

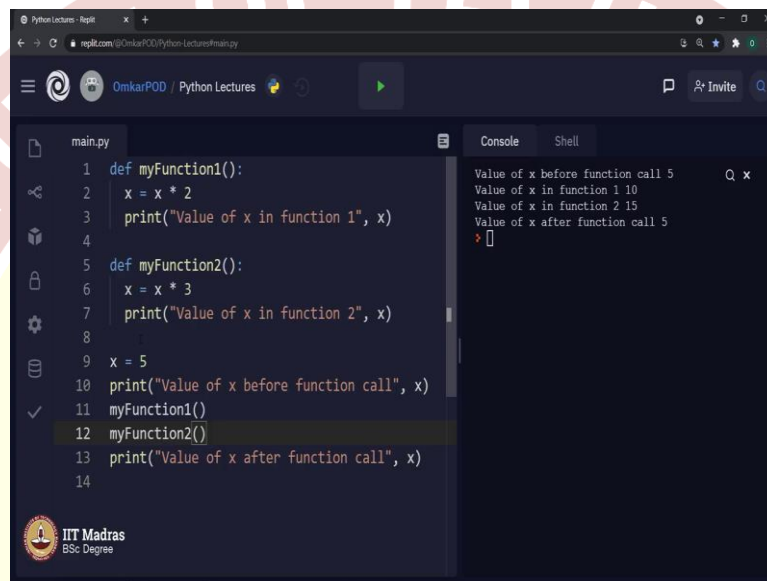


IIT Madras

ONLINE DEGREE

Programming in Python
Professor. Sudarshan Iyengar
Department of Computer Science and Engineering
Indian Institute of Technology, Ropar
Mr. Omkar Joshi
Course Instructor
Online Degree Programme
Indian Institute of Technology, Madras
Scope of Variables

(Refer Slide Time: 0:16)



The screenshot shows a Python REPL window titled 'Python Lecture - Repl'. The code in 'main.py' is as follows:

```
1 def myFunction1():
2     x = x * 2
3     print("Value of x in function 1", x)
4
5 def myFunction2():
6     x = x * 3
7     print("Value of x in function 2", x)
8
9 x = 5
10 print("Value of x before function call", x)
11 myFunction1()
12 myFunction2()
13 print("Value of x after function call", x)
14
```

The console output on the right shows the following sequence of values for x:

- Value of x before function call 5
- Value of x in function 1 10
- Value of x in function 2 15
- Value of x after function call 5

Hello Python student. Let us start this lecture by writing a piece of code using functions. Let us define one function name myFunction1, parameter x, x is equal to x multiplied to 2, value of x in function 1 x. Let us say x is equal to 5, value of x before function call x, myFunction1 pass the argument x whose value is 5, then print value of x after function call x.

What do you think, what will be the output of this particular Python program? Before execution, let us try to analyse it just by a plain logic. X is equal to 5, then print value of x before function call and x, which means at line number 6, it should print 5. Let us say 5 over here, then myFunction1 pass 5 as a value to this parameter x multiplied by 2 that is 5 into 2, 10. Updated value for x is 10, hence print value of x in function1 x it should print 10.

Come back to the call and the next line value of x after function call x, the recent value of x being 10, it should again print 10 over here. So, the output should be 5, 10, 10. Let us see whether our logical analysis of this code is correct. No, there is something wrong with this output. We are getting 5, 10, 5 instead of 5, 10, 10. The last value should have been 10, but instead we are getting 5.

I think there is something wrong with my computer. No, there is nothing wrong with the computer or this output. The mistake is in the analysis what we did with respect to this particular print statement over here. It will not print 10, it is supposed to print 5 as we are getting as a output over here.

Now, you all must be wondering why this is so? This is because of two different concepts. First, call by value and second, scope of a variable. You must be wondering what all these tricky terms are, scope of a variable, call by value? But do not worry, they are not as difficult as they sound. First let us see what is this call by value concept is.

