



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

# **Lecture Notes Python for Computational Problem Solving UE23CS151A**

***Lecture #65  
Closures in python***

**By,  
Prof. Sindhu R Pai  
Anchor, PCPS - 2023  
Prof. Apoorva MS  
Assistant Professor  
Dept. of CSE, PESU**

**Verified by,  
Prof. Sowmyashree,**

**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

A Closure in Python is a function object that remembers values in enclosing scopes even if they are not present in memory. It can also be defined as a nested function which has access to a free variable from an enclosing function that has finished its execution.

Nested functions are important in closures in Python because they allow you to access variables from the outer function even after the outer function has returned which makes the python code more concise and reusable.

### Characteristics of Closures:

- It is a nested function
- It has access to a free variable in outer scope - A free variable is a variable that is not bound in the local scope. Closures with immutable variables such as numbers and strings - use the nonlocal keyword.
- Nested function is returned by the enclosing function.

Let us understand nested functions i.e one of the characteristics of closure.

#### Example\_code\_1: Two function definitions and two calls separately. No nested functions

```
def f1():  
    print("in f1")  
def f2():  
    print("in f2")  
f1()  
f2()
```

Output:

```
in f1  
in f2
```

#### Example\_code\_2: Function f2 is the Enclosed/Nested function and f1 is the Nesting function/Enclosing function

```
def f1():  
    print("in f1")  
    def f2():  
        print("in f2")  
f1()  
f2()
```

output:

```
in f1  
Traceback (most recent call last):  
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/samy.py", line 6, in <module>  
    f2()  
NameError: name 'f2' is not defined
```

**Note:** Function f2 is defined inside the function f1 and after execution of function f1, the f2 function cannot be accessed because it is in the scope of f1 function.

**Example\_code\_3: Function f2 available inside f1.**

```
def f1():  
    print("in f1")  
    def f2():  
        print("in f2")  
    f2() #f2 is inside the function f1 so it is accessible  
f1()
```

output:

```
in f1  
in f2
```

**Example\_code\_4: Function f2 is made as global so it is valid,so f2 can be invoked from outside the f1**

```
def f1():  
    print("in f1")  
    global f2  
    def f2():  
        print("in f2")  
f1()  
f2()
```

output:

```
in f1  
in f2
```

**Example\_code\_5: Enclosing function returns the object of enclosed function. So id of f2 and f must be same**

```
def f1():  
    print("in f1")  
    def f2():  
        print("in f2")  
    print("id of f2 = ",id(f2))  
    return f2  
f = f1()  
print("id of f = ",id(f))  
#f2()
```

output:

```
in f1  
id of f2 = 48115920  
id of f = 48115920
```

**Example\_code\_6: The variable created in the enclosing scope cannot be modified in the enclosed function**

```
def f1():  
    a = 10  
    def f2():  
        print("a in f2", a)  
        a = 11  
    print("a in f1 = ", a) #10  
    return f2  
f = f1()  
f()  
#print("a outside ",a) #Error
```

Output:

```
a in f1 = 10  
Traceback (most recent call last):  
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/samy.py", line 9, in <module>  
    f()  
  File "C:/Users/HP/AppData/Local/Programs/Python/Python38/samy.py", line 4, in f2  
    print("a in f2", a)  
UnboundLocalError: local variable 'a' referenced before assignment
```

Function `f()` is called which executes `f2()`. Since '`a`' value cannot be accessed by the inner function, it shows Error. **Variable '`a`' is a free variable. Not a global variable.** We can make this variable accessible to the inner function by **declaring it as a "nonlocal" in the inner function** so that any modification to that variable in inner function will be reflected even outside the scope of inner function. Modified code is available in `Example_code_7`.

**Example\_code\_7:**

```
def f1():  
    a = 10  
    def f2():  
        nonlocal a  
        print("a in f2", a)  
        a = 11  
    print("a in f1 = ", a) #10  
    return f2
```

`f = f1()`

`f()` # invokes the inner function

**Is the above example, i.e., `Example_code_7` a closure? Yes. See the characteristics of the closures**

output:

```
a in f1 = 10  
a in f2 10  
a in f2 11
```

**Example\_code\_8:**

```
def f1():  
    a = 10  
    def f2():  
        a = 11  
        print("a in f2:", a)  
    print("a in f1:", a) #10  
    return f2
```

`f = f1()`

`f()`

**Is the above example, i.e., `Example_code_8` a closure? Yes. See the characteristics of the closures**

Output:

```
a in f1: 10  
a in f2: 11
```

**Example\_code\_9:**

```
def f1():  
    a = 0  
    def f2():  
        nonlocal a;    a = a + 1;    return a  
    return f2
```

`f = f1(); b = f()`

`print(b)`

**Is the above example, i.e., `Example_code_9` a closure?**

Output:

