ENGG 100
Measurement and Design Lab
Fall 2025, Section: Waves and Matter

Week # 2
Introduction to Python

Syed Babar Ali
School of Science and
Engineering

LUMS

PHYSLAB

Department of Physics

What is Python?

- Python is a high-level, interpreted programming language.
- It emphasizes code readability and simplicity with clean syntax.
- Python is widely used in web development, data science, AI, and automation.

- **Lecture Goals:**
  - Understand basic Python syntax and operations
  - Learn about variables, data types, and structures
  - Use conditional statements and loops
  - Create data visualizations and analysis
  - Write functions and modules

Basic arithmetic operations

- Python follows **PEMDAS** for operator precedence:
  - **P**arentheses: ()
  - **E**xponents: **
  - **M**ultiplication/Division: * /
  - **A**ddition/Subtraction: + -

```
2 + 2               # Addition
4 * 12.2            # Multiplication
3**2                # Exponentiation (** instead of ^ in MATLAB)
7 / 2               # Division (gives float result)
Sqrt(3)             # square root of 3, you will need numpy library
Sqrt(-3)            # square root of -3, you will need cmath library
1 / (1 + 3j)^(-4)   # complex number operations
```

Setting up environment

- Python's power comes from its extensive library ecosystem. Unlike MATLAB's built-in functions, Python requires importing specific libraries for different tasks.
- Essential Libraries
  - import numpy as np # For numerical operations (Numerical Python)
  - import matplotlib.pyplot as plt # For plotting and visualization
  - import math # For mathematical functions
- Key Scientific Computing Libraries
  - import pandas as pd # For data manipulation and analysis
  - import scipy # For advanced scientific computing (Scientific Python)

Variables

- Variables are like labeled containers that store values for later use.
- Good variable naming practices
  - student_name = "Alice" # Use descriptive names
  - age = 20
  - grade_average = 85.7
  - final_score = grade_average * 1.1 # Calculate using stored variables
  - is_passing = final_score >= 60 # Boolean variable
- Avoid these naming mistakes:
  - 2nd_name = "Bob“ # Can't start with number
  - student-age = 25 # Can't use hyphens (use underscores instead)
  - class = "Math101“ # Can't use Python keywords
  - def = 42 # 'def' is reserved for functions
  - import = "numpy“ # 'import' is a Python keyword

- Strings must be in quotes: "text" or 'text’

In Python, the primary way to check the data type of an object or variable is by using the built-in type() function.

Hands-on Exercise 1

(a) The distance from Jupiter to the Sun is approximately $742{,}180{,}000$ kilometers.

- Create a variable called j_dist_km and assign this value to it.
- Print the variable value by itself.
- Print a complete sentence with the distance and units.

(b) Earth orbits the Sun in an approximately circular path. Given the following data, calculate Earth's tangential orbital velocity.

- Earth's average orbital radius: 1.496×10^8 km
- Earth's orbital period: 3.156×10^7 s

Vectors

- A vector in Python can be represented as a list or NumPy array - a 1-dimensional collection of numbers.

Representation	**MATLAB**	**Python (NumPy)**	Shape in Python
Row Vector (1×n)	x = [1 2 3 4 5]	x = np.array([[1, 2, 3, 4, 5]])	(1, 5)
Column Vector (n×1)	x = [1; 2; 3; 4; 5]	x = np.array([[1], [2], [3], [4], [5]])	(5, 1)
1D Vector (just values)	x = [1 2 3 4 5] (still treated as row by default)	x = np.array([1, 2, 3, 4, 5])	(5,)

Matrices

- A *matrix* is a 2D array of numbers: 2-dimensional

- import numpy as np

- # Manual entry - equivalent to MATLAB M = [1 2; 3 4; 7 6]
- M = np.array([[1, 2],
-               [3, 4],
-               [7, 6]])

- # Alternative compact syntax
- M = np.array([[1, 2], [3, 4], [7, 6]])

- % Built-in functions

```
I =  np.eye(3)                 # 3×3 identity matrix
Z = np.zeros(3, 4);            # 3×4 matrix of zeros
O = np.ones(2, 5);             # 2×5 matrix of ones
R = np.random.rand(3, 3)       # 3×3 random numbers
```

Matrix Operations vs Element-wise Operations

Operation	MATLAB	Python/NumPy
Matrix multiplication	`A * B`	`A @ B` or `np.dot(A, B)`
Element-wise multiplication	`A .* B`	`A * B`
Matrix power	`A^2`	`np.linalg.matrix_power(A, 2)`
Element-wise power	`A.^2`	`A**2`
Element-wise division	`A ./ B`	`A / B`
Element-wise addition	`A + B`	`A + B`
Element-wise subtraction	`A - B`	`A - B`

