Day 4: Functions and Modules

1. Functions in Python

A **function** is a reusable block of code that performs a specific task. Instead of writing the same logic repeatedly, we define a function once and use it whenever required. Functions improve **code reusability, readability, and maintainability**.

In Python, functions are defined using the def keyword. A function may take input values (parameters), perform operations, and optionally return a result.

Advantages of Functions

- Reduce code duplication
- Make programs easier to understand
- Improve debugging and testing
- Allow modular program design

2. Function Naming Conventions

Function names should be meaningful and describe the task they perform. Python follows **snake_case** naming style, where words are written in lowercase and separated by underscores.

Examples:

- calculate_total
- find_maximum
- get_user_input

Good naming helps others (and yourself) understand the code easily.

3. Parameters and Arguments

Parameters are variables defined in the function definition.
Arguments are the actual values passed to the function when it is called.

Python supports different types of arguments, making functions very flexible.

[1] Positional Arguments

These arguments are passed in the same order as the parameters are defined. The position of values matters.

[2] Keyword Arguments

In keyword arguments, values are passed using the parameter name. The order does not matter, which improves clarity and reduces errors.

[3] Default Arguments

Default arguments allow a function to use a predefined value if no argument is provided. This makes functions more flexible and reduces the need for multiple function definitions.

[4] *args (Variable-length Arguments)

*args allows a function to accept any number of positional arguments. Internally, these arguments are treated as a tuple. This is useful when the number of inputs is unknown.

[5] **kwargs (Variable-length Keyword Arguments)

**kwargs allows a function to accept any number of keyword arguments. These arguments are stored as a dictionary inside the function. It is commonly used when working with dynamic data.

4. Return Values

A function can return data back to the caller using the return statement. Returning values allows functions to produce results that can be stored, processed, or reused later.

Single Return Value

A function may return one value, such as a number or a string.

Multiple Return Values

Python allows a function to return multiple values at once. These values are returned as a **tuple**, which can be unpacked into multiple variables.

Returning values makes functions more powerful and useful in real-world applications.

5. Scope of Variables

The **scope** of a variable defines where it can be accessed in a program.

Local Variables

Local variables are declared inside a function and can only be used within that function. They are created when the function is called and destroyed when the function ends.

Global Variables

Global variables are declared outside all functions and can be accessed anywhere in the program.

global Keyword

If a function needs to modify a global variable, the global keyword must be used. Without it, Python treats the variable as local.

LEGB Rule

Python follows the **LEGB rule** to search for variables:

- **L – Local** (inside the function)
- **E – Enclosing** (inside nested functions)
- **G – Global** (at the top level)
- **B – Built-in** (Python's built-in names)

Understanding scope helps avoid unexpected errors and bugs.

6. Lambda Functions

A **lambda function** is a small anonymous function written in a single line. It does not have a name and is mainly used for short, simple operations.

Lambda functions are commonly used where a function is required temporarily, especially with built-in functions like map() and filter().

Characteristics of lambda functions:

- No function name
- Single expression
- Short and concise

7. Built-in Functional Tools

Python provides several built-in functions that work efficiently with functions and collections.

map()

The map() function applies a given function to each element of an iterable and returns the transformed result.

filter()

The filter() function selects elements from an iterable based on a condition. Only elements that satisfy the condition are returned.

zip()

The zip() function combines multiple iterables into a single iterable of tuples. It is useful when working with related data.

enumerate()

The enumerate() function adds an index to elements of an iterable. It is commonly used in loops when both index and value are required.

8. Modules in Python

A **module** is a file that contains Python code such as functions, variables, and classes. Modules help organize code into logical sections and promote reusability.

Python has:

- **Built-in modules** (example: math, random)
- **User-defined modules** (created by programmers)

Importing Modules

Modules can be imported using different methods:

- Import the entire module
- Import specific items from a module
- Use aliases for shorter names

Custom Modules

A custom module is created by saving Python code in a .py file. This file can then be imported into other programs and reused.

9. Importance of Modules

- Improve code organization
- Encourage code reuse
- Make large programs manageable
- Support teamwork and scalability