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

Ans: Functions provide reusability, modularity, abstraction, code organization, easier testing and debugging, and code reusability. These advantages make functions an essential tool for building complex and maintainable software systems.

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

Ans: In most programming languages, the code inside a function runs when the function is called, not when it is specified or defined. When you define a function, you are essentially creating a reusable block of code that can be executed later by calling the function.

The code within the function is not executed until the function is invoked or called explicitly in your program. You can call the function multiple times, and each time it is called, the code inside the function will be executed.

3. What statement creates a function?

Ans: In most programming languages, including Python, a function is created using the def keyword followed by the function name and a set of parentheses containing any parameters the function accepts. Here's the general syntax:

def function\_name(parameter1, parameter2, ...):

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

Ans: In programming, a function and a function call are two related but distinct concepts.

A function is a block of code that performs a specific task. It is a reusable piece of code that can be invoked or called from various parts of a program. Functions are used to organize code into logical units, promote code reusability, and enhance the maintainability of a program. When you define a function, you specify its name, parameters (if any), and the code that gets executed when the function is called.

On the other hand, a function call is the act of invoking or executing a function. It is the point in the program where the control flow transfers to the function's code, and the instructions within the function are executed. To call a function, you use its name followed by parentheses, optionally passing any required arguments or parameters within the parentheses.

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

Ans: In a Python program, there is typically only one global scope, which is created when the program starts and remains in existence throughout the program's execution. The global scope contains variables and functions that are defined at the top level of the program, outside of any functions or classes. Variables and functions defined in the global scope can be accessed and modified from anywhere within the program.

On the other hand, the number of local scopes in a Python program depends on the number of functions or code blocks that create new scopes. Each time a function is called, a new local scope is created. Local scopes are temporary and are destroyed when the function completes its execution. Variables defined within a local scope are only accessible within that specific scope, and they are not visible outside of the function or code block where they are defined.

Therefore, the number of local scopes in a Python program can vary depending on the number of functions and code blocks that introduce new scopes.

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

Ans: When a function call returns, the local variables defined within the function's scope are typically destroyed or deallocated. The memory occupied by these variables is released, and the variables cease to exist. This process is known as variable scope or variable lifetime.

7. 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 refers to the value that a function or method provides back to the caller when it completes its execution. It is a way for functions to communicate information or results to the code that invoked them.

It is not possible to have a return value directly within an expression. The return value is generated when a function or method completes its execution, and it needs to be explicitly captured using a variable assignment or used in a statement.

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 does not have a return statement, it typically returns a special value called None. In Python, None is a built-in constant that represents the absence of a value. When a function reaches the end without encountering a return statement, it implicitly returns None by default.

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

Ans:

To make a function variable refer to a global variable, you need to use the global keyword in Python. Here's how you can do it:

1. Define the global variable outside of any function
2. Create a function and use the global keyword to reference the global variable within the function
3. Call the function to modify the global variable.

10. What is the data type of None?

Ans: None itself is not considered a data type in Python. Instead, it is an object of the NoneType class, which is a built-in type in Python. You can check the type of None using the type() function.

11. What does the sentence import areallyourpetsnamederic do?

Ans: The sentence "import areallyourpetsnamederic" is not a meaningful Python code or statement. It appears to be a random string of words that does not have a specific programming interpretation or purpose. In Python, the "import" keyword is used to import modules or packages, and "areallyourpetsnamederic" does not correspond to a valid module or package name.

12. 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 it as spam, you would call it using the module name followed by the function name. So, you would call it as spam.bacon().

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

Ans:

There are several techniques and best practices that can help prevent a program from crashing when it encounters an error. Here are some strategies you can employ:

Error handling: Implement error handling mechanisms such as exception handling or error codes. When an error occurs, catch the exception or check the error code and gracefully handle the error, providing a meaningful error message to the user and taking appropriate actions to recover or exit the program safely.

Defensive programming: Write defensive code by validating inputs and checking for potential errors or edge cases. Use conditionals and assertions to verify assumptions and prevent the program from crashing due to unexpected data or conditions.

Logging: Implement a robust logging mechanism to record important information, including errors and exceptions. This can help diagnose issues and provide valuable information for debugging and troubleshooting.

Graceful degradation: If your program relies on external resources or dependencies, handle failures in a way that allows the program to continue functioning without crashing. For example, if a network connection is lost, provide a fallback mechanism or display a friendly message instead of abruptly terminating the program.

Testing and debugging: Thoroughly test your code and use debugging tools to identify and fix potential issues before deploying the program. Automated testing, unit testing, and integration testing can help identify and resolve errors early in the development process.

Proper resource management: Make sure to release or close resources properly, such as file handles, network connections, or database connections. Failing to release resources can lead to memory leaks or other issues that may eventually cause a crash.

Robust input validation: Validate and sanitize user input to prevent errors or malicious code injections. This helps to ensure the program can handle unexpected or malicious inputs without crashing.

Error reporting and monitoring: Implement mechanisms to report errors and exceptions, either to the user or to a centralized monitoring system. This can help you track and analyze errors, identify recurring issues, and proactively fix them.

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

Ans:

In Python, the try and except clauses are used together for exception handling, allowing you to handle and manage potential errors or exceptional situations in your code.

The purpose of the try clause is to enclose a block of code that might raise an exception. By placing potentially problematic code within the try block, you are signaling to Python that you are aware that an exception could occur at that point. The try block is followed by one or more except clauses, which specify how to handle specific exceptions that may be raised within the try block.

The purpose of the except clause is to define the actions to be taken when a specific exception occurs within the corresponding try block. If an exception is raised within the try block, Python checks each except clause one by one, starting from the top, to find a matching exception type. If a match is found, the code within that except block is executed. If no matching except clause is found, the exception propagates to the surrounding code or to an outer try statement, or the program terminates with an unhandled exception.