Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #91

Functional Programming, Lambda, Map

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU
&
Dr. Manju More E
Associate Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M, (Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Introduction

There are a lot of programming languages that are known. But all of them need to follow some strategy or style when they are used in the implementation of some requirement. This methodology/strategy is known as paradigm.

Paradigm: A style of programming

It is a methodology to solve some problems or to do some tasks. A programming paradigm is an **approach to solve the problem using some programming language** or a **method to solve a problem using tools and techniques** that are available to us following some approach. Apart from varieties of programming languages, there are lots of paradigms to fulfill each and every demand as shown below.

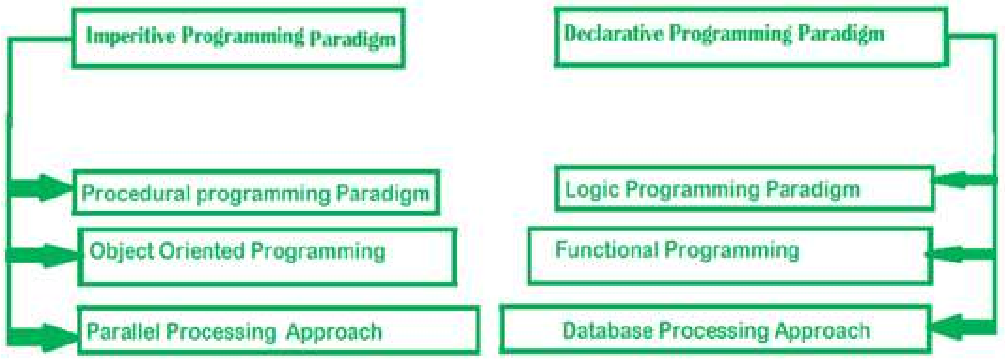

Imperitive Programming Paradigm
Declarative Programming Paradigm
Procedural programming Paradigm
Object Oriented Programming
Parallel Processing Approach
Logic Programming Paradigm
Functional Programming
Database Processing Approach

Fig - 1: Representation of Programming Paradigms

Python supports Imperative, Procedural, Functional and Object Oriented Paradigm.

Let us discuss Functional Programming in detail wrt python.

Functional Programming:

It is a paradigm in which everything is bind in pure mathematical functions style. It uses the mathematical function and **treats every statement as functional expression as an expression is executed to produce a value.** This paradigm mainly focuses on **"what to solve"** rather than "how to solve". The ability to treat functions as values and pass them as an argument make the code more readable and understandable. **Lambda functions, Recursion and callback** are the basic approaches used for its implementation.

Advantages of Functional Programming

- Simple to understand
- Making debugging and testing easier
- Enhances the comprehension and readability of the code

Example_code_1: Assigning a function to a variable. We can then use that variable, same as the function

```
def func():
   print("I am function func()!")
func()

another_name = func
another_name()
```

```
I am function func()!
I am function func()!
```

The assignment **another_name = func** creates a new reference to func() named another_name. We can then call the function by either name, func or another_name. We can display a function to the console with print() or include it as an element in a composite data object like a list, or even use it as a key in the dictionary.

Example_code_2: You can pass a function to another function as an argument – callback concept

```
>>> def inner():
```

```
I am function inner()!
```

```
...     print("I am function inner()!")
...
>>> def outer(function):
...     function()
...
>>> outer(inner)
I am function inner()!
```

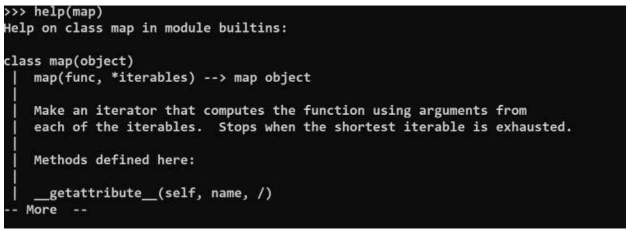
List of functional programming constructs, the python language supports is here - **> lambda, map, filter, and reduce, max, min, zip, list comprehension.**

Let us dig into map and lambda in this lecture notes. Other constructs are explained in further lecture notes.

