Элементы функционального программирования

Технологии и языки программирования

Юдинцев В. В.

Кафедра теоретической механики
Самарский университет

20 октября 2017 г.

Содержание

Функциональное программирование

Анонимная (лямбда) функция

- Короткие однострочные функции могут быть объявлены при помощи `lambda`-функций

```
1 mean = lambda x,y: (x + y)*0.5
```

Вызов функции

```
mean(1,6)
3.5
```

- Лямбда-функция обычно используется в паре с функциями `filter`, `map`, `sort`

Функция `filter`

Применение лямбда-функции с функцией `filter`

```
>> data = {1, 7, 3, 2, 6, 8}

>> filter(lambda x: x % 2 == 0, data)

{2, 6, 8}
```

Функция `filter`

Применение лямбда-функции с функцией `filter`

```
data = {1, 7, 3, 2, 6, 8}

def less_than_5(x): return x < 5

>> filter(less_than_5, data)
{1, 3, 2}
```

Функция `filter`

Применение лямбда-функции с функцией `filter`

```
data = {1, 7, 3, 2, 6, 8}

def less_than_5(x): return x < 5

>> filter(less_than_5, data)
{1, 3, 2}
```

но лучше:

```
def less_than(value): return lambda x: x < value

>> filter(less_than(5), data)
{1, 3, 2}
```

или совсем просто:

```
>> filter(lambda x: x < 5, data)
```

Функция map

Функция `map(function, data)` применяет `function` к каждому элементу последовательности `data`:

```
data = [4,5,6,7]

>> map(lambda x: x**2, data)
<builtins.map at 0xa8d36bcc>
```

Функция `map` – ленивая функция. Функция возвращает не весь обработанный список, а ссылку на функцию-генератор, которая вычисляет значения по мере необходимости, подобно функции `range`:

```
>> list(map(lambda x: x**2, data))
[16, 25, 36, 49]
```

Функция map

Функцию map можно использовать для обработки нескольких списков:

```
1 vec1 = [1,4,3,7]
2
3 vec2 = [8,5,1,3]
4
5 prod = map(lambda x,y: x*y ,vec1,vec2)
```

```
>> scalar_product = sum(prod)
52
```

Списочные выражения и генераторы

Списочные выражения

Вместо функции `map`

```
1 data = [10, 12, 13, 14]
2
3 >> list(map(lambda x: x**2, data))
4 [16, 25, 36, 49]
```

можно использовать списочные выражения

```
1 res = [x**2 for x in data]
2 print(res)
3
4 [16, 25, 36, 49]
```

Дополнительные условия

Выражение формирует список квадратов только чётных элементов из `range(10)`:

```
1 [ x**2 for x in range(10) if x % 2 == 0 ]
2
3 [0, 4, 16, 36, 64]
```

Эту же последовательность можно сгенерировать при помощи функций `map` и `filter`:

```
1 list( map( lambda x: x**2,
2       filter(lambda x: x%2==0, range(10)) ) )
3
4 [0, 4, 16, 36, 64]
```

Выражения с вложенными циклами

Списочные выражения могут содержать несколько вложенных циклов:

```
1 [ x+'–'+y for x in ('A','B') for y in ('1', '2')]
2
3 ['A–1', 'A–2', 'B–1', 'B–2']
```

Функции-генераторы

- При помощи ключевого слова `yield` создаются функции, “лениво” генерирующие последовательности, подобно функции `range`
- В определении такой функции `return` заменяется на `yield`:

```
1 def progression(a1, d, n = 5):
2     # an = a1 + d (n-1)
3     for i in range(n):
4         yield a1+d*i
```

```
>> progression(1, 3)
<generator object progression at 0xa8dc68c4>

>> list(progression(1, 3))
[1, 4, 7, 10, 13]
```

Функции-генераторы

- Функция-генератор чисел Фибоначчи

```
1 def fib(count):
2     n0, n1 = 0, 1
3     for i in range(count):
4         n0, n1 = n1, n0+n1
5         yield n0
```

- Вызов функции `fib(10)` вернёт ссылку на эту функцию, которая будет работать как итератор, а само выполнение функции “приостановится” на операторе `yield`.

Функции-генераторы

- Функция-генератор чисел Фибоначчи

```
1 def fib(count):
2     n0, n1 = 0, 1
3     for i in range(count):
4         n0, n1 = n1, n0+n1
5         yield n0
```

- Все элементы можно получить создав список по этому итератору

```
1 >> list( fib(10) )
2
3 [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Функции-генераторы

- Функция-генератор чисел Фибоначчи

```
1 def fib(count):
2     n0, n1 = 0, 1
3     for i in range(count):
4         n0, n1 = n1, n0+n1
5         yield n0
```

- Последовательно получить элементы можно при помощи функции `next`:

```
1 a = fib(10)
2 >> next(a)
3 1
4 >> next(a)
5 1
6 >> next(a)
7 2
```

Функции-генераторы

- Функция-генератор чисел Фибоначчи

```
1 def fib(count):
2     n0, n1 = 0, 1
3     for i in range(count):
4         n0, n1 = n1, n0+n1
5         yield n0
```

- `yeld` возвращает не результат работы функции а объект типа итератор, для получения результата, поэтому этот итератор можно использовать внутри конструкции `for`:

```
6 for i in fib(10):
7     print(i,end=', ')
8
9 1,1,2,3,5,8,13,21,34,55,
```

Пример

```
1 text = "Feci quod potui, faciant meliora potentes"
```

Формирование множества букв предложения:

```
2 letters_set = set(text)
3 letters_set.remove(' ')
4 letters_set.remove(',')
```

Пример

```
1 text = "Feci quod potui, faciant meliora potentes"
```

Формирование множества букв предложения:

```
2 letters_set = set(text)
3 letters_set.remove(' ')
4 letters_set.remove(',')
```

Формирование списка пар (буква, частота):

```
6 res = [ ( x, text.count(x) ) for x in letters_set]
```

Пример

```
1 text = "Feci quod potui, faciant meliora potentes"
```

Формирование множества букв предложения:

```
2 letters_set = set(text)
3 letters_set.remove(' ')
4 letters_set.remove(',')
```

Формирование списка пар (буква, частота):

```
6 res = [ ( x, text.count(x) ) for x in letters_set]
```

Сортировка по второму элементу пары:

```
7 res.sort(key = lambda pair: pair[1], reverse = True)
```

Вывод первых пяти наиболее встречающихся букв:

```
8 print(res[:5])
```

```
[('o', 4), ('t', 4), ('e', 4), ('i', 4), ('a', 3)]
```

Списочные выражения и генераторы

- Списочное выражение формирует сразу весь список (`list`):

```
1 res = [ (x, text.count(x)) for x in letters_set]
```

```
>> type(res)
list
```

- Это выражение создаёт итератор, который будет вычислять следующее значение по требованию:

```
1 res = ( (x, text.count(x)) for x in letters_set] )
```

```
>> type(res)
iterator
```

Итератор можно пройти только один раз!

Функция `zip`

Функция `zip(i1, i2, ...)` объединяет итераторы-аргументы и создаёт итератор по кортежам элементов аргументов:

```
1 word = "Qapla"
2 for i, j in zip( range(len(word)), word ):
3     print(i+1,"буква",j)
```

```
1 буква Q
2 буква a
3 буква p
4 буква l
5 буква a
```

Если аргументы-итераторы (последовательности) разной длины, то результирующий итератор будет иметь длину минимального аргумента.

Задание