Name :- Deepawali. B. Mhaisagar

Assignment 3

Questions

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

ANS

Functions are advantageous to have in programs for several reasons:

- **Code organization and reusability:** Functions allow you to organize your code into modular units, making it more structured and manageable.By breaking down a program into smaller functions, you can focus on specific tasks or functionalities. Functions can also be reused in different parts of the program or even in other programs, reducing code duplication and promoting code reuse.
- **Modularity and maintainability:** Functions encapsulate a specific set of instructions, providing a modular approach to programming.
- **Abstraction and readability**: Functions provide a level of abstraction by hiding the implementation details. Instead of dealing with the internal logic of a function, other parts of the program can simply use the function by providing the necessary inputs and receiving the outputs.
- **Code reuse and efficiency**: Functions promote code reuse, as you can define a set of instructions once and call the function multiple times whenever needed. This avoids repetitive coding and improves code efficiency.
- **Testing and debugging:** Functions enable easier testing and debugging.

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

ANS

The code inside a function runs when the function is called, not when it is specified or defined. Defining a function simply creates the function object and associates a name with it, but it does not execute the code within the function.

The actual execution of the code inside a function occurs when the function is called by its name followed by parentheses () and any required arguments. When a function call is encountered in the program, the flow of execution jumps to the function's code block, executes the instructions inside the function, and then returns to the point where the function was called.

3. What statement creates a function?

ANS

In Python, the def statement is used to create a function. It is followed by the function name, a set of parentheses (), and a colon :. The body of the function is indented below the def statement.

syntax:

```
def function_name(parameters):

    # Function body

    # Code statements

    # …
```

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

ANS

The difference between a function and a function call lies in their respective roles and actions within a program:

Function: A function is a reusable block of code that performs a specific task or carries out a particular operation. It is defined using the def statement in Python and consists of a function name, a set of parentheses ‘()’.The function encapsulates a sequence of instructions or statements that are executed when the function is called. Functions allow you to modularize your code, promote reusability, and enhance code organization.

Example

```
def greet(name):

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

Function Call: A function call is an instruction that tells the program to execute a specific function at a certain point in the code. It involves using the function name followed by parentheses () and any necessary arguments or parameters inside the parentheses. When a function call is encountered, the program transfers control to the function, executes the code inside the function's body, and then returns to the point of the function call.

Example of a function call:

```
greet("Deepawali")
```

In this example, greet("Deepawali") is a function call that invokes the greet function and passes the argument "Deepawali" to the name parameter. The program executes the code within the greet function, printing "Hello, Deepawali!

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

ANS

In a Python program, there can be one global scope and multiple local scopes.

Global Scope: The global scope refers to the outermost level of a program where variables and functions are defined outside of any function or class. The global scope is accessible from anywhere within the program. Variables and functions defined in the global scope have global visibility, meaning they can be accessed from both the global scope itself and any nested local scopes.

Local Scopes: Local scopes are created when functions or classes are defined within a program. Each function or class creates its own local scope. Variables defined within a function or class are local to that scope and are accessible only within the scope in which they are defined. Local scopes are temporary and are created when the corresponding function or class is called and destroyed when the function or class execution completes.

Example

```
global_var = 10  # Variable in the global scope

def my_function():

    local_var = 20  # Variable in the local scope of my_function

    print(global_var)  # Accessing the global variable

    print(local_var)  # Accessing the local variable

my_function()
```

O/P

```
10

20
```

In this example, there is one global scope that contains the global_var variable. The my_function() creates its own local scope and defines the local_var variable within that scope. The global_var variable can be accessed both in the global scope and within the my_function() scope. However, the local_var variable is accessible only within the my_function() scope.

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

ANS

When a function call returns, the local variables defined within the function's scope are destroyed or deallocated. This means that the memory occupied by those variables is freed up and can be used for other purposes. When a function is called, a local scope is created for that function. Any variables defined within the function, including the function's parameters, exist only within that local scope. These variables are not accessible outside of the function. Once the function execution completes and the function call returns, the local scope is destroyed, and the local variables within it cease to exist. The memory allocated for those variables is released, and the values they held are no longer accessible.

Example

```
def my_function():

    x = 20  # Local variable

    print(x)

my_function()

print(x)  # Raises NameError: name 'x' is not defined
```

In this example, the x variable is defined within the local scope of the my_function(). When the function is called, the value of x is printed within the function's scope. However, when the function call returns, attempting to access the x variable outside of the function results in a NameError because x does not exist in the global scope.

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

ANS

The concept of a return value refers to the value that a function provides back to the caller when the function is executed. It allows a function to produce a result or output that can be used by the calling code or assigned to a variable.

In Python, a function can use the return statement to specify the value it should return. When a return statement is encountered in a function, it immediately terminates the function's execution and passes the specified value back to the caller.

