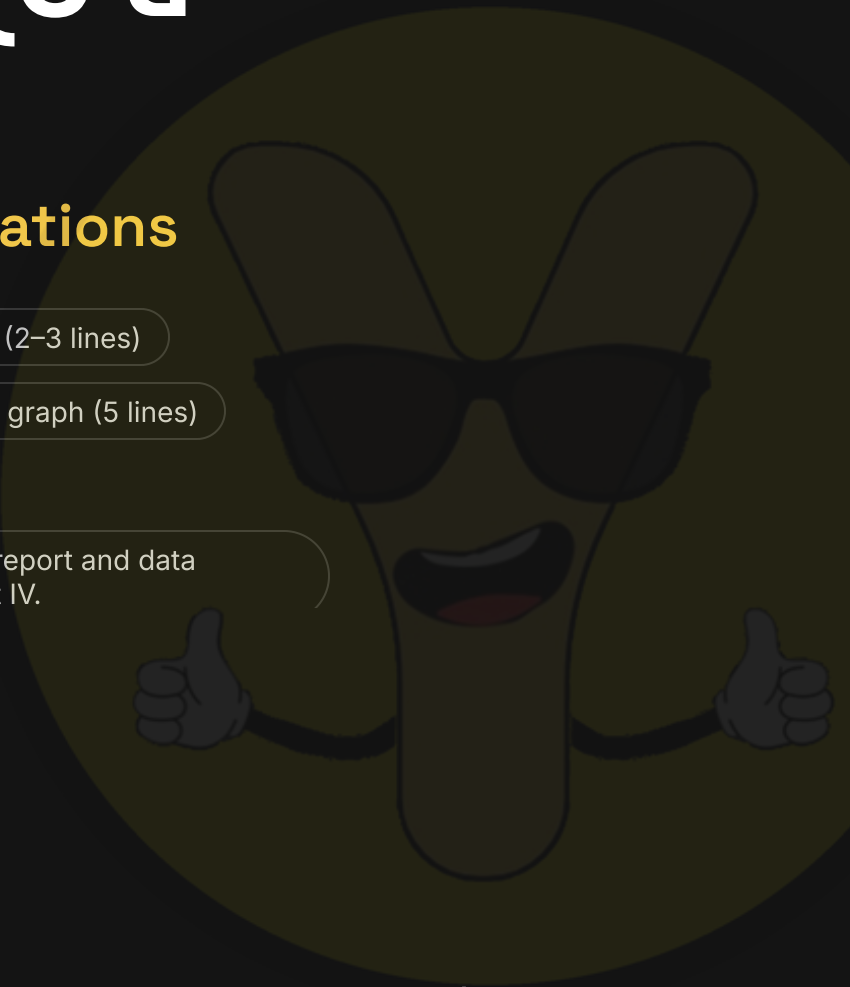
AKTU · 1ST YEAR · SEMESTER 1 · ACS101

# Programming Languages (C & Python)

## Unit 5 — Python Data Structures and Emerging Applications

- Python list, tuple, dictionary, set — basic operations only
- File handling — read and write a text file; read a CSV (2–3 lines)
- Introduction to NumPy — array, mean, simple arithmetic (5 lines)
- Introduction to Matplotlib — plot a simple line graph (5 lines)
- One mini-demo — IoT sensor simulation

Real connect: Python lists power NumPy which powers TensorFlow. Matplotlib plots are used in every engineering report and data science notebook. This unit opens the door to AI, IoT, and Data Science using the same Python basics learnt in Unit IV.




DECODED BY

**Chirag Jain**

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

# Unit 5 Overview: Python Data Structures & Applications

This unit bridges foundational programming with high-performance computing, data science, and modern engineering applications.

