BA-BEAD Big Data Engineering for Analytics

BEAD Workshop Series

Functional Thinking in Python

Table of Contents

Introduction

This set of instructions would guide you through the functional thinking workshop using Python and PyCharm IDE.

Functional programming

Functional programming is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing state and mutable data. ~ As Quoted by Wikipedia.

Python provides features like lambda, filter, map, and reduce that can easily demonstrate the concept of Functional Programming. In this workshop, students will practice creating functions using these basic features.

Functions Vs Methods

***Functions** are objects, methods are not. Functions are named, reusable expressions.*
***Methods** are associated with a class, functions are not.*

The terms **method** and object are often used interchangeably because methods can be stored in objects (class) quite easily.

General Syntax

```
class ClassName:
      def.method_name()
      . . .
```

For example:

```
class Car:
   def getAreaMethod(radius):
     PI = 3.14;
     area = PI * radius * radius
     print(area)
car = Car
car.getAreaMethod(10.0)
```

Output : 17

Member in summary has the below properties:

1. Methods are not objects, but can be converted to function objects quite easily.
2. Methods have signature, but no independent type.
3. Slightly faster and better performing.
4. Methods are not first class entities unless converted to functions.
5. Methods work fine with type parameters and parameter default values.
6. A method in python is somewhat similar to a function, except it is associated with object/classes.
7. Methods in python are very similar to functions except for two major differences.
    a. The method is implicitly used for an object for which it is called.
    b. The method is accessible to data that is contained within the class.

Function objects are first class entities on par with classes - methods are not. A function is a block of code to carry out a specific task, will contain its own scope and is called by name. All functions may contain zero(no) arguments or more than one argument. On exit, a function can or cannot return one or more values.

General Syntax

```
def functionName( arg1, arg2,….):
    …….
    # Function_body
    ……..
```

For example:

```
import math
def areaofcircle(radius):
    return math.pi * radius * radius
print(areaofcircle(10.0))
```

Output: 314.1592653589793

OR using lambda

```
import math
circlearea = lambda radius : math.pi * radius * radius
print(circlearea(10.0))
```

Output: 314.1592653589793

Functions in summary has the below properties:

1. Functions are value types; can be stored in val and var storage units.
2. Functions are objects of type function0, function1, ...
3. Functions have both a type and a signature (type is function0,function1, ..).
4. Slightly slower, and higher overhead.
5. Functions are first class entities on par with classes.
6. Functions can accept type parameters or parameter default values or partial functions.

Statements are units of code that do not return a value. Expressions are units of code that return a value. The last expression in a block is the return value for the entire block. Functions are named, reusable expressions. Expressions can be stored in values or variables and passed into functions.

Simple Examples

In this section participants will learn how to how to use the lambda, map, filter and reduce functions in Python to transform data structures.

The Lambda Expression

Lambda expressions, also known as “anonymous functions”, allow us to create and use a function in a single line. They are useful when we need a short function that will be used only once. They are mostly used in conjunction with the map, filter and the sort methods. Let’s write a function in Python, that computes the value of 5x + 2. The standard approach would be to define a function.

For example, the standard way to write a function for 5X+2

```
def f(x):
  return 5 * x + 2
print(f(3))
```

Output : 17

However, there's a quicker way to write functions on the fly, and these are called lambda functions. A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression. To create a lambda expression, we type in the keyword lambda, followed by the inputs. Next, we enter a colon, followed by the expression that will be the return value.

General Syntax

```
lambda arguments : expression
```

For example, if we rewrite the same standard expression above as lambda:

```
 lambda x: 5*x+2
```

There is a small issue in the above definition, lambda is not the name of the function. So developers usually like to set a name for the functions crafted.

