**Nonlocal variable in a nested function**

Before getting into what a closure is, we have to first understand what a nested function and nonlocal variable is.

A function defined inside another function is called a nested function. Nested functions can access variables of the enclosing scope.

In Python, these non-local variables are read only by default

def print\_msg(msg):

# This is the outer enclosing function

def printer():

# This is the nested function

print(msg)

printer()

# We execute the function

# Output: Hello

print\_msg("Hello")

We can see that the nested function printer() was able to access the non-local variable msg of the enclosing function.

## Defining a Closure Function

In the example above, what would happen if the last line of the function print\_msg()returned the printer() function instead of calling it? This means the function was defined as follows.

def print\_msg(msg):

# This is the outer enclosing function

def printer():

# This is the nested function

print(msg)

return printer # this got changed

# Now let's try calling this function.

# Output: Hello

another = print\_msg("Hello")

another()

That's unusual.

The print\_msg() function was called with the string "Hello" and the returned function was bound to the name another. On calling another(), the message was still remembered although we had already finished executing the print\_msg() function.

This technique by which some data ("Hello") gets attached to the code is called **closure in Python**.

This value in the enclosing scope is remembered even when the variable goes out of scope or the function itself is removed from the current namespace.

Try running the following in the Python shell to see the output.

>>> del print\_msg

>>> another()

Hello

>>> print\_msg("Hello")

Traceback (most recent call last):

...

NameError: name 'print\_msg' is not defined

**When do we have a closure?**

As seen from the above example, we have a closure in Python when a nested function references a value in its enclosing scope.

The criteria that must be met to create closure in Python are summarized in the following points.

* We must have a nested function (function inside a function).
* The nested function must refer to a value defined in the enclosing function.
* The enclosing function must return the nested function.

**When to use closures?**

So what are closures good for?

Closures can avoid the use of global values and provides some form of data hiding. It can also provide an object oriented solution to the problem.

When there are few methods (one method in most cases) to be implemented in a class, closures can provide an alternate and more elegant solutions. But when the number of attributes and methods get larger, better implement a class.

Here is a simple example where a closure might be more preferable than defining a class and making objects. But the preference is all yours.

def make\_multiplier\_of(n):

def multiplier(x):

return x \* n

return multiplier

# Multiplier of 3

times3 = make\_multiplier\_of(3)

# Multiplier of 5

times5 = make\_multiplier\_of(5)

# Output: 27

print(times3(9))

# Output: 15

print(times5(3))

# Output: 30

print(times5(times3(2)))