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

Ans: **Code Reusability:** Functions allow you to define a block of code that can be reused multiple times throughout your program or in different programs. Instead of writing the same code repeatedly, you can encapsulate it in a function and call the function whenever you need to perform that particular task. This saves time and effort and promotes efficient code reuse.

**Modularization:** Functions help break down complex programs into smaller, manageable modules. By dividing your code into functions, you can focus on implementing and understanding specific tasks or functionalities. This modular approach enhances code organization, readability, and maintainability. It also makes it easier to collaborate with other developers, as they can work on different functions independently.

**Abstraction:** Functions provide an abstraction layer by encapsulating a set of instructions behind a well-defined interface. When you call a function, you don't need to know the internal implementation details; you only need to understand the inputs and outputs of the function. This abstraction allows you to work at a higher level of logic and simplifies the understanding and usage of complex code.

**Code Readability:** Functions allow you to give meaningful names to blocks of code, which makes your program more readable and self-explanatory. Well-named functions serve as documentation, describing the purpose and functionality of the code. This improves code maintainability and makes it easier for others (or even yourself) to understand and modify the code in the future.

**Testing and Debugging:** Functions facilitate testing and debugging. Since functions isolate specific pieces of code, you can test and debug them independently. This makes it easier to identify and fix issues within a specific function rather than searching through the entire program. Functions also enable unit testing, where you can test each function individually to ensure they produce the expected output.

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

Ans:In Python, the code inside a function runs when the function is called, not when it is specified. When you define a function in Python, you are essentially creating a reusable block of code that will be executed only when the function is called.

Eg:

def greet():

print("Hello, world!")

print("Before function call")

greet()

print("After function call")

1. What statement creates a function?

Ans: def add\_numbers(x, y):

sum = x + y

return sum

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

Ans: Function is a defined block of code that performs a specific task, while a function call is the act of invoking or executing that function with specific arguments or parameters to obtain the desired result.

Eg: Function

def calculate\_square(number):

square = number \*\* 2

return square

Eg of Function call:

result = calculate\_square(5)

print(result)

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

Ans: The global scope refers to the outermost scope of a Python program, where global variables and functions are defined. It is accessible from anywhere within the program. There is only one global scope per program.

On the other hand, local scopes are created whenever a function is called or a code block is executed, such as inside loops or conditional statements. Each function call or code block creates a new local scope. Local variables and function parameters are defined within these local scopes and are only accessible within their respective scope. Once the function call or code block completes, the local scope is destroyed. Therefore, the number of local scopes depends on how many function calls or code blocks are executed during the program's execution.

So, while there is one global scope in a Python program, the number of local scopes can vary depending on the structure and execution flow of the program.

Top of Form

Regenerate response

Bottom of Form

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

Ans: When a function call returns, the local scope variables that were created within that function cease to exist. These variables are known as local variables and are only accessible within the scope of the function in which they were defined.

Once the function call is complete and the function's execution has finished, the local variables and any data stored in them are destroyed. The memory allocated to these variables is released, and their values cannot be accessed or used outside the function.

This means that any data or state stored in local variables during the function's execution will be lost once the function call returns. If you need to preserve data across function calls, you should consider using global variables or returning values from the function to pass information back to the calling code.

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

Ans: In programming, a return value is the value that a function or method gives back to the caller after it has finished executing. When a function is called, it may perform some operations or computations and then produce a result that is returned to the caller.

The concept of a return value allows functions to be more versatile and reusable. By returning a value, a function can provide useful information or data to the caller, which can then be used for further processing or decision-making.

Yes, it is possible to have a return value in an expression. In many programming languages, including popular ones like Python, functions or methods can be used as part of an expression and their return values can be directly used in calculations or assignments.

def add(a, b):

return a + b

result = add(3, 4) # The function call returns 7

print(result) # Output: 7

total = add(result, 5) # The function call returns 12

print(total)

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, the return value of a call to that function is usually None. In Python, None is a special object that represents the absence of a value. When a function reaches the end without encountering a return statement, it implicitly returns None.

def greet(name):

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

result = greet("Alice")

print(result)

O/P:

Hello, Alice!

None

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

Ans: global\_var = 10

def my\_function():

global global\_var

global\_var = 20

my\_function()

print(global\_var)

In the example above, the **global\_var** variable is declared as global within the **my\_function()** using the **global** keyword. Any changes made to **global\_var** within the function will affect the global variable.

1. What is the data type of None?

Ans: In Python, the data type of None is called "NoneType". It is a special built-in type that represents the absence of a value or the lack of a value. It is commonly used to indicate that a variable or expression does not have a value or has been intentionally set to no value. None is a singleton object, meaning there is only one instance of it in memory. It is often used as a default return value for functions that do not explicitly return anything.

1. What does the sentence import areallyourpetsnamederic do?

Ans: The sentence "import areallyourpetsnamederic" is not a valid Python code statement or expression. In Python, the keyword "import" is used to import modules or packages, not sentences or strings. Therefore, the sentence "import areallyourpetsnamederic" would not have any specific meaning or functionality in the context of Python programming.

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

Ans: If you had a bacon() feature in a spam module and you imported the module using the name "spam", you would call the bacon() feature as spam.bacon(). This is because when you import a module with a specific name, you need to use that name to access the functions or features within that module.

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

Ans: **Error handling**: Implement proper error handling mechanisms in your code. Use try-catch blocks or exception handling techniques to catch and handle anticipated errors gracefully. This allows you to execute alternative code paths or display meaningful error messages to the user without crashing the program.

**Input validation**: Validate user input and data to ensure it meets the expected format and constraints. Check for potential errors such as invalid characters, incorrect data types, or out-of-range values. By validating inputs, you can prevent unexpected errors and handle them appropriately.

**Defensive programming:** Adopt a defensive programming approach by anticipating potential issues and incorporating defensive measures into your code. This includes validating function arguments, checking return values, and adding appropriate conditionals to handle edge cases and prevent crashes.

**Logging:** Implement a logging system in your program to record errors and important events. This allows you to collect valuable information about the error, which can help with debugging and understanding the cause of the crash. Proper logging can provide insights and facilitate troubleshooting.

**Graceful degradation:** If encountering an error makes it impossible to continue the program's normal execution, you can implement strategies for graceful degradation. This involves having fallback mechanisms or alternative pathways to handle critical errors. It ensures that even if a specific feature or component fails, the program can still function partially or in a limited capacity.

**Testing and debugging:** Thoroughly test your code to identify and fix bugs before deploying the program. Utilize debugging tools and techniques to step through the code, identify the root cause of the error, and fix it. Proper testing and debugging minimize the chances of encountering errors in a production environment.

**Version control and backups:** Maintain version control for your codebase, and regularly create backups of your program and its data. This allows you to revert to a previous working version if a crash occurs due to a recent change. It provides an additional layer of safety and reduces the impact of errors.

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

Ans: The try clause allows you to attempt a risky operation or a block of code, while the except clause lets you define what should happen if an exception is raised during the execution of that code. By using try-except statements, you can handle exceptions and errors in a controlled manner, preventing your program from crashing and providing fallback options or error messages to the user.