



LAB NOTES OF PYTHON PROGRAMMING (Functions)

By

Dr. Samaher Hussein Ali

Bashar Hamid

Anaam Ghanem

Samaher@itnet.uobabylon.edu.iq

College of Information Technology, University of Babylon, Iraq

Notes of Lab 5

Python Functions

- A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
- As you already know, Python gives you many built-in functions like **print()**, etc. but you can also create your own functions. These functions are called **user-defined functions**.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses ()
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement **return** [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Defining a Function

- Syntax:

```
def functionname( parameters ):
    code to be executed
    return [expression]
```

Functions

- Example: This function takes a string as input parameter and prints it on standard screen.

```
def printme( str ):  
    print str  
    return
```

Calling a Function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function:

Function definition is here

```
def printme( str ):
    print str
    return
```

Now you can call printme function

```
printme("First call to user defined function!")
printme("Again second call to the same function")
```

Required arguments

- Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.
- To call the function *printme()*, you definitely need to pass one argument, otherwise it would give a syntax error as follows:

```
# Function definition is here  
def printme( str ):   
    print str  
    return  
  
# Now you can call printme function  
printme()
```

Required arguments

- When the above code is executed, it produces the following result:

```
>>>
```

```
Traceback (most recent call last):
```

```
  File "C:/Users/Bashar/Desktop/function1.py", line 4, in <module>
```

```
    printme()
```

```
TypeError: printme() takes exactly 1 argument (0 given)
```

```
>>>
```


Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. Following example gives an idea on default arguments, it would print default age if it is not passed:

```
def printinfo( name, age = 35 ):  
    print "Name: ", name  
    print "Age ", age  
    return
```

```
printinfo("Ahmed" , 20 )  
printinfo( name="Ali" )
```

Functions

- Example: This function take a,b and return the summation of it.

```
def add(a,b):  
    total= a+b  
    #print "result of a + b is : ",total  
    return total
```

```
print add(3,2)
```

```
print add(5,6)
```

```
>>>
```

```
5
```

```
11
```

Functions

- Example: This function to find the factorial of given **n**.

```
def find_fact(n):  
    fact=1  
    for i in range(1,n+1):  
        fact=fact*i  
    return fact  
print find_fact(3)  
print find_fact(5)  
print find_fact(7)  
>>>  
6  
120  
5040
```

Functions

- Example: This function to find the Fibonacci sequence.

```
def fibo(n):  
    a=0  
    b=1  
    for i in range(0,n):  
        print a ,  
        temp = a  
        a = b  
        b = a + temp  
    return a  
  
print fibo(15)  
>>>  
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
```

Functions

- Example: This function to find the gcd of two given numbers.

```
def find_gcd(x, y):  
    if x > y:  
        smaller = y  
    else:  
        smaller = x  
    for i in range(1, smaller + 1):  
        if((x % i == 0) and (y % i == 0)):  
            gcd = i  
    return gcd  
  
num1=input("Enter Number1 :")  
num2=input("Enter Number2 :")  
print "gcd is : ", find_gcd(num1, num2)
```

Scope of Variables

- **Global vs. Local variables:**
- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
- This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example:

Scope of Variables

- Example:

```
total = 0; # This is global variable.  
def sum( arg1, arg2 ):  
    total = arg1 + arg2 # Here total is local variable.  
    print "Inside the function local total : ", total  
    return total  
sum( 10, 20 )  
print "Outside the function global total : ", total  
>>>  
Inside the function local total : 30  
Outside the function global total : 0
```