One way is to give it a name. Let us call this lambda expression s. Now, we can use this like any other function.

```
s = lambda x: 5*x+2
s(3)
```

Output :17

Another example:

```
x = lambda a : a + 10
print(x(5))
```

Output: 15

The lambda function is much more powerful and concise because we can also construct anonymous functions; functions without a name, example:

```
(lambda x, y: x * y)(9, 10) #returns 90
```

Let us look at a situation where we craft a lambda expression with multiple inputs. We will use Harmonic Mean to illustrate this situation. The harmonic mean *H* of the positive real numbers {x1, x2, …. xn} is defined to be:

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \left(\frac{\sum_{i=1}^{n} x_i^{-1}}{n} \right)^{-1}.$$

Typically, Harmonic Mean is appropriate for situations when the average of rates is desired. Let us simplify the equation into three parameters for starters.

```
# Calculating Harmonic Mean using lambda function
harmonic_mean = lambda x,y,z : 3/(1/x + 1/y + 1/z)**0.5
harmonic_mean(1,2,3)
```

Output: 2.215646837627989

Let us also look at the last scenario where we craft lambda expression without inputs. Let's look at a common use of Lambda function where we do not assign it a name. Let's say we have a list of presidents of the Republic of Singapore and we'd like to sort this list by their last name. We shall create a Lambda function that extracts the last name, and uses that as the sorting value.

```
# Sorting a List by the last name using lambda expression
presidents_sg = ["Yusof Ishak", "Benjamin Sheares", "Devan Nair", "Wee
Kim Wee", "Ong Teng Cheong", "Sellappan Nathan", "Tony Tan", "Halimah
Yacob"]
presidents_sg.sort(key = lambda name: name.split(" ")[-1].lower())
print(presidents_sg)
```

*Output: ['Ong Teng **Cheong**', 'Yusof **Ishak**', 'Devan **Nair**', 'Sellappan **Nathan**', 'Benjamin **Sheares**', 'Tony **Tan**', 'Wee Kim **Wee**', 'Halimah **Yacob**']*

The Map Function

The map function applies a function to every item of iterable, yielding the results. When used with lists, Map transforms a given list into a new list by applying the function to all the items in an input_list.

General Syntax

```
map(function_to_apply, iterables)
```

As mentioned in the above syntax, the map function takes two arguments, a function and a sequence such as a list and applies the function over all the elements of the sequence. We can pass lambda function to the map without even naming them, and in this case, we refer to them as anonymous functions.

In the below example, we use map() on the lambda function, which squares all elements of the list, and we store the result in square_all.

```
# original integer list
nums = [48, 6, 9, 21, 1]
square_all = map(lambda num: num ** 2, nums)
```

```
# print the list using join and map to convert integers to str
print(' '.join(map(str, square_all)))
```

Output: 2304 36 81 441 1

Suppose we have a function that computes the volume of a cube, given the value of its edge(a). What if we need to compute the volumes for many different cubes with different edge lengths?

```
def volume(a):
    """volumne of a cube with edge 'a'"""
    return a**3
# Edge, say length in cm
edges = [1,2,3,4,5]
# calculate and print answers
print(list(map(volume,edges)))
```

Output: [1, 8, 27, 64, 125]

Let's now see an example which demonstrates the use of lambda function with the map function. We have a list of tuples containing name and heights for 5 people. Each of the height is in centimetres and we need to convert it into feet.

We will first write a converter function using a lambda expression which will accept a tuple as the input and will return a tuple with the same name.

```
# Convert height from cms to feet : 1 cm = 0.0328 feet
height_in_cms = [('Po Ping',183), ('Oogway',161), ('Shifu',130),
('Tigress',179), ('Viper',190), ('Monkey',165), ('Crane',165),
('Mantis',45)]
#Writing Convertor function using lambda expression
height_in_feet = lambda data: (data[0],round(data[1]*0.0328,1))
#Using the 'Map' function
print(list(map(height_in_feet,height_in_cms)))
```

Output: [('Po Ping', 6.0), ('Oogway', 5.3), ('Shifu', 4.3), ('Tigress', 5.9), ('Viper', 6.2), ('Monkey', 5.4), ('Crane', 5.4), ('Mantis', 1.5)]

The Filter Function

The filter function constructs an iterator from those elements of iterable for which function returns true. This means filter function is used to select certain pieces of data from a list, tuple, or other collection of data, hence the name.

