Тема 12. Элементы функционального программирования

Функциональным называется такой подход к процессу программирования, в котором, в отличие от процедурного программирования, где функции являются подпрограммами, программа основана на вычислении функций в их математическом понимании. Функциональное программирование является одной из парадигм, поддерживаемых языком Python.

12.1. Lambda-функция

Функция в Python может быть определена при помощи оператора **def** (его мы уже рассматривали выше) или при помощи **lambda**-выражения. **Lambda-выражение** в Python представляет собой анонимную функцию, т. е. функцию, у которой не задано имя. **Lambda-функция** может принимать любое количество аргументов, но обрабатывается в ней только одно выражение.

Общий вид **lambda-функции** следующий

lambda *<аргумент1>, <аргумент 2>, ... , <аргумент n> : <выражение>*

```
sum = lambda a, b: a + b
print(sum(2, 3))
```

```
5
```

В примере выше объявлена простая **lambda-функция**, вычисляющая сумму двух чисел. Это очень простой пример, в таком случае нет смысла использовать **lambda-функцию**. Как правило, их используют совместно с другими функциями, такими как **map()**, **reduce()**, **filter()**, принимающими функции в качестве одного из аргументов, когда нужно обработать итерируемые объекты.

12.2. Функция map

Функция **map()** принимает два аргумента: функцию и итерируемый объект, к которому эту функцию нужно применить. Во время выполнения функция, указанная в качестве аргумента, будет применена ко всем элементам итерируемой последовательности.

Пример 1

Дан список чисел, но все числа заданы в строковом типе данных. Нужно получить точно такой же список, но все числа в нем должны храниться уже в числовом формате.

Можно перебрать все элементы при помощи цикла и преобразовать тип:

```
str_list = ['1', '2', '3', '4', '5']
int_list = []
for i in str_list:
    int_list.append(int(i))
print(int_list)
```

```
[1, 2, 3, 4, 5]
```

А можно эту же задачу решить проще при помощи функции **map():**

```
str_list = ['1', '2', '3', '4', '5']
int_list = list(map(int, str_list))
print(int_list)
```

```
[1, 2, 3, 4, 5]
```

Пример 2

Дан список чисел. Нужно удвоить все элементы списка.

Можно перебрать все элементы при помощи цикла и умножить их на 2:

```
int_list = [1, 2, 3, 4, 5]
for i in range(len(int_list)):
    int_list[i] *= 2
print(int_list)
```

```
[2, 4, 6, 8, 10]
```

Можно написать пользовательскую функцию для удвоения числа и применить ее при помощи функции **map()** ко всему списку:

```
def redouble(item):
    return item*2
int_list = [1, 2, 3, 4, 5]
int_list = list(map(redouble, int_list))
print(int_list)
```

```
[2, 4, 6, 8, 10]
```

Можно решить эту задачу используя **lambda-функцию** для удвоения числа и применить ее при помощи функции **map()** ко всему списку:

```
int_list = [1, 2, 3, 4, 5]
int_list = list(map(lambda a: a * 2, int_list))
print(int_list)
```

```
[2, 4, 6, 8, 10]
```

Как видно, используя **lambda-функцию**, получаем наиболее компактное решение.

Пример 3

Даны 2 списка чисел. Нужно получить новый список, состоящий из элементов, равных произведению соответствующих элементов первоначальных списков. Решим эту задачу при помощи функции **map()** и **lambda-функции**:

```
list1 = [1, 2, 3, 4, 5]
list2 = [2, 3, 4, 5, 6]
list3 = list(map(lambda a,b: a*b, list1, list2))
print(list3)
```

```
[2, 6, 12, 20, 30]
```

Решение задачи получилось достаточно компактным.

12.3. Функция reduce

Функция **reduce()** принимает два аргумента: функцию и итерируемую последовательность. Передаваемая в качестве аргумента функция последовательно применяется ко всем элементам указанной последовательности и возвращает одно значение. В Python 3 для использования данной функции нужно подключать модуль **functools**.

Пример:

Дан список чисел. Нужно найти минимальный элемент, не используя функцию **min()**. Решим эту задачу при помощи функции **reduce()** и **lambda-функции**:

```
from functools import reduce

list1 = [12, 8, 31, 40, 5, 16, 7, 18]
min_item = reduce(lambda a, b: a if (a < b) else b, list1)
print(min_item)
```

```
5
```

12.4. Функция filter

Функция **filter()** используется для фильтрации элементов (отбор элементов, удовлетворяющих определенному правилу), принимает два аргумента: функцию и итерируемую последовательность, которую нужно отфильтровать. Важным моментом является то, что функция, передаваемая в качестве аргумента, должна возвращать **True** или **False**.

Пример:

Дан список чисел. Нужно получить новый список, состоящий из четных элементов данного списка. Решим эту задачу при помощи функции **filter()** и **lambda-функции**:

```
list1 = [1, 2, 3, 4, 5, 6, 7, 8]
list2 = list(filter(lambda a: a % 2 == 0, list1))
print(list2)
```

```
[2, 4, 6, 8]
```