Python Programming

Unit 03 – Lecture 03: Modules, Packages, and the Standard Library

Tofik Ali

School of Computer Science, UPES Dehradun

February 14, 2026

Repository: `https://github.com/tali7c/Python-Programming`

Quick Links

Core Concepts Demo Interactive Summary

Agenda

Learning Outcomes

- Explain what modules and packages are in Python

Learning Outcomes

- Explain what modules and packages are in Python
- Import modules using different import styles

Learning Outcomes

- Explain what modules and packages are in Python
- Import modules using different import styles
- Use `__name__ == "__main__"` to control script execution

Learning Outcomes

- Explain what modules and packages are in Python
- Import modules using different import styles
- Use `__name__ == "__main__"` to control script execution
- Use common standard modules: `sys`, `math`, `time`, `os`, `pathlib`

Why Modules?

- Organize code into reusable files

Why Modules?

- Organize code into reusable files
- Improve readability and maintainability

Why Modules?

- Organize code into reusable files
- Improve readability and maintainability
- Enable teamwork (different files for different features)

Why Modules?

- Organize code into reusable files
- Improve readability and maintainability
- Enable teamwork (different files for different features)
- Avoid copying-pasting functions between scripts

A Module is a `.py` File

Example:

- `math_helpers.py` contains functions

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

A Module is a .py File

Example:

- math_helpers.py contains functions
- another script imports and uses them

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

Import Styles

- `import module`

```
import math as m
from math import pi, sqrt
```

Import Styles

- `import module`
- `import module as alias`

```
import math as m
from math import pi, sqrt
```

Import Styles

- `import module`
- `import module as alias`
- `from module import name`

```python
import math as m
from math import pi, sqrt
```

Import Styles

- `import module`
- `import module as alias`
- `from module import name`
- `from module import name as alias`

```python
import math as m
from math import pi, sqrt
```

`__name__` and Script Entry Point

- If a file is executed directly, `__name__` is `"__main__"`

```
def main():
    print("Running as a script")

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

`__name__` and Script Entry Point

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

```
def main():
    print("Running as a script")

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

What is a Package?

- A package is a folder that groups modules

Example structure:

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

What is a Package?

- A package is a folder that groups modules
- Conventionally contains `__init__.py`

Example structure:

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

Standard Modules You Should Know

- `sys`: command-line args, Python path

Standard Modules You Should Know

- `sys`: command-line args, Python path
- `math`: math functions/constants

Standard Modules You Should Know

- `sys`: command-line args, Python path
- `math`: math functions/constants
- `time`: timestamps, delays

Standard Modules You Should Know

- `sys`: command-line args, Python path
- `math`: math functions/constants
- `time`: timestamps, delays
- `os` and `pathlib`: filesystem and paths

Example: sys.argv

```python
import sys
print(sys.argv)  # list of command-line arguments
```

- Useful when building scripts that take inputs from terminal

Demo: Create a Small Package

- Package: `demo/my_utils/`

Demo: Create a Small Package

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

Demo: Create a Small Package

- Package: `demo/my_utils/`
- Script: `demo/use_my_utils.py`
- Also demonstrates standard modules (`sys`, `math`, `time`, `pathlib`)

Checkpoint 1

Question: What is the value of `__name__` when:

- you run a file directly?
- you import that file as a module?

Checkpoint 2

Question: When should you prefer `import module` over `from module import name`?

Think-Pair-Share

You have a project with 200 lines of code in one file. Discuss how you would split it into modules:

- What goes into `utils.py`?
- What stays in `main.py`?

Key Takeaways

- Modules and packages help organize and reuse code

Key Takeaways

- Modules and packages help organize and reuse code
- `__name__ == "__main__"` controls what runs on import

Key Takeaways

- Modules and packages help organize and reuse code
- `__name__ == "__main__"` controls what runs on import
- The standard library provides powerful tools without extra installs

Exit Question

Create a package named `tools` with a module `helpers.py`.
Which file makes `tools` a package?