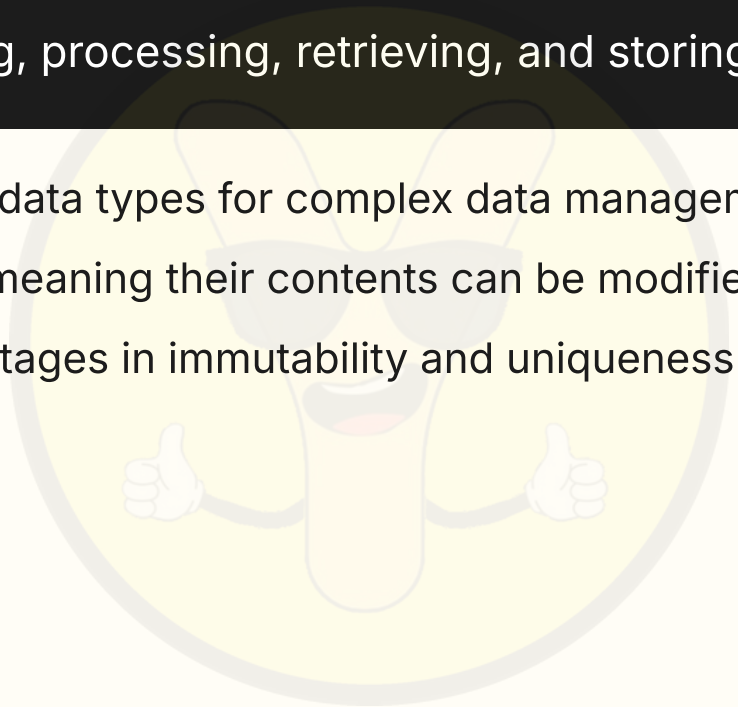
- Core Data Structures: Lists, Tuples, Dictionaries, and Sets with basic operations.
- File Handling: Reading and writing text files, and parsing CSV files.
- Scientific Computing: Introduction to NumPy arrays, mean, and basic arithmetic.
- Data Visualization: Plotting simple line graphs using Matplotlib.
- Emerging Applications: IoT sensor simulation and the pipeline from Python to AI.

# Python Data Structures

**DATA STRUCTURE**

A specialized format for organizing, processing, retrieving, and storing data efficiently in memory.

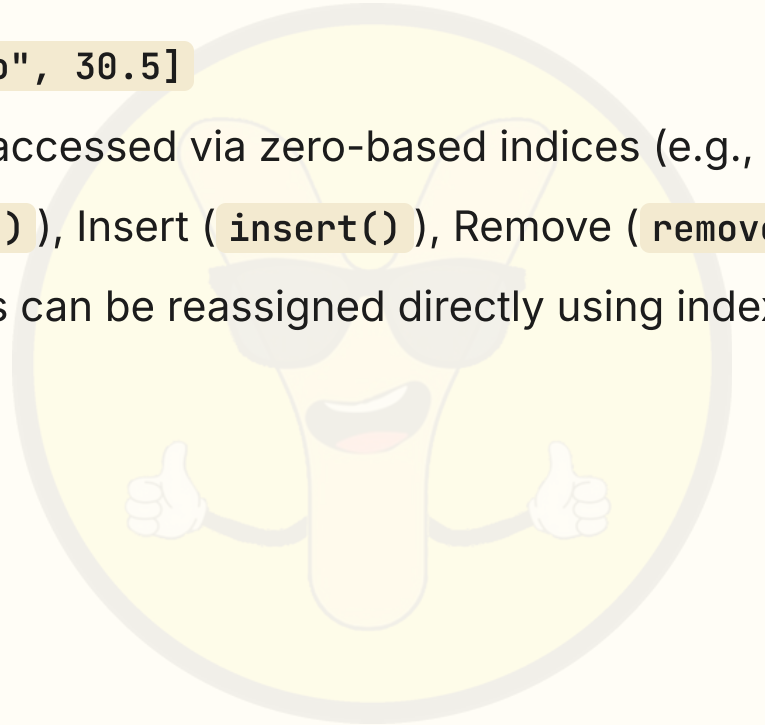
- Python provides built-in composite data types for complex data management.
- Lists and dictionaries are mutable, meaning their contents can be modified after creation.
- Tuples and sets offer distinct advantages in immutability and uniqueness enforcement.



# Python Lists

A list is an ordered, mutable collection of heterogeneous elements enclosed in square brackets.

