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

Ans:Functions are advantageous to have in your programs for several reasons:

i. Modularity and Reusability: Functions allow you to break down your code into smaller, manageable, and reusable blocks of code. Once a function is defined, it can be called multiple times from different parts of the program, reducing code duplication and promoting modularity.

ii.Code Organization: Functions help in organizing your code logically. By grouping related operations into functions, you can improve the overall structure and readability of your program, making it easier to understand and maintain.

iii.Abstraction: Functions provide an abstraction layer, hiding the implementation details from the rest of the program. This allows you to focus on what the function does rather than how it does it, making the code easier to comprehend.

iv. Testing: Functions make it easier to isolate and test specific parts of your program. Since functions are self-contained units of code, it becomes simpler to identify and fix bugs when they occur.

v. Team Collaboration: In larger projects, functions enable team members to work on different parts of the code independently. This division of labor can enhance collaboration and facilitate parallel development.

vi.Performance Optimization: By encapsulating frequently used code blocks within functions, you can optimize the code for better performance. Functions can also help in reducing the computational complexity by breaking down complex tasks into smaller, manageable steps.

vii.Library and Framework Integration: Functions are fundamental building blocks in programming, and most libraries and frameworks are designed to work with functions. By using functions, you can easily integrate external code and take advantage of pre-existing functionality.

viii Code Maintainability: As your program grows in size and complexity, functions help in maintaining a clear structure and make the codebase more maintainable. Making changes to a specific function is easier than trying to modify a large monolithic codebase.

Ix.Encapsulation and Security: Functions provide a level of encapsulation, allowing you to control access to certain parts of the code. This helps in securing sensitive or critical parts of the program from unintended modifications.

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

Ans:The code within a function runs when the function is called, not when it is specified or defined. Functions are blocks of code designed to perform a specific task or set of tasks, and they are created to be executed only when invoked by their name.

Here, the explanation how function works:

i. Function definition: When you define a function, you are essentially creating a blueprint for a specific set of operations. The code inside the function body specifies what actions the function will perform when called.

Example of function definition in Python:

def greet():

print("Hello, there!")

ii.Function call: To execute the code inside the function, you need to call the function by using its name followed by parentheses.

Example of function call in Python:

greet() # This will print "Hello, there!"

When the function is called (`greet()`), the code inside the function (`print("Hello, there!")`) will be executed, and "Hello, there!" will be printed to the console.

3. What statement creates a function?

Ans:A function in Python is created using the `def` keyword. Here's how to create a function:

i . Use the `def` keyword to start defining the function.

ii.. Choose a name for your function (follow the variable naming rules).

iii.If the function takes inputs, put them inside parentheses after the function name.

iv. Write the code for what you want the function to do inside the function body (indented under the `def` line).

V. If the function should return a value, use the `return` keyword followed by the value you want to return.

Example of a simple function that adds two numbers and returns the result:

def add\_numbers(a, b):

result = a + b

return result

To use this function, you can call it with specific values for `a` and `b`:

sum\_result = add\_numbers(2, 3)

print(sum\_result) # Output: 5

This will call the `add\_numbers` function with `a=2` and `b=3`, and it will return the sum (5), which will be stored in the `sum\_result` variable and printed to the console.

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

Ans:

\*\*Function:\*\*

- A function is like a recipe. It's a set of instructions that define a specific task or action.

- You create a function using the `def` keyword and give it a name.

- Inside the function, you write the code that does the task you want.

- Functions are like reusable tools that you can use whenever you need to do that task.

