CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NUMERICAL METHODS AND OPTIMIZATION USING PYTHON
COURSE CODE- 22CSH-259/22ITH-259
DR. HARDEEP KAUR (E15828)

DISCOVER . LEARN . EMPOWER

Python Functions

- A function is a block of organized, reusable code that is used to perform a single, related action.
- Functions provides 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

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 first statement of a function can be an optional statement - the documentation string of the function or **docstring**.
- 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.

- **Declaring Docstrings:** The docstrings are declared using “””triple double quotes””” just below the class, method or function declaration. All functions should have a docstring.

- **Accessing Docstrings:** The docstrings can be accessed using the __doc__ method of the object or using the help function.

Syntax:

```
def functionname( parameters ):
   “””function_docstring”””
   function_suite
   return [expression]
```

By default, parameters have a **positional behavior**, and you need to inform them in the same order that they were defined.

Calling a Function

- Example:

```
def printme( str ):
   “””This prints a passed string function”””
   print(str)
   return

printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

This would produce following result:

```
I'm first call to user defined function!
Again second call to the same function
```

THANK YOU

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NUMERICAL METHODS AND OPTIMIZATION USING PYTHON
COURSE CODE- 22CSH-259/22ITH-259
DR. HARDEEP KAUR (E15828)

DISCOVER . LEARN . EMPOWER

Pass by reference vs value

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. For example:

```
def changeme( mylist ):
    mylist.append([1,2,3,4])
    print("Values inside the function: ", mylist)
    return
mylist = [10,20,30]
changeme( mylist )
print("Values outside the function: ", mylist)
```

So this would produce following result:

Values inside the function: [10, 20, 30, [1, 2, 3, 4]]

Values outside the function: [10, 20, 30, [1, 2, 3, 4]]

```
# Here x is a new reference to same list lst
def myFun(x):
    x[0] = 20
lst = [10,11,12,13,14,15]
myFun(lst)
print(lst)
```

Output:

[20, 11, 12, 13, 14, 15]

When we pass a reference and change the received reference to something else, the connection between passed and received parameter is broken. For example, consider below program.

```
# Here x is a new reference to same list lst
def myFun(x):
    x=[20,30,40]
# Driver Code (Note that lst is not modified ) after function call.
lst =[10,11,12,13,14,15]
myFun(lst)
print(lst)
```

Output:

```
[10, 11, 12, 13, 14, 15]
```

There is one more example where argument is being passed by reference but inside the function, but the reference is being over-written.

```
def changeme( mylist ):
   mylist = [1,2,3,4]
   print ("Values inside the function: ", mylist )
   return
mylist = [10,20,30]
changeme( mylist )
print("Values outside the function: ", mylist)
```

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 following result:

```
Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]
```

Another example to demonstrate that reference link is broken if we assign a new value (inside the function).

```
def myFun(x):
    x=20
x=10
myFun(x)
print(x)
```

Output:
10

THANK YOU

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NUMERICAL METHODS AND OPTIMIZATION USING PYTHON
COURSE CODE- 22CSH-259/22ITH-259
DR. HARDEEP KAUR (E15828)

DISCOVER . LEARN . EMPOWER

Function Arguments:

A function by using the following types of formal arguments::

Required arguments

Keyword arguments

Default arguments

Variable-length arguments

Required arguments:

Required arguments are the arguments passed to a function in correct positional order.

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

This would produce following result:

```
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/UT1.PY", line 4, in <module>
    printme()
TypeError: printme() missing 1 required positional argument: 'str'
```

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.

```
def printme( str ):
   print (str)
   return
printme( str = "My string")
```

This would produce following result:

My string

Following example gives more clear picture. Note, here order of the parameter does not matter:

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

This would produce following result:

```
Name: miki  Age 50
```

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 idea on default arguments, it would print default age if it is not passed:

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

This would produce following result:

```
Name: miki Age 50
Name: miki Age 35
```

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.

The general 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 will hold the values of all non_keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. **For example**:

```
def printinfo( arg1, *vartuple ):
    print("Output is: ")
    print (arg1)
    for var in vartuple:
        print(var)
    return
printinfo( 10 )
printinfo( 70, 60, 50 )
```

This would produce following result:

```
Output is:
10
Output is:
70
60
50
```

THANK YOU

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NUMERICAL METHODS AND OPTIMIZATION USING PYTHON
COURSE CODE- 22CSH-259/22ITH-259
DR. HARDEEP KAUR (E15828)

DISCOVER . LEARN . EMPOWER

The Anonymous Functions:

- You can use the lambda keyword to create small anonymous functions. These functions are called anonymous because they are not declared in the standard manner by using the def keyword.
- 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 [arg1 [,arg2,.....argn]]:expression

Example:

```
sum = lambda arg1, arg2: arg1 + arg2
print("Value of total : ", sum( 10, 20 ))
```

This would produce following result:

```
Value of total : 30
```

THANK YOU

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NUMERICAL METHODS AND OPTIMIZATION USING PYTHON
COURSE CODE- 22CSH-259/22ITH-259
DR. HARDEEP KAUR (E15828)

DISCOVER . LEARN . EMPOWER

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.

Example:

```
total = 0 # This is global variable.
def sum( arg1, arg2 ):
   total = arg1 + arg2
   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:

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

- Global variables can be read from local scope.
- def demo():
        print(S)
  S=“I love python”
  demo()
 print(S)

We can use local and global vaiable with the same name.

- def demo():
        S=“I love programming”
        print(S)
  S=“I love python”
  demo()
 print(S)

The global statement

- We can use global keyword, if we want to change the value of a global variable within a function and outside the function too.
- def demo():
- global S

  S="I love programming"

  print(S)

S="I love python"

demo()

print(S)

Returning multiple values

- It is possible in python to return multiple values.
- Example 1:
- Def calculate(num1, num2):
- return num1+num2, num1-num2
- print(“ “, calculate(10,20))
- Example 2:
- Def calculate(num1, num2):
- return num1+num2, num1-num2
- add, sub=calculate(10,20)
- print(“add= “, add,”sub=“, sub)

THANK YOU