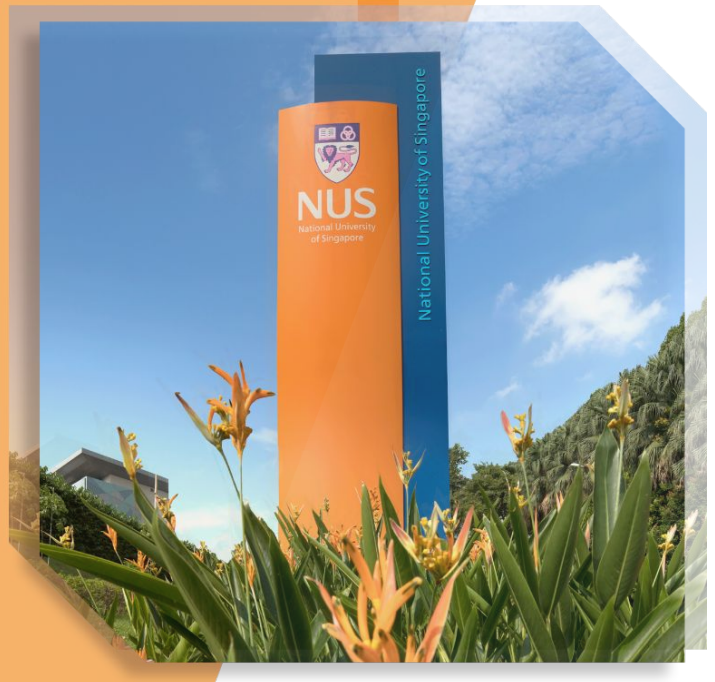
Full Stack Development With AI

Raghavendran V

Saturday, August 23, 2025
11:00 AM SGT



NUS National University of Singapore | Advanced Computing for Executives
School of Computing

Agenda

- Introduction to Python Libraries
- Numpy and its features
- Pandas and its features
- Matplotlib and its features

1) What's a "library" in Python?

- **Module**: a single `.py` file you can import.
- **Package**: a directory of modules (often with an `__init__.py`), importable as a unit.
- **Library**: informal term for one or more modules/packages that provide useful features (e.g., NumPy, requests).

Python code can come from:

- The **Standard Library** (ships with Python—`json`, `pathlib`, `datetime`, `asyncio`, …).
- **Third-party libraries** installed from **PyPI** (the Python Package Index) with `pip`.

2) Installing & managing libraries

Virtual environments (best practice)

macOS/Linux

```
python3 -m venv .venv
source .venv/bin/activate
python -m pip install --upgrade pip
```

Windows (PowerShell)

```
py -m venv .venv
.venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
```

Install & freeze:

```
pip install requests pandas matplotlib
pip freeze > requirements.txt
```

Reproduce later:

```
pip install -r requirements.txt
```

3) Import patterns & namespaces

```
import math //
```
The **`math` module** is a built-in Python library for **mathematical functions**.

```
from pathlib import Path //
```
`pathlib` is a modern Python library for working with **file system paths** (introduced in Python 3.4).

`Path` is a class that makes it easier to **read, write, and navigate files/folders** than using old

```
import numpy as np              # aliasing
from collections import Counter
```

- Absolute imports are clearer: `from mypkg.tools import helper`.
- Avoid `from x import *` (pollutes namespace).

What is NumPy?

- **NumPy** stands for **Numerical Python**.
- It is the **foundation library for scientific computing in Python**.
- Provides:
    - Fast **n-dimensional array objects (`ndarray`)**
    - Mathematical operations (linear algebra, statistics, Fourier transform)
    - Broadcasting (apply operations across arrays without writing loops)
    - Memory efficiency (compared to Python lists)

Why NumPy?

1. **Performance** – NumPy is written in C, so operations are **much faster** than Python lists.
2. **Vectorization** – You can apply operations to the entire array without loops.
3. **Convenience** – Many ML & AI libraries (Pandas, TensorFlow, PyTorch) are built on top of NumPy.

Example 1: Creating Arrays

```
import numpy as np

# 1D Array
arr1 = np.array([1, 2, 3, 4])
print("1D Array:", arr1)

# 2D Array
arr2 = np.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:\n", arr2)

# Array of zeros
zeros = np.zeros((2,3))
print("Zeros:\n", zeros)

# Array of ones
ones = np.ones((2,2))
print("Ones:\n", ones)

# Range array
rng = np.arange(0, 10, 2)  # start, stop, step
print("Range:", rng)

# Linearly spaced array
lin = np.linspace(0, 1, 5)  # start, stop, no. of points
print("Linspace:", lin)
```

Example 2: Array Operations

```
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

print("Addition:", a + b)        # [5 7 9]
print("Multiplication:", a * b)  # [4 10 18]
print("Dot Product:", np.dot(a, b))  # 1*4 + 2*5 + 3*6 = 32

# Broadcasting Example
m = np.array([[1, 2, 3], [4, 5, 6]])
print("Matrix + 10:\n", m + 10)  # adds 10 to every element
```

