Python Fundamentals day 47

Today's Agenda

- Functions as objects
- Decorators

Functions as objects

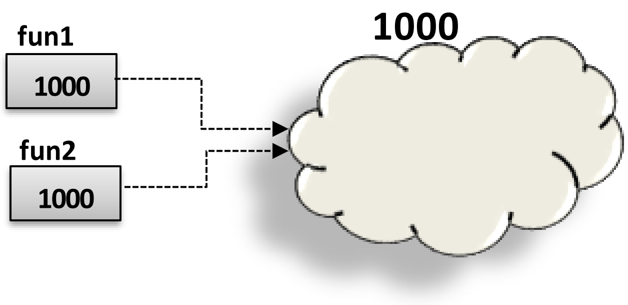
One of the most powerful features of Python is that everything is an object, including functions. Functions in Python are **first-class objects**. Everything a regular object is capable of doing such as integer, floating point numbers, complex numbers, list, set, tuple, dictionaries etc can also be done using functions.

Let us now understand first-class objects with the help of code shown below.

CODE:

```python
def fun1():
   print('Inside fun1( )')

fun1()
print(fun1)
fun2 = fun1
fun2()
print(fun2)
```



OUTPUT:

```
Inside fun1( )
<function fun1 at 0x000002008C3AC6A8>
Inside fun1( )
<function fun1 at 0x000002008C3AC6A8>
```

We see in the above output, when we simply print fun1 and fun2, we get the address of function object as output which means fun1 and fun2 are the reference variables pointing to the same object.

Can one function be sent as argument to another function?

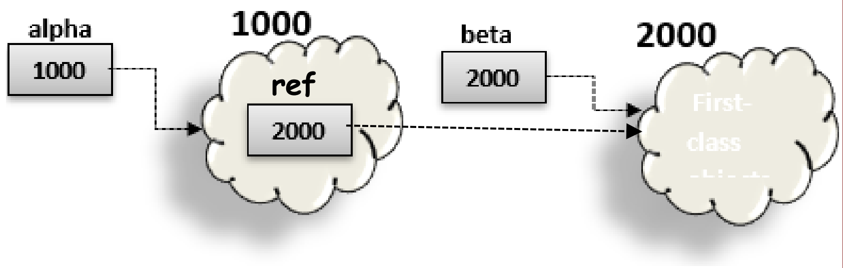
Let us now see can functions be passed as arguments in python?

CODE:

```
def alpha(ref):
    print('Inside alpha( )')
    ref()

def beta():
    print('Inside beta( )')

alpha(beta)
```



OUTPUT:

```
Inside alpha( )
Inside beta( )
```

As we can see from the above output, beta () function is passed as input to alpha () function which concludes one function can be passed as input to other function in python.

Can a function be passed as output from another function?

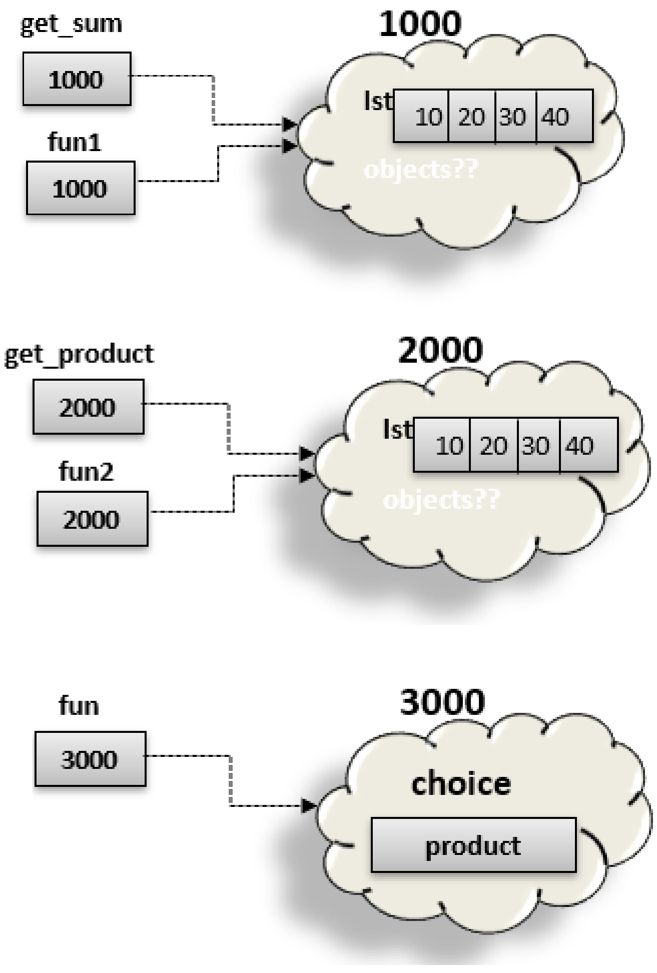
Let us now see can functions be passed as output in python?

CODE:

```python
def get_sum(lst):
    print(sum(lst))

def get_product(lst):
    p = 1
    for i in lst:
        p *= i
    print(p)

def fun(choice):
    if choice == 'sum':
        return get_sum
    else:
        return get_product
fun1 = fun('sum')
fun1([10,20,30,40])
fun2 = fun('product')
fun2([10,20,30,40])
```



OUTPUT:

```
100
240000
```

From the above output, we can conclude that not only can functions be passed as input but also can be returned as output from another function.