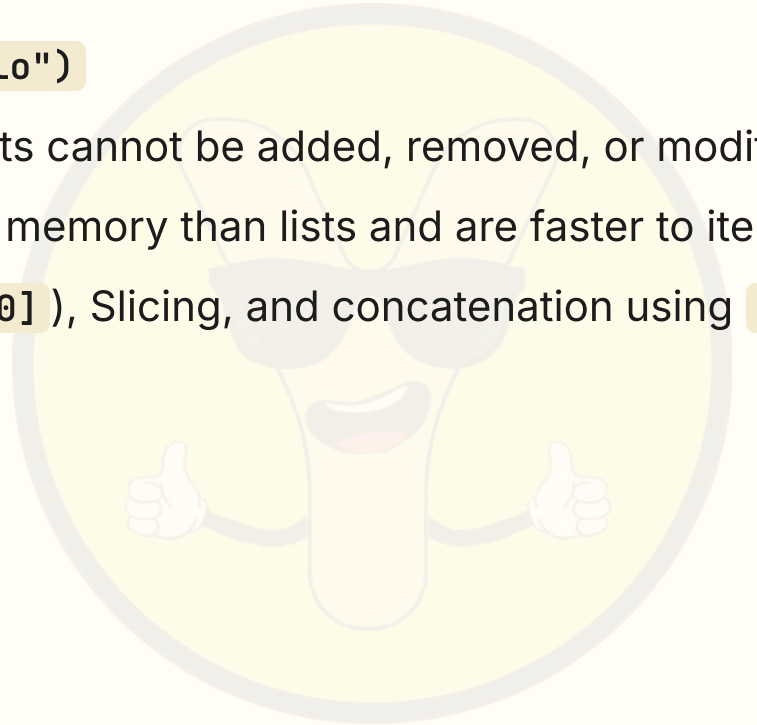
- Syntax: `my_list = [10, 20, "Hello", 30.5]`
- Indexing and Slicing: Elements are accessed via zero-based indices (e.g., `my_list[0]`).
- Basic Operations: Append (`append()`), Insert (`insert()`), Remove (`remove()`), and Sort (`sort()`).
- Mutable Nature: Individual elements can be reassigned directly using index assignment.



# Python Tuples

A tuple is an ordered, immutable collection of elements enclosed in parentheses.

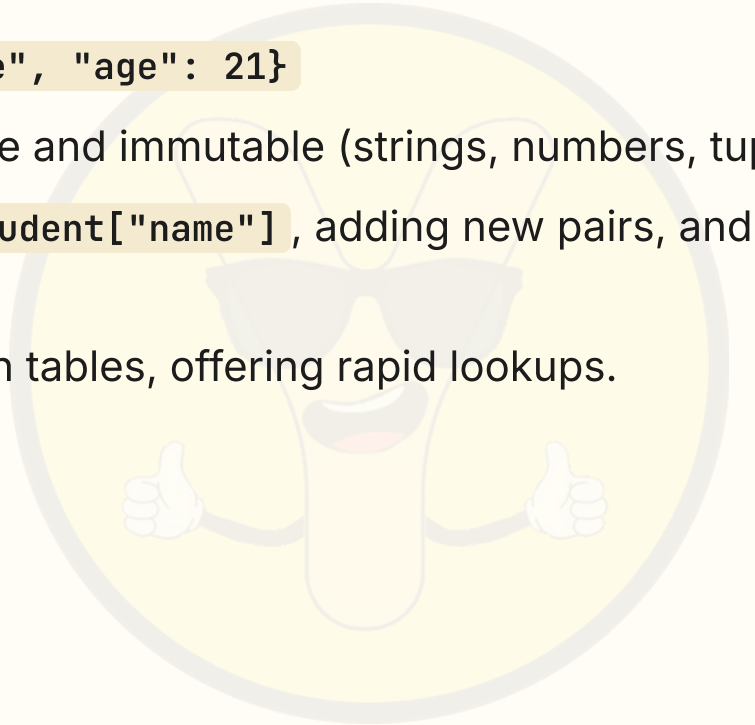
- Syntax: `my_tuple = (10, 20, "Hello")`
- Immutability: Once created, elements cannot be added, removed, or modified.
- Performance: Tuples consume less memory than lists and are faster to iterate over.
- Basic Operations: Indexing ( `tuple[0]` ), Slicing, and concatenation using `+` .



# Python Dictionaries

A dictionary stores data in key-value pairs, optimized for fast retrieval based on unique keys.

