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

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

1. Reusability: Functions allow you to encapsulate a piece of code that performs a specific task. This code can then be reused multiple times throughout your program or in different programs altogether. By defining functions, you avoid duplicating code, which promotes cleaner and more maintainable code.

2. Modularity: Functions promote modularity by breaking down a program into smaller, manageable chunks. Each function can focus on a specific task, making the overall program easier to understand and debug. This modular structure also enables multiple developers to work on different parts of a program simultaneously.

3. Abstraction: Functions allow you to abstract away complex implementation details and provide a higher-level interface. This means that the users of your function don't need to know the internal workings of the function, but can still use it to accomplish a specific task. This abstraction helps in reducing complexity and making the code more readable.

4. Readability and Maintainability: Functions enhance the readability and maintainability of your code. By using meaningful function names, you can convey the purpose of a particular piece of code. Moreover, breaking down a program into smaller functions makes it easier to understand and maintain, as each function has a specific responsibility.

5. Testing and Debugging: Functions make it easier to test and debug your code. You can isolate a function and test it independently, ensuring that it behaves correctly. When issues arise, functions help in narrowing down the problem area, allowing you to focus your debugging efforts on a specific function.

6. Code Organization: Functions provide a structured approach to organizing your code. You can group related code together in functions, making it easier to navigate and locate specific functionality within your program. This organization is particularly useful as programs grow larger and more complex.

1. **When does the code in a function run: when it&#39;s specified or when it&#39;s called?**

Ans - The code inside a function runs when the function is called, not when it is specified or defined.

Defining a function involves writing the code block that will be executed when the function is called, but the actual execution of that code occurs only when the function is invoked or called in your program.

For example, consider the following function definition:

```

def greet():

print("Hello, world!")

```

In this case, the code inside the function is `print("Hello, world!")`. This code will only execute when the function `greet()` is called.

To call the function and execute its code, you would write:

```

greet()

```

When this line of code is encountered during program execution, it triggers the execution of the code within the `greet()` function, resulting in the output `Hello, world!`.

So, remember that defining a function sets up the code that will run when the function is called, and the actual execution of that code occurs when the function is invoked in your program.

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

Ans - In Python, the `def` statement is used to create a function.

The syntax for creating a function using the `def` statement is as follows:

```python

def function\_name(parameters):

# Function code block

# ...

# ...

# (optional) return statement

```

Here's a breakdown of the components:

- `def`: This keyword is used to indicate the start of a function definition.

- `function\_name`: This is the name you choose for your function, following the Python naming conventions. It should be a descriptive and meaningful name.

- `parameters`: These are optional input parameters that you can define for your function. They are enclosed in parentheses and separated by commas. Parameters allow you to pass data into the function for it to operate on.

- Function code block: This is the indented block of code that constitutes the body of the function. It contains the instructions or operations that the function performs when called. This block is executed when the function is invoked.

- `return` statement (optional): A function can have a `return` statement to specify the value or values that the function should return when it finishes executing. If no `return` statement is present, the function will return `None` by default.

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

Ans - A function and a function call are two related but distinct concepts in programming.

A function is a named block of code that performs a specific task or set of tasks. It encapsulates a sequence of instructions that can be executed when the function is called. Functions are defined using the `def` statement in Python and can take parameters, perform operations, and optionally return a value.

A function is a named block of code that defines a specific task, while a function call is the actual execution of that function, using the function name followed by parentheses and passing any required arguments. Functions are defined once and can be called multiple times, allowing for code reuse and modular programming.

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

Ans - In a Python program, there is only one global scope, which is the outermost scope accessible throughout the entire program. The global scope is created when the program starts and remains active until the program terminates. Variables defined in the global scope are accessible from any part of the program, including inside functions.

On the other hand, the number of local scopes in a Python program can vary based on the number of functions and nested code blocks present in the program. Each time a function is called or a code block is entered, a new local scope is created. Local scopes are temporary and exist only during the execution of the function or code block. Variables defined inside a local scope are only accessible within that scope and its nested scopes.

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

Ans -When a function call returns in Python, the local scope associated with that function is destroyed, and any variables defined within that local scope cease to exist.

Here's what happens to variables in a local scope when a function call returns:

1. Variable Deletion: All the variables defined within the local scope of the function are deleted. This means that the memory occupied by those variables is freed up and can be reused by other parts of the program.

2. Scope Cleanup: Any other cleanup operations specific to the local scope are performed. For example, if there were any open files or acquired resources within the function, they are typically closed or released during this phase.

