Intermediate Python Programming – Lesson 6

Facilitated by Kent State University
Topic: Organizing Code with Modules and Packages
Duration: 1 Hour

Learning Objectives

By the end of this lesson, participants will be able to:

- Create and import Python modules
- Structure a project using packages and sub-packages
- Understand Python's import system and module resolution rules

---

Lesson 6: Organizing Code with Modules and Packages

I. Introduction to Code Organization (10 minutes)

As projects grow, organizing code becomes essential. Python provides a powerful and simple way to structure programs into reusable and maintainable pieces through modules and packages.

- **Modules**: Any `.py` file that defines functions, classes, or variables
- **Packages**: Directories that contain a special `__init__.py` file, allowing grouping of related modules

Benefits:

- Encourages reuse
- Improves maintainability
- Supports collaboration and separation of concerns

Multiple-Choice Question

What is a module in Python?

A. A special variable that stores system paths
B. A compiled C extension
C. A single Python file with functions and classes
D. A folder of Python scripts

Answer: C. A single Python file with functions and classes

Short Answer Question

What is the main purpose of using modules?

Expected Answer: To separate code into reusable, manageable parts.

II. Creating and Importing Modules (15 minutes)

To create a module, simply write a Python file with function or class definitions. To use it, import it in another script.

Example: `math_utils.py`

```
def square(x):
    return x * x

def cube(x):
    return x * x * x
```

Importing the module:

```
import math_utils
print(math_utils.square(3))  # Output: 9
```

Or import specific functions:

```
from math_utils import square
print(square(5))  # Output: 25
```

Modules can be located in the same directory or anywhere on the Python path.

Exercise 1:

Create a file called `greetings.py` with a function `say_hello(name)` that prints "Hello, !". Import and call it from another file.

III. Structuring Projects with Packages (15 minutes)

A **package** is a directory that contains an `__init__.py` file and one or more modules. It allows grouping related code in a hierarchical structure.

Project Example:

```
my_project/
|— main.py
|— utils/
    |— __init__.py
    |— math_tools.py
    |— string_tools.py
```

You can then import using:

```python
from utils.math_tools import square
```

`__init__.py`

This file can be empty or used to define what gets imported when the package is imported. It marks the directory as a package.

Exercise 2:

Create a package `shapes/` with:

- `__init__.py`
- `circle.py` with an `area(radius)` function
- `square.py` with an `area(side)` function

Write `main.py` that imports both and uses the area functions.

IV. Understanding Python's Import System (10 minutes)

Python searches for modules in a specific order:

1. The current directory
2. Directories in `PYTHONPATH`
3. Standard library directories
4. Site-packages for installed packages

You can inspect the module search path:

```python
import sys
print(sys.path)
```

You can dynamically import modules using `__import__()` or `importlib`, but use this only when necessary.

Exercise 3:

Print the contents of the Python search path and locate where a module is being imported from.

```python
import math
print(math.__file__)
```

V. Recap and Q&A (10 minutes)

- A module is a single `.py` file; a package is a folder with an `__init__.py`
- Use `import`, `from ... import ...`, and aliases (`import x as y`) to control scope
- Structure projects using directories and submodules for readability and reuse
- Use `sys.path` and `__file__` to explore the import system

Final Exercise:

Create a reusable package named `tools/` with submodules for logging, math operations, and file utilities. Import and use them from a script in the project root.
