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

1. Reusability: Functions allow you to write a block of code that can be used repeatedly throughout your program. This makes your code more efficient, easier to read, and less prone to errors.
2. Modularity: Functions help break down complex programs into smaller, more manageable parts. This makes your code easier to understand and maintain.
3. Abstraction: Functions allow you to hide the complexity of your code from the rest of the program. This means that other parts of your program only need to know how to use the function, not how it works.
4. Debugging: Functions make it easier to debug your code. By isolating a particular function, you can test it separately and fix any errors without affecting the rest of the program.
5. Scalability: Functions make it easier to scale your programs. As your program grows, you can add new functions to handle new functionality, without having to rewrite or modify existing code.

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

When a function is defined, its code is not executed, but rather stored in memory.

When the function is called, the code inside the function is executed, and any parameters passed to the function are processed. Once the function has completed its task, it can return a value (if specified) or simply exit.

It's important to note that a function can be called multiple times with different arguments or input values, each time executing the same block of code. This makes functions a powerful tool for reuse and abstraction in programming.

Eg:

def cube\_sum(n):

  sum = 0

  for i in range(1,n+1):

    sum += i\*\*3

  return sum

n = int(input("Enter number : "))

print(f"cube sum of first {n} natural numbers is {cube\_sum(n)}")

3. What statement creates a function?

In Python, the "def" statement is used to create a function.

def function\_name(parameters):

# code block

return output\_value

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

1. A function is a named block of code that performs a specific task.
2. A function call is the act of executing a function by using its name and passing any required arguments.

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

1. There is one global scope in a Python program.
2. Local scopes are created dynamically whenever a function is called or a block is entered. Each function call or block entry creates a new local scope, and variables defined within those scopes are only accessible within their respective scopes. Local scopes are destroyed once the function call or block is completed.

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

When a function call returns, the local scope of that function is destroyed, and any variables defined within that local scope are also destroyed. These variables are no longer accessible or available for use once the function call has completed.

EG:

def my\_function():

x = 10 # Local variable

print(x)

my\_function()

print(x)

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

* The concept of a return value refers to the value that a function can optionally provide when it completes its execution. When a function is defined, you can specify what value it should return using the return statement. This return value can be of any data type in Python, including numbers, strings, lists, dictionaries, or even custom objects.
* The return statement allows the function to send a computed value or result back to the caller. Once a return statement is encountered within a function, the function's execution stops, and the specified value is returned to the caller. The return value can then be stored in a variable or used in an expression for further computation.

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, or if the return statement is not explicitly specified, the function will implicitly return None. None is a special Python object that represents the absence of a value. It is often used to indicate that a function does not produce a meaningful result or does not need to return a specific value.

Eg:

def greet(name):

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

result = greet("Alice")

print(result) # Output: None

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

In Python, if you want to make a function variable refer to the global variable of the same name, you can use the global keyword.

Eg:

x = 10 # Global variable

def modify\_global():

global x # Declare x as a global variable

x = 20 # Modify the global variable

print(x) # Output: 10

modify\_global()

print(x) # Output: 20

10. What is the data type of None?

In Python, None is a special value that represents the absence of a value.

11. What does the sentence import areallyourpetsnamederic do?

The sentence "import areallyourpetsnamederic" does not have any special meaning or functionality in Python. It is an invalid import statement and would result in a ModuleNotFoundError because there is no module named "areallyourpetsnamederic" in the Python standard library or any commonly used external libraries.

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

If you have a function called bacon() in a module named spam, after importing the spam module, you can call the bacon() function using the following syntax:

import spam

spam.bacon()

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

To save a program from crashing when it encounters an error, you can implement error handling techniques. By handling errors appropriately, you can prevent the program from abruptly terminating and provide a graceful way to handle unexpected situations. Here are a few strategies to consider:

1. Try-Except Blocks: Surround the code that might raise an error with a try-except block. The code within the try block is executed, and if an exception occurs, it is caught by the except block, allowing you to handle the error gracefully. You can specify different except blocks to handle specific types of exceptions.

try:

# Code that might raise an error

# ...

except ExceptionType:

# Code to handle the specific exception

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

The try and except clauses in Python are used together to implement error handling mechanisms. Here's the purpose of each clause:

1. Try Clause: The try clause is used to enclose the block of code that might raise an exception or encounter an error. It is the section where you expect potential errors to occur. The purpose of the try clause is to monitor the code within it and detect any raised exceptions.

If an exception occurs within the try block, the execution of the try block is immediately halted, and the program flow jumps to the corresponding except block (if defined).

2. Except Clause: The except clause is used to define the block of code that should be executed when a specific exception occurs within the corresponding try block. It allows you to specify how the program should handle the exception.

The purpose of the except clause is to catch and handle specific types of exceptions that may be raised within the try block. By providing an except block, you can define the actions or error-handling logic to be executed when a particular exception occurs. You can have multiple except blocks to handle different types of exceptions separately.

Eg:

try:

# Code that might raise an exception

# ...

except ValueError:

# Code to handle ValueError

except FileNotFoundError:

# Code to handle FileNotFoundError

except Exception as e:

# Code to handle other exceptions