15 Functional Programming

"Map it, reduce it, filter it......"

Contents

Functional Programming

- In functional programming a problem is treated as evaluation of one or more functions.

- Hence a given problem is decomposed into a set of functions. These functions provide the main source of logic in the program.

Functions as First Class Values

- Python facilitates functional programming by treating functions as 'first-class' data values. This means that:
  - Functions can be assigned to variables and then called using these variables.
  - Functions can be passed as arguments to function and returned from function.
  - Functions can be built at execution time, the way lists, tuples, etc. can be.

- Example of assigning a function to a variable and calling the function using the variable:

```
def func( ) :
    print('Hello')

def sum(x, y) :
    print(x + y)

f = func              # assignment of function to a variable
f( )                  # call to func( )
g = sum               # assignment of function to a variable
g(10, 20)             # call to sum( )
```

- Example of passing a function as argument to a function:

```
def sum(x, y, f) :
    print(x + y)
    f( )              # calls func( )

def func( ) :
    print('Hello')
```

```
f = func           # assignment of function to a variable
sum(10, 20, f)     # pass function as argument to a function
```

- Example of building function at execution time is discussed in the next section on lambda functions.

Lambda Functions

- Normal functions have names. They are defined using the **def** keyword.

- Lambda functions do not have names. They are defined using the **lambda** keyword and are built at execution time.

- Lambda functions are commonly used for short functions that are convenient to define at the point where they are called.

- Lambda functions are also called anonymous functions or inline functions.

- A lambda function can take any number of arguments but can return only one value. Its syntax is:

  lambda arguments : expression

  : separates the parameters to be passed to the lambda function and the function body. The result of running the function body is returned implicitly.

- A few examples of lambda functions

  ```
  #  function that receives an argument and returns  its cube
  lambda  n : n * n * n

  #  function that receives 3 arguments and returns average of them
  lambda x, y, z : (x + y + z) / 3

  #  function that receives a string, strips any whitespace and returns
  # the uppercase version of the string
  lambda s : s.trim( ).upper( )
  ```

- Lambda functions are often used as an argument to other functions. For example, the above lambdas can be passed to **print( )** function to print the value that they return.

```
print((lambda  n : n * n * n)(3))                      # prints 27
print((lambda  x, y, z : (x + y + z) / 3)(10, 20, 30))   # prints 20.0
```

```
print((lambda  s : s.lstrip( ).rstrip( ).upper( ))('   Ngp   '))  # prints NGP
```

- The lambda can also be assigned to a variable and then invoked.

```
p = lambda  n : n * n * n
q = lambda  x, y, z : (x + y + z) / 3
r = lambda  s : s.lstrip( ).rstrip( ).upper( )
print(p(3))                        # calls first lambda function
print(q(10, 20, 30))               # calls second lambda function
print(r('   Nagpur   '))           # calls third lambda function
```

- Container types can also be passed to a lambda function. For example, a lambda function that calculates average of numbers in a list can be passed to **print( )** function:

```
lst1 = [1, 2, 3, 4, 5]
lst2 = [10, 20, 30, 40, 50]
print((lambda l : sum(l) / len(l)) (lst1))
print((lambda l : sum(l) / len(l)) (lst2))
```

Here instead of assigning a lambda function to a variable and then passing the variable to **print( )**, we have passed the lambda function itself to **print( )**.

Higher Order Functions

- A higher order function is a function that can receive other functions as arguments or return them.

- For example, we can pass a lambda function to the built-in **sorted( )** function to sort a dictionary by values.

```
d = {'Oil' : 230, 'Clip' : 150, 'Stud' : 175, 'Nut' : 35}
# lambda takes a dictionary item and returns a value
d1 = sorted(d.items( ), key = lambda kv : kv[1])
print(d1)  # prints [('Nut', 35), ('Clip', 150), ('Stud', 175), ('Oil', 230)]
```

The **sorted( )** function uses a parameter **key**. It specifies a function of one argument that is used to extract a comparison for each element in the first argument of **sorted( )**. The default value of key is **None**, indicating that the elements in first argument are to be compared directly.

