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

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #61 & #62
Callback and Programs on callback

By,
PCPS Team-2022
Prof. Apoorva MS

Verified by,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant 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

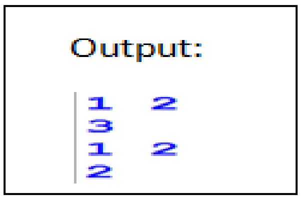
Callback is nothing but passing a function name as an argument to a another function and calling the passed function inside the another function usually based on the processed data. Assume a very big project has been given by the client to a very good team. For every request from the client, rather than modifying the initial code, add functions and use the concept of callback.

Usage of Callback:

- Used in **event-driven programming**, where a function is called in response to a specific event or action, such as a button press or the completion of a network request.
- **Extensively used in functional programming** - [Unit 4], where a function is passed as an argument to another function to be used as a "hook" for performing specific operations.
- Helps to **separate functionality and it's call** and make the code more reusable and modular.
- Used for **asynchronous programming.**

Consider an Example code to add two number and multiple same two numbers using a callback function

```
def doMath(x, y, func):
   print(x, y)
   func(x, y)
def add(x, y):
   print(x + y)
def multiply(x, y):
   print(x * y)
doMath(1, 2, add)
doMath(1, 2, multiply)
```



Output:

```
1  2
3
1  2
2
```

In the above example, the doMath() function takes three arguments: x, y, and func. x and y are the values to be operated on, and func is the callback function to be called. The doMath() function first prints x and y, then calls the callback function with x and y as arguments.
The add() and multiply() functions are the callback functions.

Scenario based example of callback

Suppose a person enters the smart home→This is an event. Light and AC gets switched on immediately -> This is an action in response to an event. So, there will be some other interface/helper function say it as Sensor function, which invokes Light and AC to reach the "ON" state as and when someone enters the home and if no one in home, the state of AC and light will be moved to state "OFF". **Here light and AC are callback functions which gets invoked by interface function.**

Implementation of Callbacks using built-in functions

Consider an example in which you have **n numbers in a list. How do you sort it**? Obviously, the answer is use the **sort() function** or use the **sorted() function.** Let us write the code to do this.

```
li = [23,91,909,27,217]
print(li)
li.sort()   # li = sorted(li)
print(li)
```

Output:
```
[23, 91, 909, 27, 217]
[23, 27, 91, 217, 909]
```

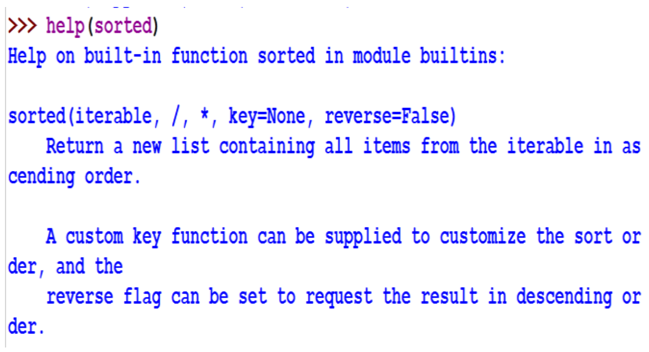
But if you have few constraints like, sort it based on the second digit, sort it based on the product of first two digits and etc, how do you proceed? Here comes the usage of callback. Define a function which will be called by the sort function.

```
def extract_second_digit(x):
        return int(str(x)[1])
li = [23,909,91, 56,27]
print(li)
li.sort(key = extract_second_digit)
#function name only for key
print(li)
```

Output:
```
[23, 909, 91, 56, 27]
[909, 91, 23, 56, 27]
```

key is a default parameter of the sort(). It can take any function name as the value. We are not calling this extract_second_digit. Sort function internally calls this on every element of the iterable and sorting is done based on this temporary. Temporary variable doesn't have any name. It is created in activation record. The sort() picks the corresponding element from the original list.

Note: Same can be accomplished using the sorted() function from builtins. help(sorted) is as below.



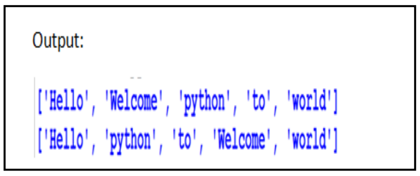
```
>>> help(sorted)
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in as
cending order.

    A custom key function can be supplied to customize the sort or
der, and the
    reverse flag can be set to request the result in descending or
der.
```

Now, we shall try to sort a list of strings? Code is as below.

```
l = ["Hello", "Welcome", "to", "python", "world"]
print(sorted(l))
print(sorted(l, key=str.lower))
```



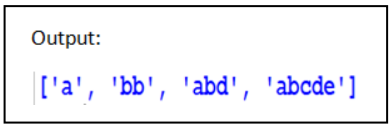
```
Output:

['Hello', 'Welcome', 'python', 'to', 'world']
['Hello', 'python', 'to', 'Welcome', 'world']
```

We have used sorted with two arguments :List-> l and Key function-> str.lower .Sorted is an inbuilt function and we are passing function str.lower as an argument. Hence it is call back function. Here, the str.lower is first implemented on the iterable list. That is, the strings in the list are converted into lowercase and then passed to the sorted function.

