Built-in Modules, and Third-Party Packages

Duration: 1 hour
Level: Beginner to Intermediate
Prerequisites: Basic Python (variables, functions, loops), Python 3.x with pip, and a text editor/IDE (juypter notebook).

Objectives

By the end of this lab, you will be able to:

1. Organize code into separate local files and import functions from them.
2. Utilize Python's built-in modules for common tasks.
3. Install third-party packages using pip and integrate them into your code.

Introduction

In Python, code reusability is key to efficient programming. This lab explores three ways to achieve this:

- importing functions from your own local files (modules).
- using Python's standard built-in modules.
- installing/using external third-party packages.

We'll work through hands-on exercises to demonstrate each concept. Allocate ~15–20 minutes per section, leaving time for testing and debugging.

Introduction:

In Python, code reusability is key to efficient programming. This lab explores three ways to achieve this: importing functions from your own local files (modules), using Python's standard built-in modules, and installing/using external third-party packages. We'll work through hands-

on exercises to demonstrate each concept. Allocate about 15-20 minutes per section, with time for testing and debugging.

Section 1: Using Functions from Local Files

Python allows you to split code across multiple files for better organization. These files act as **modules** that you can import into your main script.

Step 1: Create a Local Module `mymodule.py`

1. Open your text editor or IDE.
2. Create a new file named `mymodule.py` in a directory (e.g., create a folder called `week8_lab`).
3. Add the following code to `mymodule.py`

In [4]:
```python
def greet(name):
    """A simple function to greet someone."""
    return f"Hello, {name}! Welcome to Week 8."

def add_numbers(a, b):
    """Adds two numbers and returns the result."""
    return a + b
```

4. Save the file.

Step 2: Import and Use the Module

1. In the same directory, create another file named main.py.
2. Add the following code

A) Import the entire module

In [5]:
```python
# Import the entire module
import mymodule
```

```
# Use the functions
print(mymodule.greet("Student"))  # Output: Hello, Student! Welcome to Week 8.
result = mymodule.add_numbers(5, 10)
print(f"The sum is: {result}")    # Output: The sum is: 15
```

```
Hello, Student! Welcome to Week 8.
The sum is: 15
```

3. Alternatively, import specific functions

In [6]:
```
from mymodule import greet, add_numbers

print(greet("Student"))
result = add_numbers(5, 10)
print(f"The sum is: {result}")
```

```
Hello, Student! Welcome to Week 8.
The sum is: 15
```

4. Run `main.py` from the command line:

- Open a terminal/command prompt.
- Navigate to your directory (e.g., cd `week8_lab`).
- Type python `main.py` and press Enter.
- Verify the output.

Exercise 1

- Modify mymodule.py to add a new function multiply_numbers(a, b) that returns the product of two numbers
- Update main.py to import and use this new function (e.g., multiply 3 and 4).
- Run the script and note any errors if the import is incorrect (e.g., if files are in different directories).
- Tip: If modules are in subdirectories, use dot notation (e.g., from subdirectory.mymodule import greet).

> **Discussion:** Why is modular code better?
> Hints: Easier maintenance, reusability across projects, clear separation of concerns.

Section 2: Using Built-in Modules

Python includes a rich **standard library** of built-in modules—no installation needed! These provide ready-made functions for tasks like math, dates, file handling, randomness, and more.

Step 1: Explore Built-in Modules — `math` and `datetime`

In [7]:
```python
# math and datetime examples
import math
from datetime import datetime

# Calculate square root
sqrt_value = math.sqrt(16)
print(f"Square root of 16: {sqrt_value}")  # 4.0

# Use pi constant
circle_area = math.pi * (5 ** 2)
print(f"Area of circle with radius 5: {circle_area:.2f}")  # ~78.54

# Current date/time and formatting
now = datetime.now()
print(f"Current date and time: {now}")
formatted_date = now.strftime('%Y-%m-%d %H:%M:%S')
print(f"Formatted: {formatted_date}")
```

```
Square root of 16: 4.0
Area of circle with radius 5: 78.54
Current date and time: 2025-10-26 17:40:39.142641
Formatted: 2025-10-26 17:40:39
```

Exercise 2

Use additional built-in modules: `random` and `os`. Generate a random number and save it to a file.

In [8]:
```
import random, os
random_num = random.randint(1, 100)
with open('random_number.txt', 'w', encoding='utf-8') as file:
    file.write(str(random_num))
print(f'Random number: {random_num}')
print(f"Random number saved to: {os.path.abspath('random_number.txt')}")
```

```
Random number: 93
Random number saved to: C:\Users\chaze\Downloads\AI Labs\random_number.txt
```

> **Discussion:** What are other useful built-ins? Examples: `sys` (system info), `json` (data parsing), `pathlib` (paths).

Section 3: Using Third-Party Packages and Installation

Third-party packages extend Python's capabilities. They are installed with **pip** and hosted on **PyPI (Python Package Index)**.

Step 1: Install a Package (run in your terminal)

```
pip install requests
```

If you get permission errors, try `pip install --user requests` or use a virtual environment:

```
python -m venv myenv
myenv\Scripts\activate  # Windows
source myenv/bin/activate # macOS/Linux
```

Step 2: Use the Package — `requests`

The code below is safe to run here; it includes error handling in case outbound internet is blocked.

In [10]:
```
try:
    import requests
    resp = requests.get('https://api.github.com', timeout=10)
```

```
    if resp.status_code == 200:
        data = resp.json()
        # The top-level dict may not always contain 'rate'; print a couple of keys safely
        print('GitHub API Response (keys):', list(data.keys())[:5])
    else:
        print(f'HTTP Error: {resp.status_code}')
except Exception as e:
    print('Could not import/use requests. Install it with pip and try again.\n', e)
```

```
GitHub API Response (keys): ['current_user_url', 'current_user_authorizations_html_url', 'authorizations_url', 'code_search_ur
l', 'commit_search_url']
```

Exercise 3 — `numpy`

Install another package:

`pip install numpy`
Then run:

In [12]:
```
try:
    import numpy as np
    arr = np.array([1, 2, 3])
    print('Array:', arr)
    print('Mean:', np.mean(arr))
except Exception as e:
    print('Numpy not available. Install with pip and retry.\n', e)
```

```
Array: [1 2 3]
Mean: 2.0
```

> To uninstall a package: `pip uninstall numpy` (confirm with `y` only if you want to remove it).

Conclusion and Wrap-Up

Review: You used local modules for custom code, built-in modules for standard tasks, and third-party packages for advanced features.

Common Pitfalls:

- Make sure files are in the correct directory for imports.
- Use `pip` correctly (or a virtual environment) to avoid permission/version conflicts.
- Handle `ModuleNotFoundError` by checking your environment and paths.

Extension: Try packages like `pandas` (data analysis) or `matplotlib` (plots). Build a small project combining all three concepts.