Генераторы списков (List comprehensions)

Мы продолжаем курс по языку программирования Python. На этом занятии мы с вами познакомимся с еще одной довольно полезной и популярной конструкцией – генератором списков. По-английски это записывается как:

List comprehension

Для лучшего понимания, давайте я начну сразу с конкретного примера. Предположим, мы хотели бы сформировать список из квадратов целых чисел от 0 до N. Используя наши текущие знания, очевидно, это можно было бы сделать, следующим образом:

```
N = 6
a = [0] * N

for i in range(N):
    a[i] = i ** 2

print(a)
```

Но это не лучший вариант. Все то же самое можно реализовать буквально в одну строчку:

```
a = [x ** 2 for x in range(N)]
```

И, кроме того, этот вариант будет работать быстрее предыдущей программы, т.к. Python оптимизирует работу таких конструкций.

Давайте теперь разберемся в этом синтаксисе. Вначале мы указываем, что будем делать с переменной x. Казалось бы, переменная нигде не задана, а мы уже говорим что с ней делать. Да, это так, это такой элемент синтаксиса list comprehensions. Далее, через пробел мы записываем цикл for и уже там указываем эту переменную x и говорим как она будет меняться. То есть, эта временная переменная x существует только внутри списка и пропадает после его создания.

Такой подход и называется генератором списков. Вначале указываем, как формируются значения списка, а затем, описываем изменение параметра x через ключевое слово for. Причем, здесь можно указывать только его. К примеру, ключевое слово while прописывать нельзя.

Итак, для генерации списков следует придерживаться следующего синтаксиса (определения):

[<способ формирования значения> for <переменная> in <итерируемый объект>]

В самом простом варианте мы можем сформировать список и так:

```
a = [1 for x in range(N)]
```

Хотя, в таких случаях, когда все значения одинаковы, обычно, используют оператор *: `b = [1] * N`

Но, в отличие от простого дублирования с помощью List comprehension можно формировать более сложные последовательности, например, состоящих из остатков от деления на 4: `a = [x % 4 for x in range(N)]`

То есть, мы можем использовать любые доступные нам операторы для формирования текущего значения. Если, например, взять оператор определения четного и нечетного значений: `a = [x % 2 == 0 for x in range(N)`

то получим список из булевых величин True и False. Или же сформировать значения линейной функции: `a = [0.5*x+1 for x in range(N)]`

И так далее. Кроме того, мы здесь можем вызывать и функции. Пусть в программе пользователь вводит несколько целых чисел через пробел:

```
d_inp = input("Целые числа через пробел")
```

Тогда с помощью генератора списков мы легко можем преобразовать их в набор чисел: `a = [int(d) for d in d_inp.split()]`

Здесь конструкция d_inp.split() возвращает список строк из чисел, а функция int(d) преобразовывает каждую строку в целое число. Если убрать функцию int() и записать просто d: `a = [d for d in d_inp.split()]`

то у нас получится уже список из строк, то есть, функция int() действительно делает нужное нам преобразование.

Работа этой реализации очень похожа на функцию map(), которую мы с вами использовали в подобных задачах: `d_inp = list(map(int, input("Целы")))`

(О функции map() подробнее мы еще будем говорить).

Вообще, в генераторе списков можно использовать любые итерируемые объекты, в том числе и строки. Например, вполне допустима такая реализация:

```
a = [d for d in "python"]
```

получим список из отдельных символов. Или, можно каждый символ превратить в его код: `a = [ord(d) for d in "python"]`

Имеем список из соответствующих кодов. Если же взять список из слов:

`t = ["Я", "б", "Python", "выучил"]` то генератор списка:

```
a = [d for d in t]
```

сформирует точно такой же список.

Условия в генераторах списков

В генераторах дополнительно можно прописывать условия. Например, выберем все числа меньше нуля: `a = [x for x in range(-5, 5) if x < 0]`

Мы здесь после оператора for записали необязательный оператор if. В результате, в список будут попадать только отрицательные значения x. Или, можно сделать проверку на нечетность: `a = [x for x in range(-5, 5) if x%2]`

Получаем только нечетные числа из диапазона [-5; 5). Также можно прописывать составные условия, например:

```
a = [x for x in range(-6, 7) if x % 2 == 0 and x % 3 == 0]
```

Выберем все числа, которые кратны 2 и 3 одновременно. Или, такой пример, выберем из списка городов только те, длина которых меньше семи:

```
cities = ["Москва", "Тверь", "Рязань'']
 a = [city for city in cities if len(city) < 7]
```

Ну и последнее, что я хочу отметить на этом занятии – это возможность использования тернарного условного оператора внутри генераторов списков. Так как первым элементом мы можем прописывать любые конструкции языка Python, то кто нам мешает сделать следующее. Пусть имеется список чисел:

```
d = [4, 3, -5, 0, 2, 11, 122, -8,]
```

и мы хотим преобразовать его в последовательность со словами «четное» и «нечетное». Сделать это можно как раз с помощью тернарного условного оператора: `a = ["четное" if x % 2 == 0 else 'не четное' for x in d]`

Обратите еще раз внимание, что этот оператор не является какой-то особой частью синтаксиса генератора списка. Мы здесь всего лишь используем его как обычный оператор языка Python. Это такой же оператор как сложение, умножение и другие: `a = [x + 2 for x in d]`

которые мы также можем использовать при формировании списков.