

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**.
 - □ 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

Math and Numbers

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

Date and Time

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

File and Directory Handling

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

Data and Text Processing

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

Internet and Networking

Module	Purpose	Example
<code>http</code>	HTTP handling	<code>http.client</code>
<code>urllib</code>	Access web pages	<code>urllib.request.urlopen()</code>
<code>socket</code>	Network programming	<code>socket.socket()</code>

Data Storage and Compression

Module	Purpose	Example
<code>sqlite3</code>	Database operations	<code>sqlite3.connect()</code>
<code>zipfile</code>	ZIP file handling	<code>zipfile.ZipFile()</code>
<code>gzip</code>	File compression	<code>gzip.open()</code>

Advanced / Utility

Module	Purpose	Example
<code>collections</code>	Specialized containers	<code>Counter</code> , <code>deque</code>
<code>itertools</code>	Iteration tools	<code>itertools.permutations()</code>
<code>functools</code>	Functional tools	<code>functools.reduce()</code>
<code>enum</code>	Enumerations	<code>Enum('Color', 'RED GREEN BLUE')</code>

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

```
project/  
|  
├─ main.py  
└─ mypackage/  
    ├─ __init__.py  
    └─ operations.py
```

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 <code>.py</code> file (e.g., <code>operations.py</code>)
Package	A folder containing multiple modules + <code>__init__.py</code>
Import	Allows access to modules and their functions
Purpose	Organize, reuse, and maintain code easily