

DATA SCIENCE COURSE TUTORIAL # 39

3.26 Modules and Packages in Python

What Is a Module

A module is a Python file that contains functions, variables and classes. It helps you organize code into separate files. Instead of writing all code in one file, you can split it into modules for better structure and readability.

Why Modules Are Important

Modules make code reusable. They keep programs clean and organized. They help manage large projects easily. They also improve maintainability and readability of code.

Creating a Module

A module is created by saving Python code in a file with a .py extension.

Example module file named math_utils.py.

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

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

Using a Module

You can use a module by importing it into another Python file.

```
import math_utils

print(math_utils.add(5, 3))
print(math_utils.subtract(10, 4))
```

Importing Specific Functions

You can import specific functions from a module.

```
from math_utils import add  
  
print(add(4, 6))
```

Using Alias with Modules

Alias allows you to use a short name for a module.

```
import math_utils as mu  
  
print(mu.add(7, 2))
```

Built in Modules

Python provides many built in modules like math, random, datetime and os.

Example using math module.

```
import math  
  
print(math.sqrt(16))
```

What Is a Package

A package is a collection of related modules stored inside a folder. It helps organize multiple modules together. Packages are useful for large applications.

Creating a Package

A package is created by making a folder and placing modules inside it. An `init.py` file is used to mark the folder as a package.

Folder structure example.

```
my_package/  
    __init__.py  
    module1.py  
    module2.py
```

Using a Package

You can import modules from a package.

```
from my_package import module1  
  
module1.some_function()
```

Importing from Sub Modules

You can import specific items from package modules.

```
from my_package.module2 import another_function  
  
another_function()
```

init File Role

The `init.py` file runs when a package is imported. It can initialize package variables or control what gets imported.

name == "main" Statement

The `name == "main"` statement is used to control the execution of code in a Python file. It helps decide whether a file is being run directly or imported as a module.

When a Python file is run directly, Python sets the value of `name` to `"main"`. When the same file is imported into another file, `name` is set to the file name instead.

Why name == "main" Is Important

This statement prevents certain code from running automatically when a module is imported. It allows you to separate reusable code from test or execution code. It is very useful in real world projects and large programs.

Example Without name == "main"

```
# file: demo.py  
print("This code will always run")
```

If this file is imported, the print statement will run automatically.

Example With name == "main"

```
# file: demo.py

def greet():
    print("Hello from module")

if __name__ == "__main__":
    greet()
```

Now the greet function will run only when the file is executed directly, not when it is imported.

Using `name == "main"` With Modules

```
# file: math_utils.py

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

if __name__ == "__main__":
    print(add(5, 3))
```

When this file is imported into another program, the print statement will not run.

Benefits of Using `name == "main"`

It prevents unwanted code execution. It makes modules reusable. It helps in testing code safely. It improves project structure and clarity.

Benefits of Modules and Packages

Modules and packages improve code organization. They support code reuse. They make debugging and maintenance easier. They are essential for building large Python projects.

Summary

- A module is a single Python file that contains code.
- A package is a folder that contains related modules.
- The `name == "main"` statement controls code execution.
- It prevents automatic execution when importing modules.
- Modules and packages improve structure, safety and reusability of code.