UNIT-II

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Functions

- A Function is a self block of code.
- A Function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability.
- A Function is a subprogram that works on data and produce some output.

Types of Functions:

- There are two types of Functions.
  - **Built-in Functions**: Functions that are predefined. We have used many predefined functions in Python.
  - **User- Defined**: Functions that are created according to the requirements.

Benefits of Modularizing a Program with Functions

- A program benefits in the following ways when it is broken down into functions:
- **Simpler Code**
- A program's code tends to be simpler and easier to understand when it is broken down into functions.
- **Code Reuse**
- Functions also reduce the duplication of code within a program. If a specific operation is performed in several places in a program, a function can be written once to perform that operation and then be executed any time it is needed
- **Better Testing**
- When each task within a program is contained in its own function, testing and debugging becomes simpler

- **Faster Development**
- It doesn't make sense to write the code for these tasks multiple times. Instead, functions can be written for commonly needed tasks & those functions can be incorporated into each program that needs them.
- **Easier Facilitation of Teamwork**
- Functions also make it easier for programmers to work in teams. When a program is developed as a set of functions that each performs an individual task, then different programmers can be assigned the job of writing different functions.

- **Void Functions and Value-Returning Functions**

When you call a void function, it simply executes the statements it contains and then terminates. When you call a value-returning function, it executes the statements that it contains, and then it returns a value back to the statement that called it.

Defining a Function

- A Function defined in Python should follow the following format:

1) Keyword **def** is used to start the Function Definition. def specifies the starting of Function block.

2) def is followed by function-name followed by parenthesis.

3) Parameters are passed inside the parenthesis.

4)At the end a colon is marked.

Syntax:

```
def <function_name>([parameters]):
</function_name>
```

- **Ex:**

```
def sum(a,b):
```

4) Before writing a code, an Indentation (space) is provided before every statement. It should be same for all statements inside the function.

5) The first statement of the function is optional. It is ?Documentation string? of function.

6) Following is the statement to be executed.

Syntax:

```
def  < function_name> ([ parameters]):
                                # where def is  a keyword
    statement_1
    statement_2
    ……
    …..
```

Invoking a Function:

- To execute a function it needs to be called. This is called function calling.
- Function Definition provides the information about function name, parameters and the definition what operation is to be performed. In order to execute the Function Definition it is to be called.
- **Syntax:**
- <function_name>(parameters)
- </function_name>
- eg:
- sum(a,b)

here sum is the function and a, b are the parameters passed to the Function Definition.

```
#This program demonstrates a function.
# First, we define a function named message.
def message( ):
    print("Iam working on it,")
    print("shall let you know once I finish it off")
# Call the message function.
message( )
```

- NOTE: A function definition specifies what a function does, but it does not cause the function to execute. To execute a function, you must call it. This is how we would call the message function

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

```
# This program has two functions. First we
# define the main function.
def main( ):
    print('I have a message for you.')
    message( )
    print('Goodbye!')
# Next we define the message function.
def message( ):
    print("Iam working on it,")
    print("shall let you know once I finish it off")
# Call the main function.
main( )
```

- **Ex:**

```
    #Providing Function Definition
def sum(x,y):
    "Going to add x and y"
    s=x+y
    print("Sum of two numbers is")
    print(s)
   #Calling the sum Function
sum(10,20)
sum(20,30)
```

Output:

- NOTE: Function call will be executed in the order in which it is called

return Statement:

- return[expression] is used to send back the control to the caller with the expression.
- In case no expression is given after return it will return None.
- In other words return statement is used to exit the Function definition.

- **Ex:**

```
def sum(a,b):
    "Adding the two values"
    print("Printing within Function")
    print(a+b)
    return (a+b)     # Try with out return
def msg():
    print("Hello")
    return
total=sum(10,20)
print("Printing Outside:",total)
msg( )
print("Rest of code")
```

Argument and Parameter:

- There can be two types of data passed in the function.

1) The First type of data is the data passed in the function call. This data is called arguments.

2) The second type of data is the data received in the function definition. This data is called parameters.

- Arguments can be variables and expressions.
- Parameters must be variable to hold incoming values.
- Alternatively, arguments can be called as actual parameters or actual arguments and parameters can be called as formal parameters or formal arguments.

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

- **Ex:**

```
def add(x,y):
    print(x+y)
x=15
add(x,10)
add(x,x)
y=20
add(x,y)
```

Output:

Passing Parameters

- Apart from matching the parameters, there are other ways of matching the parameters.
- Python supports following types of formal argument:

  1) Positional argument (Required argument).

  2) Default argument.

