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

Functions are an essential concept in programming, including in Python:

1. Modularization and Code Organization: Functions allow you to break down a complex problem into smaller, manageable pieces. Each function can represent a specific task or operation,

2. Reusability 3. Readability and Maintainability 4. Abstraction and Encapsulation:

5. Testing 6. Code Reusability

7. Parameterization: Functions can take input parameters, allowing you to create flexible and customizable solutions.

8. Collaboration: When working on a larger project, functions allow multiple developers to work on different parts of the code simultaneously, knowing that they are interacting with well-defined interfaces.

9. Libraries and Modules: In Python, functions can be grouped into libraries and modules, which can be imported and used in other programs.

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

The code within a function runs when the function is called, not when it is specified or defined. In Python, defining a function involves creating a blueprint or template for a specific set of instructions, but those instructions are executed only when the function is invoked (called) during the runtime of the program.

def my\_function():

print("This is inside the function.")

print("This is outside the function.")

my\_function()

1. **What statement creates a function?**

In Python, the `def` statement is used to create a function. The `def` statement defines a function by specifying its name, parameters, and the code block (body) that will be executed when the function is called.

Here's the basic syntax :

def function\_name(parameters):

# Function body

# Code to be executed when the function is called

# ...

1. **What is the difference between a function and a function call?**

1. Function:

A function is a block of code that performs a specific task or set of tasks. It encapsulates a series of instructions that can be executed whenever the function is called. Functions are defined using the `def` statement, and they can have parameters (inputs) and a return value (output). Functions provide modularity, code reusability, and abstraction.

2. Function Call:

A function call (also known as invoking a function) is the action of actually using the function that you have defined. It involves providing the necessary arguments (inputs) to the function, which are then used to execute the code within the function body. When a function is called, the program jumps to the function's definition, executes its code, and returns the result (if applicable) back to the caller.

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

In a Python program, there is one global scope and multiple local scopes.

1. Global Scope: The global scope refers to the outermost level of the program, where variables and functions are defined outside of any function or class. Variables defined in the global scope are accessible throughout the entire program.

2. Local Scopes: Local scopes are created within functions or blocks of code. Each function or block creates its own local scope. Variables defined within a local scope are only accessible within that specific block

**Example:**

global\_variable = 10 # This is a global variable

def my\_function():

local\_variable = 5 # This is a local variable within the function

print(global\_variable)

print(local\_variable)

my\_function()

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

When a function call returns in Python, the variables that were defined within the local scope of that function are no longer accessible. This is because local variables have a limited lifetime and scope—they are created when the function is called and destroyed when the function call completes.

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

The concept of a return value refers to the value that a function provides as output when it is executed. In Python, functions can have a return value, which is a way for the function to communicate information back to the caller.

Yes, it is possible to have a return value in an expression. This allows you to directly use the return value of a function within an expression, without the need to assign it to a separate variable first.

def multiply(a, b):

return a \* b

result = multiply(4, 6) + 2

print(result) # This will print 26 (4 \* 6 + 2)

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

If a function does not have a `return` statement, the return value of a call to that function is `None`. In Python, `None` is a special built-in object that represents the absence of a value or a null value.

**EXAMPLE:**  def greet(name):

print(f"Hello, {name}!")

result = greet("Alice")

print(result) # This will print: None

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

In Python, to make a function variable refer to a global variable, you need to use the `global` keyword within the function. This allows you to explicitly indicate that you want to work with the global variable of the same name, rather than creating a new local variable within the function.

**Syntax:**

global variable\_name

**Example :**

global\_variable = 10

def modify\_global():

global global\_variable

global\_variable = 20

print("Before function call:", global\_variable)

modify\_global()

print("After function call:", global\_variable)

1. **What is the data type of None?**

In Python, `None` is a special constant that represents the absence of a value or a null value. It is often used to indicate that a variable or a function does not have a meaningful value or result. The data type of `None` is itself, and it is typically considered a singleton object.

**Example:**

result = None

print(type(result)) # Output: <class 'NoneType'>

1. **What does the sentence import areallyourpetsnamederic do?**

The sentence "import areallyourpetsnamederic" is not a valid Python import statement and would result in a `ModuleNotFoundError`.

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

After importing the `spam` module in Python, you would call the `bacon()` function from the `spam` module using the following syntax:

import spam

spam.bacon()

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

To prevent a Python program from crashing when it encounters an error, you can use error handling techniques. These techniques allow you to gracefully handle exceptions (errors)

1. Try-Except Blocks

2. Exception Handling with Multiple Except Blocks

3. Using the `else` Clause

4. Using the `finally` Clause

5. Exception Base Classes

6. Raising Custom Exceptions

1. **What is the purpose of the try clause? What is the purpose of the except clause?**

In Python, the `try` and `except` clauses are used for error handling and exception handling. They provide a way to gracefully handle exceptions (errors)

1. **Try Clause:**

The `try` clause is used to enclose the code that might potentially raise an exception. You place the code that you want to monitor for exceptions within this block. If an exception occurs within the `try` block, the program will jump to the corresponding `except` block (if one is defined) to handle the exception.

try:

# Code that might raise an exception

result = 10 / 0 # This will raise a ZeroDivisionError

except ZeroDivisionError:

print("Error: Division by zero")

2**. Except Clause:**

The `except` clause follows the `try` clause and defines what actions should be taken if a specific type of exception occurs within the `try` block. When an exception of the specified type occurs, the code within the corresponding `except` block is executed.

try:

# Code that might raise an exception

result = int("abc") # This will raise a ValueError

except ValueError:

print("Error: Invalid value")

except ZeroDivisionError:

print("Error: Division by zero")

```