- **Closures**

What is a Closure?
A Closure is an inner function that remembers and has access to variables in the local scope in which it was created even after the outer function has finished executing.

```
def outer_func():
    message = 'Hi'

    def inner_func():
        print(message)

    return inner_func() # returning and executing inner_func

outer_func()
```

output:

_____
Hi

Example: instead of returning and executing the inner_func, let's just return it without executing it.

```
def outer_func():
    message = 'Hi'

    def inner_func():
        print(message)

    return inner_func # no parenthesis, i.e., we are not executing

my_func = outer_func()
```

Explanation:

Same as above, but instead of returning and executing the inner_func, we only returned the inner_func. So my_func variable is actually a function that is equal to inner_func. To make sure, try to print my_func.
=======

```
my_func = outer_func()
print(my_func)
```

output:

<function outer_func.<locals>.inner_func at 0x01282078>

=======
my_func = outer_func()
print(my_func.__name__)
output:
------
inner_func

========
my_func = outer_func()
my_func()
my_func()
my_func()

output:

Hi
Hi
Hi

Note:

That eventhough we are done with the execution of our outer_func, but the inner_func that we returned is still having access to the message variable that is printed out.

Example: adding parameters to the outer_func. The inner_func is still without arguments. therefore, to execute the inner_func, we only need to add ().

```
def outer_func(msg): # msg
	message = msg # msg

	def inner_func():
		print(message)

	return inner_func # inner function still does not receive any arguments

hi_func = outer_func('Hi')
hello_func = outer_func('Hello')

hi_func()
hello_func()
```

output:
_____
Hi
Hello