Can one function be present within another function?

A function can be present within another function in python and such functions are only called as Inner functions.

An **inner function** is simply a **function** that is defined **inside** another **function**. The **inner function** is able to access the variables that have been defined within the scope of the **outer function**, but it cannot change them.

We can call an inner function in two ways as shown below:

CODE:

```
def outer():
    print('Inside outer()')

    def inner():
        print('Inside innner()')
    inner()

outer()
```

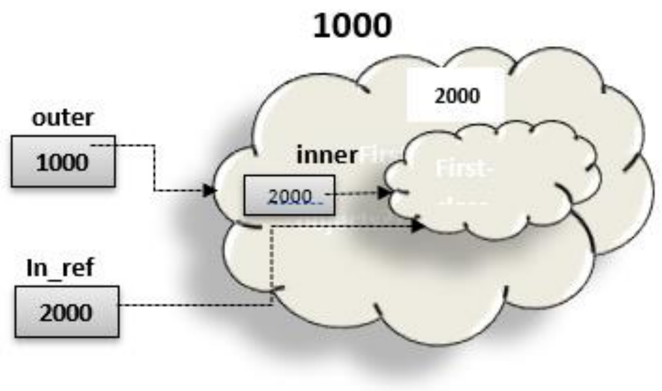
Or

```
def outer():
    print('Inside outer()')

    def inner():
        print('Inside innner()')

    return inner

in_ref = outer()
in_ref()
```



OUTPUT:

```
Inside outer()
Inside innner()
```

After getting to know all the important points about functions as objects in python, let us now try to code a scenario where we have a get_product() which calculates the product of all numbers present in a list but condition is product should not be calculated if 0 is present in list without modifying get_product().

Now how do we achieve it?? Have a look at the code given below!

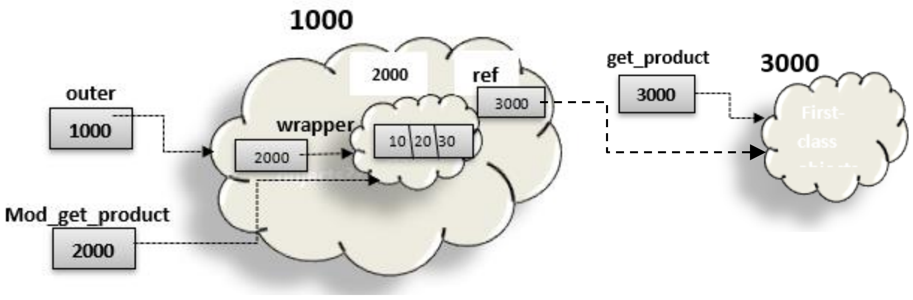
CODE:

```
def outer(ref):

    def wrapper(lst):
        if 0 in lst:
            print('0 is present')
        else:
            ref(lst)
    return wrapper

def get_product(lst):
    p = 1
    for i in lst:
        p *= i
    print(p)

mod_get_product = outer(get_product)
mod_get_product([10,20,30])
mod_get_product([10,0,30])
```



OUTPUT:

```
6000
0 is present
```

The above output can also be achieved by using the concept of decorators in python.

Decorators in python

Decorators in Python allows programmers to modify the behaviour of function or class. Decorators allow us to wrap another function in order to extend the behaviour of wrapped function, without permanently modifying it. In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.

Let us now understand how can we achieve the above output using decorators.

CODE:

```
def outer(ref):

    def wrapper(lst):
        if 0 in lst:
            print('0 is present')
        else:
            ref(lst)
    return wrapper
@outer
def get_product(lst):
    p = 1
    for i in lst:
        p *= i
    print(p)
```

$\xrightarrow{}$ **Decorator**

```
get_product([10,20,30])
get_product([10,0,30])
```

OUTPUT:

```
6000
0 is present
```

Let’s jump to another example to understand decorators in detail.

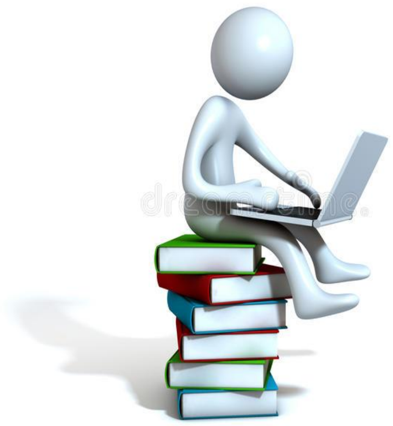
CODE:

```
def outer(ref):

    def wrapper(a,b):
        if b == 0:
            print("Please provide a non zero denominator")
        else:
            ref(a,b)
    return wrapper

@outer
def div(a,b):
    print(a/b)

#mod_div = outer(div)
#mod_div(10,2)
#mod+div(10,0)
div(10,2)
div(10,0)
```



OUTPUT:

```
5.0
Please provide a non zero denominator
```

DID YOU KNOW??

Python is one of the official programming languages at Google and YouTube is one of Google's products that are powered by Python.