Python Fundamentals day 14

Today's Agenda

- Scope of variable
- Global variable
- Local variable
- globals() & locals()
- Recursive function

Scope of variable

A variable is only available from inside the region it is created. This is called **scope**. There are two types of variable

- **Global variable with global scope:**
  Variable created outside all functions are global variables. These variables can be accessed through out the code.
- **Local Variable with local scope:**
  Variables created within a function are local variables. These are accessed only within that function.

Global variables

Let us understand using an example

```
x=99 #global variable

def fun():
    y=999 #local variable
    print(y)

fun()
```

Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
999
```

We have declared global and local variable, let us see where these can be accessed

```
x=99 #global variable
print(x) #in begining of program
def fun():
    y=999 #local variable
    print(y)
    print(x) #in the function

fun()
print(x) #at the end of program
```

Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
99
999
99
99
```

In the above example we can see that global variable is accessible throughout the program. Therefore the scope is said to be global.

Local variables

```
1 x=99 #global variable
2
3 def fun():
4     y=999 #local variable
5     print(y)
6
7 fun()
8 print(y) #outside the function
```

Output:

```
  File "C:/Users/rooman/OneDrive/Desktop/
python/test.py", line 8, in <module>
    print(y) #outside the function

NameError: name 'y' is not defined
```

In the beginning we saw that y can be accessed inside the function. But in the above example when tried to access y outside the function we are getting an error that "y is not defined". That is because the scope of local variable is restricted inside the function only.

globals( ) & locals( )

Is there a way to know which variable is local and which is global? Certainly there is a way. Let us see what that is

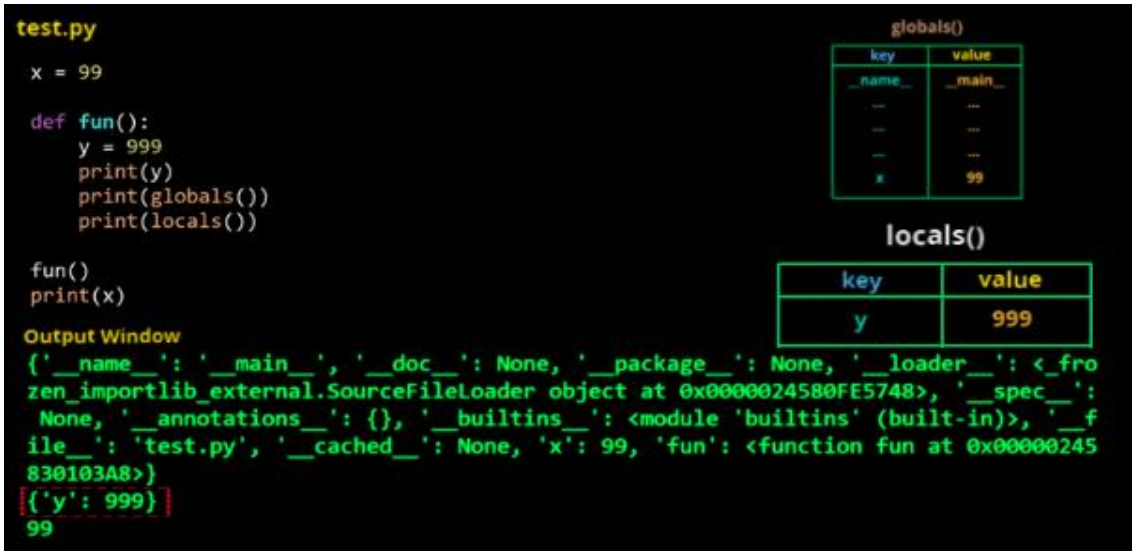
```
test.py

x = 99

def fun():
    y = 999
    print(y)
    print(globals())

fun()
print(x)
```

globals()

key	value
__name__	__main__
...	...
...	...
...	...
x	99

```
Output Window
C:\python>python test.py
999
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <_fro
zen_importlib_external.SourceFileLoader object at 0x000002B216765748>, '__spec__':
 None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, '__f
ile__': 'test.py', '__cached__': None, 'x': 99, 'fun': <function fun at 0x000002B2
168B03A8>}
99
```

All the global variables are mapped to their respective values and these are stored in a dictionary. To access this dictionary, you have to call globals(). As seen in above output, there are several global variables apart from x. For example dunder name, function fun is having its object as its value etc.