Now shall we sort the list of strings based on the length of the string?

```
words = ["abd","a","abcde","bb"]
words.sort(key=len)
print(words)
```



```
Output:
['a', 'bb', 'abd', 'abcde']
```

Implementation of Callbacks using user defined functions

Let us implement a callback concept for booking a cab (ola/uber) through google map interface function.

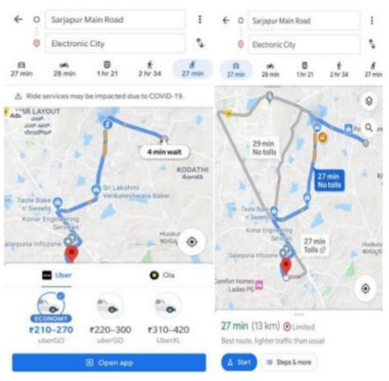
Step - 1: Create a function called google maps(interface function)

Step - 2: Create two functions ola and uber(callback functions)

Step - 3: Take inputs from the user - source, destination, type of cab service(uber or ola)

Step - 4: Using interface function invoke the callback function indirectly.

```
def ola(s,d): #function to book an OLA
        print('booked an ola from:',s,'to',d)
def uber(s,d): #function to book an UBER
        print('booked an uber from:',s,'to',d)
def maps(s,d,app):
        app(s,d)
s,d,service = input('Enter source, destination and taxi service [space separated]: ').split()
if service == 'ola':
        maps(s,d,ola)
elif service == 'uber':
        maps(s,d,uber)
```



Output:

```
Enter source,destination and taxi service: a b ola
booked an ola from: a to b
```

If I have to multiply two numbers of the list, consider the example of callback.

```
def multiply(x):
   return num_list[0]*num_list[1]
def compute(func,x):
   return func(x)
num_list=[2,3]
product=compute(multiply,num_list)
print("Multiplication=",product)
```

Output:

```
Multiplication= 6
```

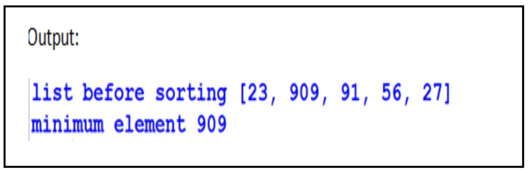
Advantages of using Callback:

- Calling function(outer function) can call the callback function as many as possible to complete the specified task.
- Calling function can pass appropriate parameters according to the task to be completed, to the called functions. This allows information hiding.
- Calling function acts as an interface through which other functions can be called.

Now let us consider few programs

Example_code_1: Find the minimum element from the list based on the second digit of every element of list.

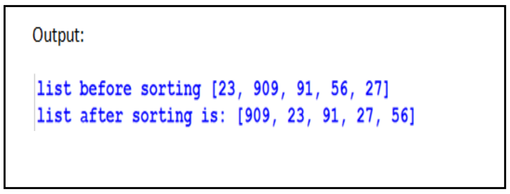
```
def extract_second_digit(x):
   return int(str(x)[1])
def product(x):
   return int(str(x)[0])* int(str(x)[1])
li = [23,909,91,56,27]
print("list before sorting",li)
print("minimum element based on 2nd digit is", min(li,key = extract_second_digit))
```


Output:

```
list before sorting [23, 909, 91, 56, 27]
minimum element 909
```

Example_code_2: Sorting the numbers in a list based on product of digits of every number.

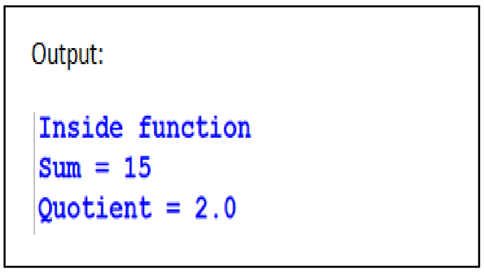
```
def product(x):
   return int(str(x)[0])* int(str(x)[1])
li = [23,909,91,56,27]
print("list before sorting",li)
li.sort(key = product)
print("list after sorting is:",li)
```


Output:

```
list before sorting [23, 909, 91, 56, 27]
list after sorting is: [909, 23, 91, 27, 56]
```

Example_code_3: Can you store the function name in the list and pass this to a function to call multiple callback functions.

```
def function(func_list, x, y):
   print("Inside function")
   for func in func_list:
      func(x,y)
def add(x,y):
   z = x+y
   print('Sum =',z)
def divide(x,y):
   z = x/y
   print('Quotient =',z)
cb_list=[add, divide]
function(cb_list, 10, 5)
```



Output:

```
Inside function
Sum = 15
Quotient = 2.0
```

-END–