ASSIGNMENT -3

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

Answer: ‘Functions’ are advantageous to have in our programs in the following ways:

Functions are advantageous to have in programs for the following reasons:

1. Reusability: Functions can be reused multiple times within a program or in different programs, which saves time and effort in writing repetitive code.
2. Modularity: Functions allow you to break down a program into smaller, more manageable pieces. This makes it easier to understand and debug the program as a whole.
3. Readability: Functions can improve the readability of a program by breaking down complex tasks into smaller, more understandable parts. This makes the code easier to follow and maintain.
4. Encapsulation: Functions provide a way to encapsulate code and data within a well-defined boundary. This makes it easier to manage the code and data, and reduces the likelihood of errors and bugs.

2. When does the code in a function run: when it’s specified or when it’s called?
   Answer: The code inside a function runs only when it is called.
3. What statement creates a function?

   Answer:

   def fun1(parameters):

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

   Answer:

   A function is a block of code that performs a specific task when called. It is a self-contained unit of code that can be reused multiple times within a program or in different programs.

   A function call, on the other hand, is the act of invoking or executing a function. It is the code that instructs the program to run a specific function with a given set of arguments.

   In other words, a function is a named block of code that performs a specific task, while a function call is the instruction that tells the program to execute that block of code with specific inputs.

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

   Answer:

   In a Python program, there is only one global scope, which is created when the program starts running. The global scope is used to define variables and functions that are accessible from anywhere in the program, including within other functions and modules.

   Local scopes, on the other hand, are created whenever a function is called. Each function call creates a new local scope that is separate from the global scope and any other local scopes. Variables defined inside a function are only accessible within that function and its inner functions (if any).

   So, the number of local scopes in a Python program depends on the number of function calls made in the program. Each function call creates a new local scope, so the number of local scopes can vary depending on the structure of the program and how many functions are called. However, there is always only one global scope.

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

   Answer:

   When a function call returns, the local scope of that function is destroyed, and any variables defined within it are deleted. This means that the values of those variables are no longer accessible outside of the function.

   For example, consider the following function that defines a local variable **x**:

   def my_function(): x = 10 print("The value of x is:", x) my_function()

   When this function is called, it creates a local scope in which the variable **x** is defined and given a value of 10. The function then prints the value of **x** to the console. When the function call returns, the local scope is destroyed and the variable **x** is deleted.

   If we try to access **x** outside of the function, we will get a **NameError**, because the variable is no longer defined:

   def my_function(): x = 10 my_function() print("The value of x is:", x) # This will raise a NameError

   In this example, **x** is only defined within the local scope of **my_function()**. When the function returns, **x** is deleted and no longer accessible.

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

   Answer:

In programming, a return value is the value that a function returns after it has completed its task. When a function is called, it may perform some operations and then return a value that can be used by the calling code.

For example, consider the following function that calculates the sum of two numbers:

def add_numbers(a, b): return a + b

This function takes two parameters **a** and **b**, adds them together, and then returns the result using the **return** keyword. When this function is called, it returns the sum of the two numbers:

result = add_numbers(3, 5) print(result)

Output: 8

Yes, it is possible to have a return value in an expression. This means that the result of a function call can be used as part of a larger expression.

For example, consider the following code that uses the **add_numbers()** function in an expression:

result = add_numbers(3, 5) * 2 print(result)

Output: 16

In this example, the function call **add_numbers(3, 5)** returns the value 8, which is then multiplied by 2 to produce the final result of 16.

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**.

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

   Answer:

   In Python,we can use the **global** keyword to indicate that a variable is a global variable, rather than a local variable. This means that the variable should be accessed and modified from the global scope, rather than the local scope of the function.

   Here's an example:

x = 10

```
def my_function():
   global x
   x = 20
   return x
my_function()


#output – 20
```

In this example, we define a global variable **x** with the value **10**. We then define a function **my_function()** that uses the **global** keyword to indicate that **x** should be accessed and modified as a global variable.

When we call **my_function()**, it sets the value of **x** to **20**. After the function call, we print the value of **x** and see that it has been modified to **20**.

Note that using global variables can sometimes lead to code that is difficult to read and maintain, so it is generally a good practice to avoid using global variables when possible.

10. What is the data type of None?

    Answer:

    The data type of **None** in Python is **NoneType**.

    None is a special value in Python that represents the absence of a value or the lack of a value. It is often used to indicate that a variable has not been assigned a value, or to represent a default value.

11. What does the sentence import areallyourpetsnamederic do?

    Answer:

    The sentence **import areallyourpetsnamederic** is not a valid Python statement, as **areallyourpetsnamederic** is not a valid Python module or package.

    In Python, the **import** statement is used to import modules or packages that contain reusable code. When we import a module or package, we make its functions, classes, and variables available in your current Python script.

    import mymodule

    result = mymodule.myfunction()

In this example, we use the **import** statement to import the **mymodule** module. We can then call the **myfunction()** function from the module and assign its return value to the variable **result**.

If you try to import a module or package that does not exist or has not been installed, you will get a **ModuleNotFoundError** or **ImportError** indicating that Python was unable to find the specified module or package.

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

    Answer:

    import spam

    result = spam.bacon()

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

    Answer:

    There are several techniques you can use to save a program from crashing if it encounters an error:

    1. Use exception handling: Exception handling allows you to handle errors in a controlled way, by "catching" the error and taking appropriate action. You can use a **try-except** block to catch errors and handle them gracefully, without causing the program to crash.

    2. Validate user input: If your program accepts input from users, you can validate that input to ensure it is in the expected format or range. For example, if your program expects a number, you can check that the input is actually a number before trying to perform calculations on it.

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 might raise an exception during its execution. The **try** block is followed by one or more **except** clauses that define what to do when a specific exception is raised within the **try** block.
    The **try** clause is used to attempt a piece of code that might raise an exception, while the **except** clause is used to catch and handle that exception. If an exception is raised within the **try** block, the interpreter looks for an **except** clause that matches the type of exception raised. If a matching **except** clause is found, the code within that clause is

executed to handle the exception. If no matching **except** clause is found, the exception is propagated up the call stack until it is either caught by an **except** clause or causes the program to terminate with an error message.
In short, the purpose of the **try** clause is to define a block of code to be attempted, while the purpose of the **except** clause is to define how to handle any exceptions that might be raised during the execution of the **try** block.