**Module 4: Functions & Modules**

A **function** is a block of reusable code that performs a specific task.

**Create your own functions**

**Syntax:**

def function\_name(parameters):

# code block

return result

**Example:**

def greet(name):

return f"Hello, {name}!"

print(greet("Amit"))

**Types of Functions:**

**Built-in Functions: Predefined (e.g., len(), print(), sum())**

**User-defined Functions: Defined by the user using def**

**Example 2: Function with Parameters**

def add\_numbers(a, b):

result = a + b

return result

# Calling the function

sum\_value = add\_numbers(10, 5)

print("Sum is:", sum\_value)

**Example 3: Function with Return Value**

def multiply(x, y):

return x \* y

output = multiply(4, 3)

print("Multiplication result:", output)

**Example 5: Function that Checks Even or Odd**

def check\_even\_odd(num):

if num % 2 == 0: # (% is Modulo Operator, It returns the **remainder** after division)

return "Even"

else:

return "Odd"

print(check\_even\_odd(7)) # Output: Odd

**Function Parameters**

**Parameter Type Description**

**Required-------------------------------------Must be passed in the correct order**

**Default---------------------------------------Takes a default value if no argument is passed**

**Keyword-------------------------------------Arguments passed by name, not position**

**Arbitrary Positional \*args---------------Accepts variable number of arguments as a tuple**

**Arbitrary Keyword \*\*kwargs-----------** **Accepts variable number of keyword arguments as a dictionary**

1. **Required Parameters**

def greet(name):

print("Hello", name)

greet("Amit")

# greet() # TypeError: missing 1 required argument

1. **Default Parameters**

def greet(name="Guest"):

print("Hello", name)

greet("Amit") # Output: Hello Amit

greet() # Output: Hello Guest

1. **Keyword Arguments**

def student(name, age):

print(f"Name: {name}, Age: {age}")

student(age=21, name="Rahul")

1. **Arbitrary Positional Parameters (\*args)**

**Use when you don't know how many arguments will be passed:**

def add\_numbers(\*args):

total = sum(args)

print("Sum is:", total)

add\_numbers(10, 20, 30)

add\_numbers(5, 15)

1. **Arbitrary Keyword Parameters (\*\*kwargs)**

**Use when you want to accept any number of keyword arguments:**

def print\_info(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key}: {value}")

print\_info(name="Amit", age=30, city="Delhi")

**Variable Arguments**

In Python, variable arguments allow a function to accept a variable number of inputs. There are two main types:

1. \*args – Non-keyword Variable Arguments

* Used when you want to pass a **variable number of positional arguments** to a function.
* Inside the function, args is a **tuple** of all the passed values.

Example:

def add\_numbers(\*args):

total = 0

for num in args:

total += num

print("Sum:", total)

add\_numbers(5, 10, 15)

2. \*\*kwargs – Keyword Variable Arguments

* Used when you want to pass a variable number of keyword arguments (i.e., named arguments).
* Inside the function, kwargs is a dictionary.

Example:

def print\_info(\*\*kwargs):

for key, value in kwargs.items():

print(f"{key}: {value}")

print\_info(name="Amit", course="Python", level="Advanced")

**Scope of a Function**

The scope refers to the region of the program where a variable is recognized or accessible.

**Types of Scope in Python**

Python uses the **LEGB rule** to determine variable scope

**Level Description**

L (Local) Names defined inside a function.

E (Enclosing) Names in the local scope of enclosing functions (for nested functions).

G (Global) Names defined at the top-level of a script/module.

B (Built-in) Names preassigned in Python (like len(), range() etc).

Example:

x = "global"

def my\_func():

x = "local"

print("Inside function:", x)

my\_func()

print("Outside function:", x)

x = "local" is local to the function and does not affect the global variable.

**Global Keyword:**

To modify a global variable inside a function, use the global keyword.

x = 5

def modify():

global x

x = 10

modify()

print(x) # Output: 10

**Function Documentation**

In Python, function documentation refers to the use of docstrings to describe what a function does, its parameters, return values, and other relevant information. This makes your code more readable and maintainable, especially when shared with others.