Accessing elements using indexing

Operation	MATLAB	Python/NumPy
Indexing starts at	1	0
Single element	`A(1,1)`	`A[0,0]`
Last element	`A(end,end)`	`A[-1,-1]`
First row	`A(1,:)`	`A[0,:]` or `A[0]`
Second column	`A(:,2)`	`A[:,1]`
Submatrix	`A(1:2,2:3)`	`A[0:2,1:3]`
Last row	`A(end,:)`	`A[-1,:]` or `A[-1]`

- Python uses 0-based indexing (MATLAB uses 1-based)
- Python slicing is exclusive of end (A[0:2] gets indices 0 and 1, not 2)
- Use -1 for last index (equivalent to MATLAB's end)
- Square brackets [] instead of parentheses ()

Plotting

- Plot the function $y = e^{-t}\cos(2\pi t)$ for the range $0 \leq t \leq 5$.
- Plot $t^2$and t^3 on the same figure for the range $0 \leq t \leq 5$.
- Using Python's matplotlib library, create a figure with two subplots side by side:
  - In the first subplot, plot the function $y = \sin(x)$ for $0 \leq x \leq 2\pi$.
  - In the second subplot, plot the function $y = \cos(x)$ for the same range of x.

Logical (Boolean) Operations

- To see if a particular condition is True or False, we must explore the comparison operators.

Boolean	**Python Syntax**	**MATLAB Syntax**
Equal	x == y	x == y
Not equal	x != y	x ~= y
Greater than	x > y	x > y
Less than	x < y	x < y
Greater than or equal to	x >= y	x >= y
Less than or equal to	x <= y	x <= y

The conditional statements: if-else

Concept	**MATLAB**	**Python**
Start condition	if condition	if condition:
Multiple conditions	elseif condition	elif condition:
Final fallback	else	else:
Block ending	end keyword	**Indentation** (no end)

```
if condition:            # Python uses ':' and indentation
      # code if true
else:
     # code if none true
```

Note: No end in Python, instead, indentation marks code blocks. : colon after if / else is required.

Example: The conditional statements:

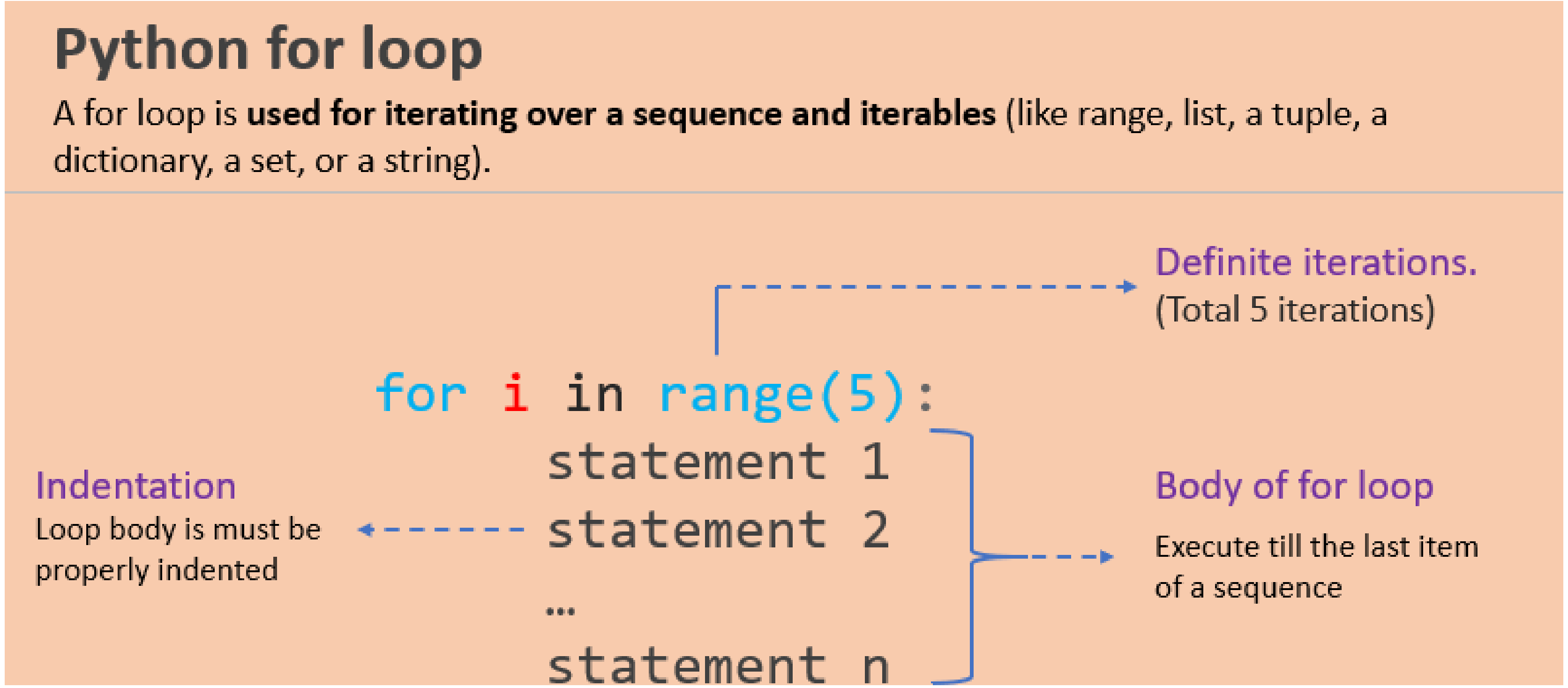
- **Example1:**
  - velocity = 350 # m/s, speed of an object
  - speed_of_sound = 343 # m/s