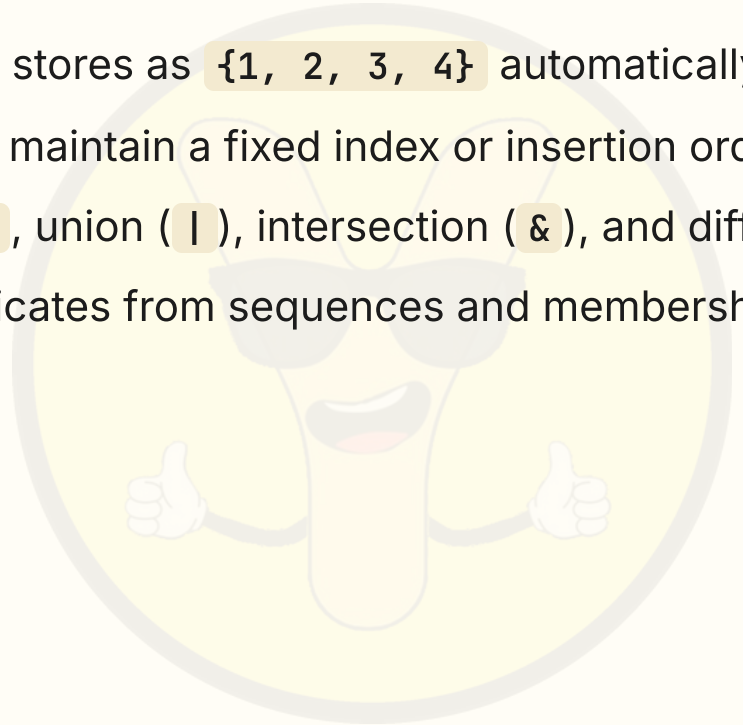
- Syntax: `student = {"name": "Alice", "age": 21}`
- Key Properties: Keys must be unique and immutable (strings, numbers, tuples). Values can be anything.
- Basic Operations: Accessing via `student["name"]`, adding new pairs, and using `keys()` and `values()` methods.
- Search Efficiency: Operates on hash tables, offering rapid lookups.



# Python Sets

A set is an unordered collection of unique elements with no duplicate values.

- Syntax: `my_set = {1, 2, 3, 4, 4}` stores as `{1, 2, 3, 4}` automatically.
- Unordered Nature: Elements do not maintain a fixed index or insertion order.
- Basic Operations: `add()`, `remove()`, union ( `|` ), intersection ( `&` ), and difference ( `-` ).
- Use Case: Efficient removal of duplicates from sequences and membership testing.



# Comparison of Python Data Structures

| Feature | List | Tuple | Dictionary | Set |
|---|---|---|---|---|
| **Ordered** | Yes | Yes | No (Key-based) | No |
| **Mutable** | Yes | No | Yes | Yes |
| **Duplicates** | Allowed | Allowed | Keys: No, Vals: Yes | Not Allowed |
| **Syntax** | [] | () | {} | {} |

# File Handling: Text Files

```
# Writing to a text file
file = open("data.txt", "w")
file.write("Hello AKTU Students\nLearning Python File
Handling")
file.close()

# Reading from a text file
file = open("data.txt", "r")
content = file.read()
print(content)
file.close()
```

