



MODULE 1: CORE PYTHON & DATA

WEEK: 2 LECTURE: 10

DATE: 29/08/2025

INSTRUCTOR: HAMZA SAJID

A NEW WAY OF BUILDING: INTRODUCTION TO MODULES

- Welcome back! You've learned to write your own functions. But what if the function you need already exists? And how do you organize your own code as it gets bigger? Python's power comes from its modules. Today, we'll learn how to use pre-built modules from the Standard Library and, just as importantly, how to turn your own code into a reusable module.

TODAY'S AGENDA

- The "Why" - What is a Module?
 - Analogy: The Toolbox
- Creating Your Own Modules
 - From Python File to Reusable Tool
 - Example: Making a greeter module
- Exploring the Standard Library
 - The math Module: For advanced calculations
 - The random Module: For generating random numbers and choices
- Putting It All Together & The Hands-On Lab
 - Hands-On Lab: The Password Generator
 - Q&A and Wrap-up

THE "WHY" - WHAT IS A MODULE?

- Imagine you're building a chair. You could create your own screws and your own screwdriver from scratch, but that's a lot of work. It's much easier to get a toolbox that already has all the tools and parts you need.
- A module in Python is like a toolbox. It's a file containing Python code—functions, variables, and classes. You can import this toolbox into your program to use its tools without having to build them yourself.

CREATING YOUR OWN MODULES

- The best part? Any Python file can be a module. If you write useful functions in a file, you can import that file into another script to reuse those functions. This is the key to organizing large projects and avoiding copy-pasting code.
- The Concept:
 - You write functions or classes in a Python file (e.g., `my_tools.py`).
 - In another file (e.g., `main_project.py`), you write `import my_tools`.
 - Now you can use any function from the first file by calling `my_tools.function_name()`.

EXAMPLE: MAKING A GREETER MODULE

- Let's create our own custom toolbox for greeting people.

STEP 1: CREATE THE MODULE FILE.

- Save this code in a file named greeter.py:
- `def say_hello(name):`
 - `"""A simple function to say hello.""" return f"Hello, {name}!"`
- `def say_goodbye(name):`
 - `"""A simple function to say goodbye.""" return f"Goodbye, {name}!"`

STEP 2: CREATE THE MAIN SCRIPT TO USE THE MODULE.

- Save this code in the same folder as app.py:
- `# We import our own module just like a built-in one!`
- `import greeter`
- `# Now we can use the functions from greeter.py`
- `print(greeter.say_hello("Alice"))`
- `print(greeter.say_goodbye("Bob"))`
- When you run app.py, it will import greeter.py and use its functions to print the messages. You've successfully created and used your own module!

THE MATH MODULE: YOUR POCKET CALCULATOR

- Now let's explore some of Python's pre-built toolboxes. The math module gives you access to advanced mathematical functions.
 - What it's for: Square roots, rounding, and constants like Pi.
 - How to use it: Start by writing `import math`.
- Key Functions:
 - `math.sqrt(x)`: Returns the square root of `x`.
 - `math.ceil(x)`: Rounds `x` up to the nearest whole number.
 - `math.pi`: A variable that holds the value of Pi (3.14159...).

EXAMPLE: USING THE MATH MODULE

- `import math`
- `# Calculate the square root of 81 result = math.sqrt(81)`
- `print(f"The square root of 81 is {result}")` # -> The square root of 81 is 9.0
- `# Use the value of Pi to find the circumference of a circle with a radius of 5`
- `radius = 5`
- `circumference = 2 * math.pi * radius`
- `print(f"The circumference is {circumference}")` # -> The circumference is 31.415...

IN-CLASS EXERCISE: AREA OF A CIRCLE

- Import the math module.
- Create a variable radius and set it to 10.
- Calculate the area of the circle using the formula: $\text{Area} = \pi * r^2$.
- Print the result in a user-friendly sentence.

THE RANDOM MODULE: EMBRACING UNPREDICTABILITY

- The random module allows you to generate random numbers, pick random items from a list, and even shuffle a list.
 - What it's for: Games, simulations, and creating unpredictable data like secure passwords.
 - How to use it: Start by writing `import random`.
- Key Functions:
 - `random.randint(a, b)`: Returns a random integer between a and b (inclusive).
 - `random.choice(sequence)`: Returns a single random element from a sequence (like a list or a string).
 - `random.shuffle(x)`: Shuffles the items in a list x in place.

EXAMPLE: USING THE RANDOM MODULE

- `import random`
- `# Simulate a six-sided dice roll dice_`
- `roll = random.randint(1, 6)`
- `print(f"You rolled a {dice_roll}")`
- `# Pick a random winner from a list of participants`
- `participants = ["Alice", "Bob", "Charlie", "Dana"]`
- `winner = random.choice(participants)`
- `print(f"The winner is {winner}!")`


IN-CLASS EXERCISE: COIN FLIP

- Import the random module.
- Create a list called outcomes that contains two strings: "Heads" and "Tails".
- Use random.choice() to select one of the outcomes from your list.
- Print the result.

PYTHON STANDARD LIBRARY — MUST-KNOW MODULES

- `os`
 - Interact with files, folders, and the operating system
- `pathlib`
 - Modern, object-oriented file path handling
- `datetime`
 - Work with dates, times, and timestamps
- `sys`
 - Access Python runtime and command-line arguments
- `json`
 - Convert between JSON and Python datacollections
 - Use advanced data types like Counter, deque, etc.
- `unittest`
 - Write and run automated tests
- `hashlib`
 - Create secure hashes (e.g., for passwords)
- These are built-in — no installation needed! Just import and go.

HANDS-ON LAB: SECURE PASSWORD GENERATOR (CLASS + MODULE)

- Goal
 - Build a password generator using OOP that:
 - Generates 8-character secure passwords
 - Must include: 
 - 1 uppercase letter
 - 1 lowercase letter
 - 1 digit
 - 1 special character
 - Uses a class structure placed in a separate module and imported in main script

STEP 1: CREATE A MODULE FILE → PASSWORD_GENERATOR.PY

- Define a class PasswordGenerator
- Create a constructor method `__init__()`
 - No arguments required
 - Inside the constructor:
 - Set `self.length = 8`
 - Store character groups as class attributes:
 - `self.uppercase_letters = A-Z`
 - `self.lowercase_letters = a-z`
 - `self.digits = 0-9`
 - `self.special_chars = !@#$%^&*`

STEP 2: DEFINE METHOD → GENERATE_PASSWORD(SELF)

- Select 1 character from each group:
 - 1 uppercase
 - 1 lowercase
 - 1 digit
 - 1 special character
- Fill the rest of the password with random characters from all groups combined
- Shuffle the characters to ensure randomness
- Join and return the password string

STEP 3: CREATE MAIN FILE → APP.PY

- Import the class
- Create an object
- Call the method and print the result

TESTING

- Run the program multiple times
- Check that each password:
 - Has exactly 8 characters
 - Includes at least one from each required category

BONUS CHALLENGE

- Allow optional length as argument in `__init__()`
- Add a method to generate a list of multiple passwords
- Add a setting to exclude special characters