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

Functions are advantageous to have in our programs because they:

**Increase code readability:** Functions allow you to break your code into smaller, more manageable chunks. This makes your code easier to read and understand, especially for other programmers.

**Improve code reusability:** Functions can be reused in different parts of your program. This saves you time and effort from having to rewrite the same code repeatedly.

**Improve code modularity:** Functions can be grouped together into modules. This makes your code easier to maintain and update.

**Improve code testability:** Functions can be easily tested in isolation. This makes it easier to find and fix bugs in your code.

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

The code in a function runs when it is called. The function is defined first, but it is not executed until it is called. When the function is called, the code inside the function is executed.

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

The def statement creates a function. The def statement takes two arguments: the name of the function and the body of the function. The body of the function is a block of code that is executed when the function is called.

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

A function is a block of code that is defined with the def statement. A function call is the act of executing a function. When you call a function, the code inside the function is executed.

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

There is only one global scope in a Python program. There can be many local scopes, one for each function that is defined. The global scope is the outermost scope in a Python program. The local scopes are nested within the global scope.

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

The variables in a local scope are destroyed when the function call returns. This is because the local scope is only created when the function is called, and it is destroyed when the function call returns.

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

The return value of a function is the value that is returned when the function is called. The return value of a function can be any type of value, including a string, a number, or a list.

It is possible to have a return value in an expression. For example, the following expression returns the value 10:

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

print(factorial(5))

The code print(factorial(5)) calls the function factorial() and passes the value 5 as an argument. The function factorial() then returns the factorial of 5, which is 120. The code then prints the value 120.

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

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

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

You can make a function variable refer to the global variable by using the global keyword. The global keyword tells Python that the variable is defined in the global scope, not in the local scope of the function.

For example, the following code defines a global variable called spam and a function called foo(). The function foo() uses the global keyword to make the variable spam refer to the global variable spam.

spam = 10

def foo():

global spam

spam = 20

foo()

print(spam)

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

The data type of None is NoneType. NoneType is a special data type in Python that represents the absence of a value

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

The sentence import actuallycurrentknowledge imports the module actuallycurrentknowledge into the current namespace. The module actuallycurrentknowledge is a hypothetical module that contains information about the current state of knowledge.

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

After importing the spam module, you would call the bacon() feature by using the spam.bacon() syntax. For example, the following code imports the spam module and then calls the bacon() feature:

import spam

spam.bacon()

The spam.bacon() syntax tells Python to look for the bacon() feature in the spam module. The spam module is a namespace, which is a collection of names that can be used to access objects. The bacon() feature is a name in the spam namespace.

When you call the spam.bacon() syntax, Python looks for the bacon() feature in the spam namespace. If the bacon() feature is found, Python executes the code that is associated with the bacon() feature.

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

There are a few things you can do to save a program from crashing if it encounters an error. One is to use the try and except blocks. The try block contains the code that you are trying to run, and the except block contains the code that is executed if an error occurs.

Another thing you can do is to use the logging module. The logging module allows you to log errors to a file or to the console. This can help you to track down the source of the error.

Finally, you can also use the breakpoint() statement. The breakpoint() statement will stop the program at the point where it is executed. This can be helpful if you want to inspect the state of the program when the error occurs.

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

The try clause is used to execute a block of code. The except clause is used to handle errors that occur in the try clause.

The try clause is executed first. If no errors occur, the except clause is not executed. However, if an error occurs in the try clause, the except clause is executed. The except clause can be used to handle the error in a variety of ways.

For example, the except clause can be used to print an error message, to log the error to a file, or to continue the execution of the program.

Below is an example of how the try and except clauses can be used:

def divide\_numbers(x, y):

try:

return x / y

except ZeroDivisionError:

print("Division by zero")

print(divide\_numbers(10, 0))

The divide\_numbers() function takes two numbers as input and returns the result of dividing them. However, if the second number is 0, the divide\_numbers() function will raise a ZeroDivisionError exception.

The try clause in the divide\_numbers() function ensures that the except clause is only executed if an error occurs. If no errors occur, the except clause is not executed and the divide\_numbers() function simply returns the result of dividing the two numbers.

The except clause in the divide\_numbers() function prints an error message if the second number is 0. This helps the user to understand why the divide\_numbers() function failed