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

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

1.Reusability: Functions allow you to write a block of code once and use it multiple times throughout your program. This saves time and reduces the amount of code you need to write.

2.Modularity: Functions break down a program into smaller, more manageable pieces. This makes the code easier to read, understand, and maintain.

3.Encapsulation: Functions can be used to encapsulate code and data, making it easier to control access to them. This helps to prevent accidental modification of important data and reduces the risk of errors.

4.Abstraction: Functions allow you to abstract away the implementation details of a program and focus on the higher-level logic. This makes it easier to write complex programs and reduces the amount of cognitive overhead required to understand the program.

5.Testing: Functions can be tested independently of the rest of the program. This makes it easier to identify and fix bugs and ensures that the program behaves as expected.

Overall, functions are a powerful tool for creating clean, efficient, and maintainable code.Top of Form

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

Ans: The code in a function is executed when the function is called. The code inside the function definition is not executed until the function is called. When the function is called, the code inside the function is executed, and any values returned by the function are passed back to the calling code.

It's worth noting that in Python, defining a function does not automatically execute the code inside the function. The code inside the function will only run when the function is called. This means that you can define functions at the beginning of your code, and then call them later when needed, and the code inside the function will only run when it's called.

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:

def function name(parameters):

# code to be executed

return value

Here, "def" is the keyword used to define a function. "function name" is the name of the function, which should be descriptive and meaningful. "parameters" are the input arguments that the function takes (if any). The code to be executed inside the function should be indented after the colon (:) and the "return" statement is used to specify the value that the function should return (if any).

Once the function is defined using the "def" statement, it can be called by its name followed by parentheses with any required arguments.

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

Ans: A function is a block of code that performs a specific task and can be reused throughout a program. It is defined using the statement in Python and consists of a function name, parameters (if any), and the code to be executed when the function is called.

On the other hand, a function call is the act of executing the code inside a function. When a function is called, control is transferred to the function, and the code inside the function is executed. Any values returned by the function are passed back to the calling code. A function call is made by using the name of the function followed by parentheses with any required arguments.

To summarize, a function is a block of code that performs a specific task, while a function call is the act of executing that code by calling the function with any required arguments. A function is defined once, while a function call can be made multiple times with different arguments to achieve different results.

1. Top of Form
2. How many global scopes are there in a Python program? How many local scopes?

Ans:In Python, there is only one global scope per program. This means that any variable or function defined outside of a function is considered to be in the global scope and can be accessed from anywhere within the program.

On the other hand, local scopes are created every time a function is called. Each function call creates a new local scope, which is a separate namespace from the global scope. Any variables or functions defined within a function are considered to be in the local scope and can only be accessed from within that function.

It's important to note that local scopes cannot access variables or functions defined in the global scope by default. To access global variables or functions from within a local scope, the global keyword can be used to explicitly declare the variable or function as global.

To summarize, a Python program has only one global scope, while local scopes are created every time a function is called.

1. Top of Form

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

When a function call returns and its execution completes, the local scope of the function is destroyed and all the variables defined within that local scope are deleted. This means that any variables that were created or modified within the function's local scope cannot be accessed outside the function.

For example, consider the following Python code:

def my function ():

x = 10

y = 20

z = x + y

return z

result = my function ()

print(result)

print(x) # This will raise an error because x is not defined outside the function

In this example x, y and z are all variables defined within the local scope of my function. When my function is called and executed, these variables are created and assigned values. However, once the function returns and its execution is complete, the local scope is destroyed and these variables are deleted.

Therefore, when the code tries to access the variable x outside of the function, it raises a name error because x is not defined in the global scope (i.e., outside of the function).

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

Ans:In computer programming, a return value is a value that a function or method returns to the caller after its execution. The return value can be any valid data type, such as a number, a string, a Boolean value, or even an object.

The concept of a return value is used to communicate the result of a function or method back to the calling code. The calling code can then use this value for further processing or as input to other functions or methods.

In most programming languages, a return value is specified using the return keyword followed by the value that needs to be returned. For example, consider the following Python function that calculates the sum of two numbers and returns the result:

def sum numbers (a, b):

return a + b

In this example, the sum number function returns the sum of the input parameters a and b using the return keyword.