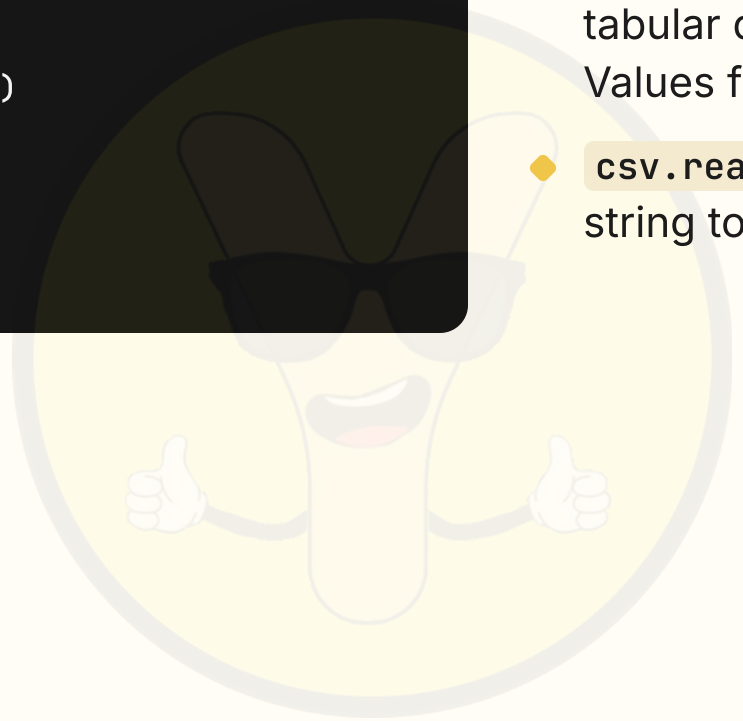


- Modes: `"w"` creates/overwrites a file, `"r"` reads a file, and `"a"` appends data.
- Always close files using `.close()` to free system resources, or use the `with` context manager.

# Reading a CSV File

```
import csv

# Reading a CSV file (2-3 lines format)
with open("grades.csv", "r") as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        print(row)
```



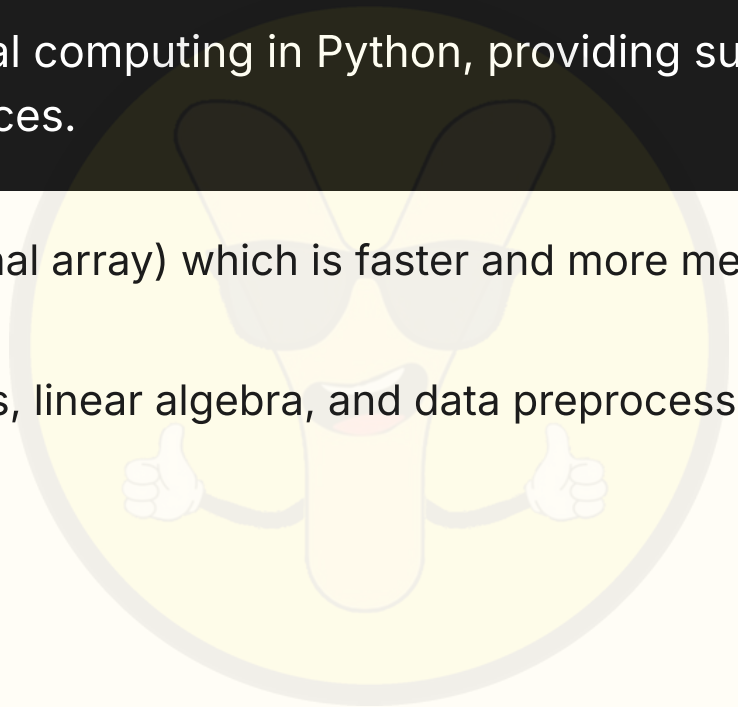
- The built-in `csv` module simplifies parsing tabular data stored in Comma-Separated Values format.
- `csv.reader()` parses each line into a list of string tokens for easy processing.

# Introduction to NumPy

**NUMPY (NUMERICAL PYTHON)**

A foundational library for numerical computing in Python, providing support for large multidimensional arrays and matrices.

- Core object: `ndarray` (N-dimensional array) which is faster and more memory-efficient than standard Python lists.
- Essential for scientific computations, linear algebra, and data preprocessing.

