CHAPTER 2: NUMERICAL COMPUTATION AND NUMPY

(Week 2: Matrices, Statistics, and Data Manipulation)

NumPy is the engine of data science. Giant libraries like Pandas, Scikit-Learn, and TensorFlow all use NumPy in the background. This week, we will learn to convert data into "numerical matrices".

2.1. WHY NUMPY? (Speed and Performance)

Python lists are flexible but slow with big data. NumPy, on the other hand, runs at the speed of C language.

- **Speed:** Lists are stored scattered in memory. NumPy arrays are in contiguous blocks.
- **Vectorization:** It processes the entire list mathematically at once without setting up a loop (for).

(Application: Speed Test) Run the code below and see the difference with your own eyes:

In [1]:
```
import numpy as np
import time

# Let's generate 1 Million numbers
py_list = list(range(1000000))
np_arr = np.arange(1000000)

# Squaring with Python List
start = time.time()
[x**2 for x in py_list]
print(f"Python Time: {time.time() - start:.5f} seconds")

# Squaring with NumPy
start = time.time()
np_arr ** 2
print(f"NumPy Time : {time.time() - start:.5f} seconds")

# Result: NumPy is usually 50-100 times faster.
```

```
Python Time: 0.03249 seconds
NumPy Time : 0.00000 seconds
```

2.2. ARRAY CREATION METHODS

In engineering, we usually do not enter data manually; we either read it from a file or simulate (generate) it.

In [2]:
```python
import numpy as np

# A) Converting from List
arr = np.array([1, 2, 3])

# B) Number Sequences (arange instead of range)
# From 0 to 10, skipping by 2
numbers = np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]

# C) Equal Partitioning (Linspace) - Vital for Graphs
# Divide the interval between 0 and 1 into 5 equal parts
parts = np.linspace(0, 1, 5) # [0.0, 0.25, 0.50, 0.75, 1.0]

# D) Generating Random Data (Simulation - IMPORTANT)
# This is how initial weights are assigned in machine learning.

# 5 random integers between 0 and 100 (e.g., Rolling dice)
random_int = np.random.randint(0, 100, 5)

# Random float numbers between 0 and 1 (e.g., Adding noise)
random_float = np.random.rand(3)

print(f"Random Integers: {random_int}")
print(f"Random Floats: {random_float}")
```

```
Random Integers: [87 32 59 34 48]
Random Floats: [0.68128238 0.33719549 0.06575217]
```

2.3. DIMENSIONS, TYPES, AND RESHAPE (NEW)

90% of the errors received in data science are due to "Shape Mismatch". We must know how to change the shape of the data (Reshape) well.

In [3]:
```python
# Matrix with 2 Rows, 3 Columns
matrix = np.array([[1, 2, 3],
                   [4, 5, 6]])

# Analysis
print("Number of Dimensions (ndim):", matrix.ndim)  # 2
print("Shape (shape):", matrix.shape)        # (2, 3) -> First ROW, then COLUMN
print("Data Type (dtype):", matrix.dtype)    # int64 or int32

# CRITICAL: Reshape
# Let's turn this 6-element matrix into (3 rows, 2 columns)
new_matrix = matrix.reshape(3, 2)
print("\nReshaped Matrix:\n", new_matrix)
```

```
Number of Dimensions (ndim): 2
Shape (shape): (2, 3)
Data Type (dtype): int32

Reshaped Matrix:
 [[1 2]
 [3 4]
 [5 6]]
```

2.4. INDEXING AND SLICING

Picking specific parts from inside the data "like pulling with tweezers". Rule: `[Row Range, Column Range]`

In [4]:
```
matrix = np.array([[10, 20, 30],
                   [40, 50, 60],
                   [70, 80, 90]])

# 1. Selecting a Single Element
print(matrix[1, 1])  # The element in the middle (50)

# 2. Selecting a Row
print(matrix[0, :])  # The entire 0th row -> [10, 20, 30]

# 3. Selecting a Column (Used a lot in Engineering)
# Question: Give me only the last column
print(matrix[:, 2])  # All rows, only the 2nd column -> [30, 60, 90]

# 4. Taking a Sub-matrix
print(matrix[0:2, 0:2]) # Top-left 2x2 square
```

```
50
[10 20 30]
[30 60 90]
[[10 20]
 [40 50]]
```

2.5. FILTERING (Conditional Selection)

This is the heart of data analysis. We ask questions like "Bring me only the data containing errors" here. NumPy does in a single line what is done with `if` and loops in classic Python.

In [5]:
```
data = np.array([5, 10, 15, 20, 25])

# Question: Which values are greater than 12?
filter_cond = data > 12  # Returns [False, False, True, True, True]
result = data[filter_cond]

print(result) # [15, 20, 25]

# Single-line usage (Common method):
# Let's find the even numbers
even_numbers = data[data % 2 == 0]
print(even_numbers)
```

```
[15 20 25]
[10 20]
```

2.6. STATISTICS AND AXIS LOGIC

When taking the average of a matrix, average of what? The whole class, or each student's own average?

In [6]:
```
grades = np.array([[60, 70],    # Student A (Midterm, Final)
                   [80, 90],    # Student B
```

```
                    [50, 50]])   # Student C

# 1. General Average (All numbers)
print("School Average:", grades.mean())       # Average of the whole school

# 2. Column-Based (Axis=0) -> "Crush from Top to Bottom"
# Midterm average and Final average separately
print("Exam Averages:", grades.mean(axis=0)) # [63.3, 70.0]

# 3. Row-Based (Axis=1) -> "Crush from Left to Right"
# Passing grade for each student
print("Student Averages:", grades.mean(axis=1)) # [65.0, 85.0, 50.0]
```

```
School Average: 66.66666666666667
Exam Averages: [63.33333333 70.        ]
Student Averages: [65. 85. 50.]
```

MINI PROJECT OF THE WEEK: SENSOR DATA ANALYSIS

Scenario: 10 hours of data is coming from a temperature sensor. However, the sensor sometimes malfunctions and measures abnormal values. Our goal is to clean these errors and find the true average.

1. Simulate the data (With normal distribution).
2. Mix incorrect data (Outliers) in between.
3. Clean with filtering and report.

In [7]:
```
# 1. Data Simulation (Normal distribution + Noise)
# 100 measurements with Mean 20 degrees, std deviation 5
temperatures = np.random.normal(20, 5, 100)

# Let's mix in 2 errors (Outliers)
temperatures[10] = 500  # Sensor broke
temperatures[50] = -100 # Sensor froze

# 2. Cleaning
# Logical range: Let it be between -10 and 50 degrees
clean_data = temperatures[(temperatures > -10) & (temperatures < 50)]

# 3. Result Report
print(f"Raw data count: {len(temperatures)}")
print(f"Cleaned data count: {len(temperatures) - len(clean_data)}")
print(f"Real Average: {clean_data.mean():.2f}")
```

```
Raw data count: 100
Cleaned data count: 2
Real Average: 20.46
```

3. WEEKLY HOMEWORK AND STUDY

Task 1: W3Schools Python Review (Mandatory) Anyone taking this course and feeling a lack of Python knowledge must complete the following topics at `w3schools.com/python` until the next lesson:

1. Python Syntax
2. Python Variables
3. Python Lists
4. Python If...Else
5. Python For Loops *(This study will not take more than 1 hour in total but will save the semester).*

Task 2: Colab Practice In your own Google Colab page;

1. Create a NumPy array consisting of numbers from 1 to 100.
2. Find the average of the numbers in this array.
3. Filter only the numbers greater than 90 and print them to the screen.