- To facilitate functional programming Python provides 3 higher order functions—**map( )**, **filter( )** and **reduce( )**. Before we see how to use these functions, we need to understand the map, filter and reduce operations.

Map, Filter, Reduce

- A map operation applies a function to each element in the sequence like list, tuple, etc. and returns a new sequence containing the results. For example:
  - Finding square root of all numbers in the list and returning a list of these roots.
  - Converting all characters in the list to uppercase and returning the uppercase characters' list.

- A filter operation applies a function to all the elements of a sequence. A sequence of those elements for which the function returns True is returned. For example:
  - Checking whether each element in a list is an alphabet and returning a list of alphabets.
  - Checking whether each element in a list is odd and returning a list of odd numbers.

- A reduce operation performs a rolling computation to sequential pairs of values in a sequence and returns the result. For example:
  - Obtaining product of a list of integers and returning the product.
  - Concatenating all strings in a list and returning the final string.

- Usually, map, filter, reduce operations mentioned above would need a **for** loop and/or **if** statement to control the flow while iterating over elements of sequence types like strings, lists, tuples.

- If we use Python functions **map( )**, **filter( )**, **reduce( )** we do not need a **for** loop or **if** statement to control the flow. This lets the programmer focus on the actual computation rather than on the details of loops, branches, and control flow.

map( ) Function

- Use of **map( )** function:

```
import math
def fun(n) :
    return n * n

lst = [5, 10, 15, 20, 25]
m1 = map(math.radians, lst)
m2 = map(math.factorial, lst)
m3 = map(fun, lst)
print(list(m1))          # prints list of radians of all values in lst
print(list(m2))          # prints list of factorial of all values in lst
print(list(m3))          # prints list of squares of all values in lst
```

- General form of **map( )** function is

  map(function_to_apply, list_of_inputs)

  map( ) returns a **map** object which can be converted to a list using **list( )** function.

filter( ) Function

- Use of **filter( )** function:

```
def fun(n) :
    if n % 5 == 0 :
        return True
    else :
        return False

lst1 = ['A', 'X', 'Y', '3', 'M', '4', 'D']
f1 = filter(str.isalpha, lst1)
print(list(f1))                # prints ['A', 'X', 'Y', 'M', 'D']

lst2 = [5, 10, 18, 27, 25]
f2 = filter(fun, lst2)
print(list(f2))                # prints  [5, 10, 25]
```

- General form of **filter( )** function is:

  filter(function_to_apply, list_of_inputs)

  filter( ) returns a **filter** object which can be converted to a list using **list( )** function.

reduce( ) Function

- Use of **reduce( )** function:

```
from functools import reduce
```

```
def getsum(x, y) :
    return x + y

def getprod(x, y) :
    return x * y

lst = [1, 2, 3, 4, 5]
s = reduce(getsum, lst)
p = reduce(getprod, lst)
print(s)                  # prints 15
print(p)                  # prints 120
```

Here the result of addition of previous two elements is added to the next element, till the end of the list. In our program this translates into operations like ((((1 + 2) + 3) + 4) + 5) and ((((1 * 2) * 3) * 4) * 5).

- General form of **reduce( )** function is:

  reduce(function_to_apply, list_of_inputs)

  The **reduce( )** function operation performs a rolling computation to sequential pairs of values in a sequence and returns the result.

- You can observe that **map( )**, **filter( )** and **reduce( )** abstract away control flow code.

Using Lambda with *map( ), filter( ), reduce( )*

- We can use **map( )**, **filter( )** and **reduce( )** with lambda functions to simplify the implementation of functions that operate over sequence types like, strings, lists and tuples.