```
def add_numbers(a, b):

    result = a + b

    return result
```

```
sum_result = add_numbers(3, 4)

print(sum_result)  # Output: 7
```

In this example, the add_numbers function takes two parameters, a and b, adds them together, and stores the result in the result variable. The return statement is then used to return the value of result back to the caller. When the function is called with add_numbers(3, 4), it returns the value 7, which is assigned to the variable sum_result. Finally, the value of sum_result is printed, resulting in 7 being displayed.

Regarding the second part of your question, a return value cannot be directly used within an expression. However, you can assign the return value of a function to a variable and then use that variable within an expression. This allows you to use the result of the function call in calculations, comparisons, or other operations.

8.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, the return value of a call to that function is None. In Python, None is a special value that represents the absence of a value or the lack of a return value.

When a function without a return statement is called, it executes its code and reaches the end of the function body without explicitly returning a value. In such cases, Python automatically returns None as the default return value.

Example

```
def greet(name):

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

result = greet("Deepawali")

print(result)  # Output: None
```

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

ANS

To make a function variable refer to a global variable, you can use the global keyword within the function. The global keyword allows you to indicate that a variable inside the function should refer to the global variable with the same name.

Example

```
global_var = 10  # Global variable
```

```
def modify_global_var():

    global global_var  # Declare the variable as global within the function

    global_var += 5  # Modify the global variable

print(global_var)  # Output: 10

modify_global_var()

print(global_var)  # Output: 15
```

In this example, global_var is a global variable initially set to 10. Inside the modify_global_var() function, we use the global keyword to declare that the variable global_var should refer to the global variable with the same name. We then increment the value of global_var by 5. When the function is called with modify_global_var(), it modifies the global variable global_var. The first print statement before calling the function displays the initial value of global_var as 10, and the second print statement after calling the function shows the modified value of global_var as 15.

By using the global keyword, you can explicitly indicate that a variable inside a function should refer to the global variable rather than creating a new local variable with the same name. This allows you to modify or access the global variable from within the function.

10.What is the data type of None?

ANS

The data type of None in Python is called NoneType. It represents the absence of a value or the lack of a specific object.

11.What does the sentence import areallyourpetsnamederic do?

ANS

The sentence "import areallyourpetsnamederic" does not have any specific meaning or functionality in Python. It is not a valid import statement because "areallyourpetsnamederic" is not a recognized module or library in the Python ecosystem.

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

ANS

After importing the spam module, you can call the bacon() function using the following syntax:

```
import spam

spam.bacon()
```

In this example, the spam module is imported using the import statement. Then, to access the bacon() function within the spam module, you use the dot notation (spam.bacon()). This syntax specifies that the bacon() function is part of the spam module. By calling spam.bacon(), you can execute the code within the bacon() function and utilize its functionality.

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

ANS

To save a program from crashing when it encounters an error, you can use error handling techniques to catch and handle exceptions. In Python, exceptions are errors that occur during the execution of a program, and handling them allows you to gracefully recover from those errors and continue the program's execution.

There are several ways to handle exceptions in Python:

Using a try-except block: You can enclose the code that may raise an exception within a try block and then use an except block to specify how to handle the exception

```
try:

    # Code that may raise an exception

except SomeException:

    # Code to handle the exception
```

Using multiple except blocks: You can handle different types of exceptions separately by using multiple except blocks. This allows you to specify different handling mechanisms for different types of exceptions.

```
try:

    # Code that may raise exceptions

except ExceptionType1:

    # Handling code for ExceptionType1
```

```
except ExceptionType2:

    # Handling code for ExceptionType2
```

Using an else block: You can include an else block after the try-except block, which executes if no exceptions are raised. This allows you to define code that should run only when the try block is successfully executed.

```
try:

    # Code that may raise an exception

except SomeException:

    # Code to handle the exception

else:

    # Code to run if no exceptions occur
```

Using a finally block: You can include a finally block after the try-except block, which always executes, regardless of whether an exception occurred or not. This block is useful for cleaning up resources or performing tasks that need to be done regardless of exceptions

```
try:

    # Code that may raise an exception

except SomeException:

    # Code to handle the exception

finally:

    # Code to always run, regardless of exceptions
```

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

ANS

The try and except clauses are used together in Python for error handling, allowing you to catch and handle exceptions that may occur during the execution of code.

The purpose of the try clause is to enclose the code that may raise an exception. It defines a block of code where you anticipate the occurrence of an exception. If an exception occurs within the try block, the execution of that block is interrupted, and the program jumps to the corresponding except block.

The purpose of the except clause is to specify how to handle the exception that occurred within the try block. It defines a block of code that is executed when a specific exception occurs. By catching the exception, you can prevent it from propagating further and crashing the program. Within the except block, you can include code to handle the exception, perform necessary actions, provide error messages, or take any other appropriate steps.

Here's an example to illustrate the use of try and except:

```
try:

    # Code that may raise an exception

    result = 10 / 0  # Division by zero raises a ZeroDivisionError

except ZeroDivisionError:

    # Code to handle the ZeroDivisionError

    print("Error: Division by zero!")
```

In this example, the try block contains code that performs a division by zero, which raises a ZeroDivisionError exception. The except block specifies that it should handle ZeroDivisionError exceptions. When the exception occurs, the program jumps to the except block, and the code inside it is executed. In this case, it prints an error message indicating the division by zero error.

By using the try-except construct, you can catch and handle exceptions, allowing your program to gracefully handle errors and continue its execution without crashing. It provides a way to anticipate and respond to potential errors, enhancing the robustness and reliability of your code.