The function – map()

Returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable. It causes iteration through the iterable, applies the callable on each element of the iterable and creates a map object which is a lazy object. Means**, unless the map object is iterated, you will not be able to access any of the elements from it**.

We can force an **iteration of the map object** by passing the map object as an argument for the **list() or set() or tuple().** The **for loop** of the client iterates through the map object and displays the elements.



```
>>> help(map)
Help on class map in module builtins:

class map(object)
 |  map(func, *iterables) --> map object
 |
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is exhausted.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
-- More  --
```

Note: You can pass one or more iterable to the map() function.

Consider the requirement to create a list of all upper case words from the given list of words. We observe this kind of requirement at number of places in programming. Given an iterable, walk through every element of it, do some operation, collect the result in an iterable. **The number of elements in the output will be same as the number of elements in the input.** Builtin function map() helps us to do this. Refer to Example_code_1.

Example_code_1: Converting all strings in the list to upper case in the new list.

```
a = [ 'apple', 'pineapple', 'fig', 'mangoes' ]
b = list(map(str.upper, a))
print(b)
```

```
['APPLE', 'PINEAPPLE', 'FIG', 'MANGOES']
```

Example_code_2: Creating a list of length of all words from the given list of strings.

```
a = [ 'apple', 'pineapple', 'fig', 'mangoes' ]
b = list(map(len, a))
print(b)
```

```
[5, 9, 3, 7 ]
```

Can we use the user defined function name as a first argument to map? Yes. Refer to below examples.

Example_code_3: Creating a list of squares of all numbers from the given list of numbers.

```
def get_square(n):  #user defined function
        return n*n
a = [11, 33, 22, 44]
b = list(map(get_square, a))
print(b)
```

```
[121, 1089, 484, 1936]
```

Observe that iterable is a list. The callable is a user defined function which returns the square of n. **The returned value from the function is directly added to the map object.**

Example_code_4: Given a list of numbers, double every even element of this list and create a new list. If the element is odd, retain in its position as it is in the new list.

```
def double_num(n):   #user defined function
   if n% 2 == 0:
      return n * 2
   else:
      return n
numbers = [1, 2, 3, 4, 5]
result = list(map(double_num, numbers))
print("The modified list is ",result)
```

```
The modified list is  [1, 4, 3, 8, 5]
```

Think about this:

Can you pass more than one iterable to a map function? If yes, what must be the number of parameters for the callback function passed as the first arg of map? If no, you are wrong! See the help(map) shown above to understand map in detail.

Example_code_5: Given two lists of numbers, create a new list by multiplying the corresponding elements of both the lists.

```
num1 = [4, 2, 5]
num2 = [6, 8, 9]
def product(x,y):
        return x*y
result = map(product, num1, num2)
print(list(result))
```

```
[24, 16, 45]
```

Few points to think!

- **What happens if the length of two iterables are different in Example_code_5? See help(map) to get the answer for this question.**
- **Can we avoid writing the user defined function and have some function inline with the map() ? Yes. Using lambda it is possible. Discussed in detail in the next section.**

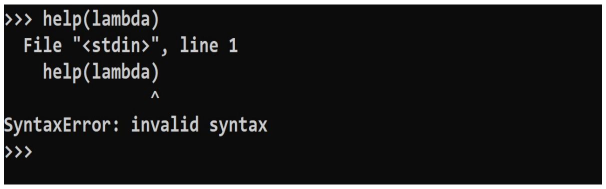
The keyword – lambda

This is **used to create a function without any name** for it. Also known as **throw away functions or Anonymous functions.** Power of lambda is appreciated only when it is used as a part of another function. Used frequently when you need a particular function for a **short period of time and when the function is not needed outside this scope.** Hence, it is helpful in creating **inline functions.**

Syntax: "lambda" [parameter_list] ":" expression

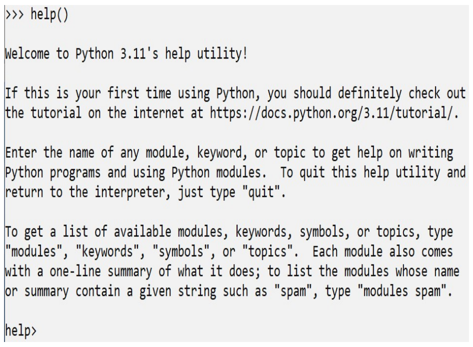
Note: Expression gets evaluated and result is returned

As lambda is the keyword, no help() is available directly.

