Practical No 3

Aim: - Implementing functions in Python.

Theory: - A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.
A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Defining a Function
You can define functions to provide the required functionality. 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.

Syntax
def functionname(parameters):
   "function_docstring"
    function_suite
    return [expression]

Creating a Function
In Python a function is defined using the **def** keyword:

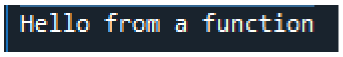
```
def my_function():
  print("Hello from a function")
```

Calling a Function
To call a function, use the function name followed by parenthesis:

```
def my_function():
  print("Hello from a function")

my_function()
```

Output:



```
Hello from a function
```

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.

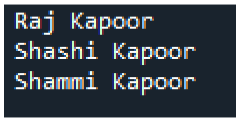
The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Program:

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

my_function("Raj")
my_function("Shashi")
my_function("Shammi")
```

Output:



```
Raj Kapoor
Shashi Kapoor
Shammi Kapoor
```

Parameters or Arguments

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

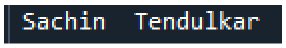
Program:

This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Sachin", "Tendulkar")
```

Output:



```
Sachin  Tendulkar
```

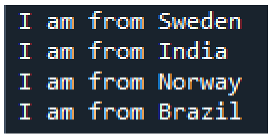
Default Parameter Value

The following example shows how to use a default parameter value.
If we call the function without argument, it uses the default value:
Program:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Output:



```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Passing a List as an Argument

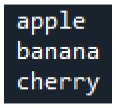
You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
E.g. if you send a List as an argument, it will still be a List when it reaches the function:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Output:



```
apple
banana
cherry
```

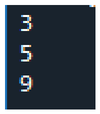
Return Values

To let a function return a value, use the return statement:

```
def my_function(x):
  return x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output:



```
3
5
9
```