1. Why are functions advantageous to have in your programs?

A function f(x) is a piece of code that performs a specified task when called or executed. It may or may not return any value.
Functions are advantageous in two main possible manners:

- **Avoiding repeatability of the code**: There may be certain set of statements that need to be executed multiple times with varying inputs at different segments of the program. The set of sentences that need to repeated can be stored as a function, f(inputs) and called at the whenever required by executing just line of the code instead of retyping the whole set of statements.
- **Compartmentalising the code:** Certain parts of the code may involve some significant complexity. These can be defined at isolated locations and just be called whenever required so that the basic structure of the code remains simple and easy to understand.

2. When does the code in a function run: when it's specified or when it's called?

Function definition is process of defining the purpose of the particular function. A function by itself contains the bunch of statements which performs the desired task. But the function does not get executed at the definition phase. Function gets executed only when the function is called. 'Function call' is the statement that tells the program to call the function. If the function takes some inputs, called arguments, this can be supplied at the time of function of function call.

Following example defines a function to take 3 integers, a, b and c as inputs, evaluates a^(b+c) and returns the value of a^(b+c) when called.

```
#Function Definition
#a, b and c are the parameters that the function definition exp_function() takes as inputs at the time
of execution
def exp_fucntion(a,b,c):
        output= a**(b+c)
        return output    #return statement returns the value specified at the time of function call

#Main section of the code
x,y,z=input('Enter three integers delimited by single whitespace ' ' : ').split()
x=int(x)
y=int(y)
z=int(z)

#Function Call, this is the part of the main code where the function is called and it gets executed
#x,y and z are supplied as the input arguments to the functions which will replace the positional

arguments defined in the function definition
desired_output=exp_function(x,y,z)
print(f'The desired output returned by the function is {desired_output}')
```

The same code has been stored in GitHub repository in a Jupyter notebook format:
https://github.com/arvindhhp/iNeuronAssignments/blob/main/Python_Basics/Python_Basics_Assignment_3_Question_2.ipynb

3. What statement creates a function?

Function definition statement creates a function. It is represented by the following statement (def is the keyword for function definition)

def f(x):

4. What is the difference between a function and a function call?

Function definition is process of defining the purpose of the particular function. A function by itself contains the bunch of statements which performs the desired task. But the function does not get executed at the definition phase. Function gets executed only when the function is called. 'Function call' is the statement that tells the program to call the function. If the function takes some inputs, called arguments, this can be supplied at the time of function of function call.

Refer to **Question 2** for example and additional explanation.

5. How many global scopes are there in a Python program? How many local scopes?

There is only **ONE** global scope in Python

The number of local scopes is equal to the **number of time a particular function is called** (or executed) within which the variable is locally defined.

6. What happens to variables in a local scope when the function call returns?

The expression in the return statement of the function call gets evaluated based on the values stored within the local scope of the variables (unless global scope is explicitly mentioned).

Once the function gets called and the return statement is executed, Python goes back to global scope and the local scope variable and the data stored in it is erased from the memory.

7. What is the concept of a return value? Is it possible to have a return value in an expression?

'return' is statement at the end of a function definition that controls whether the function should return any value when called. A return statement is not mandatory. However, a return could have anything defined there, it could be just a variable retuned or an expression or even a print statement.
For example:

- In the example code in Question 2, the return statement returns the value of the local variable 'output' evaluated by the function
- We can also rewrite the return statement by replacing the variable name with the expression 'a**(b+c)', i.e ' return a**(b+c)'. Even this will execute the function and return the same value at the time of function call.

8. If a function does not have a return statement, what is the return value of a call to that function?

- When the function definition does not have any return statement, it returns nothing. This can be stated as the function returns 'None'.
- 'None' refers to the fact that function returns nothing.

9. How do you make a function variable refer to the global variable?

Usage of the keyword 'global' ahead of the variable name makes the function variable to refer the global variable itself.

An example code explaining the global reference using global() has been stored in GitHub repository in a Jupyter notebook format:
https://github.com/arvindhhp/iNeuronAssignments/blob/main/Python_Basics/Python_Basics_Assignment_3_Question_9.ipynb

10. What is the data type of None?

It is 'NoneType'

11. What does the sentence import areallyourpetsnamederic do?

It imports the module 'areallyourpetsnamederic'

12. If you had a bacon() feature in a spam module, what would you call it after importing spam?

Assuming spam module is already imported, the following statement will call bacon()

spam.bacon()

13. What can you do to save a programme from crashing if it encounters an error?

Usage of "Exception Handling" prevents the program from crashing on encountering any error. It consists of try and except blocks.

- The 'try' block contains the actual code block to be executed.
- The 'except' block contains set of commands to be executed when a particular type of error is encountered within the 'try' block.

14. What is the purpose of the try clause? What is the purpose of the except clause?

Usage of "Exception Handling" prevents the program from crashing on encountering any error. It consists of try and except blocks.

- The 'try' block contains the actual code block to be executed.
- The 'except' block contains set of commands to be executed when a particular type of error is encountered within the 'try' block.

An example code explaining the usage of try and except clauses has been stored in GitHub repository in a Jupyter notebook format:
https://github.com/arvindhhp/PyPro_ahhp/blob/main/Part_004_ListOperations.ipynb
(Refer to the sample program in the 3rd last cell of the notebook)