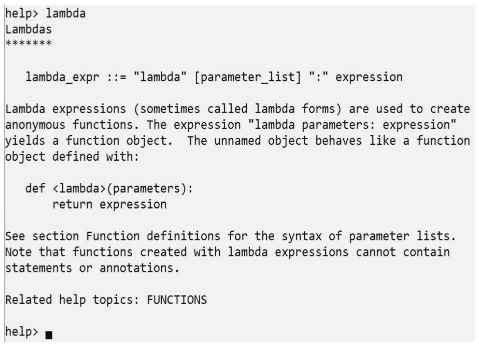


```
>>> help(lambda)
  File "<stdin>", line 1
    help(lambda)
               ^
SyntaxError: invalid syntax
>>>
```

Indirectly, first say help() and press enter. Once you get the help prompt, type lambda and press enter to get the usage of lambda.



```
>>> help()

Welcome to Python 3.11's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.11/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
```



```
help> lambda
Lambdas
*******

   lambda_expr ::= "lambda" [parameter_list] ":" expression

Lambda expressions (sometimes called lambda forms) are used to create
anonymous functions. The expression "lambda parameters: expression"
yields a function object.  The unnamed object behaves like a function
object defined with:

   def <lambda>(parameters):
       return expression

See section Function definitions for the syntax of parameter lists.
Note that functions created with lambda expressions cannot contain
statements or annotations.

Related help topics: FUNCTIONS

help>
```

Consider the below example code.

```
def func(x, y, z):
        return x + y + z
k=func(2, 3, 4)
print(k)
```

```
9
```

Can we implement the same requirement using lambda now?

```
f = lambda x,y,z : x + y + z
print(f(2,3,4))
```

Here, f gets a function object that the lambda expression creates. def **works in a similar fashion, but its assignment is automatic.** But in this code, we named the function which is created using lambda. Name of the function is f. Now f can be used anywhere again and again. But if you create the anonymous function using lambda, it cannot be called again and again wherever possible. It is accessible only within the scope of the callback function again and again. Let us see few example codes

Example_code_6: Given a list of words, display the list of upper case words.

```
a = [ 'apple', 'pineapple', 'fig', 'mangoes' ]
print(list(map(lambda x: x.upper(), a)))
```

```
['APPLE', 'PINEAPPLE', 'FIG', 'MANGOES']
```

#This function created using lambda, cannot be used outside.

Example_code_7: Creating a list of squares of all numbers from the given list of numbers.

```
a = [11, 33, 22, 44]
b = list(map(lambda x: x*x, a))

# rather than creating user defined function,
#better use lambda if you do not want to
#call it outside this.
print(b)
```

```
[121, 1089, 484, 1936]
```

Example_code_8: Given a list of numbers, double every element of this list and create a new list.

```
numbers = [1, 2, 3, 4, 5]
result = list(map(lambda x: x*2, numbers))
print("The modified list is ",result)
```

```
The modified list is  [2, 4, 6, 8, 10]
```

Can the function created by lambda take more than one parameter? Yes. Refer to below example code.

Example_code_9: Given two lists of numbers, create a new list by multiplying the corresponding elements of both the lists.

```
num1 = [4, 2, 5]
num2 = [6, 8, 9]
result = map(lambda x,y: x*y, num1, num2)
print(list(result))
```

```
[24, 16, 45]
```

Example_code_10: Generate multiplication table for a given number n from 1 to 10.

```
n = int(input("Enter the number : "))
def generate(i) :
        prod = n * i # prod : local variable
        return str(n) + " X " + str(i) + " = " + str(prod)
for line in map(generate, range(1, 11)):
        print(line)
```

```
Enter the number :
5
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
```

-END-