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

Answer:

Functions are advantageous to have in programs for several reasons:

1. Modularity and Reusability: Functions allow you to break down your program into smaller, manageable chunks of code. Each function performs a specific task or solves a particular problem. This modular approach makes the code more organized and easier to understand, maintain, and debug. Functions can be reused in different parts of the program or even in other programs, saving time and effort by avoiding code duplication.
2. Abstraction and Encapsulation: Functions provide an abstraction layer that hides the internal implementation details from the rest of the program. By defining inputs (parameters) and outputs (return values), functions encapsulate a specific behavior or functionality, allowing other parts of the program to interact with them without knowing the internal workings. This promotes code readability, reduces complexity, and enables collaborative development.

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

Answer:

When you define a function, you are essentially creating a reusable block of code that contains a series of instructions to be executed. However, these instructions are not executed immediately upon defining the function. Instead, they are executed when you call the function in your program.

So, the code in a function runs when you explicitly call the function during the execution of your program.

3. What statement creates a function?

Answer:

def function\_name(arguments):

# Function body

# Statements and code to be executed when the function is called

return result

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

Answer:

a function is a reusable code block with a defined behavior, while a function call is the specific point in your program where you invoke and execute the function with specific arguments (if any). The function call triggers the execution of the code within the function's body, allowing you to utilize its functionality.

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

Answer:

a Python program has one global scope, accessible throughout the program, and local scopes that are created for each function or class, allowing for encapsulation and separate namespaces within those scopes.

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

Answer:

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

1. Variables are inaccessible: Once the function call returns, any variables defined within the local scope of that function become inaccessible from outside the function. Attempting to access those variables would result in an error.
2. Memory is freed: The memory allocated to store the local variables is freed up by the Python interpreter. This memory can then be used for other purposes in the program.
3. Variable values are lost: Since the local variables cease to exist, any values stored in those variables are lost. If you need to retain or use the values computed within the function, you should store them in global variables or return them from the function before it returns.

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

Answer:

The concept of a return value refers to the value that a function can send back to the caller when it is executed. When a function is called, it can perform certain operations and computations, and optionally, it can produce a result or a value that can be used by the calling code.

The **return** statement is used in most programming languages, including Python, to specify the return value of a function. It allows the function to pass a value back to the caller. The return value can be of any data type, such as numbers, strings, lists, dictionaries, or even custom objects.

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

Answer:

If a function does not have a return statement, the return value of a call to that function is **None**. **None** is a special Python object that represents the absence of a value.

When a function reaches the end of its execution without encountering a **return** statement, or if the **return** statement is without any value, Python automatically returns **None** as the default return value.

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

Answer:

In Python, if you want to make a function variable refer to a global variable, you can use the **global** keyword to declare that variable as a global variable within the function. This allows the function to access and modify the global variable instead of creating a new local variable with the same name.

10. What is the data type of None?

Answer:

The data type of **None** in Python is called **NoneType**. **None** represents the absence of a value or the lack of a specific object. It is commonly used to indicate the absence of a meaningful result or to initialize variables or placeholders

11. What does the sentence import areallyourpetsnamederic do?

Answer:

The sentence "import areallyourpetsnamederic" is not a valid Python import statement. In Python, the **import** statement is used to import modules or packages that provide additional functionality or resources for your code.

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

Answer:

After importing the **spam** module, you can access the **bacon()** feature by referencing it with the module name as a prefix

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

Asnwer:

To prevent a program from crashing when encountering an error, you can implement error handling techniques. Error handling allows you to gracefully handle and recover from exceptions or errors that may occur during program execution.

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

Answer:

The purpose of the **try** clause in Python is to enclose a block of code that may potentially raise an exception or error. It allows you to handle potential errors or exceptions that might occur during the execution of that code. The **try** block essentially sets up an area where you can test for and catch exceptions.