Python Programming
Unit 03 – Lecture 03 Notes
Modules, Packages, and the Standard Library

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

As programs grow, keeping everything in one file becomes messy. Modules and packages help you:

- reuse code (write once, import anywhere),
- keep a clean structure,
- and avoid "copy-paste programming".

This lecture also introduces useful standard library modules.

2 Core Concepts

2.1 What is a Module?

A **module** is simply a `.py` file. When you import a module, Python executes that file (top to bottom) once, and then you can use its variables and functions.

2.2 Import Styles (and When to Use Them)

Style 1: import module

```
import math
print(math.sqrt(25))
```

Pros: clear namespace (`math.sqrt`). Less name collision.

Style 2: import module as alias

```
import math as m
print(m.pi)
```

Pros: shorter, still keeps namespace.

Style 3: from module import name

```
from math import sqrt, pi
print(sqrt(9), pi)
```

Pros: shorter calls. Cons: can shadow your own variable names.

2.3 `__name__` and the Script Entry Point

Python sets a special variable `__name__`:

- If a file is executed directly, `__name__ == "__main__"`.
- If a file is imported, `__name__` is the module name.

This is why many scripts use:

```
def main():
    print("Hello")

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

So that importing does not accidentally run the script logic.

2.4 What is a Package?

A **package** is a folder that groups related modules. Traditionally, it contains `__init__.py`.

Example structure used in the demo:

```
demo/
  my_utils/
    __init__.py
    math_helpers.py
    text_helpers.py
  use_my_utils.py
```

2.5 A Few Standard Modules You Should Know

`sys`

Common uses:

- `sys.argv` to read command-line arguments
- `sys.path` to see the module search paths

```
import sys
print(sys.argv)
```

`math`

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

`time`

```
import time
start = time.time()
time.sleep(1)
print("Elapsed:", time.time() - start)
```

`os` **and** `pathlib`

`pathlib` is often nicer for paths:

```
from pathlib import Path
root = Path(".")
print(root.resolve())
```

3 Demo Walkthrough

Package: `demo/my_utils/`
Script: `demo/use_my_utils.py`

How to run

From the lecture folder:

```
python demo/use_my_utils.py "hello world"
```

What to observe

- How functions are imported from a local package.
- How `sys.argv` can accept input from terminal.
- How standard modules (`math`, `time`, `pathlib`) are used.

4 Interactive Checkpoints (with Solutions)

Checkpoint 1 Solution

Question: value of `__name__` when run directly vs imported?
Answer:

- Run directly: `"__main__"`
- Imported: module name (e.g., `"math_helpers"`)

Checkpoint 2 Solution

Question: when prefer `import module` over `from module import name`?
Answer:

- When you want to avoid name conflicts and keep a clear namespace.
- When importing many things; `module.name` stays readable.

5 Practice Exercises (with Solutions)

Exercise 1: Create a Module

Task: Create `calc.py` with function `add(a,b)` and import it in another file.
Solution (idea):

```
# calc.py
def add(a, b):
    return a + b
```

```
# main.py
import calc
print(calc.add(2, 3))
```

Exercise 2: Use `sys.argv`

Task: Print the first command-line argument (if provided).
Solution:

```
import sys
if len(sys.argv) >= 2:
    print("Arg1:", sys.argv[1])
else:
    print("No argument provided.")
```

6 Exit Question (with Solution)

Question: Which file makes a folder a package?
Answer: `__init__.py`