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

- Functions are advantageous to have in your programs because they allow you to reuse code. This can save you time and effort, and it can also make your code more readable and maintainable. When you define a function, you are essentially creating a new block of code that can be called from anywhere in your program. This means that you can write the code once and then use it as many times as you need to. This can be very helpful if you have a piece of code that you need to use in multiple places in your program.
- Functions can also make your code more readable and maintainable. When you define a function, you can give it a name that describes what it does. This makes it easier to understand what the function is doing, and it also makes it easier to find the function when you need to use it.
- Functions can also help you to keep your code organized. When you have a lot of code, it can be easy to get lost. Functions can help you to break your code down into smaller, more manageable pieces. This makes it easier to understand and debug your code.
- Overall, functions are a powerful tool that can help you to write better, more efficient, and more maintainable 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. When you call a function, the function's code is executed, and the function returns a value. For example, the following code defines a function called `add()` that adds two numbers together:

- def add(x, y):

  return x + y

To call this function, you would use the following syntax:

- z = add(1, 2)

This would return the value 3, which is the sum of 1 and 2. Functions are a powerful tool that can be used to reuse code and make your programs more modular.

3. What statement creates a function?

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

For e.g :

- def sub(x,y):
  return x - y

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

A function is a block of code that is executed when it is called. A function call is the act of invoking a function.

- For example, the following code defines a function called `add()` that takes two numbers as input and returns their sum:

```
def add(x, y):
    return x + y
```

- The following code calls the `add()` function with the numbers 1 and 2 as input:

```
print(add(1, 2))
```

This code will print the number 3, which is the sum of 1 and 2

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. This is the scope that is created when the program starts running. All variables that are defined outside of any function or class are defined in the global scope.
- There can be multiple local scopes in a Python program. Each time a function or class is defined, a new local scope is created. Variables that are defined inside a function or class are defined in the local scope of that function or class.

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.
- This means that all variables that were defined in that local scope are no longer accessible.

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

A return value is the value that is returned by a function when it is called. The return value can be any type of value, including a number, a string, a list, or a dictionary. It is possible to have a return value in an expression.

- For example, the following code returns the value of the “add()” function:

```
return add(1, 2)
```

- The return value of a function can be used in any expression, just like any other value.

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 a call to that function is “None”

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

To make a function variable refer to the global variable, you can use the “global” keyword. For example, the following code defines a function called “add()” that takes two numbers as input and returns their sum.

- The function also has a global variable called “total” that is initialized to 0. The function then adds the two numbers to the global variable `total` and returns the sum.

```
def add(x, y):
    global total
    total += x + y
    return total
```

- The following code calls the “add()” function with the numbers 1 and 2 as input. The function then prints the value of the global variable “total”.

```
print(add(1, 2))
```

This code will print the number 3, which is the sum of 1 and 2.

10. What is the data type of None?

- The data type of **“None”** is **“NoneType”.**

11. What does the sentence import areallyourpetsnamederic do?

The sentence “import areallyourpetsnamederic” imports the “areallyourpetsnamederic” module into the current namespace. This means that the functions and variables defined in the “areallyourpetsnamederic” module can be used in the current program

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 can call the `bacon()` feature by using the following syntax:

  spam.bacon()

- This will call the `bacon()` function in the `spam` module and return the result.

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 in Python. One option is to use the “try” and “except” statements. The “try” statement allows you to execute a block of code that may contain an error. The “except” statement allows you to handle the error if it occurs. For example, the following code will try to divide “a” by “b”. If “b” is zero, the “except” statement will be executed.

```
try:
a / b
except ZeroDivisionError:
print("Cannot divide by zero")
```

- Another option is to use the “assert” statement. The “assert” statement checks if a condition is true. If the condition is not true, the “assert” statement will raise an error. For example, the following code will raise an error if “a” is less than zero.

  ```
  assert a >= 0
  ```

- Finally, you can also use the “logging” module to log errors. The “logging” module allows you to record errors in a log file. This can be helpful for debugging problems.

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 that may contain an error. The “except” clause is used to handle the error if it occurs.
- For example, the following code will try to divide “a” by “b”. If “b” is zero, the “except” clause will be executed.

  ```
  try:
      a / b
      except ZeroDivisionError:
  ```

```
    print("Cannot divide by zero")
```

- The “try” clause can also be used with an “else” clause. The “else” clause will be executed if the “try” clause does not raise an error. For example, the following code will try to divide “a” by “b”. If “b” is not zero, the “else” clause will be executed.

```
try:
    a / b
    except ZeroDivisionError:
print("Cannot divide by zero")
else:
    print("Division successful")
```

- The “try” and “except” clauses can be used together to handle different types of errors. For example, the following code will try to open a file. If the file does not exist, the “FileNotFoundError” exception will be raised. The “except” clause will handle this exception by printing a message to the console.

```
try:
    f = open("file.txt")
except FileNotFoundError:
    print("File not found")
```

- The “try” and “except” clauses can be used to make your code more robust and error-proof.