- Since **map( )**, **filter( )** and **reduce( )** expect a function to be passed to them, we can also pass lambda functions to them, as shown below.

```
# using lambda with map( )
lst1 = [5, 10, 15, 20, 25]
m = map(lambda n : n * n, lst1)
print(list(m))                # prints [25, 100, 225, 400, 625]

# using lambda with filter( )
lst2 = [5, 10, 18, 27, 25]
f = filter(lambda n : n % 5 == 0, lst2)
print(list(f))                # prints [5, 10, 25]

# using lambda with reduce( )
```

```
from functools import reduce
lst3 = [1, 2, 3, 4, 5]
s = reduce(lambda x, y : x + y, lst3)
p = reduce(lambda x, y : x * y, lst3)
print(s, p)        # prints 15  120
```

- If required **map( )**, **filter( )** and **reduce( )** can be used together.

```
def fun(n) :
   return n > 1000

lst = [10, 20, 30, 40, 50]
l = filter(fun, map(lambda x : x * x, lst))
print(list(l))
```

- Here **map( )** and **filter( )** are used together. **map( )** obtains a list of square of all elements in a list. **filter( )** then filters out only those squares which are bigger than 1000.

Where are they Useful?

- Relational databases use the map/filter/reduce paradigm. A typical SQL query to obtain the maximum salary that a skilled worker gets from an Employees table will be:

```
SELECT max(salary) FROM Employees WHERE grade = 'Skilled'
```

The same query can be written in terms of **map( )**, **filter( )** and **reduce( )** as:

```
reduce(max, map(get_salary, filter(lambda x : x.grade( ) ==
               'Skilled', employees)))
```

Here employees is a sequence, i.e. a list of lists, where each list has the data for one employee

grade = 'Skilled' is a filter

get_salary is a map which returns the salary field from the list

and max is a reduce

In SQL terminology map, filter and reduce are called project, select and aggregate respectively.

- If we can manage our program using map, filter, and reduce, and lambda functions then we can run each operation in separate threads and/or different processors and still get the same results. Multithreading is discussed in detail in Chapter 25.

P</> Programs

Problem 15.1

Define three functions **fun( )**, **disp( )** and **msg( )**, store them in a list and call them one by one in a loop.

Program

```
def fun( ) :
    print('In fun')

def disp( ) :
    print('In disp')

def msg( ) :
    print('In msg')

lst = [fun, disp, msg]
for f in lst :
    f( )
```

Output

```
In fun
In disp
In msg
```

Problem 15.2

Suppose there are two lists, one containing numbers from 1 to 6, and other containing umbers from 6 to 1. Write a program to obtain a list that contains elements obtained by adding corresponding elements of the two lists.

Program

```
lst1 = [1, 2, 3, 4, 5, 6]
lst2 = [6, 5, 4, 3, 2, 1]
result = map(lambda n1, n2: n1 + n2, lst1, lst2)
print(list(result))
```

Output

```
[7, 7, 7, 7, 7, 7]
```

Tips

- lambda function receives two numbers and returns their sum.
- **map( )** function applies lambda function to each pair of elements from **lst1** and **lst2**.
- The **map( )** function returns a **map** object which is then converted into a list using **list( )** before printing.

Problem 15.3

Write a program to create a new list by obtaining square of all numbers in a list.

Program

```
lst1 = [5, 7, 9, -3, 4, 2, 6]
lst2 = list(map(lambda n : n ** 2, lst1))
print(lst2)
```

Output

```
[25, 49, 81, 9, 16, 4, 36]
```

Tips

- lambda function receives a number and returns its square.
- **map( )** function applies lambda function to each element from **lst1**.

- The **map( )** function returns a **map** object which is then converted into a list using **list( )** before printing.

Problem 15.4

Though **map( )** function is available ready-made in Python, can you define one yourself and test it?

Program

```
def my_map(fun, seq) :
    result = [ ]
    for ele in seq :
        result.append(fun(ele))
    return result
lst1 = [5, 7, 9, -3, 4, 2, 6]
lst2 = list(my_map(lambda n : n ** 2, lst1))
print(lst2)
```

Output

```
[25, 49, 81, 9, 16, 4, 36]
```

Tips

- lambda function receives a number and returns its square.
- **my_map( )** function applies lambda function to each element from **lst1**.
- The **my_map( )** function returns a **map** object which is then converted into a list using **list( )** before printing.

