HOPE foundation

PYTHON PROGRAMMING (Basics)

Lecture: 17

I. Python Functions
   - Function Definition
   - Function Calling

II. Function Arguments

III. Recursive Functions

IV. Anonymous Function(lambda)

PYTHON FUNCTIONS:

In Python, a function is a group of related statements that performs a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

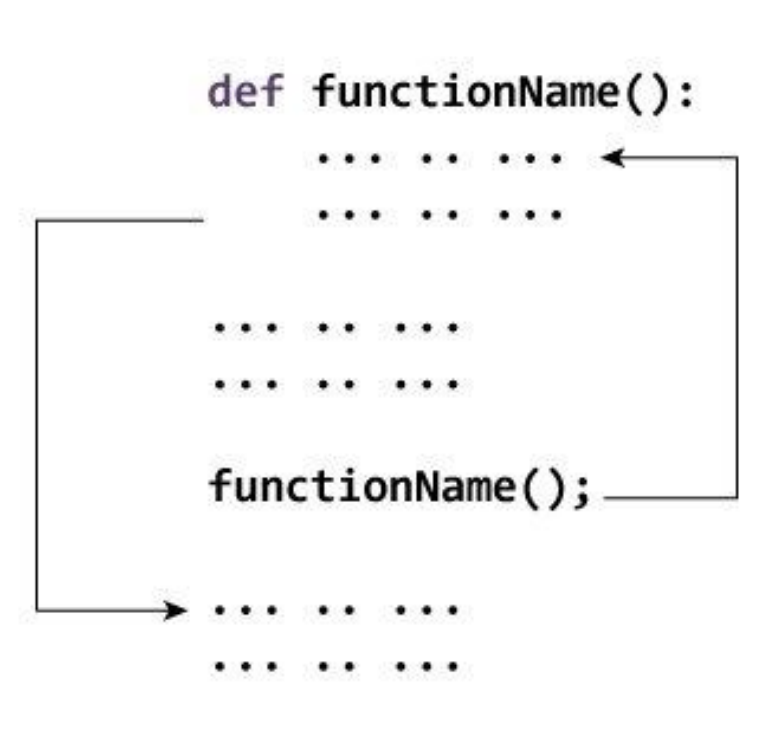
Syntax of Function:

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

1. Keyword **def** that marks the start of the function header.
2. A function name to uniquely identify the function. Function naming follows the same rules.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of the function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
7. An optional **return** statement to return a value from the function.

```
def greet(name):
    """ This function greets
     to the person passed in
    as a parameter """
    print("Hello, " + name + ". Good morning!")
greet('aila')
```

How Function works in Python?



How to call a function in python?

Once we have defined a function, we can call it from another function, program or even the Python prompt.

To call a function we simply type the function name with appropriate parameters.

```
>>> greet(‘aila')
```

Output: Hello, aila. Good morning!

The return statement:

The return statement is used to exit a function and go back to the place from where it was called.

Syntax of return

return [expression_list]

This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

```
def absolute_value(num):
        """This function returns the
        absolute value of the entered
         number"""
        if num >= 0:
            return num
        else:
             return -num
print(absolute_value(2))
print(absolute_value(-4))
```

```
def greet(name):
    """ This function greets
     to the person passed in
    as a parameter """
    print("Hello, " + name + ". Good morning!")
greet('aila')
```

Output: Hello aila. Good morning!

Scope and Lifetime of variables:

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a local scope.

```
def my_func():
    x = 10
        print("Value inside function:",x)
x = 20
     my_func()
print("Value outside function:",x)
```

Function Arguments:

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

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

*ARGUMENTS

* Information can be passed into functions as arguments.
* Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

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 gives a syntax error as follows −

```
def printme( name, age ):
    "This prints a passed string
     into this function"
      print str
      return
printme('aila',10)
```

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 –

```
def printme( str ):
    print str
    return;
printme( str=‘hello ‘)
```

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 −

```
def printinfo( name, age = 35 ):
    print("Name: ", name)
    print("Age ", age)
    return
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

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.

```
def greet(*names):
        print("Hello", name)
greet("Monica", "Luke", "Steve", "John")
```

Python Recursive Function:

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

```
def factorial(x):
    if x == 1:
        return 1
    else:
    return (x * factorial(x-1))
num = 3
print("The factorial of", num, "is", factorial(num))
```

Anonymous Functions (lambda):

- In Python, an anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.
- Hence, anonymous functions are also called lambda functions.

syntax: lambda arguments: expression

```
sum=lambda a,b: a+b
print(sum(5,6))
```

Example:

```
double = lambda x: x * 2

print(double(5))
```

filter() function in lambda:

The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence “sequence”, for which the function returns True. Here is a small program that returns the odd numbers from an input list.

```
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambda x: (x%2 = 0) , li))
print(final_list)
```

Use of lambda with map() function:

The map() function in Python takes in a function and a list as argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item.

```
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(map(lambda x: x*2 , li))
print(final_list)
```