Assignment 3

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

Ans: Functions in Python (or any programming language) offer several advantages:

- Modularity: Functions allow you to break your program into smaller, manageable pieces. This makes it easier to understand, maintain, and debug your code.
- Reusability: Once defined, a function can be used multiple times throughout your program. This avoids code duplication and promotes the DRY (Don't Repeat Yourself) principle.
- Organization: Functions help organize your code into logical sections, each responsible for a specific task. This improves readability and structure.
- Abstraction: Functions can hide complex implementation details. You can use a function without needing to understand its internal workings, as long as you know what it does and how to use it.
- Ease of Testing: Functions can be tested individually. This makes it easier to identify and fix bugs in specific parts of your code without having to test the entire program.
- Parameters and Return Values: Functions can take inputs (parameters) and produce outputs (return values). This allows you to customize behavior and use functions in different contexts.
- Recursion: Functions can call themselves, enabling powerful techniques for solving problems that are naturally recursive.

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's defined.

- **When Defined**: Just defining a function (using the def keyword) doesn’t execute the code inside it. It merely sets up the function and tells Python what to do when the function is called.
- **When Called**: The code inside the function actually runs only when you call the function by its name and provide any required arguments.

3. What statement creates a function?

Ans: In Python, you create a function using the def statement.

Syntax:

```
def function_name(parameters):
```

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

Ans:

- **Function**: A function is a block of reusable code that performs a specific task. It’s defined once using the def keyword and consists of a name, parameters (optional), and a code block. Defining a function doesn't execute its code; it merely sets up the instructions.

Ex:

```
def greet(name):
        print(f"Hello, {name}!")
```

- **Function Call**: A function call is the act of executing or invoking the function you've defined. When you call a function, you use its name followed by parentheses, optionally passing arguments if the function requires them. This triggers the execution of the code inside the function.

  Ex:

  ```
  greet("Alice")
  ```

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

Ans:

- Global Scope: There is one global scope per Python program. This is the scope at the top level of the script or module. Variables defined in the global scope are accessible throughout the entire program, including inside functions, unless shadowed by local variables.
- **Local Scopes**: Each function or method creates a new local scope. So, there can be multiple local scopes, one for each function or method in your program. Variables defined inside a function or method are local to that function and are not accessible outside it.

Ex:

```
# Global scope
x = 10
def my_function():
    # Local scope
    y = 20
    print(x) # Can access global variable
    print(y) # Can access global variable
my_function()
print(x) # Can access global variable
print(y) # Raises an error, y is not accessible outside the function
```

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

Ans: When a function call returns, the variables in the local scope of that function are typically destroyed. This means that:

Local Variables: These variables are removed from memory, and their names are no longer accessible. They only exist for the duration of the function's execution.

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

Ans: In Python, the concept of a **return value** refers to the value that a function sends back to the caller when it finishes executing. The return statement is used to specify this value. Once a return statement is executed, the function terminates, and the specified value is returned to the point where the function was called.

Yes, it is possible to use a return value in an expression. When you call a function in an expression, the function call is replaced by the return value of the function.

Ex:

```
def multiply(x, y):
    return x * y
result = multiply(2, 3) + 10
print(result)  # This will print: 16
```

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 in Python does not have a return statement, or if the return statement is used without specifying a value, the function will return None by default. None is a special value in Python that represents the absence of a value.

Ex:

```
def greet(name):
    print(f"Hello, {name}!")  # No return statement
result = greet("Alice")
print(result)  # This will print: 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 inside the function. This tells Python that you want to use the global version of the variable, not a local one.

Ex:

```
x = 10  # Global variable
def modify_global():
    global x  # Refers to the global variable x
    x = 20  # Modifies the global variable
modify_global()
print(x)  # This will print: 20
```

10. What is the data type of None?

Ans: The data type of None in Python is NoneType.

11. What does the sentence import areallyourpetsnamederic do?

Ans: The sentence import areallyourpetsnamederic attempts to import a module named areallyourpetsnamederic into your Python program.

However, unless there is a module with that exact name in your Python environment or directory, this import statement will raise an ImportError, indicating that the module doesn't exist.

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

Ans:

```
import spam

spam.bacon()
```

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

Ans: To prevent a program from crashing when it encounters an error, you can use exception handling in Python. This is done using the try, except, else, and finally blocks, which allow you to catch and handle errors gracefully.

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

Ans:

- The **try clause** is used to wrap code that might potentially raise an exception (an error). The idea is to "try" running this code and see if it works without any issues. If an error occurs within the try block, the program doesn’t crash; instead, control is passed to the except block, allowing you to handle the error gracefully.
- The **except clause** is used to handle exceptions (errors) that are raised in the try block. If an error occurs, the code in the except block is executed. This allows you to define how your program should respond to specific errors, such as logging the error, displaying an error message to the user, or taking corrective action.