  - if velocity > speed_of_sound: # check if supersonic
    print("The object is supersonic")
    else:
    print("The object is subsonic")

- **Example 2:**
  - num = -3
  - if num > 0:
    print(num, 'is positive’)
    elif num == 0:
    print(num, 'is zero’)
    else:
    print(num, 'is negative')

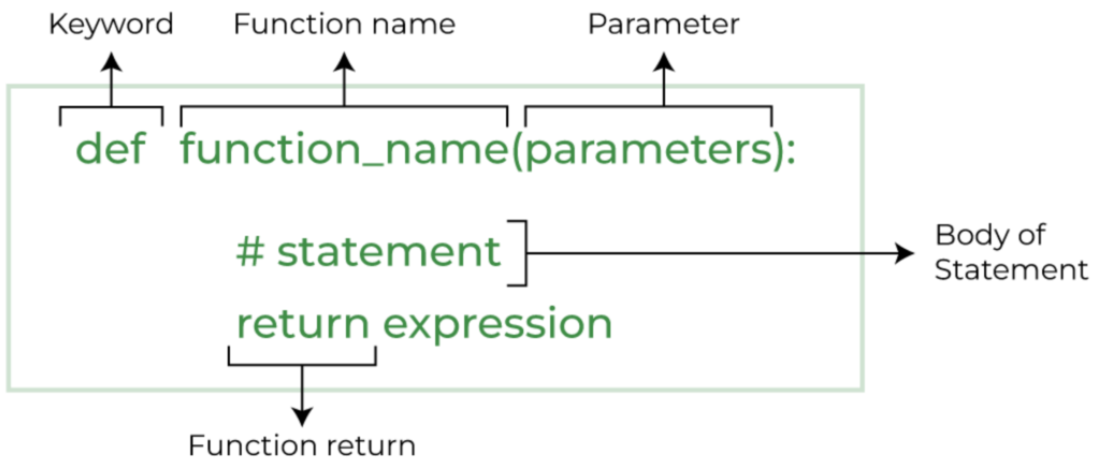
Loops: for

- Computers are great at repetitive tasks, and loops are a good way to do them! Combined with iterables, you can do a lot with loops.Syntax:



- **Example**
- odds = [1, 3, 5, 7, 9, 11]
- for num in odds:
    print(num)

How to create your own function




```
def fahr_to_celsius(temp):
# Assign the converted value to a variable
converted = ((temp - 32) * (5/9))
# Return the value of the new variable
return converted
```

