Python Basic Assignment: - 3

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

Answer:- Functions are advantageous to have in programs for several reasons:

- Reusability: Functions allow you to define a block of code that can be executed multiple times from different parts of the program. This promotes code reuse and avoids duplicating the same code in multiple places.

- Modularity: Functions help in breaking down complex tasks into smaller, manageable units. Each function can focus on a specific task, making the overall code structure more organized and easier to understand, maintain, and debug.

- Abstraction: Functions can encapsulate complex operations and hide the implementation details from the calling code. This allows you to use functions as building blocks without worrying about the internal workings, enhancing code readability and maintainability.

- Testing and Debugging: Functions enable isolated testing and debugging of specific functionality. You can identify and fix issues by unit testing individual functions more efficiently.

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

Answer: - The code in a function runs when it is called, not when it is specified. When a function is called, the program transfers control to the function, executes the code inside the function block, and then returns to the point where the function was called.

3. What statement creates a function?

Answer: - The `def` statement is used to create a function in Python. It is followed by the function name, parentheses (which may contain optional parameters), and a colon. The function code block is indented below the `def` statement.

Example:

```
def my_function():

    # Function code goes here
```

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

Answer: - A function is a block of reusable code that performs a specific task when called. It is a defined piece of code that may or may not take parameters and may or may not return a value. On the other hand, a function call is the actual invocation or execution of the function. It is the statement that triggers the function to run with the provided arguments (if any).

Example:

```
# Function definition

def greet(name):

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


# Function call

greet("Alice")
```

In this example, `greet` is the function, and `greet("Alice")` is the function call.

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

Answer: - In a Python program, there is one global scope, which is the outermost level of the program, accessible from any part of the code. It is where global variables and functions are defined.

The number of local scopes depends on the number of function calls. Each time a function is called, a new local scope is created for that function, which is destroyed once the function execution is complete. Local variables defined within the function are only accessible within that particular function's scope.

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

Answer: - When a function call returns, the local variables defined within the function scope are destroyed, and their values are no longer accessible. Only the return value (if any) is passed back to the calling code.

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

Answer: - The concept of a return value refers to the value that a function can send back to the caller. It allows a function to compute a value or perform a task and provide the result to the part of the program that is called the function.

Yes, it is possible to have a return value in an expression. You can assign the return value of a function to a variable or use it directly in an expression.

Example:

```
def add(a, b):
    return a + b


result = add(3, 4)  # Assigning return value to a variable
print(result)      # Output: 7


print(add(2, 5))    # Using return value in an expression
```

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 will be `None`. `None` is a special Python object that represents the absence of a value.

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

Answer: - To make a function variable refer to a global variable, you can use the `global` keyword within the function. By declaring a variable as global, any assignments made to that variable inside the function will affect the global variable with the same name.

Example:

```
count = 0


def increment():
    global count
    count += 1


increment()
print(count)  # Output: 1
```

10. What is the data type of None?

Answer: - The data type of `None` is `NoneType`. It is a built-in Python object that represents the absence of a value or a null value.

11. What does the sentence import areallyourpetsnamederic do?

Answer: - That import statement imports a module named areallyourpetsnamederic.

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

Answer: - If you had a `bacon()` feature in a `spam` module, after importing the `spam` module, you would call the `bacon()` function using the following syntax:

```
import spam

spam.bacon()
```

This assumes that the `bacon()` function is defined within the `spam` module.

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

Answer: - To save a program from crashing if it encounters an error, you can use error handling techniques such as using try-except blocks. By enclosing potentially error-prone code within a try block and specifying how to handle specific types of errors in the except block, you can gracefully handle exceptions and prevent the program from abruptly terminating.

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. It allows you to test a block of code for errors. If an exception occurs within the try block, the control flow is transferred to the corresponding except block that matches the exception type.

The purpose of the except clause is to define how to handle a specific type of exception. It specifies the code to be executed when a particular exception is raised within the corresponding try block. Multiple except blocks can be used to handle different types of exceptions, allowing for granular error handling.

Example:

```
try:
    # Code that might raise an exception
    result = x / y
except ZeroDivisionError:
    # Code to handle the ZeroDivisionError exception
    print("Error: Division by zero")
except TypeError:
    # Code to handle the TypeError exception
    print("Error: Invalid types for division")
else:
    # Code to execute if no exception occurs
    print("Result:", result)
```

In this example, the code within the try block performs division. If a `ZeroDivisionError` occurs, the first except block is executed. If a `TypeError` occurs, the second except block is executed. If no exception occurs, the else block is executed.