Problem 15.5

Following data shows names, ages and marks of students in a class:

Anil, 21, 80
Sohail, 20, 90
Sunil, 20, 91
Shobha, 18, 93
Anil, 19, 85

Write a program to sort this data on multiple keys in the order name, age and marks.

Program

```
import operator
lst = [('Anil', 21, 80), ('Sohail', 20, 90), ('Sunil', 20, 91),
       ('Shobha', 18, 93), ('Anil', 19, 85), ('Shobha', 20, 92)]
print(sorted(lst, key = operator.itemgetter(0, 1, 2)))
print(sorted(lst, key = lambda tpl : (tpl[0], tpl[1], tpl[2])))
```

Output

```
[('Anil', 19, 85), ('Anil', 21, 80), ('Shobha', 18, 93), ('Shobha', 20, 92),
('Sohail', 20, 90), ('Sunil', 20, 91)]
[('Anil', 19, 85), ('Anil', 21, 80), ('Shobha', 18, 93), ('Shobha', 20, 92),
('Sohail', 20, 90), ('Sunil', 20, 91)]
```

Tips

- Since there are multiple data items about a student, they have been put into a tuple.
- Since there are multiple students, all tuples have been put in a list.
- Two sorting methods have been used. In the first method **itemgetter( )** specifies the sorting order. In the second method a lambda has been used to specify the sorting order.

Problem 15.6

Suppose a dictionary contain key-value pairs, where key is an alphabet and value is a number. Write a program that obtains the maximum and minimum values from the dictionary.

Program

```
d = {'x' : 500, 'y' : 5874, 'z' : 560}

key_max = max(d.keys( ), key = (lambda k: d[k]))
key_min = min(d.keys( ), key = (lambda k: d[k]))
```

```
print('Maximum Value: ', d[key_max])
print('Minimum Value: ', d[key_min])
```

Output

```
Maximum Value:  5874
Minimum Value:  500
```

Exercises

[A] State whether the following statements are True or False:

(a) lambda function cannot be used with **reduce( )** function.

(b) lambda, **map( )**, **filter( )**, **reduce( )** can be combined in one single expression.

(c) Though functions can be assigned to variables, they cannot be called using these variables.

(d) Functions can be passed as arguments to function and returned from function.

(e) Functions can be built at execution time, the way lists, tuples, etc. can be.

(f) Lambda functions are always nameless.

[B] Using lambda, **map( )**, **filter( )** and **reduce( )** or a combination thereof to perform the following tasks:

(a) Suppose a dictionary contains type of pet (cat, dog, etc.), name of pet and age of pet. Write a program that obtains the sum of all dog's ages.

(b) Consider the following list:

lst = [1.25, 3.22, 4.68, 10.95, 32.55, 12.54]

The numbers in the list represent radii of circles. Write a program to obtain a list of areas of these circles rounded off to two decimal places.

(c) Consider the following lists:

nums = [10, 20, 30, 40, 50, 60, 70, 80]

strs = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

Write a program to obtain a list of tuples, where each tuple contains a number from one list and a string from another, in the same order in which they appear in the original lists.

(d) Suppose a dictionary contains names of students and marks obtained by them in an examination. Write a program to obtain a list of students who obtained more than 40 marks in the examination.

(e) Consider the following list:

lst = ['Malayalam', 'Drawing', 'madamlamadam', '1234321']

Write a program to print those strings which are palindromes.

(f) A list contains names of employees. Write a program to filter out those names whose length is more than 8 characters.

(g) A dictionary contains following information about 5 employees:

First name
Last name
Age
Grade (Skilled, Semi-skilled, Highly-skilled)

Write a program to obtain a list of employees (first name + last name) who are Highly-skilled.

(h) Consider the following list:

lst = ['Benevolent', 'Dictator', 'For', 'Life']

Write a program to obtain a string 'Benevolent Dictator For Life'.

(i) Consider the following list of students in a class.

lst = ['Rahul', 'Priya', 'Chaaya', 'Narendra', 'Prashant']

Write a program to obtain a list in which all the names are converted to uppercase.