1

### Advantages of using Closures:

- We can avoid the use of global values by calling the inner function outside its scope, and accessing the enclosed variable outside its scope
- It provides data hiding.
- It is used while building python micro services

Let us discuss few more examples to know about **nested Functions and closures in detail.**

**Example\_code\_10: Program that shows that, even after deletion of the outer function, the variables of outer function are accessible. Is this closure? Yes.**

```
def outerfunc(x):  
    def innerfunc():  
        print(x)  
    return innerfunc  
myfunc=outerfunc(7)  
myfunc()#refers to innerfunc()  
del outerfunc  
print("After deletion of outer function")  
myfunc()
```

**Output:**

```
7  
After deletion of outer function  
7
```

**Example\_code\_11: Is this closure?**

```
def make_printer(msg):  
    def printer(msg=msg):  
        print(msg)  
    return printer  
printer = make_printer("Hello!")  
printer()
```

**Output:**  
Hello!

Here, the nested function printer() is not a closure because it does not access the free variable variables. The variable msg inside the printer() function is local to printer function and is not accessing the free variable of outer function.

**Example\_code\_12: Is this closure?**

```
def make_printer(msg):  
    def printer(m):  
        print(m, msg)  
    return printer  
printer = make_printer("Hello!")  
printer("world")
```

**Output:**  
world Hello!

#Above code if you change to m = msg in the parameter of printer() and call printer function using printer(), what is the inference?

### Example\_code\_13: Is this closure?

```
def delete_last_item(list):
    def get_last_item():# inner function definition
        list.pop(); print(list)
    return get_last_item #returning inner function
test_list = [1,2,3,4,5]
a=delete_last_item(test_list)#a receives inner function object returned by outer function
a()#function object that invokes inner function only
a()#a can remember the list recieved from delete_last_item function
a()
```

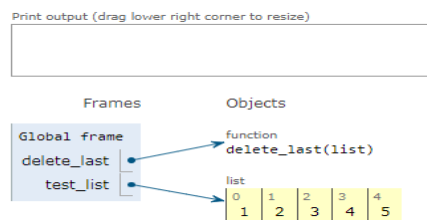
Output:

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

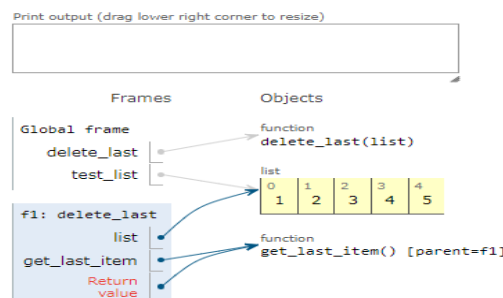
In the above example, list variable is accessed from the outer function delete\_last\_item to the inner function get\_last\_item. Variable “a” is able to remember the outer function variable “list” even after execution of outer function completely. Whenever we make a function call like “a()” the inner function get\_last\_item gets invoked.

## Visualization using Python Tutor

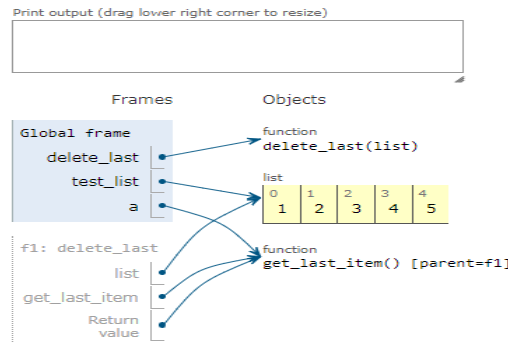
### 1. Funtion delete\_last()is processed as global frame



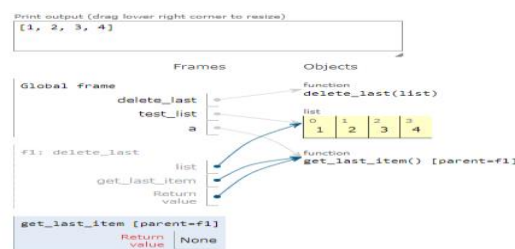
### 2. Inner function get\_last\_item is processed and activation record is created as f1



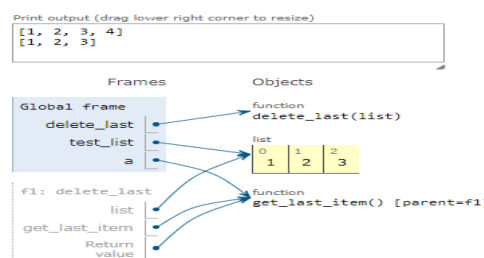
3. Function object a is called which refers to inner function get\_last\_item and Activation f1 got deleted



4. Execution of inner function get\_last\_item and removal of last item 5



5. On calling a() again invokes the inner function get\_last\_item and Removes one more last item 4 from the list .



Important points to note:

- All Nested functions are not closures. But all closures are nested functions.
- Use cases are applicable to python decorators.

-END-