Whenever we call a function along with an argument, we are not actually passing this variable `x` to this function, what we are actually doing is, only passing the value of that variable to this function. Whenever we say `myFunction` and pass this variable as an argument over here, computer is not actually passing this `x` from here to here. What actually it is doing is, it is taking the value of `x` from here and passing that value from this particular line number 7 to this line number 1.

Because of that whenever computer starts the execution of this particular function, it knows the value of `x` is 5. So, in this particular code block, which is the definition of function, we do not have this `x` over there. We have a different `x` variable, and this `x` variable also has value which is 5. When we are passing `x` as a argument and we have `x` as a parameter, still this variable `x` in these 4 lines is different than this variable `x` in these 3 lines.

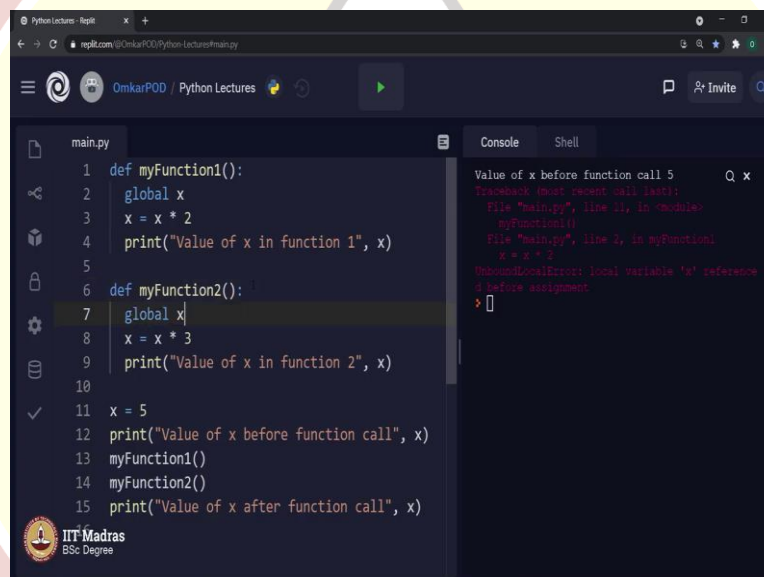
Because a computer maintains two different types of variables. One, which is global and second which is called as local. Now, you must be wondering what is this local and global business. Let me write one more function then we will see. `myFunction2` and over here we are multiplying `x` with 3, value of `x` in function 2 as `x`. Let us call that function `function2` and execute the code.

Still, we are getting a strange output; 5, 10, 15 again 5. Ideally it should have done `x`, so it should do 5 into 2, which is 10. Then function 2 it should do 10 into 3 which is 30, but no, it is doing 5 into 3 which is 15 and when it comes back over here, it is once again saying 5. Now, computer is maintaining three different variables with same name which is `x`; one over here, which is called a global variable which means, this `x` is accessible wherever you want throughout the entire problem whereas, this variable `x` is accessible, or it is useful only in this function definition. Whereas this particular variable `x` is useful and accessible only in this particular function definition.

Therefore, the scope of this variable `x` is referred as local, which means it is local to this function definition or the scope of this variable `x`, it is local to this function definition. Whereas this variable `x` is global across the entire Python program. Now, one may ask, if we are calling this variable global, then why it is not accessible inside these functions? Why it is creating a new variable `x` every time as a local variable? Why not modify this variable itself?

Yes, we can do that. We can access and modify this original variable `x` which is global variable inside these functions as well. But in order to do that we have to explicitly tell the computer that I do not want you to create a new variable `x` over here. I want you to use this same variable and modify it. We should remove it from here so that computer will not create it and now let us try to execute it.

