Python Basic Assignment – 3

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

* **Modularity:** Breaking down code into smaller, self-contained units for better organization and reusability.
* **Abstraction**: Hiding implementation details, using functions without knowing their internal workings.
* **Code Reusability**: Writing functions once and using them in multiple places or projects.
* **Readability:** Enhancing code comprehension by dividing it into smaller logical units.
* **Debugging Ease:** Isolating errors to specific functions for easier troubleshooting.
* **Collaborative Development:** Allowing multiple developers to work on different functions independently.
* **Unit Testing:** Testing individual functions to ensure their correctness.
* **Performance Optimization:** Optimizing functions individually to improve overall program performance.

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

The code in a function runs when the function is called, not when it's specified. When you define a function, you are essentially creating a reusable block of code with a specific name and set of parameters. The function definition specifies what the function does and what parameters it expects, but it doesn't execute the code inside the function at that moment.

To execute the code inside the function, you need to call the function by its name and provide any required arguments or parameters, if applicable. When the function is called, the program jumps to the function's definition, executes the code inside the function body, and then returns to the point where the function was called.

# Function definition

def greet(name):

print(f"Hello, {name}!")

# Function call

greet("John") # This is when the code inside the 'greet' function runs

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

In Python, the def statement is used to create a function. The def statement defines the function's name, parameters (if any), and the code block that will be executed when the function is called.

Here's an example of a simple Python function that adds two numbers:

def add\_numbers(a, b):

result = a + b

return result

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

* **Function:** A function is a block of code that performs a specific task or operation. It is defined using a function declaration, such as def in Python and it includes a name, parameters (optional), and a code block. These are reusable, modular, and organized pieces of code that can be called multiple times from different parts of a program.
* **Function Call:** A function call is the act of invoking or executing a function to perform its defined task. When a function is called, the program jumps to the function's definition, executes the code inside the function's code block, and then returns to the point where the function was called. To call a function, you use the function's name followed by parentheses, optionally passing arguments inside the parentheses if the function expects any parameters.

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

In a Python program, there is only one global scope and multiple local scopes.

* **Global Scope:** The global scope refers to the top-level scope of a Python program. Variables defined in the global scope are accessible throughout the entire program. These are typically defined outside of any function or class and can be accessed from within functions, as long as they are not shadowed by local variables.

global\_var = 10

def my\_function():

print(global\_var)

# Accessing the global variable

my\_function() # Output: 10

* **Local Scopes:** Local scopes are created whenever a function is called. Variables defined inside a function belong to the local scope and are only accessible within that function. Each function call creates a new local scope, and when the function completes execution, the local scope is destroyed along with its variables.

def my\_function():

local\_var = 20 # This is a local variable

print(local\_var)

my\_function() # Output: 20

# print(local\_var)

# This would raise an error since local\_var is not defined in the global scope

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

When a function call returns, the local scope associated with that function is destroyed, and the variables defined within that local scope cease to exist.

Here's what happens to variables in a local scope when a function call returns:

* **Variable Deletion:** All the variables defined within the local scope of the function are deleted. This means that these variables are no longer accessible or usable after the function call returns.
* **Memory Release**: The memory occupied by the local variables is released back to the system, making it available for other parts of the program or other processes.
* **Control Returns to Caller**: After the function execution is complete, the program control returns to the point where the function was called, and the execution continues from that point.

1. **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 in programming refers to the value that a function produces and passes back to the caller after the function is executed. When a function is called, it can perform some operations, manipulate data, or calculate a result, and then use the return statement to send that result back to the point in the program where it was called.

The return statement is used to specify the value that the function should return. Once the return statement is encountered in the function, the function's execution stops, and the specified value (if any) is sent back as the function's return value.

def add\_numbers(a, b):

result = a + b

return result

sum\_result = add\_numbers(10, 20)

print(sum\_result) # Output: 30

1. **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 a special value called None. In Python, None is used to represent the absence of a value, and it serves as the default return value for functions that do not explicitly return anything.

When a function without a return statement is called, it executes its code and completes its operations, but it doesn't provide any specific value to be returned to the caller. In such cases, Python automatically returns None as the implicit return value.

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

To make a function variable refer to the global variable with the same name, you need to use the global keyword within the function. This informs the function that the variable you are using inside the function should be considered as the global variable with that name, rather than creating a new local variable.

global\_var = 10

def modify\_global():

global global\_var

global\_var = 20

print(global\_var) # Output: 10

modify\_global()

print(global\_var) # Output: 20

Without the global keyword, the function would create a new local variable global\_var instead of modifying the global variable, and the changes wouldn't be reflected outside the function. Using global is essential when you want to modify global variables from within a function. However, using global variables should be done with caution, as they can make code harder to understand and maintain, and can lead to unintended side effects. In general, it's often better to pass variables as arguments to functions and return results explicitly rather than relying heavily on global variables.

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

In Python, None is a special constant that represents the absence of a value or a null value. It is often used to indicate that a variable or a function has no specific value or result.

In Python, None is a unique object of the NoneType class. It is used as a placeholder when a value is not available or when you want to initialize a variable without assigning any meaningful value to it.

# Assigning None to a variable

my\_variable = None

# Checking the type of the variable

print(type(my\_variable))

# Output: <class 'NoneType'>

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

In Python, the import keyword is used to import modules or packages, allowing you to access their functions, classes, and variables in your current code.

The statement "import areallyourpetsnamederic" suggests that there might be a Python module named "areallyourpetsnamederic" that the code wants to import. If such a module exists, it would be imported, and you could use its contents in the code that follows.

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

If we have imported a module named "spam" in your Python code, and the "spam" module contains a feature or function called "bacon()", you would call it using the following syntax:

import spam

spam.bacon()

In this example, "spam" is the name of the module you imported, and "bacon()" is the function you want to use from the "spam" module. By using the dot notation (spam.bacon()), you are specifying that you want to access the "bacon()" function that is defined within the "spam" module.

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

To prevent a program from crashing when it encounters an error, you can implement error handling techniques to gracefully handle exceptions that might occur during the program's execution.

* **try-except block:** If an exception occurs within the try block, the corresponding except block will be executed, allowing you to handle the error gracefully.
* **Multiple except blocks:** You can have multiple except blocks to handle different types of exceptions separately.
* **else block:** The else block is executed if no exception occurs in the try block. It can be used to handle the normal flow of code when there are no exceptions.
* **finally block:** The finally block is executed regardless of whether an exception occurred or not. It is typically used for cleanup operations that need to be performed regardless of the outcome.

By using these error handling techniques, you can prevent a program from crashing when it encounters errors and provide a better user experience by presenting informative error messages or handling exceptional cases gracefully.

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

* **Try Block:** The purpose of the try clause in Python is to enclose a block of code that might raise an exception or encounter an error during its execution. The try block acts as a protective container around the potentially problematic code. It allows you to anticipate exceptions and handle them gracefully without causing the entire program to crash.
* **Except Block:** The purpose of the except clause is to define the actions that should be taken when a specific exception occurs within the corresponding try block. If an exception occurs in the try block, Python looks for the appropriate except block that matches the raised exception's type and executes the code inside that block.

You can have multiple except blocks to handle different types of exceptions, each targeting a specific exception type. If no except block matches the exception, the program will terminate with an unhandled exception and display an error message.

try:

num = int(input("Enter a number: "))

result = 10 / num

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

print("Error. Please enter a valid number.")