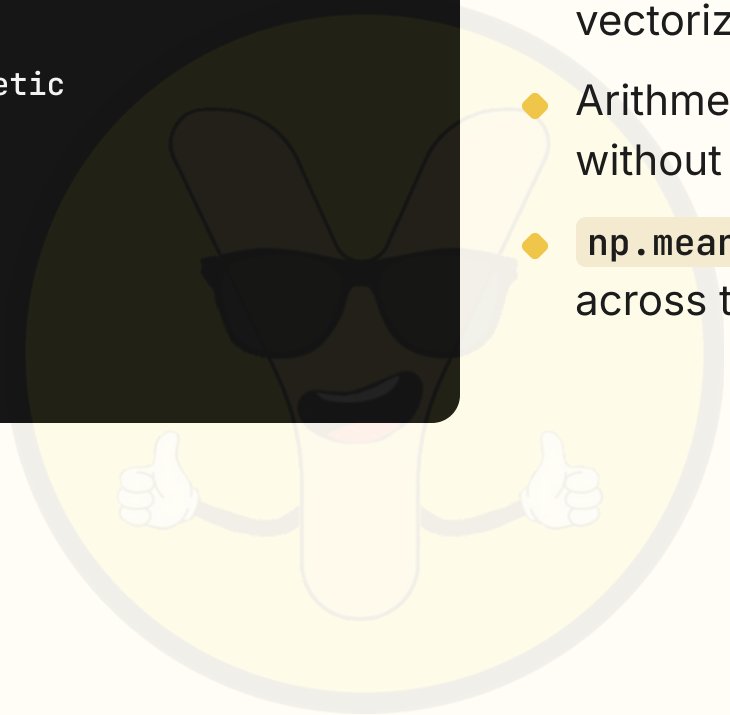


# NumPy Array, Mean, and Arithmetic

```
import numpy as np

# Create a 1D array and perform arithmetic
arr = np.array([10, 20, 30, 40, 50])
result = arr * 2
mean_val = np.mean(arr)

print("Scaled:", result)
print("Mean:", mean_val)
```



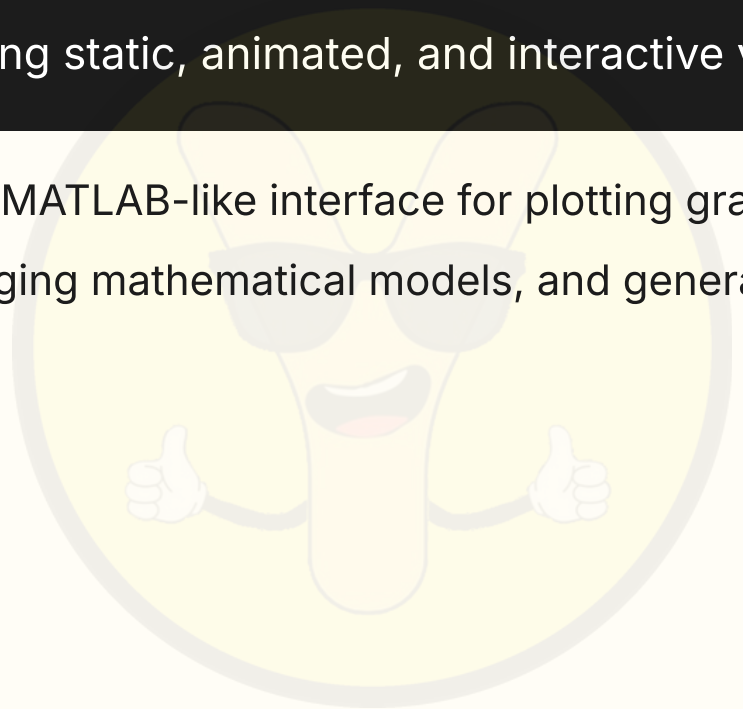
- `np.array()` converts sequences into vectorized numeric arrays.
- Arithmetic operations apply element-wise without requiring explicit loops.
- `np.mean()` calculates the arithmetic average across the array elements efficiently.

# Introduction to Matplotlib

**MATPLOTLIB**

A comprehensive library for creating static, animated, and interactive visualizations in Python.

- The `pyplot` submodule provides a MATLAB-like interface for plotting graphs easily.
- Crucial for analyzing trends, debugging mathematical models, and generating engineering reports.

