BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on Pass by reference, function arguments and Python Lambda Function.

Pass by reference:-

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

In pass by reference the argument is being passed by reference and the reference is being overwritten inside the called function.

Example:-

```
#!/usr/bin/python

# Function definition is here

def changeme( mylist ):

  "This changes a passed list into this function"

  mylist = [1,2,3,4]; # This would assig new reference in
mylist

  print "Values inside the function: ", mylist

  return

# Now you can call changeme function
```

```
mylist = [10,20,30];

changeme( mylist );

print "Values outside the function: ", mylist
```

Output:-

The parameter mylist is local to the function changeme. Changing mylist within the function does not affect mylist. The function accomplishes nothing and finally this would produce the following result −

```
Values inside the function:  [1, 2, 3, 4]

Values outside the function:  [10, 20, 30]
```

Function Arguments:-

You can call a function by using the following types of formal arguments −

1) Required arguments
2) Keyword arguments
3) Default arguments
4) Variable-length arguments

1) 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.

Example:-

```
#!/usr/bin/python

# Function definition is here

def printme( str ):

   #This prints a passed string into this function

   print str

   return;

# Now you can call printme function

printme("Hello")
```

Output:- Hello

To call the function printme(), you definitely need to pass one argument, otherwise it gives a syntax error as follows−

```
#!/usr/bin/python

# Function definition is here

def printme( str ):
   #This prints a passed string into this function
   print str
   return;
# Now you can call printme function
printme()
```

Output:-

When the above code is executed, it produces the following result −

```
Traceback (most recent call last):
   File "test.py", line 11, in <module>
      printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

2) Keyword arguments:-

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the printme() function in the following ways –

Example:- 1

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
   #This prints a passed string into this function
   print str
   return;
# Now you can call printme function
printme( str = "My string")
```

Output:-

When the above code is executed, it produces the following result –

```
My string
```

Example:- 2

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age ):
   #This prints a passed info into this function
   print "Name: ", name
   print "Age ", age
   return 0;
# Now you can call printinfo function
printinfo( age=29, name="Rahul" )
```

Output:-

When the above code is executed, it produces the following result –

```
Name:  Rahul
Age:  29
```

3) 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. The following example gives an idea on default arguments, it prints default age if it is not passed –

Example:-

```
#!/usr/bin/python
# Function definition is here
def printinfo( name, age = 35 ):
   #This prints a passed info into this function
   print "Name: ", name
   print "Age: ", age
   return;
# Now you can call printinfo function
printinfo( age=29, name="Rahul" )
printinfo( name="Roshan" )
```

When the above code is executed, it produces the following result −

```
Name:  Rahul
Age:  29
Name:  Roshan
Age:  35
```

4) Variable-length arguments

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is this −

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
```

```
  function_suite
  return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example −

```
#!/usr/bin/python
# Function definition is here
def printinfo( arg1, *vartuple ):
  #This prints a variable passed arguments
  print "Output is: "
  print arg1
  for var in vartuple:
    print var
  return;
# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Output:-

When the above code is executed, it produces the following result –

```
10

Output is:

70

60

50
```

Python Lambda (Anonymous) Functions:-

A lambda function is a small anonymous function. These functions are called anonymous because they are not declared in the standard manner by using the **def** keyword. You can use the **lambda** keyword to create small anonymous functions.

- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

- Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Syntax

lambda arguments : expression

The expression is executed and the result is returned:

Example:-

```
#!/usr/bin/python

# Function definition is here

sum = lambda arg1, arg2: arg1 + arg2;

# Now you can call sum as a function

print "Value of total : ", sum( 10, 20 )

print "Value of total : ", sum( 20, 20 )
```

Output:-

When the above code is executed, it produces the following result −

```
Value of total :  30

Value of total :  40
```

Lambda functions can take any number of arguments

Example:- 1 Lambda function with one argument

```
x = lambda a: a + 10
print(x(5))
```

Output:- 15:

Example:- 2 Lambda function with two argument

```
x = lambda a, b: a * b
print(x(5, 6))
```

Output:- 30

Example:- 3 Lambda function with three argument

```
x = lambda a, b, c: a + b + c
print(x(5, 6, 2))
```

Output :- 13

Why we Use Lambda Functions?

The power of lambda is better shown when you use them as an anonymous function inside another function. We Use lambda functions when an anonymous function is required for a short period of time.

Example:-

We have a function definition that takes one argument, and that argument will be multiplied with an unknown number.

Use that function definition to make a function that always doubles the number you send in.

```
def myfunc(n):
  return lambda a : a * n
mydoubler = myfunc(2)
print(mydoubler(11))
```

Output:- 22

Use the same function definition to make a function that always triples the number you send in.

```
def myfunc(n):
  return lambda a : a * n
mytripler = myfunc(3)
print(mytripler(11))
```

Output:- 33

Use the same function definition to make both functions, in the same program.

```
def myfunc(n):
  return lambda a : a * n
mydoubler = myfunc(2)
mytripler = myfunc(3)


print(mydoubler(11))
print(mytripler(11))
```

Output:-
```
22
33
```

The return Statement:-

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.

All the above examples are not returning any value. You can return a value from a function as follows −

Example:-

```
#!/usr/bin/python
# Function definition is here
```

```
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2
   print "Inside the function : ", total
   return total;
# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

Output:-

When the above code is executed, it produces the following result −

```
Inside the function :  30
Outside the function :  30
```

Scope of Variables:-

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier.

There are two basic scopes of variables in Python –

1) Global variables
2) Local 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 –

```
#!/usr/bin/python

total = 0; # This is global variable.

# Function definition is here

def sum( arg1, arg2 ):

   # Add both the parameters and return them."

   total = arg1 + arg2; # Here total is local variable.

   print "Inside the function local total : ", total
   return total;
# Now you can call sum function
sum( 10, 20 );
print "Outside the function global total : ", total
```

Output:-

When the above code is executed, it produces the following result −

```
Inside the function local total :  30
Outside the function global total :  0
```