**Modules and Packages**

**Modules:**

**So, lets us assume modules as files and folders as packages.**

Module is nothing but a file that contains python code in that in form of functions classes and variables which we can use in other programs.

Modules are used to reuse code.

Any file with .py extension is called as a module.

So here assume we are developing a big project ABCD so we divide this into module A,B,C,D so we will be using this modules in another project if it is needed so code reusability is a very important thing in programming.

In modules we have 2 types of modules:

* User defined modules
* Built-in modules

**Built-in modules :**

The modules which are already predefined in python

So lets have a look on some built-in modules:

1.Math

2. Random

3. Datetime

These are few predefined modules in python let’s see with one simple examples

Import math

Print(math.pi) #output: 3.141592653589793

\*\*\*\*

import random

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

\*\*\*\*

import datetime

now = datetime.datetime.now()

print(now) # Output: Current date and time

**Userdefined modules :**

1. **Code Reusability: You can reuse the same code in multiple scripts without rewriting it.**
2. **Organization: Code can be organized into logical components, making it easier to maintain.**
3. **Namespace Management: By importing modules, you can avoid naming conflicts in your code.**

The modules which are created by programmers is called userdefined modules

Here I have defined some functions in the module file

def add(a,b):  
 return a+b  
def sub(a,b):  
 return a-b  
def mul(a,b):  
 return a\*b  
def div(a,b):  
 return a/b

So I need to do some operations on add.sub,mul and div to do these operations no need to do the coding from the scratch so just I will import those modules.

from module import \*

now just I will do the operations which ever I needed.

a = 9  
b = 7  
  
c = add(a,b)  
print(c) # 16  
  
c = sub(a,b)  
print(c) # 2

c= mul(a,b)  
print(c) # 63

c = div(a,b)  
print(c) # 1.2857142857142858

**Package**

Package is the collection of modules and subpackages

Each package contains a special file called as \_\_init\_\_.py

When a directory contains a \_\_init\_\_.py file python treats the directory as a package allowing u to import modules and sub packages.

In python we have larger developer community so code which is written by the developer he will add that package to the Pypl

So knowing about the pypl(python package index) we call this pypl as a repository more than 2 lakh packages are present in this repository so we will install those packages from this repository with the help of pip we can say this as package manager we can install, uninstall, search , upgrade, list operations in with this pip.

**Structure of a Python Package**

my\_package/

\_\_init\_\_.py

module1.py

module2.py

sub\_package/

\_\_init\_\_.py

submodule1.py

submodule2.py

* **my\_package/**: The main package directory.
* **\_\_init\_\_.py**: A special file that marks the directory as a package.
* **module1.py, module2.py**: Python module files that are part of the package.
* **sub\_package/**: A sub-package within the main package, which can contain its own modules and \_\_init\_\_.py.

**Creating a Package**

To create a package in Python, follow these steps:

1. **Create a directory** for your package.
2. **Add an \_\_init\_\_.py file** (this can be empty or contain initialization code).
3. **Add Python module files** (.py files) for your code.
4. **Optional**: You can create sub-packages by creating nested directories with their own \_\_init\_\_.py.

**Example: Creating a Package**

1. Create the following directory structure:

markdown

Copy code

my\_package/

\_\_init\_\_.py

module1.py

module2.py

1. Inside module1.py:

python

Copy code

# module1.py

def greet(name):

return f"Hello, {name}!"

1. Inside module2.py:

python

Copy code

# module2.py

def add(a, b):

return a + b

1. Inside \_\_init\_\_.py (can be empty):

python

Copy code

# \_\_init\_\_.py

# This file can be empty or contain package initialization code.

**Using the Package**

Once the package is created, you can import and use it in another Python script.

1. **Importing the package**:

python

Copy code

# main.py

import my\_package.module1

import my\_package.module2

print(my\_package.module1.greet("Alice")) # Output: Hello, Alice!

print(my\_package.module2.add(5, 3)) # Output: 8

1. **Using the from ... import ... statement**:

python

Copy code

# main.py

from my\_package.module1 import greet

from my\_package.module2 import add

print(greet("Bob")) # Output: Hello, Bob!

print(add(10, 5)) # Output: 15

**Creating Sub-Packages**

A **sub-package** is simply a package inside another package. You can create sub-packages by adding another directory with an \_\_init\_\_.py file inside your main package.

**Example of a Sub-Package:**

Let's extend our package by adding a sub-package inside my\_package:

1. Directory structure:

markdown

Copy code

my\_package/

\_\_init\_\_.py

module1.py

module2.py

sub\_package/

\_\_init\_\_.py

submodule1.py

submodule2.py

1. **Inside submodule1.py**:

python

Copy code

# submodule1.py

def square(n):

return n \* n

1. **Inside submodule2.py**:

python

Copy code

# submodule2.py

def cube(n):

return n \* n \* n

1. **Using the sub-package**:

python

Copy code

# main.py

from my\_package.sub\_package import submodule1, submodule2

print(submodule1.square(4)) # Output: 16

print(submodule2.cube(3)) # Output: 27

**Importing Packages**

You can import the whole package, specific modules, or specific functions from a package:

* **Import the entire package**:

python

Copy code

import my\_package

* **Import specific modules**:

python

Copy code

import my\_package.module1

* **Import specific functions from modules**:

python

Copy code

from my\_package.module1 import greet

**Advantages of Using Packages**

1. **Organization**: Packages help keep your code organized and structured, especially for large projects.
2. **Code Reusability**: You can reuse the same modules across multiple projects by importing the package.
3. **Namespace Management**: Packages help avoid naming conflicts by grouping related code into different namespaces.