3. Control Returns: The program flow returns to the point immediately after the function call that initiated the local scope. The execution continues from that point, potentially using the return value (if any) from the function call.

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

Ans - The concept of a return value in programming refers to the value that a function can send back to the caller after its execution. When a function finishes executing, it can use the `return` statement to specify a value (or multiple values) that will be returned to the caller.

The return value serves as the result of the function's operation and can be used by the caller in further computations, assignments, or any other desired use. The return value can be of any valid data type in Python, such as numbers, strings, lists, or even custom objects.

So, return values allow functions to provide computed results back to the caller, and these return values can be used in expressions or passed as arguments to other functions.

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

Ans - If a function does not have a `return` statement or if it reaches the end of the function without encountering a `return` statement, the return value of a call to that function is `None`.

`None` is a special built-in value in Python that represents the absence of a value. It is often used to indicate the lack of a meaningful or specific result.

Here's an example of a function without a `return` statement:

```

def greet(name):

print("Hello, " + name + "!")

result = greet("Alice")

print(result) # Output: None

```

In this example, the `greet` function prints a greeting message but does not have a `return` statement. When the function is called with the argument `"Alice"`, it executes the code to print the message. However, since there is no explicit `return` statement, the function implicitly returns `None`.

The variable `result` is assigned the return value of the `greet` function, which is `None`. When `None` is printed to the console, it appears as `None`.

It's worth noting that if you don't need to return a value from a function, you can omit the `return` statement, and the function will still execute its code and return `None` by default. However, if you intend for a function to have a meaningful return value, it's important to include a `return` statement to specify that value explicitly.

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

Ans - To make a function variable refer to a global variable in Python, you can use the `global` keyword within the function. This tells Python that the variable being used is the global variable with the same name, rather than creating a new local variable.

Here's an example to illustrate the usage of the `global` keyword:

```python

x = 10 # Global variable

def modify\_global():

global x

x = 20 # Assign a new value to the global variable

print("Before function call:", x) # Output: 10

modify\_global()

print("After function call:", x) # Output: 20

```

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

Ans - The data type of `None` in Python is `NoneType`.

`None` is a special value that represents the absence of a value. It is commonly used to indicate the lack of a meaningful or specific result. It is often used as a default return value for functions that do not explicitly return anything.

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

Ans - The sentence "import areallyourpetsnamederic" does not have any built-in meaning in Python. It is not a valid Python statement or import statement.

In Python, the `import` statement is used to import modules or packages, which are pre-defined collections of code that provide additional functionality. When you import a module, you can access its functions, classes, and variables in your program.

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

Ans - After importing the `spam` module, you can call the `bacon()` feature by using the following syntax:

```python

import spam

spam.bacon()

```

In this case, `spam` is the name of the module, and `bacon()` is a feature or function within that module. By prefixing `bacon()` with `spam.`, you can access and call the function within the `spam` module.

This approach is necessary to avoid naming conflicts in case there are multiple modules with the same feature names or when you want to be explicit about which module the feature belongs to.

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

Ans - To save a program from crashing when encountering an error, you can implement error handling techniques to gracefully handle exceptions. This prevents the program from abruptly terminating and allows you to handle the error situation in a controlled manner. Here are some techniques you can use:

1. Try-Except Blocks: Wrap the potentially error-prone code within a `try` block, and use one or more `except` blocks to catch specific exceptions or handle any unexpected errors. This allows you to provide alternative actions or error messages when an exception occurs.

```python

try:

# Potentially error-prone code

# ...

except ExceptionType1:

# Handle ExceptionType1

# ...

except ExceptionType2:

# Handle ExceptionType2

# ...

```

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

Ans - The purpose of the `try` clause in Python is to enclose a block of code that might raise exceptions. It allows you to identify sections of code where exceptions may occur and handle them appropriately. The `try` clause is used in conjunction with the `except` clause.

The primary purpose of the `try` clause is to provide a structured way to catch and handle exceptions. It allows you to anticipate potential errors and write code to handle them gracefully. The code within the `try` block is executed, and if an exception occurs, the execution of the `try` block is immediately halted, and the control is transferred to the corresponding `except` block (if one is defined) based on the type of exception raised.

The purpose of the `except` clause is to define the specific exception(s) that you want to catch and specify the corresponding actions to be taken when those exceptions occur. The `except` block provides a way to handle exceptions in a controlled manner. You can have multiple `except` blocks to handle different types of exceptions or perform different actions based on the specific exception that occurred.