  3) Keyword argument (Named argument)

- **Positional/Required Arguments:**
- When the function call statement must match the number and order of arguments as defined in the function definition it is Positional Argument matching

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

- **Eg:** #Function definition of sum

```
def sum(a,b):
 "Function having two parameters"
 c=a+b
 print(c)
sum(10,20)
sum(20)
```

Output:

```
>>>
```

- **Explanation:**

1) In the first case, when sum() function is called passing two values i.e., 10 and 20 it matches with function definition parameter and hence 10 and 20 is assigned to a and b respectively. The sum is calculated and printed.

2) In the second case, when sum() function is called passing a single value i.e., 20 , it is passed to function definition. Function definition accepts two parameters whereas only one value is being passed, hence it will show an error.

Default Arguments

- Default Argument is the argument which provides the default values to the parameters passed in the function definition, in case value is not provided in the function call.

Eg: #Function Definition

```
def msg(Id,Name,Age=21):
        "Printing the passed value"
        print(Id)
        print(Name)
        print(Age)
        return
        #Function call
msg(Id=100,Name='Ravi',Age=20)
msg(Id=101,Name='Ratan')
```

- **Output:**

100

Ravi

20

101

Ratan

21

- **Explanation:**

1) In first case, when msg() function is called passing three different values i.e., 100 , Ravi and 20, these values will be assigned to respective parameters and thus respective values will be printed.

2) In second case, when msg() function is called passing two values i.e., 101 and Ratan, these values will be assigned to Id and Name respectively. No value is assigned for third argument via function call and hence it will retain its default value i.e, 21.

Keyword Arguments:

- Using the Keyword Argument, the argument passed in function call is matched with function definition on the basis of the name of the parameter.

Ex:

```
def msg(id,name):
        "Printing passed value"
        print(id)
        print(name)
        return
msg(id=100,name='Raj')
msg(name='Rahul',id=101)
```

- **Output:**

```
>>>
100
Raj
101
Rahul
```

- **Explanation:**
- 1) In the first case, when msg() function is called passing two values i.e., id and name the position of parameter passed is same as that of function definition and hence values are initialized to respective parameters in function definition. This is done on the basis of the name of the parameter.
- 2) In second case, when msg() function is called passing two values i.e., name and id, although the position of two parameters is different it initialize the value of id in Function call to id in Function Definition. same with name parameter. Hence, values are initialized on the basis of name of the parameter.

Lambda or Anonymous Function:

- ‘lambda keyword is used to define anonymous functions
- Anonymous functions are not defined in the standard manner using ‘def’ keword
- Can take any number of arguments but, return only one value in the form of an expression
- Cannot be direct call to print as lambda requires an expression
- They have their own namespace and cannot access variables in either in local or global namespaces

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

- **Syntax:**

  lambda [arg1,[args2,[args3…..argsn]:expression

EX:

```
#Function Definiton
total= lambda num1,num2:num1+num2
 # Function Call
print("Value of total:",total(10,20))
print("Value of total:", total(100,200))
```

Ex:

```
square=lambda a:a*a
print("Square of number is",square(10))
```

Output : Square of number is 100

- **Difference between Normal Functions and Anonymous Function:**

Have a look over two examples:

Eg:

Normal function:

```
#Function Definiton
def square(x):
    return x*x

#Calling square function
print("Square of number is",square(10))
```

- **Anonymous function:**

```
#Function Definiton
square=lambda x1: x1*x1

#Calling square as a function
print("Square of number is",square(10))
```

Explanation:

Anonymous is created without using def keyword.

lambda keyword is used to create anonymous function.

It returns the evaluated expression.

- **Scope of Variable:**
- Scope of a variable can be determined by the part in which variable is defined.
- Each variable cannot be accessed in each part of a program.
- There are two types of variables based on Scope:

1) Local Variable.

2) Global Variable.

1) Local Variables:

- Variables declared inside a function body is known as Local Variable. These have a local access thus these variables cannot be accessed outside the function body in which they are declared.

Eg:

```
def msg( ):
        a=10
        print("Value of a is",a)
        return
msg( )
print(a )#it will show error since variable is local
```

- **Output:**

```
>>>
Value of a is 10

Traceback (most recent call last):
  File "C:/Python27/lam.py", line 7, in <module>
   print a #it will show error since variable is local
NameError: name 'a' is not defined
>>>
</module>
```

- **b) Global Variable:**
- Variable defined outside the function is called Global Variable. Global variable is accessed all over program thus global variable have widest accessibility.
- **Ex:**

```
b=20
def msg( ):
        a=10
        print ("Value of a is",a)
        print ("Value of b is",b)
        return
        msg( )
        print (b)
```

- **Output:**

```
>>>
Value of a is 10
Value of b is 20
20
>>>
```

Recursion

- A method invoking itself is referred to as a Recursion
- Typically when a program employs recursion the function invokes itself with a smaller argument
- Computing factorial(5) involves computing factorial(4), computing factorial(4) involves computing factorial(3) and so on
- Try out with Factorial & GCD Problems

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language