* Use triple quotes (''' or """) just below the function definition:

def greet(name):

"""

Greets the person with the provided name.

Parameters:

name (str): The name of the person to greet.

Returns:

str: A greeting message.

"""

return f"Hello, {name}!"

**Lambda Functions & map**

A **lambda function** is a small, anonymous function in Python. It is used when you need a function for a short period and usually consists of a single expression.

Syntax:

lambda arguments: expression

Example:

square = lambda x: x \* x

print(square(5)) # Output: 25

The **map() function** applies a given function to all items in an iterable (like a list).

Syntax:

map(function, iterable)

It returns a **map object**, so you often convert it to a list.

* **Using Lambda with map()**

Example 1: Square all numbers in a list

numbers = [1, 2, 3, 4, 5]

squared = list(map(lambda x: x \* x, numbers))

print(squared) # Output: [1, 4, 9, 16, 25]

Example 2: Convert all strings to uppercase

names = ['amit', 'raj', 'sita']

upper\_names = list(map(lambda name: name.upper(), names))

print(upper\_names) # Output: ['AMIT', 'RAJ', 'SITA']

Real-Life Example 3: Rupees convert to dollars

assume 1 dollar = 75 rupees

prices\_in\_rupees = [750, 1500, 2250]

prices\_in\_dollars = list(map(lambda x: x / 75, prices\_in\_rupees))

print(prices\_in\_dollars) # Output: [10.0, 20.0, 30.0]

**Create a Module**

Creating a module in Python is simple — a module is just a Python file (with a .py extension) that contains functions, variables, or classes which you can import into other Python programs.

Step 1: Create a Python File as a Module

Create a file named mymodule.py with the following code:

# mymodule.py

def greet(name):

return f"Hello, {name}!"

def add(a, b):

return a + b

PI = 3.14159

This file is now a module with:

* A function greet
* A function add
* A variable PI

Step 2: Use the Module in Another Python File

Now, create another Python file (e.g. main.py) in the same directory, and use the module:

# main.py

import mymodule

print(mymodule.greet("Amit"))

print("Sum:", mymodule.add(10, 5))

print("Value of PI:", mymodule.PI)

Output

Hello, Amit

Sum: 15

Value of PI: 3.14159

**Standard Modules**

standard modules are built-in modules that come with the Python Standard Library. These modules provide a wide range of functionalities like math operations, file handling, date/time, regular expressions, system interaction, and more — without needing to install anything extra.

**Mathematical Modules**

* math – Mathematical functions like sqrt(), pow(), sin(), etc.
* random – Functions to generate random numbers.
* statistics – Functions for statistical operations like mean(), median().

**Date and Time Modules**

* datetime – Date and time manipulation.
* time – Working with time-related functions like sleep(), time().

**File and Directory Management**

* os – Interact with the operating system (e.g., list files, paths).
* shutil – File operations like copy, move, delete.

**System and Environment Interaction**

* sys – Access to system-specific parameters and functions.
* platform – Information about the platform (OS, processor, etc.).
* getpass – Secure password input.

**Data Handling**

* json – Reading and writing JSON data.
* csv – Working with CSV files.
* pickle – Serializing and deserializing Python objects.

**Text and Pattern Matching**

* re – Regular expression operations.
* string – String constants and functions.

**Internet and Web Services**

* urllib – Handling URLs (open, parse).
* http – Modules for handling HTTP.
* socket – Network connections using TCP/IP.

**Error and Exception Handling**

* logging – Logging messages for tracking.
* warnings – Issue warning messages.
* traceback – Print or retrieve a stack traceback.

**Utilities**

* itertools – Functions for efficient looping.
* functools – Higher-order functions (e.g., lru\_cache, reduce()).
* collections – Specialized container datatypes (deque, Counter).

**Example**

import math

print(math.sqrt(25)) # Output: 5.0

import datetime

print(datetime.datetime.now()) # Current date and time

import random

print(random.randint(1, 10)) # Random number between 1 and 10