General Syntax

```
filter(filter(function, iterable)
```

Let's look at an example where we want to get the list of all numbers that are greater than 5, from a given input list.

```
# Filter out all the numbers greater than 4 from a list
some_list = [1,2,3,4,5,6,7,8,9]
output_list = filter(lambda x : x>4, some_list)
print(list(output_list))
```

Output: [6, 7, 8, 9]

Let us consider another filter example in connection to data cleansing. Consider is a list containing some of the countries in Asia. You also observe that there are numerous empty strings. We'll use the filter function to remove these missing values. We'll simply pass none as the first argument and the second argument is the list of data as before.

```
# Removing missing values from a list
countries_asia = ["Bhutan", "", "","China","Myanmar","", "India",
"Indonesia","", "Philippines", "Singapore"]
print(list(filter(None,countries_asia)))
```

Output: ['Bhutan', 'China', 'Myanmar', 'India', 'Indonesia', 'Philippines', 'Singapore']

This filters out all values that are treated as false in a boolean setting.

The Reduce Function

The Reduce function is a bit unusual and in fact, as of Python 3, it is no longer a built-in function. Instead, it has been moved to the **`functools`** module. The **`'reduce'`** function transforms a given list into a single value by applying a function cumulatively to the items of sequence, from left to right.

General Syntax

```
reduce(func, seq)
```

where reduce continually applies the function func() to the sequence seq and returns a single value. Let's illustrate the working of the reduce function with the help of a simple example that computes the product of a list of integers.

```
# importing functools for reduce()
import functools
# Compute the product of a list of integers using
#'reduce'function from functools import reduce
product = functools.reduce((lambda x,y : x*y), [1,2,3,4,5])
print(product)
```

Output: 120

The reduce function can determine the maximum of a list containing integers in a single line of code. There does exist a built-in function called max() in Python which is normally used for this purpose as max(list_name).

```
# Determining the maximum number in a given list
from functools import reduce
f = lambda a,b : a if (a>b) else b
print(reduce(f,[58,698,12,158,6,79,82]))
```

Output: 698

Exercise

1. Write a function that computes the area of a circle given its radius.
2. Write a function calculation() such that it can accept two variables and calculate the addition and subtraction of it. And also it must return both addition and subtraction in a single return call
3. Write a recursive function that prints the product of all values from an array of integer numbers, without using `for` or `while` loops. Can you make it tail-recursive? *(Hint use Sum Example as inspiration)*
4. Write a function that takes a milliseconds value and returns a string describing the value in days, hours, minutes, and seconds. What's the optimal type for the input value?
5. Create a function showSalary() in such a way that it should accept employee name, and it's salary and display both, and if the salary is missing in function call it should show it as 5000
6. Write a function that calculates the difference between a pair of 2D points (x and y) and returns the result as a point.
   (Hint: this would be a good use for tuples - Search Tuples Examples for inspiration).
7. Write a function that takes a 3-sized tuple and returns a 6-sized tuple, with each original parameter followed by its `String` representation. For example, invoking the function with `(true, 22.25, "yes")` should return, `(true, "true", 22.5, "22.5", "yes", "yes")`. Can you ensure that tuples of all possible types are compatible with your function? When you invoke this function, can you do so with explicit types not only in the function result but also in the value that you use to store the result?

(OPTIONAL EXCERCISE)

Write a sort function for a list of 4-tuples. Below is a list of the nearest stars and some of their properties. The list elements are 4-tuples containing the name of the star, the distance from the sun in light years, the apparent brightness, and the luminosity. The apparent brightness is how bright the stars look in our sky compared to the brightness of Sirius A. The luminosity, or the true brightness, is how bright the stars would look if all were at the same distance compared to the Sun.

```
data = [
('Alpha Centauri A', 4.3, 0.26, 1.56),
('Alpha Centauri B', 4.3, 0.077, 0.45),
('Alpha Centauri C', 4.2, 0.00001, 0.00006),
("Barnard's Star", 6.0, 0.00004, 0.0005),
('Wolf 359', 7.7, 0.000001, 0.00002),
('BD +36 degrees 2147', 8.2, 0.0003, 0.006),
('Luyten 726-8 A', 8.4, 0.000003, 0.00006),
('Luyten 726-8 B', 8.4, 0.000002, 0.00004),
('Sirius A', 8.6, 1.00, 23.6),
('Sirius B', 8.6, 0.001, 0.003),
('Ross 154', 9.4, 0.00002, 0.0005),
]
```

The purpose of this exercise is to sort this list with respect to distance, apparent brightness, and luminosity. Write a program that initializes the data list as above and writes out three sorted tables: star name versus distance, star name versus apparent brightness, and star name versus luminosity. *Hint To sort a list data, one can call sorted(data).*