Day 11: Modules and Packages in Python

1. Introduction

In Python, **modules** and **packages** help organize code efficiently. Instead of writing everything in one large file, Python allows us to create **modules** (single .py files) and **packages** (directories containing multiple modules).

Additionally, Python provides **built-in libraries** such as:

- math (mathematical operations)
- random (random number generation)
- datetime (date and time management)

These libraries simplify complex tasks, making Python powerful and versatile.

---

2. Importing Modules

A **module** is simply a Python file containing reusable code (functions, classes, and variables). Python allows us to import these modules and use their functionalities.

a) Importing Built-in Modules

Python has many built-in modules, which can be imported using the import statement.

Example 1: Using the math module

```
import math

print("Square root of 25:", math.sqrt(25))

print("Factorial of 5:", math.factorial(5))

print("Value of Pi:", math.pi)
```

Example 2: Using the random module

```
import random

print("Random integer between 1 and 10:", random.randint(1, 10))

print("Random floating-point number:", random.uniform(1.5, 5.5))

print("Random choice from a list:", random.choice(["apple", "banana", "cherry"]))
```

b) Importing Specific Functions

Instead of importing the entire module, we can import specific functions.

```
from math import sqrt, pi
from random import randint

print("Square root of 49:", sqrt(49))
print("Value of Pi:", pi)
print("Random number:", randint(1, 100))
```

---

c) Using Aliases for Modules

We can use an alias to make module names shorter.

```
import datetime as dt

current_time = dt.datetime.now()
print("Current Date and Time:", current_time)
```

---

d) Importing All Functions from a Module (Not Recommended)

We can import everything using *, but it's discouraged because of potential naming conflicts.

```
from math import *

print("Cosine of 0:", cos(0))
print("Exponential of 2:", exp(2))
```

---

3. Creating Custom Modules

We can create our own modules by writing Python functions in separate files.

Steps to Create a Custom Module

1. Create a Python file (custom_module.py).
2. Define functions inside it.
3. Import and use it in another script.

Example 3: Creating a Custom Module

Step 1: Create custom_module.py

```
# custom_module.py


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


def square(num):
    return num * num
```

Step 2: Import and Use the Module

```
import custom_module


print(custom_module.greet("Alice"))
print("Square of 7:", custom_module.square(7))
```

Step 3: Import Specific Functions

```
from custom_module import greet


print(greet("Bob"))
```

---

4. Using Python Libraries

Python comes with powerful libraries that simplify various tasks.

a) math Module – Advanced Examples

```
import math

angle = math.radians(30)  # Convert degrees to radians
print("Sin of 30 degrees:", math.sin(angle))

print("Logarithm (base 10) of 1000:", math.log10(1000))
```

b) random Module – More Randomization Examples

```
import random

random_list = random.sample(range(1, 100), 5)  # Generate 5 random numbers
print("Random sample of 5 numbers:", random_list)

random.shuffle(random_list)  # Shuffle the list
print("Shuffled list:", random_list)
```

---

5. Creating and Using Python Packages

A **package** is a collection of related modules stored in a directory. It must contain an __init__.py file.

Example 4: Creating and Using a Package

Step 1: Create Package Structure

mypackage/

|— __init__.py

|— math_operations.py

|— string_operations.py

Step 2: Define Modules

math_operations.py

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


def multiply(a, b):
    return a * b
```

string_operations.py

```
def uppercase(text):
    return text.upper()


def lowercase(text):
    return text.lower()
```

Step 3: Import and Use the Package

```
from mypackage import math_operations, string_operations


print("Addition:", math_operations.add(4, 5))
print("Uppercase:", string_operations.uppercase("hello"))
```

---

6. Advantages of Using Modules and Packages

✓ **Code Reusability** – Avoid rewriting the same code multiple times.
✓ **Better Code Organization** – Keep related functions together.
✓ **Easy Debugging** – Modular structure makes debugging manageable.
✓ **Namespace Management** – Prevents variable/function name conflicts.

---

7. Summary

Feature	Module	Package
Definition	A single Python file (.py)	A directory with multiple modules
Purpose	Organize reusable code	Organize multiple related modules
Example	math.py, random.py	numpy, pandas
Import Syntax	import module_name	from package import module