It is possible to use a return value in an expression. For example, consider the following Python code:

def square number(x):

return x \*\* 2

result = square number (5) + 10

print(result)

In this example, the square number function returns the square of the input parameter x. The return value of the function is then used in an expression to calculate result, which is the sum of the square of 5 and 10. The value of result will be 35.

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, it will still execute the code inside the function but it will not return any value. In this case, the return value of a call to that function will be none

For example, consider the following Python function:

def print hello ():

print ("Hello World!")

This function simply prints "Hello World!" to the console but does not have a return statement. If we call this function like this:

result = print hello ()

print(result)

The output will be:

Hello World!

None

Here, the function print Hello () is called and it prints "Hello World!" to the console. However, when we try to assign the return value of the function to a variable result we get none as the value of result. This is because the function does not have a return statement, so it does not return any value.

In Python, none is a special built-in object that represents the absence of a value. Therefore, if a function does not have a return statement, its return value will be none.

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

Ans:In Python, you can make a function variable refer to a global variable by using the global keyword inside the function. This tells Python that you want to use the global variable, rather than create a new variable with the same name inside the function's scope.

Here is the example

x = 10 # global variable

def my\_function():

global x # use the global variable 'x'

x = 20 # update the value of 'x'

my\_function()

print(x) # outputs 20

In this example, x is a global variable with a value of 10. The my\_function() function uses the global keyword to tell Python that it wants to use the global variable x. It then updates the value of x to 20. Finally, the value of x is printed, which is now 20 because it was updated by the function.

1. What is the data type of None?

Ans: In Python, None is a special constant that represents the absence of a value. It is commonly used to indicate that a variable or expression does not have a meaningful value. None is of type NoneType, which is a built-in data type in Python. NoneType has only one value, which is None.

1. What does the sentence import areallyourpetsnamederic do?

Ans: The sentence "import areallyourpetsnamederic" is not a valid Python import statement and would raise a ModuleNotFoundError exception if executed in Python.

In general, import statements are used in Python to load and make available external modules or packages in the current program. A valid import statement would typically include the name of the module or package being imported, such as import pandas or import numpy as np

Therefore, the sentence "import areallyourpetsnamederic" would not have any effect on the Python program, as there is no such module or package named "areallyourpetsnamederic" that could be imported.

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, you could call it after importing spam by using dot notation to access the bacon() function:

import spam

spam.bacon()

This will execute the bacon () function that is defined within the spam module.

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

Ans: As an AI language model, I don't have the ability to directly interact with running programs. However, in general, there are several things that can be done to prevent a program from crashing or to recover from an error if it does occur:

exception handling: Wrap code in a try-except block to catch any errors that may occur and handle them in a way that prevents the program from crashing. For example, you can display an error message to the user or log the error for later analysis.

Use defensive programming techniques: Write code that anticipates potential errors and handles them proactively. For example, check that user input is valid before using it in calculations.

Write robust code: Test code thoroughly before deploying it to production. Use automated testing tools to catch potential issues before they cause problems.

Use monitoring and alerting tools: Monitor the program's performance and error logs in real-time, and set up alerts to notify you when errors occur.

Use version control: Keep track of changes to the codebase and roll back to a previous version if a bug is introduced

These are just a few examples of things that can be done to prevent a program from crashing or to recover from an error if it does occur. It's important to be proactive in your approach to error handling and to continuously monitor and improve your code to prevent issues from occurring in the first place.in the first place.

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 for exception handling.

The purpose of the try clause is to enclose the code that might raise an exception. The statements within the try block are executed, and if no exceptions are raised, the program continues to execute the statements following the try block. However, if an exception is raised, the normal flow of the program is disrupted, and the program jumps to the nearest except block.

The purpose of the except clause is to catch and handle exceptions that are raised within the try block. The except clause specifies the type of exception to catch and the code to execute if that exception occurs. Multiple except clauses can be used to catch different types of exceptions. If no exceptions are caught by any of the except blocks, the exception is passed up the call stack, and the program terminates with an error message.

try:

# code that might raise an exception

except TypeError:

# code to handle a TypeError exception

except ValueError:

# code to handle a ValueError exception

except:

# code to handle any other exception

In this example, the try block contains code that might raise an exception. If a TypeError or ValueError exception is raised, the corresponding except block is executed. If any other exception is raised, the final except block is executed.