**Experiment Number – 9**

**Title – Write a program to count digits of number using function**

**Theory –**

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]

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

### Example of a User-Defined Function

We will define a function that when called will return the square of the number passed to it as an argument.

# Example Python Code for User-Defined function

def square():

    " This function computes the square of the number."

     num = int(input(“Enter a number ”)

return num\*\*2

object = square()

print( "The square of the given number is: ", object )

### ****Creating a function with parameter****s

Let’s create a function that takes two parameters and displays their values.

In this example, we are creating function with two parameters ‘ name’ and ‘age’.

# function

def course\_func(name, course\_name):

print("Hello", name, "Welcome to FYMCA")

print("Your course name is", course\_name)

# call function

course\_func('Arun', 'Python')

## Return Values

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

def my\_function(x):  
  **return 5 \* x**  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

## The pass Statement

Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

def myfunction():  
  pass

## Scope and Lifetime of Variables

When we define a function with [variables](https://pynative.com/python-variables/), then those variables’ scope is limited to that function. In Python, the scope of a variable is an area where a variable is declared. It is called the variable’s local scope.

We cannot access the local variables from outside of the function. Because the scope is local, those variables are not visible from the outside of the function.

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

If you use the global keyword, the variable belongs to the global scope:

def myfunc():  
  global x  
  x = "fantastic"  
myfunc()  
print("Python is " , x)

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

x = "awesome"  
def myfunc():  
  global x  
  x = "fantastic"  
myfunc()  
print("Python is " ,x)

**Nested Functions**

A function which is created inside another function is called a **nested function** or an **inner function**. A nested function is defined by simply creating it using the def keyword inside another function. Here is an example.

def outer(): # outer function

print("This is outer function")

def inner():

print("This is inner function")

inner() # calling inner function

outer() # calling outer function

Output

This is outer function

This is inner function

We defined a function named inner() inside another function named outer(). Thus, inner() is a nested function. When the outer() function is called, its body is executed. Inside its body, the inner() function is defined and then called. Thus, we first called the outer() function which in turn called the inner() function. Note that an inner function is always called from inside the function in which it is defined. Thus, to call the inner function, we need to call the outer function. A nested function can access the variables defined in the function in which it is created. This is demonstrated in the following example.

def outer(): # outer function

x = 10

def inner():

print("Inside inner func", x)

inner() # calling inner function

print("Inside outer func", x)

outer() # calling outer function

Output

Inside inner func 10

Inside outer func 10

In this example, we defined a variable x having a value of 10 in the outer() function. When we printed the value of x in the nested function, it got printed. This means that the nested function can access the variable x defined in its parent function.

Exercise-

1. Write a Python function to multiply all the numbers in a list.
2. Write a Python function that accepts a string and counts the number of upper and lower case letters.
3. Write a Python function that takes a list and returns a new list with distinct elements from the first list.
4. Write a Python program to access a function inside a function.
5. Write a Python function that accepts lowercase words and returns uppercase words.