Example 3: Useful Functions

```
import numpy as np

arr = np.array([10, 20, 30, 40, 50])

print("Mean:", np.mean(arr))
print("Standard Deviation:", np.std(arr))
print("Sum:", np.sum(arr))
print("Max:", np.max(arr))
print("Min:", np.min(arr))
```

Example 4: Random Numbers

```
import numpy as np

print("Random Float [0-1):", np.random.rand())
print("Random 2x2 Matrix:\n", np.random.rand(2,2))
print("Random Integers:", np.random.randint(1, 10, 5))  # 5 random ints between 1-9
```

Example 5: Reshaping & Indexing

```
import numpy as np

arr = np.arange(1, 13)  # [1 2 3 ... 12]
print("Original:", arr)

reshaped = arr.reshape(3,4)
print("Reshaped 3x4:\n", reshaped)

print("Element at (2,3):", reshaped[1,2])   # row=1 col=2
print("First Row:", reshaped[0])
print("First Column:", reshaped[:,0])
```

Summary

- **Pandas** is the **go-to tool for data analysis in Python**.
- Key features:
  - **Series & DataFrame** for handling structured data
  - **Read/write data** from multiple sources
  - **Filtering, grouping, cleaning** operations

🐼 Pandas: Introduction

- **Pandas = Python Data Analysis Library**.
- Built on **NumPy**.
- Used for working with **tabular / structured data** (like Excel sheets or SQL tables).

It provides **two main data structures**:

1. **Series** → 1D labeled array (like one column).
2. **DataFrame** → 2D labeled table (rows & columns).

📝 Examples

1. Import Pandas

```
import pandas as pd
```

2. Creating a Series

```
# A simple 1D Series
s = pd.Series([10, 20, 30, 40], index=["a", "b", "c", "d"])
print(s)

# Output:
# a    10
# b    20
# c    30
# d    40
```

3. Creating a DataFrame

```
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "City": ["NY", "LA", "Chicago"]
}

df = pd.DataFrame(data)
print(df)

# Output:
#       Name  Age     City
# 0    Alice   25       NY
# 1      Bob   30       LA
# 2  Charlie   35  Chicago
```

4. Reading & Writing Data

```
# Read data from CSV file
df = pd.read_csv("data.csv")

# Write to CSV file
df.to_csv("output.csv", index=False)
```

5. Selecting Data

```
print(df["Name"])        # Select column
print(df.iloc[0])        # First row
print(df.loc[1, "City"]) # Specific cell
print(df[df["Age"] > 28]) # Filter rows
```

6. Cleaning Data

```
df = pd.DataFrame({
    "A": [1, 2, None, 4],
    "B": [5, None, None, 8]
})

print(df.dropna())        # Remove rows with missing values
print(df.fillna(0))       # Replace NaN with 0
print(df.isna())          # Check missing values
```

7. Analyzing Data

```
print(df.describe())      # Summary statistics
print(df["A"].mean())     # Average of column A
```

8. Grouping Data

```
grouped = df.groupby("City")["Age"].mean()
print(grouped)
```

9. Quick Visualization

```
import matplotlib.pyplot as plt

df["Age"].plot(kind="bar")
plt.show()
```

📊 Matplotlib: Introduction

- **Matplotlib** is a **Python library for data visualization**.
- It helps to create **static, animated, and interactive plots**.
- Often used with **NumPy** and **Pandas** for plotting arrays or DataFrames.

📝 Examples

1. Import Matplotlib

```
import matplotlib.pyplot as plt
```

2. Line Plot

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

plt.plot(x, y)                  # Line plot
plt.title("Line Plot Example")  # Title
plt.xlabel("X-axis")            # X label
plt.ylabel("Y-axis")            # Y label
plt.show()
```

3. Bar Chart

```
fruits = ["Apple", "Banana", "Orange", "Mango"]
values = [10, 25, 15, 30]

plt.bar(fruits, values, color="orange")
plt.title("Fruit Sales")
plt.show()
```

4. Scatter Plot

```
x = [5, 7, 8, 7, 6, 9, 5, 6, 7, 8]
y = [99, 86, 87, 88, 100, 86, 103, 87, 94, 78]

plt.scatter(x, y, color="red")
plt.title("Scatter Plot Example")
plt.show()
```

5. Histogram

```
import numpy as np

data = np.random.randn(1000)  # 1000 random numbers
plt.hist(data, bins=20, color="green", alpha=0.7)
plt.title("Histogram Example")
plt.show()
```

6. Pie Chart

```
sizes = [40, 30, 20, 10]
labels = ["Apples", "Bananas", "Cherries", "Mangoes"]

plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Fruit Distribution")
plt.show()
```

LIVE SESSION EXPERIENCE SURVEY

Before we proceed to Q&A
Take 2 minutes to share your feedback with Us!