PYTHON FUNCTIONS

Prof. Rachana V. Modi

Python Functions

- Functions are the most important aspect of an application.
- A function can be defined as the organized block of reusable code which can be called whenever required.
- Python allows us to divide a large program into the basic building blocks known as function.
- Python provide us various inbuilt functions like range() or print().
- Although, the user can create its functions which can be called user-defined functions.

- **Advantage of Functions in Python**
  - we can avoid rewriting same logic/code again and again in a program.
  - We can call python functions any number of times in a program and from any place in a program.
  - We can track a large python program easily when it is divided into multiple functions.
  - Reusability is the main achievement of python functions.
  - However, Function calling is always overhead in a python program.

Creating a function

- In python, we can use **def** keyword to define the function.
- Syntax:

```
def my_function():
    function-suite
    return <expression>
```

- The function block is started with the colon (:) and all the same level block statements remain at the same indentation.
- A function can accept any number of parameters that must be the same in the definition and function calling.
- 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.

Function calling

- A function must be defined before the function calling otherwise the python interpreter gives an error.
- Once the function is defined, we can call it from another function or the python prompt.
- To call the function, use the function name followed by the parentheses.
- Example:

```
def hello_world():
    print("hello world")

hello_world()
```

Parameters in function

- The information into the functions can be passed as the parameters.
- The parameters are specified in the parentheses.
- We can give any number of parameters, but we have to separate them with a comma.
- **Example 1:**

```
def func (name):     #defining the function
    print("Hi ",name);

func("Shidharth")        #calling the function
```

Example 2:

```
#python function to calculate the sum of two variables

def sum (a,b):                          #defining the function
    return a+b;

a = int(input("Enter a: "))      #taking values from the user
b = int(input("Enter b: "))

print("Sum = ",sum(a,b))        #printing the sum of a and b
```

Call by reference

- All the functions are called by reference.
- All the changes made to the reference inside the function revert back to the original value referred by the reference.
- **Example 1 Passing Immutable Object (List)**

```
def change_list(list1):                          #defining the function
    list1.append(20);
    list1.append(30);
    print("list inside function = ",list1)

list1 = [10,30,40,50]                             #defining the list
change_list(list1);                               #calling the function
print("list outside function = ",list1);
```

Output:
```
list inside function = [10, 30, 40, 50, 20, 30]
list outside function = [10, 30, 40, 50, 20, 30]
```

- **Example 2 Passing Mutable Object (String)**

```
def change_string (str):          #defining the function
    str = str + " Hows you";
    print("printing the string inside function :",str);

string1 = "Hi I am there"

change_string(string1)           #calling the function

print("printing the string outside function :",string1)
```

Output: printing the string inside function : Hi I am there Hows you
printing the string outside function : Hi I am there

Types of arguments

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

Required arguments

- The arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition.
- If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.
- **Example:**

```
def func(name):
    message = "Hi "+name;
    return message;
name = input("Enter the name?")
print(func(name))
```

Keyword arguments

- This kind of function call will enable us to pass the arguments in the random order.
- The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.
- Example:

```
def func(name,message):
    print("printing the message with",name,"and ",message)
func(name = "John",message="hello")
#name and message is copied with the values John and hello respectively
```

Default Arguments

- To initialize the arguments at the function definition.
- If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.
- Example:

```
def printme(name,age=22):
    print("My name is",name,"and age is",age)
printme(name = "john")
```

Variable length Arguments

- In the large projects, sometimes we may not know the number of arguments to be passed in advance.
- In such cases, Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.
- However, at the function definition, we have to define the variable with * (star) as *<variable - name >.
- **Example:**

```
def printme(*names):
    print("type of passed argument is ",type(names))
    print("printing the passed arguments...")
    for name in names:
        print(name)
printme("john","David","smith","nick")
```