# Chapter 11.1 – Python Modules

Teaching Notes and Student Activities

---

## 1 Learning Objectives

- Understand what a **module** is in Python and why modules are used.
- Learn how to **import** and use modules.
- Explain the **importing process** and Python's `sys.modules`.
- Distinguish between **scripts** and **modules**.
- Practice writing and importing custom modules.

## 2 Key Concepts and Vocabulary

**Core Terms**

- **Script**: A file ending in `.py` that executes directly to perform a task.
- **Module**: A `.py` file containing definitions (functions, variables, or classes) that can be imported into another program.
- **Import**: The act of bringing one module's functionality into another using the `import` keyword.
- **Namespace**: The space that keeps variable and function names distinct between different modules.
- **Dependency**: A module that another program relies on.
- **sys.modules**: A Python dictionary tracking all currently loaded modules.

## 3 Introduction to Modules

The interactive Python interpreter provides the most basic way to execute Python code. However, any defined variables, functions, or classes are lost when the interpreter closes.

To reuse code, programmers save it in a **script** file (e.g., `my_script.py`) that can be executed again later.

If the same function definitions are needed across multiple programs, Python offers a better approach: create a **module**. Modules allow us to group related code and import it wherever it's needed.

**Participation Activity – What Is a Module?**

**Step 1:** Create a file `math_utils.py`

```
def double(x):
    return 2 * x

def triple(x):
    return 3 * x
```

**Step 2:** Create a file `test_math.py`

```
import math_utils

print(math_utils.double(5))
print(math_utils.triple(7))
```

**Discussion:** What is the advantage of separating these functions into their own file?

# 4 How Import Works

When Python executes an `import` statement, several things occur:

1. Python checks whether the module is already loaded in `sys.modules`.
2. If not, a new **module object** is created.
3. Python executes the code within the module.
4. The module is added to `sys.modules`.
5. The module name becomes available in the importer's namespace.

**Demonstration – The Importing Process**

**File:** `tools.py`

```
print("tools.py is running!")

def greet(name):
    print(f"Hello, {name}!")
```

**File:** `main.py`

```
import tools

print("main.py continues")
tools.greet("Jeremy")
```

**Expected Output:**

```
tools.py is running!
main.py continues
Hello, Jeremy!
```

**Discussion:** Why does the message from `tools.py` appear before `main.py continues`? (Hint: the module's code runs at import time.)

# 5 Using Imported Modules

Once a module is imported, its functions and variables can be accessed using the dot operator.

**Example:**

```
import math_utils
print(math_utils.double(10))
```

You can also overwrite module attributes temporarily:

```
import math_utils
math_utils.double = lambda x: x * 10
print(math_utils.double(2))  # prints 20
```

However, note that such changes vanish when the program ends.

# 6 Guided Practice – Importing Multiple Modules

**Try This!**

**File:** `weather.py`

```
def forecast():
    return "Sunny and warm"
```

**File:** `mood.py`

```
def today():
    return "Feeling great!"
```

**File:** `main.py`

```
import weather
import mood

print(weather.forecast(), "and", mood.today())
```

**Your Task:**

1. Predict the output.
2. Modify one module to print a message when imported.
3. Observe the order in which imports occur.

# 7 Student Challenges

## Challenge 1 – Module Check

Create a module `stats_tools.py`:

```
def mean(nums):
    return sum(nums) / len(nums)

def median(nums):
    nums.sort()
    mid = len(nums) // 2
    return nums[mid]
```

Then use it in `analyze.py`:

```
import stats_tools

data = [4, 7, 2, 9, 6]
print(stats_tools.mean(data))
print(stats_tools.median(data))
```

## Challenge 2 – Reimport Experiment

```
import stats_tools
import stats_tools
```

Why does the print statement in `stats_tools.py` only appear once?

### Challenge 3 – Namespace Play

```
# cat.py
def speak():
    return "Meow!"

# dog.py
def speak():
    return "Woof!"

# zoo.py
import cat, dog
print(cat.speak(), dog.speak())
```

How do namespaces prevent conflicts between `cat.speak()` and `dog.speak()`?

**Reflection and Why It Matters**

- Modules keep your code **organized**, **reusable**, and easier to maintain.
- They allow collaboration across files and projects.
- Understanding module imports helps debug issues like "module not found" or circular imports.

## 8 Extension Prompts

- Explore what happens when you use `from module import function`.
- Try creating a subpackage structure with multiple folders.
- Investigate the `importlib.reload()` function.