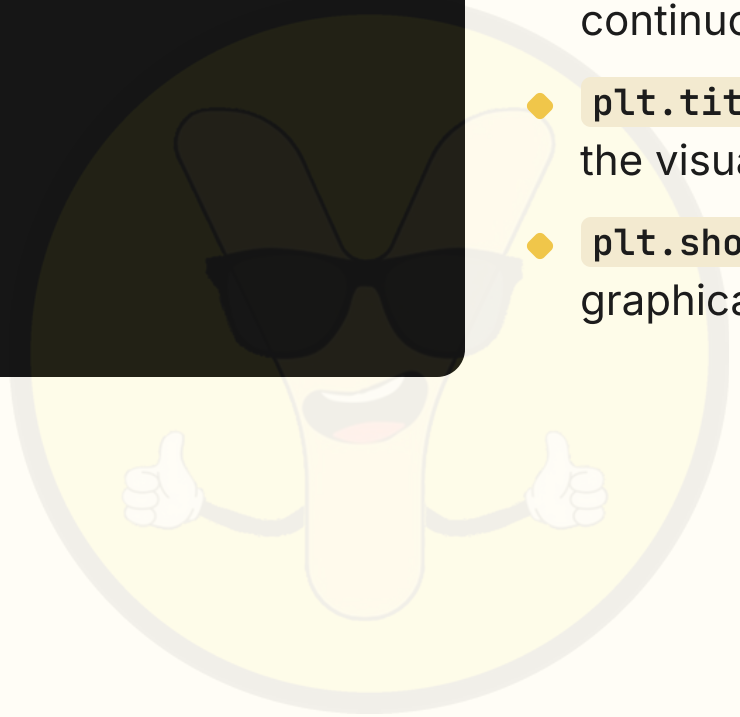


# Matplotlib: Simple Line Graph

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

plt.plot(x, y)
plt.title("Simple Line Graph")
plt.show()
```



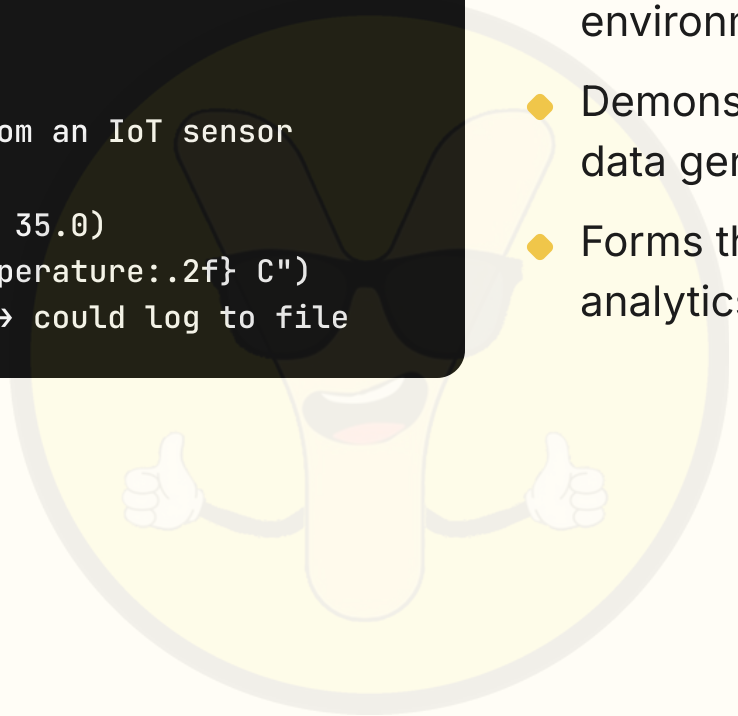
- `plt.plot(x, y)` connects data points with a continuous line.
- `plt.title()` adds a descriptive heading to the visualization window.
- `plt.show()` renders and displays the graphical output.

# Mini-Demo: IoT Sensor Simulation

```
import random
import time

# Simulating 3 temperature readings from an IoT sensor
for i in range(3):
    temperature = random.uniform(20.0, 35.0)
    print(f"Sensor Reading {i+1}: {temperature:.2f} C")
    # file.write(f"{temperature}\n") → could log to file
```

- Simulates edge devices collecting environmental parameters.
- Demonstrates practical integration of random data generation and logging loops.
- Forms the data pipeline foundation for IoT analytics.



# Real Connect: From Python Basics to AI

Python Lists & Collections (Base Data Structures)

NumPy Arrays (Vectorized High-Performance Compute)

TensorFlow & PyTorch (Deep Learning & AI Engines)

Matplotlib (Engineering Reports & Data Science Notebooks)

How foundational Python data structures scale up to power modern AI and scientific data pipelines.

# Exam corner

- **Q1** Differentiate between Python lists and tuples with suitable syntax and memory allocation examples.
- **Q2** Explain mutable and immutable data structures in Python, giving examples of each.
- **Q3** Write a Python program to read a text file, count the total number of lines, and write results to a new file.
- **Q4** Explain the significance of NumPy arrays over standard Python lists in scientific computing.
- **Q5** Write a Python script using NumPy to create an array of 5 numbers and compute their mean and standard deviation.
- **Q6** How is Matplotlib used in engineering applications? Write code to plot a simple line graph.
- **Q7** Discuss the role of dictionaries and sets in data preprocessing for machine learning algorithms.

# Unit 5 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 1 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

