Python Packages and Modules

S1 MCA

Packages

 In Python, a package is a way to organize related modules into a hierarchical directory structure. It serves as a container for multiple modules and sub-packages, allowing for better code organization, reusability, and management, especially in larger projects.

• Directory Structure:

 A package is essentially a directory that contains Python modules (individual .py files) and potentially other sub-directories, which can themselves be sub-packages.

• __init__.py file (Optional in modern Python):

Historically, a directory needed to contain a special file named __init__.py to be recognized as a Python package. This file could be empty or contain initialization code that runs when the package is imported. While still commonly seen, in Python 3.3 and later, a directory can be recognized as a package even without an __init__.py file.

Modules:

Within a package directory, individual .py files represent modules. These modules can contain functions, classes, and variables.

Sub-packages:

A package can contain sub-directories that are themselves packages, creating a nested structure. Each sub-package can also contain its own modules and __init__.py file (if needed for older Python versions or specific initialization logic).

Key benefits of using Python packages:

- Organization: Packages help in logically grouping related code, making large projects more manageable and easier to navigate.
- Namespace Management: They provide a distinct namespace, preventing naming conflicts between modules or variables from different parts of your application.
- Reusability: Packages facilitate code reuse by allowing you to import and utilize specific modules or functionalities from a package in different parts of your project or in other projects.
- Distribution: Packages are the standard way to distribute Python code to others, often through tools like pip and platforms like PyPI (Python Package Index).

Built-in Packages

- Built-in packages (also called the Python Standard Library) are collections of ready-made modules that come *pre-installed* with Python.
- That means:
 - → ☐ You don't need to install them using pip.
 - $\rightarrow \square$ You can **import and use** them directly in your program.
- They give you ready-to-use functions for tasks like:

Math operations, Date and time handling, File and directory access, Random numbers, Internet access, Data compression, Regular expressions And much more!

Commonly Used Built-in Modules

III Math and Numbers		
Module	Purpose	Example
math	Mathematical functions	math.sqrt(25)
random	Random numbers	random.randint(1, 10)
statistics	Mean, median, mode	<pre>statistics.mean([1,2,3])</pre>

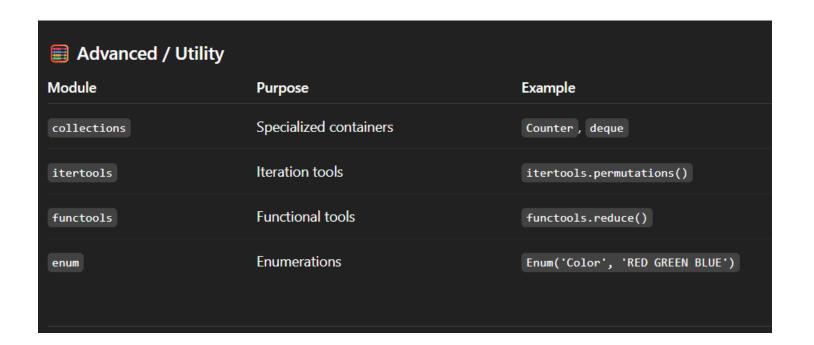
■ Date and Time		
Module	Purpose	Example
datetime	Date and time handling	<pre>datetime.now()</pre>
time	Time delays, timestamps	time.sleep(2)

File and Directory Handling		
Module	Purpose	Example
os	Operating system functions	os.listdir()
sys	System-specific info	sys.argv
shutil	File copying/moving	<pre>shutil.copy('a.txt','b.txt')</pre>

Data and Text Processing		
Module	Purpose	Example
json	JSON parsing	json.loads('{"a":1}')
CSV	Read/write CSV files	csv.reader(file)
re	Regular expressions	re.findall(r'\d+', 'abc123')
string	Common string constants	string.ascii_letters

Internet and Networking		
Module	Purpose	Example
http	HTTP handling	http.client
urllib	Access web pages	urllib.request.urlopen()
socket	Network programming	socket.socket()

🚍 Data Storage and Compression		
Module	Purpose	Example
sqlite3	Database operations	<pre>sqlite3.connect()</pre>
zipfile	ZIP file handling	<pre>zipfile.ZipFile()</pre>
gzip	File compression	<pre>gzip.open()</pre>



Example:

import math, random, datetime, os print("Random number:", random.randint(1, 100)) print("Square root of 64:", math.sqrt(64)) print("Today:", datetime.date.today()) print("Current directory:", os.getcwd())

What Are User-Defined Packages?

- A user-defined package is a collection of modules (Python files) that you create yourself to organize your code.
- It helps you:
- Reuse your own code easily
- Organize large projects
- Avoid repeating the same functions in multiple files

Basic Components

- A package is simply a folder that contains:
- One or more .py files (called modules)
- A special file named __init__.py

The "__init__.py" file tells Python that this folder should be treated as a package.

Creating a User-Defined Package

Creating a User-Defined Package

Folder Structure

Step 1 – Create the package folder mkdir mypackage cd mypackage New-Item __init__.py
New-Item operations.py

Step 2 – Write code in operations.py

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

Step 3 – Write code in main.py (outside the package)

```
# Import the module from your package
from mypackage import operations
print("Addition:", operations.add(10, 5))
print("Subtraction:", operations.subtract(10, 5))
```

Step 4 – Run the program

Import entire module

import mypackage.operations
print(mypackage.operations.add(3, 4))

Import specific functions

from mypackage.operations import add print(add(3, 4))

Import all functions

from mypackage.operations import * print(add(10, 2)) print(subtract(10, 2))

Concept	Description
Module	A single .py file (e.g., operations.py)
Package	A folder containing multiple modules +initpy
Import	Allows access to modules and their functions
Purpose	Organize, reuse, and maintain code easily