**Q 1.** In Python, what is the difference between a built-in function and a user-defined function? Provide an example of each.

**Answer** –

**Built-in function** - A built in function is a function that is already defined in python. These functions are ready to use, we can simply start utilizing these functions without any additional code.

Example - **type(), print() etc.**

**User Defined function -** User defined function is defined by the user. The user can write custom code inside the function according to the need of the user. ***def*** keyword is used to define the user defined function.

Example -

```
def sum(a,b):

   return a+b

sum(1,2) # calling the function
```

Output - 3

**Q 2.** How can you pass arguments to a function in Python? Explain the difference between positional arguments and keyword arguments.

**Answer** –

Following are the ways to pass arguments to a function in python

- Default Arguments - In this case the functions are having default parameter values. This means that if the user does not provide any arguments while calling the function then the default values are picked. Following we define a function with default arguments and while calling the function we don't give any argument, in this case the default values are picked by the function.

```
 def test_function(name="akhil", message="hello"):
            print(message + ", " + name + "!")
test_function()
```

- Positional arguments - These are the arguments that are provided in a order. Following we define a test function with name and message parameters. While calling the function we provide the arguments. The first value is assigned to the first parameter of the function, the second value is assigned to the second parameter and so on.

```
def test_function(name, message):
        print(message + ", " + name + "!")
test_function(“akhil”,”hello”)
```

- Keyword arguments - These are the  arguments that are passed to a function using the parameter names explicitly. The order of arguments does not matter. Following we provide the arguments by specifying the parameter names.

```
def test_function(name, message):
        print(message + ", " + name + "!")
test_function(name=“akhil”,message=”hello”)
```

**Q 3.** What is the purpose of the return statement in a function? Can a function have multiple return statements? Explain with an example.

**Answer** –

The return statement in a function helps us exit the function. The return statement takes the program execution back to the function calling statement with a certain value that the function wants to send/return. Yes, a function can have multiple return statements. This is possible when we have if statements inside the function and each condition wants us to return different values.

Example:

```
def test_evenodd(number):
    if (number %2 ==0):
        return "even"
    else:
        return "odd"
```

**Q 4.** What are lambda functions in Python? How are they different from regular functions? Provide an example where a lambda function can be useful.

**Answer** –

Lambda functions are unnamed functions defined using the *lambda* keyword. Lambda functions are different from regular functions in following ways

Naming - The regular functions have a name and lambda functions are anonymous.

Lines of code - The lambda functions are defined in a single expression, whereas the regular functions can have multiple lines of code.

Functionality - The lambda functions have limited functionality due to their concise structure. Whereas regular functions can have varied levels of functionality.

Example -

```
x = lambda a : a + 10

print(x(5))
```

**Q 5.** How does the concept of "scope" apply to functions in Python? Explain the difference between local scope and global scope.

**Answer** –

*Scope* in python refers to the accessibility of a variable inside the program. Following are different types of *scopes* that come into picture when we talk about functions.

Local scope -The scope of the variables are limited to inside the functions. Which means we can access the variable only inside the function.

Global scope - The variables defined outside the function can also be accessed inside the function. Such variables are said to have a global scope.

Enclosed scope - When a function contains another function inside it then in that case the variables inside the nested function is said to have enclosed scope. This means that the variable can be accessed only inside the nested scope.

The difference between a local scope and global scope is that the variable with global scope is accessed throughout the program. Whereas the variable with local scope is accessed inside only a particular function.

**Q 6.** How can you use the "return" statement in a Python function to return multiple values?

**Answer** – To return multiple values from python functions we need to separate the variables by a comma. The values returned will be in a tuple format by default. Following is the example where we return multiple values from a function. The variable *test_tuple* will store the values returned in the form of a tuple. Second way to store the returned values is to assign them to respective variables as shown below.

```
def test_function():

        first_name = "Akhil"
        last_name = "Kumar"
        age = 30
        return first_name, last_name, age

test_tuple = test_function()
return first_name, last_name, age = test_function()
print(test_tuple)
```

**Q 7.** What is the difference between the "pass by value" and "pass by reference" concepts when it comes to function arguments in Python?

**Answer** –

Pass by value - In this case the copy of the value of the variable is passed to the function. This means that any changes that are made to the variable inside the function does not affect the original value. In python this concept applies to immutable data types such as string, tuples. In the following example we can see we defined an integer variable var2. We passed this to a function where 1 is added to it. We can see that change does not reflect outside the function.

Pass by reference - In this case the reference/ memory address of the variable is provided to the function. This means that the changes made to the variable inside the function will also reflect outside the function. In the case of python this happens when a variable with mutable data type is given to the function. Following we have a variable *var1* which stores a list. We provide it as an argument to a function, where we make changes to this list. We can see that the changes made to this list inside the function also gets reflected outside the function as well.

```
var1= [1,2,3]
var2= 1
print(var1)
def test_function(var1,var2):
        var1.append(4)
        var2= var2+1
        print(var2)
```

```
test_function(var1,var2)
print(var1)
print(var2)
```

Output -

```
[1, 2, 3]
2
[1, 2, 3, 4]
1
```

**Q 8.** Create a function that can intake integer or decimal value and do following operations:

a. Logarithmic function (log x)
b. Exponential function (exp(x))
c. Power function with base 2(2^x)
d. Square root

**Answer** –

```
import math
def make_operations(number):
    log = math.log(number)
    exp = math.exp(number)
    power = math.pow(2,number)
    sqrt = math.sqrt(number)
    return log, exp, power, sqrt
```

**Q 9.** Create a function that takes a full name as an argument and returns first name and last name.

**Answer** –

```
def get_names(fullname):
    fn = fullname.split(" ")
    return fn[0], fn[1]

first_name, last_name = get_names("akhil kumar")
print("first name is :", first_name)
print("last name is :", last_name)
```

Output -

```
first name is : akhil
last name is : kumar
```