Curve Fitting Using Figure Tools

Example

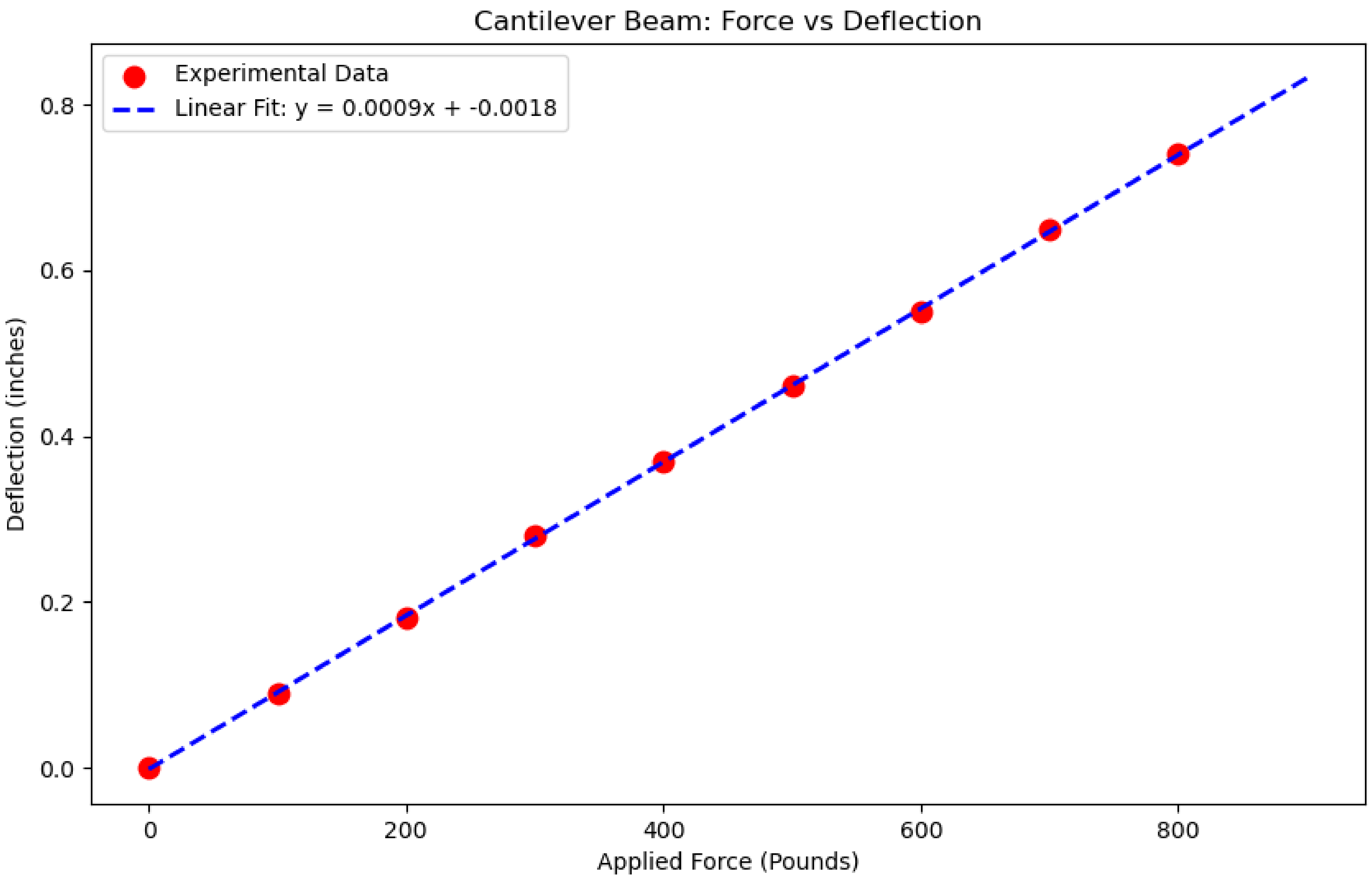
Q. A cantilever beam's deflection is measured under different applied forces. Find the linear relationship between force and deflection.

Force f (Pounds)	0	100	200	300	400	500	600	700	800
Deflection x (inches)	0	0.09	0.18	0.28	0.37	0.46	0.55	0.65	0.74

Table. An experiment to measure force and deflection in cantilever beam.

Solution

```python
import numpy as np
import matplotlib.pyplot as plt

# Experimental data from cantilever beam test
force = np.array([0, 100, 200, 300, 400, 500, 600, 700, 800])  # Pounds
deflection = np.array([0, 0.09, 0.18, 0.28, 0.37, 0.46, 0.55, 0.65, 0.74])  # inches

# Task 1: Plot the experimental data
plt.figure(figsize=(10, 6))
plt.scatter(force, deflection, color='red', s=80, label='Experimental Data')
plt.xlabel('Applied Force (Pounds)')
plt.ylabel('Deflection (inches)')
plt.title('Cantilever Beam: Force vs Deflection')

# Using np.polyfit for linear fit (degree = 1)
slope, intercept = np.polyfit(force, deflection, 1)

# Plot the fitted line with data
force_line = np.linspace(0, 900, 100)  # Smooth line for plotting
deflection_line = slope * force_line + intercept

# Plot the linear regression line with equation in legend
plt.plot(force_line, deflection_line, 'b--', linewidth=2,
        label=f'Linear Fit: y = {slope:.4f}x + {intercept:.4f}')  # Uses f-string formatting to insert actual values

plt.legend()
```


Cantilever Beam: Force vs Deflection
Experimental Data
Linear Fit: y = 0.0009x + -0.0018
Deflection (inches)
0.0
0.2
0.4
0.6
0.8
0
200
400
600
800
Applied Force (Pounds)