DEFINING A FUNCTION

- **Syntax**-using the **def** keyword, followed by the function **name** and **parameters** enclosed in parentheses. The **function body, which** contains code to be executed, is indented.
- **Parameters and Arguments**- **Parameters** are the variables listed inside the parentheses in the function definition. **Arguments** are the values passed into the function when it is called.
- **Return Statement-**You can return values using the return statement. The function stops executing once a return statement is encountered.
- **Docstrings-**are documentation strings enclosed in triple quotes. They provide information about the purpose and usage of a function.
- **Default Parameters-**values can be specified in the function definition. If an argument is not provided for a parameter with a default value, the default value is used.
- **Arbitrary Number of Arguments-**can accept a variable number of arguments by using *args and **kwargs syntax.
- **Scope-**Variables defined inside a function are scoped to that function and are not accessible outside of it unless explicitly returned.
- **Lambda Functions-**are small anonymous functions defined using the lambda keyword.

REASONS OF USING FUNCTION AND

ADVANTAGES OF USER - DEFINED FUNCTION (user-defined functions play a crucial role in Python programming by promoting modularity, reusability, abstraction, readability, testing, debugging, code organization, scalability, and collaboration. They are a fundamental concept in software development that helps manage complexity and improve efficiency.)

- **Modularity**-It allows the function to break down a program into smaller and more manageable chunks of code. It can perform a specific task or operation and make the codebase easier to understand and work with.
- **Reusability**-When your function is defined, it can be reused multiple times throughout the program. This can save time and reduce the chance of errors if you want to rewrite the same code for similar tasks.
- **Abstraction**-provides a level of abstraction by hiding the implementation details of a particular operation. The users of this function only need to know what the function does and how to use it, without needing to understand how it achieves its functionality internally.
- **Readability**-This function improves its readability because the program breaks down the program into smaller functions. Descriptive function names and well-defined parameters make it easier for other developers to understand the purpose of each piece of code.
- **Testing and Debugging**-isolates the specific parts of the code. It's function performs a specific task; it can be tested individually to ensure it behaves as expected. If a bug is found, it's often easier to pinpoint and fixed within a smaller function rather than searching through the entire codebase.
- **Code Organization**-It helps in organizing code into logical units. By grouping related functionality together, it makes it easier to navigate and maintain the codebase, especially as it grows larger and more complex.

- **Scalability**-allows you to scale your codebase more efficiently, and you can create new functions or update existing ones without affecting other parts of the program, as long as the function interfaces remain consistent.
- **Collaboration**-provides a clear interface for communication and collaboration. Each function acts as a building block that can be developed, tested, and maintained independently, enabling parallel development and teamwork.

TYPES OF FUNCTIONS IN PYTHON

- **Built-in library functions**-These are standard functions in Python that are available to use.Examples include **print()**, **len()**, **type()**, **range()**, **etc**.
- **User-defined function**-We can create our own functions based on our requirements.Using the **'def'** keyword followed by the **function name, parameters, and a block of code.**
- **Anonymous Functions (Lambda Functions)**-They are typically used for simple operations and are often passed as arguments to higher-order functions.
- **Recursive Functions-**calls directly or indirectly in order to solve a problem. They are commonly used in algorithms such as factorial calculation, Fibonacci series generation, and tree traversal.
- **Higher-Order Functions**-can take other functions as arguments or return functions as results. Examples include map(), filter(), and reduce().
- **Generator Functions**-use the **yield** keyword to return values one at a time, rather than all at once. They generate a sequence of values lazily, which can be more memory-efficient than returning a list of values.
- **Decorators**-used for adding functionality to existing functions, such as logging, authentication, or caching.

Rules in Declaring a Function in Python

- **Use the `def` keyword-**Begin to declare the `def` keyword, followed by the function name.
- **Function Name-**It should be lowercase and may contain letters, numbers, or underscores. They cannot start with a number.
- **Parameters**-When the function accepts input, specify the parameters inside the parentheses. Parameters are separated by commas.
- **Indentation**-When you indent the function body, use four spaces (or a tab) to define the scope of the function. Everything indented under the function definition belongs to the function.
- **Docstring (Optional)**-Add a docstring when needed because that describes the purpose of the function, its parameters, and its return value (if any). Docstrings are enclosed in triple quotes.
- **Return Statement (Optional)**-use the `return` statement to return the value. It has a function that can have multiple `return` statements, but once a `return` statement is executed, the function exits.
- **Whitespace**-Separate the function name and the opening parenthesis with a single space. Place one space after each comma in the parameter list.
- **Naming Conventions**-Follow Python's naming conventions (PEP 8) for function names. Use lowercase letters with words separated by underscores (`snake_case`) for function names.
- **Function Call**-When you call a function, use its name followed by parentheses. If the function accepts arguments, provide them within the parentheses.

Python Function Syntax

- **def Keyword**-used to declare a function in Python.
- **Function Name**: This is the identifier for the function.
- **Parameters (optional)**- are enclosed within parentheses. You can specify multiple parameters separated by commas.
- **Colon (:)**-It signifies the end of the function header and the start of the function body.
- **Docstring (optional)**-are enclosed in triple quotes (""" """).
- **Function Body**-contains the statements that define what the function does. The body is indented and can contain any valid Python code.
- **Return Statement (Optional)**-It is used to return a value from the function. The `return` keyword is followed by the value to be returned. If the `return` statement is absent, the function returns `None` by default.

Function Argument and Parameter

Argument:

- are the actual values passed to the function when it is called.
- the data that the function operates on.
- When a function is called, arguments are supplied to match the parameters defined in the function declaration.
- specified inside the parentheses when calling a function.

Parameters:

- are the placeholders defined in the function declaration.
- represent the input values that a function expects to receive when it is called.
- are like variables that are used within the function's body.
- defined inside the parentheses following the function name.

THE RETURN STATEMENT

- **Basic Syntax:**
  ```
  def my_function():
  # Function body
  return value
  ```
- **Exiting the Function**-When the `return` statement is encountered, the function immediately exits, and no further statements in the function body are executed.
- **Returning a Value**-The `return` statement can be followed by an expression or a value that is computed by the function. This value is passed back to the caller, allowing the function to communicate results or computed values.
- **Returning Multiple Values**-Python allows returning multiple values from a function by separating them with commas, and multiple values are returned as a tuple.
- **Returning None**-If the `return` statement is used without an expression, it returns `None`. Functions without a `return` statement implicitly return `None` at the end.
- **Early Exit**-You can use `return` to exit a function early if certain conditions are met.