(Refer Slide Time: 8:31)

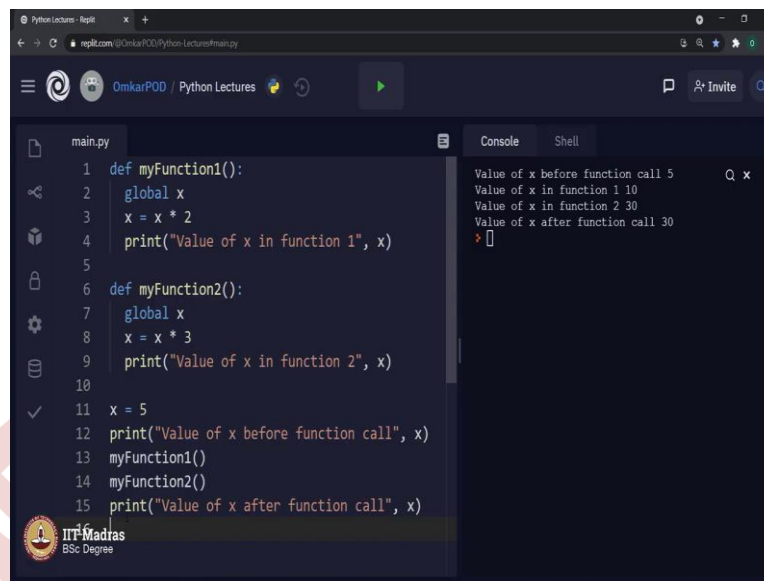


```
main.py
1 def myFunction1():
2     global x
3     x = x * 2
4     print("Value of x in function 1", x)
5
6 def myFunction2():
7     global x
8     x = x * 3
9     print("Value of x in function 2", x)
10
11 x = 5
12 print("Value of x before function call", x)
13 myFunction1()
14 myFunction2()
15 print("Value of x after function call", x)
```

```
Value of x before function call 5
Traceback (most recent call last):
  File "main.py", line 11, in <module>
    myFunction1()
  File "main.py", line 2, in myFunction1
    x = x * 2
NameError: local variable 'x' referenced before assignment
```

Still there is a error because now computer says local variable `x` referenced before assignment which means, even after removing this variable `x` from function definition, still variable `x` is considered as local variable by the computer. Now what to do? How to tell computer that variable `x` is not a local variable, it is the global variable which is defined over here in line number 9. In order to do that, Python has a specific keyword called `global` and followed by variable name which is `x` in this case. Similarly, over here. Let us try this code.

(Refer Slide Time: 9:32)



```
main.py
1 def myFunction1():
2     global x
3     x = x * 2
4     print("Value of x in function 1", x)
5
6 def myFunction2():
7     global x
8     x = x * 3
9     print("Value of x in function 2", x)
10
11 x = 5
12 print("Value of x before function call", x)
13 myFunction1()
14 myFunction2()
15 print("Value of x after function call", x)
```

Console

```
Value of x before function call 5
Value of x in function 1 10
Value of x in function 2 30
Value of x after function call 30
>
```

Now we are getting the output as we predicted initially. 5, then because of this x into 2, we got 10. Then this same variable 10 is multiplied by 3 hence, we got 30 and then, we are printing that 30 over here in line number 15. So, let us see what happens with this particular keyword global. Whenever we say global x, computer realizes there is one variable name x somewhere in the global code which we have written, then it looks for that particular variable and finds its value which is 5.

And then in that particular function, it always refers to this particular variable x. Because of that this x into 2 is 5 into 2, results into 10. When we execute second function, once again we are saying global x, once again computer realizes that x is a global variable that means, there is only single version of this variable x. Therefore, it will not create another copy of x which is local to this function. It will simply use the existing global copy and as the value of that x is 10, it will do 10 into 3 and result into 30.

Few of you must be wondering why I am explaining this simple concept of global and local variables for so much of time. Whereas there will be another set of students who may think this very difficult. So, do not break your head over this concept. This is not as difficult as it sounds. Maybe watch this lecture few more times and you will understand the difference between local and global variable, what is this call by value concept and so on. With respect to writing your own programs, this should not be a problem, but when you try to answer assignment questions, this will be very helpful. Thank you for watching this lecture. Happy learning.