BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on Python Lambda Function and return statement.

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 −

 Global variables

 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
```