создавать большие списки. Делать это вручную неудобно. Воспользуемся генератором создания списков и рандомизируем его. Подключим библиотеку `random`, которая содержит функции, генерирующие случайные объекты. Подключить библиотеку можно командой `import`:

```
import random
```

Рассмотрим на примерах, как пользоваться некоторыми функциями модуля `random`. Допустим, мы хотим сгенерировать список из целых чисел в диапазоне от -100 до 100. Для этого используем функцию `random.randint(a, b)`, которая при каждом обращении генерирует случайное число от `a` до `b` включительно:

```
arr = [random.randint(-100, 100) for i in range(5)]
```

Или, например, мы хотим получить простейший генератор паролей из 9 символов. Сначала создадим список из символов, которые могут входить в пароль (в нашем примере это будут прописные буквы английского алфавита). Затем, с помощью функции `random.sample` запросим из этого списка подсписок длины 9, преобразовав этот список в строку при помощи знакомого нам метода `join`:

```
simb = [chr(x) for x in range(65, 91)]  # список из прописных букв
                                        # латинского алфавита
arr = [''.join(random.sample(simb, 9)) for i in range(10)]
print(arr)
```

```
GKDEUTHFP
```

4 Несколько последовательностей в генераторе

Генератор может использовать несколько последовательностей для создания списка.

Рассмотрим следующий пример. В программировании встречаются задачи, где объект перемещается по правилам хода шахматного коня – на две клетки в одном направлении, и на одну клетку в другом. Допустим, нам нужно пройти по всем координатам, достижимым из текущей точки (`x`, `y`) (например, один из ходов будет в клетку (`x + 1`, `y - 2`)).

Для решения этой задачи удобно создать список из изменений координат. После этого пройти циклом по списку изменений и прибавлять значения к текущей координате. Сгенерируем список автоматически. Изменения каждой координате могут быть на -2, -1, на 1 и на 2. Но не все комбинации дают ход коня, например комбинация (-2, -2) недопустимая. Посмотрим на существующие изменения и заметим, что приращения координат никогда не бывают одинаковой четности. Напишем все эти условия в одном генераторе со вложенными циклами по списку изменений координат. Каждый элемент нашего нового списка — это кортеж из двух элементов:

```
pr = [-2, -1, 1, 2]
h = [(x, y) for x in pr for y in pr if x % 2 != y % 2]
print(h)
```

```
[(-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)]
```

В нашем генераторе присутствуют вложенные циклы – переменные `x` и `y` каждая принимают все значение из `pr`, во всех возможных комбинациях; условие `if x % 2 != y % 2` отсеивает неподходящие.

Полученный список `h` можно использовать, чтобы получить все координаты, достижимые из заданной клетки, например клетки (3, 5):

```
px, py = 3, 5
for x, y in h:
    print(px + x, py + y)
```

```
[(1, 4), (1, 6), (2, 3), (2, 7), 4, 3), (4, 7), (5, 4), (5, 6)]
```

В этом примере мы идем по списку `h` и каждый элемент списка (кортеж из двух элементов) сразу помещаем в переменные `x` и `y`.

5 Создание двумерных списков

В некоторых задачах необходимо создать двумерный список (список, каждый элемент которого – тоже список), заполненный одинаковыми значениями, например нулями. При работе с одномерными списками мы использовали умножение одномерного списка на число. Что будет, если использовать тот же приём для создания двумерного списка? Попробуем написать и запустить следующий код:

```
a = [[0] * 4] * 3
print(a)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Визуально кажется, что мы получили двумерный список, заполненный нулями, т.е. решили поставленную задачу. Но не всё так просто! Давайте изменим нулевой элемент первой строки и второй элемент второй строки и снова выведем список:

```
a[0][0] = 8
a[1][2] = 7
print(a)
```

```
[[8, 0, 7, 0], [8, 0, 7, 0], [8, 0, 7, 0]]
```

Мы видим, что значения нулевого и второго элемента изменились у каждого вложенного списка! При умножении сложного объекта (списка из 4 нулей) на число 3 этот список не копировался; вместо этого у нас создался список из трех ссылок на один и тот же объект, одномерный список из четырех нулей. В нашем примере мы сначала обратились к нулевому элементу этого списка по одной ссылке, а потом — ко второму элементу того же списка по другой ссылке.

Как узнать что элементы `a[0]` и `a[1]` – это ссылки на один и тот же объект? Для этого в Python есть специальный оператор `is`, который проверяет, находятся ли два значения или две переменные по одному адресу в памяти. В нашем случае `a[0]` is `a[1]` вернет значение `True`:

```
print(a[0] is a[1])
```

```
True
```

Для создания полноценного двумерного списка нужно обратиться к другим методам. Например, использовать генераторы.

5.1 Генератор двумерных списков

Генератор двумерного списка создаёт одномерный список как элемент другого одномерного списка. Такая конструкция называется вложенные генераторы и напоминает использование вложенных циклов:

```
a = [[0 for j in range(4)] for i in range(3)]
```

Внешний генератор формирует список из 3 элементов, на каждой итерации он будет вызывать внутренний генератор, который сформирует список из 4 нулей. Эту конструкцию можно немного сократить, если использовать корректную операцию умножения списка из числа на число (т.к. число – простой объект, то при умножении оно копируется):

```
b = [[0] * 4  for i in range(3)]
```

Теперь, двумерный список сохдан корректно – каждый его элемент – отдельный список:

```
a[0][0] = 1
b[0][0] = 4
```

```
[[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
[[4, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Используя вложенные генераторы, можно создавать и более интересные варианты двумерных списков. Создадим таблицу умножения чисел от 1 до 9. Внешний цикл будет отвечать за формирование элементов-списков, поэтому вся содержательная часть будет находится во внутреннем генераторе:

```
a = [[i * j for j in range(1, 10)] for i in range(1, 10)]
```

Индекс `i` определяется внешним циклом, он нумерует строки нашей таблицы, а индекс `j` нумерует элементы внутри строки (часто это называют нумерацией столбца в таблице). Перемножение индексов строки и столбца и дадут нам нужный элемент таблицы умножения.

5.2 Считываение двумерного списка с клавиатуры

Использовать генератор возможно и для считывания двумерного списка с клавиатуры. Например, рассмотрим ситуацию, когда нам на вход подается `n` строк, в каждой из которых целые числа, разделенные пробелами. Количество чисел в разных строках может быть различным.

Наша задача сводится к тому, чтобы взять генератор считывания одномерного списка целых чисел и поместить его внутрь генератора, который создает список из `n` элементов:

```
a = [[int(x) for x in input().split()] for i in range(n)]
```

На следующем занятии мы подробнее остановимся на способах вывода и обработки двумерных списков.