Similarly all local variables are mapped to their values and are stored in a dictionary, which can be accessed by calling locals().

A doubt might arise in back of your mind. What if both global and local variable have same name??? Let's see what happens then

```
x=99 #global variable

def fun():
    x=999 #local variable
    print(x)

fun()
```

Output:

```
In [7]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
999
```

Whenever local & global variables have same name, if tried accessing inside a function then local variable is what you'll get.

If at all you want to access global variable inside the function then you have to use a keyword called global as shown below

```
x=99 #global variable

def fun():
    global x
    x=999 #global variable
    print(x)

fun()
```

Output:

```
In [9]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
999
```

When we say global x, wherever x is used it acts like a global variable. To verify this we can print x outside the function and cross check the value.

```
x=99 #global variable

def fun():
    global x
    x=999 #global variable
    print(x)

fun()
print(x)
```

Output:

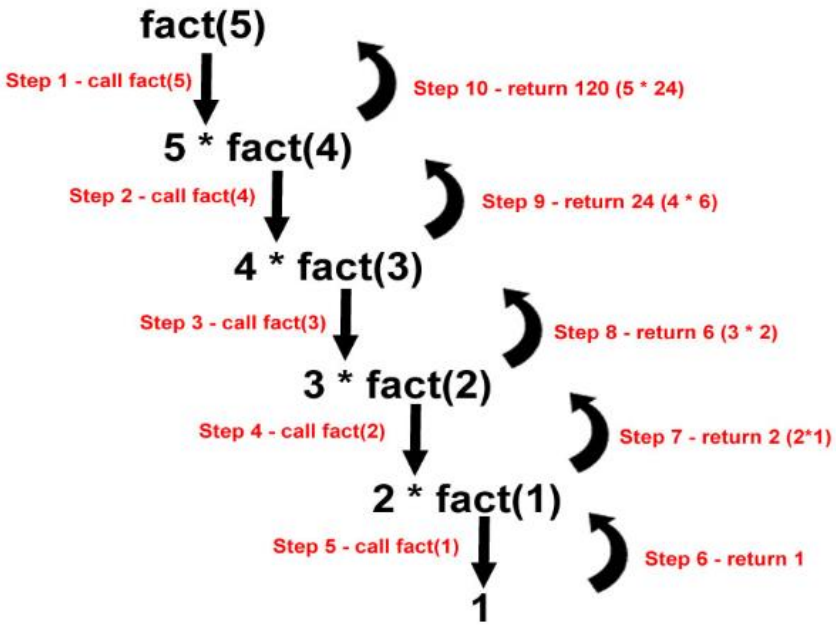
```
In [10]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
999
999
```

Note that integer is immutable, therefore a new object will get created with new address and the address will be reassigned whenever we are trying to change the values.

Recursive function

In simple words, a function that calls itself is called as recursive function. Let us take a basic example and look further

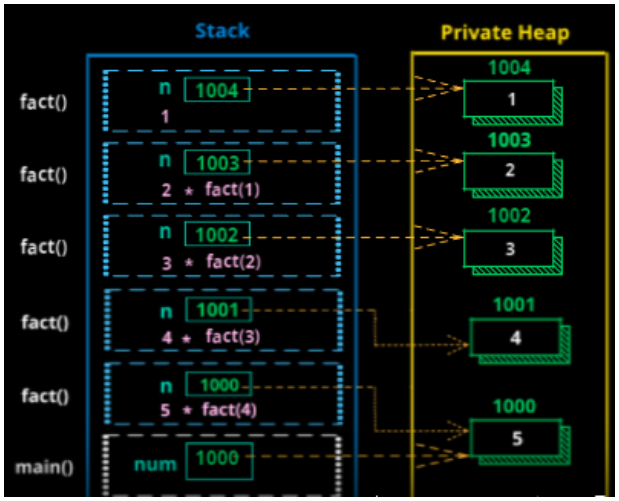
Considering that we want to calculate the factorial of 5



As we can see to calculate the factorial of a number we have to calculate factorial of its previous number. Which in general can we written as **fact(n) = n*fact(n-1)**. Let us try to code the above logic

```
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)

num = int(input("Enter the number:\n"))
print(fact(num))
```

Memory perspective



Output:

```
In [12]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter the number:
5
120
```