"""
Package:
A package is collection of sub-packages and modules.

Note:
A Package consists a __init__.py file which is usually empty.
But, We can include the code in that file which should be executed
when the package is imported.
(A Folder with multiple python files and a __init__.py file
is called a package).

Syntax:
1. from <package_name> import <module_name>
Usage:
<module_name>.<variable / function / class>

2. from <package_name>.<module_name> import <variable_name> / <function_name> /
<class_name>
Usage:
<variable / function / class>

3. from <package_name>.<sub_package>.<module_name> import <variable_name> /
<function_name> / <class_name>
Usage:
<variable / function / class>

In Python, a package is a way to organize related modules and sub-packages into a hierarchical directory structure. This structure helps manage larger projects by providing a clear namespace and promoting code reusability.
Key characteristics of Python packages:
Directory-based: A package is essentially a directory that contains Python modules and potentially other sub-packages.
__init__.py file: Every package directory must contain a special file named __init__.py. This file can be empty, but its presence signifies to Python that the directory is a package and should be treated as such when imported. It can also contain initialization code for the package.
Modules: Packages contain individual Python files (modules) that define functions, classes, and variables related to a specific aspect of the package's functionality.
Sub-packages: Packages can contain other packages nested within them, creating a deeper level of organization.
Namespace: Packages provide a namespace, which helps prevent naming conflicts between modules or variables in different parts of a large project.
Benefits of using packages:
Code organization: Packages help structure code logically, making it easier to navigate and understand, especially in complex projects.
Modularity and reusability: They promote modular design by grouping related functionalities, allowing for easier reuse of code across different projects or within the same project.
Collaboration: Packages facilitate collaboration by providing a clear structure for different developers to work on specific parts of a larger application.
Distribution: Packages are the standard way to distribute Python libraries and tools to others, typically through package managers like pip and repositories like PyPI.
Example:
Consider a project for a "math operations" library. You could organize it into a package structure like this:
Code

```
math_operations/
    __init__.py
    basic_operations.py
    advanced_operations/
        __init__.py
        calculus.py
        statistics.py
```

In this example:

math_operations is the main package.
basic_operations.py is a module within the main package.
advanced_operations is a sub-package within math_operations.
calculus.py and statistics.py are modules within the advanced_operations sub-package.
You can then import and use these modules using dot notation, for example:
Python

```
from math_operations import basic_operations
from math_operations.advanced_operations import calculus

# Use functions from the imported modules
result = basic_operations.add(5, 3)
derivative = calculus.differentiate(expression, variable)

"""
#
from MyPackage1 import my_module1

my_module1.foo()


#
from MyPackage1.my_module1 import foo

foo()


#
from MyPackage1.InnerPackage import inner_module

inner_module.bar()


#
from MyPackage1.InnerPackage.inner_module import bar

bar()


#
import MyPackage2
```