Example of a simple function in Python:

def say\_hello():

print("Hello, there!")

```

\*\*Function Call:\*\*

- A function call is like using that recipe to cook something.

- When you "call" a function, you're telling the program to execute the code inside the function.

- You call a function by using its name followed by parentheses.

Example of a function call in Python:

say\_hello() # This will print "Hello, there!"

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

Ans:In a Python program, there is always one global scope, and local scopes are created whenever a function is called.

i. Global Scope:

The global scope refers to the top-level scope of a Python program. It exists outside of any functions or classes and includes all the variables and functions defined at the module level. There is only one global scope per Python module.

ii. Local Scopes:

Local scopes are created whenever a function is called. Each time a function is invoked, it creates its own local scope, and any variables defined inside that function will belong to this local scope. When the function execution ends, the local scope is destroyed.

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

Ans:When a function finishes running and returns, any variables that were created inside the function (local variables) disappear. They only exist while the function is running and cannot be accessed from outside the function.

Imagine the function as a small workspace. Once the function is done, that workspace is cleaned up, and anything inside it vanishes, leaving the rest of the program unaffected.

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

Ans:i. Return Value: When you use a function, it can give you a result back, which is called the "return value." It's like a gift from the function to the rest of the code that called it.

ii.Example: Imagine you have a function called `add\_numbers(a, b)` that adds two numbers `a` and `b` and gives you the answer back. When you call `add\_numbers(3, 5)`, it returns `8`. You can use this result in your code, like storing it in a variable or using it in expressions, just like you would with any other number.

In short, a return value is like a little present from a function, and you can use it in your code to do whatever you want!

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

Ans:

If a function doesn't have a `return` statement, it automatically returns a special value called `None`.

Imagine you're asking someone to do a task for you, but they forget to give you anything back. In Python, when you call a function without a `return` statement, it's like the function forgetting to give you a return value, so it gives you `None` instead.

So, if you call a function that doesn't have a `return` statement, the return value is `None`.

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

Ans:

To make a function variable refer to the global variable with the same name, you need to use the `global` keyword before the variable inside the function.

Example:

global\_variable = 10

def access\_global():

global global\_variable

print("Inside the function:", global\_variable)

access\_global() # Output: Inside the function: 10

In this example, the function `access\_global` uses the `global global\_variable` statement inside the function to refer to the global variable `global\_variable`. When you call `access\_global()`, it prints the value of the global variable.

10. What is the data type of None?

Ans:The data type of `None` in Python is called "NoneType." `None` is a special value that represents the absence of a value or a null value.

In Python, you can use `None` as a placeholder for variables that are not assigned a value or to indicate that a function doesn't return anything. It is often used to represent the absence of a meaningful value.

To check the data type of `None`, you can use the `type()` function:

result = None

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

As we can see, `type(result)` returns `<class 'NoneType'>`, confirming that `None` is of type "NoneType."

11. What does the sentence import areallyourpetsnamederic do?

Ans:The sentence "import areallyourpetsnamederic" doesn't have any meaning or function in standard Python programming. It is not a valid import statement because there is no module called "areallyourpetsnamederic" in Python.

In Python, the `import` statement is used to include external modules or libraries into your program, but it requires the name of a valid module. Without a valid module name, the import statement won't do anything useful, and it may raise an error.

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

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

import spam

spam.bacon()

In this example, we assume that the "spam" module contains a function called `bacon()`. When you import the "spam" module using `import spam`, you can access the `bacon()` function with `spam.bacon()`. The dot (`.`) is used to access the functions or variables inside the imported module.

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

Ans:To prevent a program from crashing when it encounters an error, you can use `try` and `except` blocks in Python. Here's how it works:

1. `try`: You put the code that might cause an error inside the `try` block.

2. `except`: If an error occurs in the `try` block, Python jumps to the corresponding `except` block instead of crashing. You can handle the error gracefully in the `except` block.

Example:

try:

result = 10 / 0 # This might cause a division by zero error

print(result) # This line won't be executed if there's an error

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

```

In this example, the `try` block contains the division operation, which may cause a "division by zero" error. If that happens, Python immediately jumps to the `except` block, which prints an error message instead of crashing.

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

Ans:1. Purpose of the `try` clause:

The `try` clause is like a protective shield around code that might cause an error. It allows you to test this code and see if any errors occur while running it.

2. Purpose of the `except` clause:

The `except` clause is like a safety net that catches errors that occurred inside the `try` block. If an error happens, instead of crashing the program, the `except` block jumps into action and deals with the error gracefully.