Welcome to Python Programming!

Teaching Assistant of Introduction to Numerical Analysis Class

Class Date: 2024-2025

Instructor: Dr. Mina Zarei

Class Link: Telegram group.

Exercise class time: **Monday at 9:30 P002 Class** (determined by student vote)

Session 1 (Install Python in VS Code)

Follow the instructions in the GitHub link.

link: www.github.com/AliSeif96/Introduction-to-Numerical-Analysis/).

---

session 2 (Basic training)

1. Variables and Data Types
2. Taking Output and Input from the User
3. Conditionals (if, elif, else)
4. Loops
5. Functions

---

session 3 (Basic training 2)

6. Lists
7. Arrays
8. Matrices
9. Using `NumPy`

---

session 4 (Root finding)

10.1. Bisection Method (روش تصنیف)

10.2. False Position Method (روش نابجایی)

10.3. Newton-Raphson Method (روش نیوتن رافسون)

10.4. Secant Method (روش وتری)

10.5. Fixed-Point Iteration (روش تکرار ساده)

10.6. Brent's Method

---

session 5 (Solving and displaying the Kuramoto model)

11. Matplotlib
12. Kuramoto model

Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

The source code for Matplotlib is located at this github repository: https://github.com/matplotlib/matplotlib

`pip install matplotlib`

In [2]:

```python
import matplotlib
```

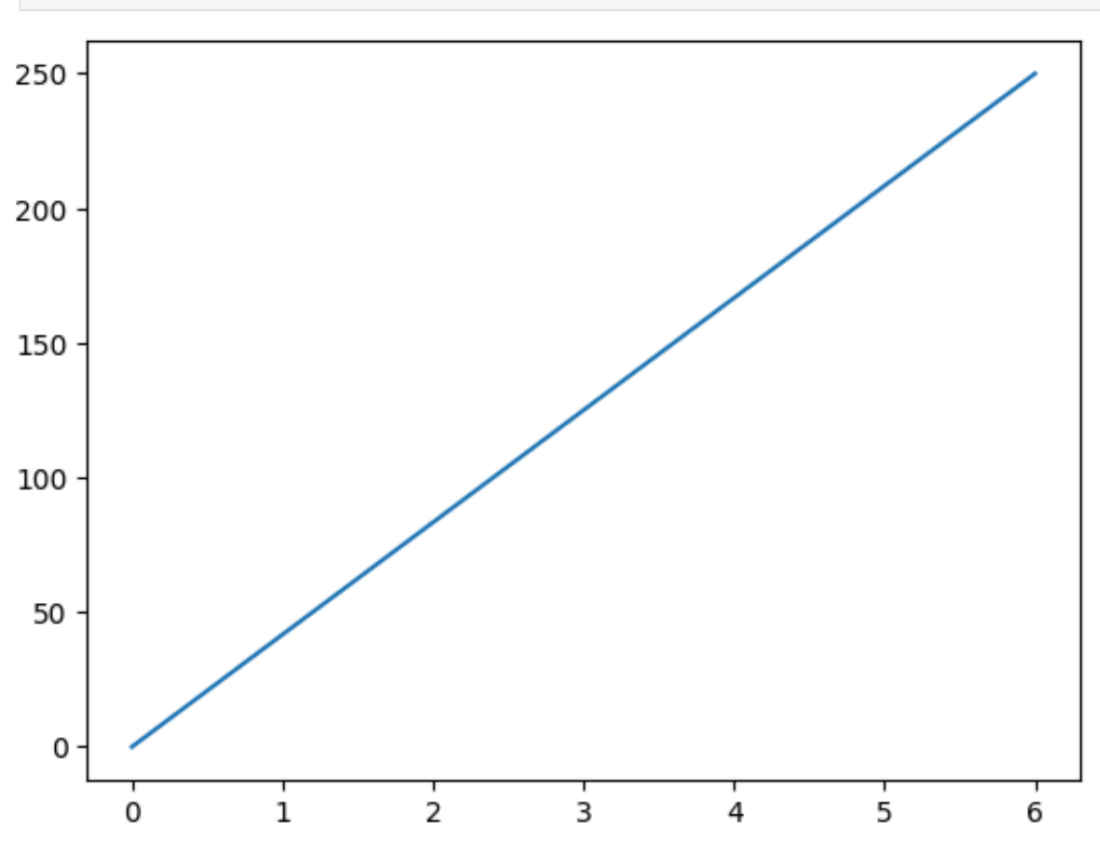
Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

In [3]:

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

In [4]:

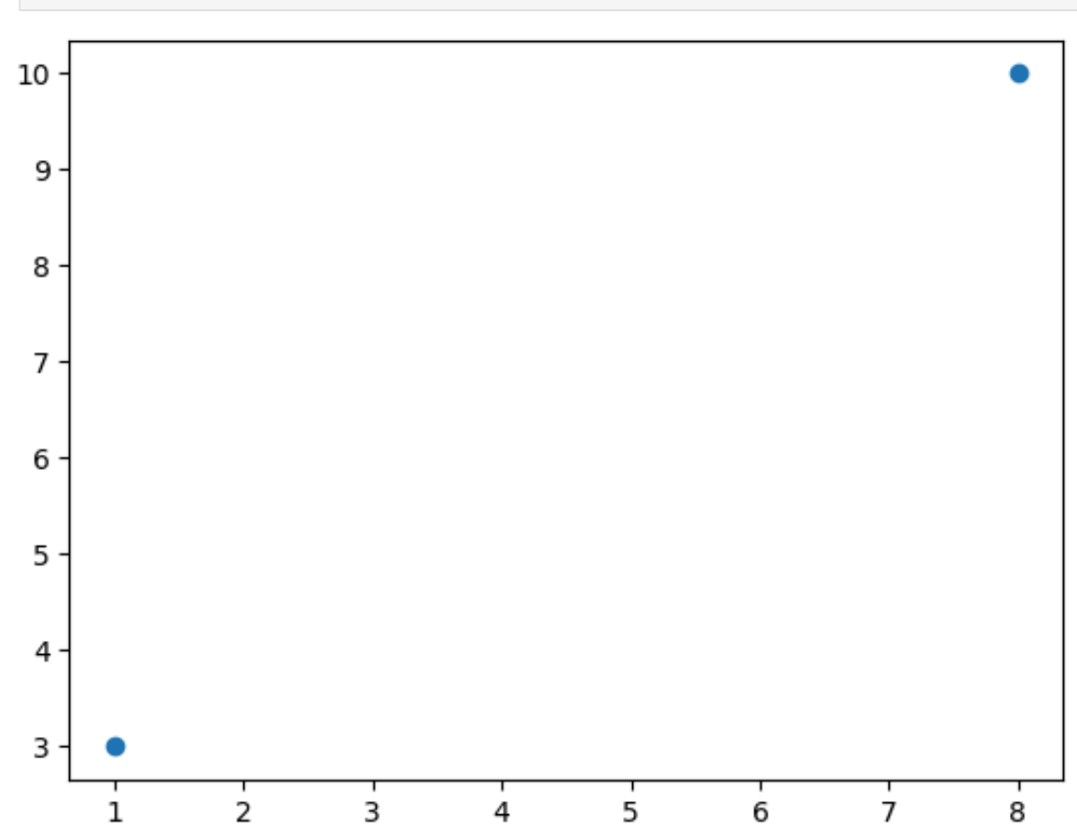
```python
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



Plotting Without Line

In [10]:

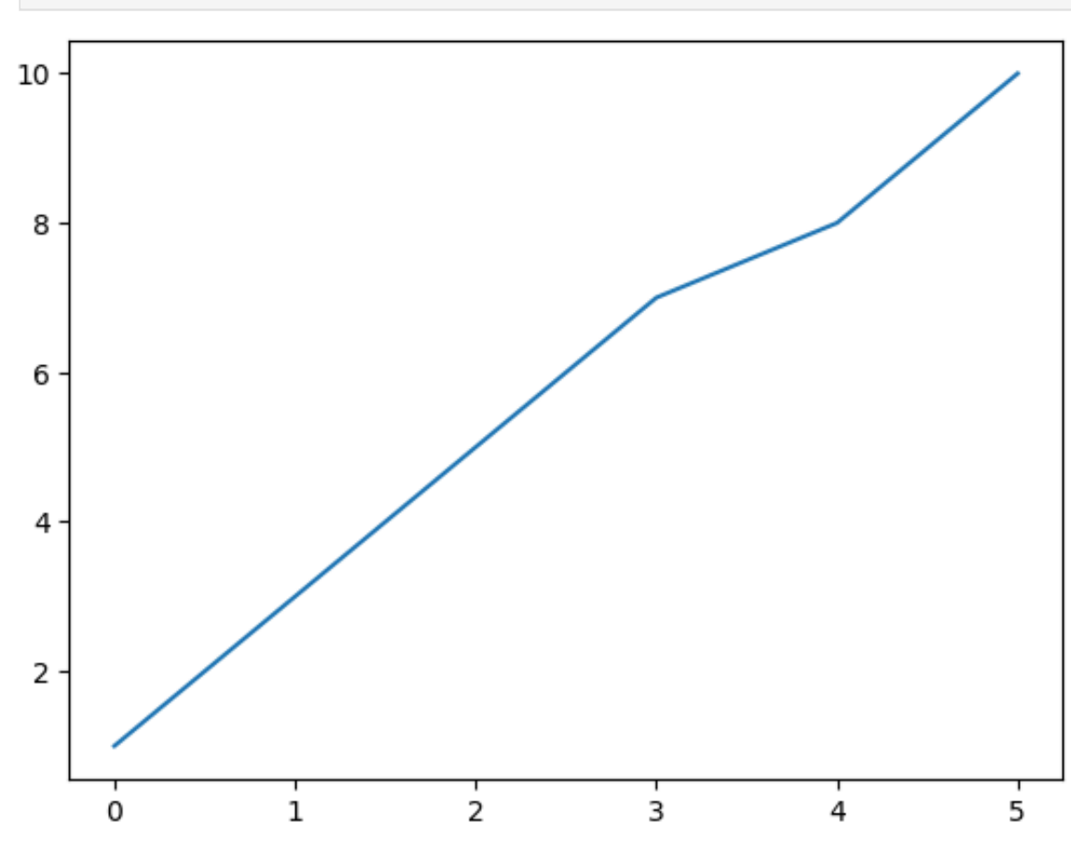
```python
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```



Plotting without x-points:

In [16]:

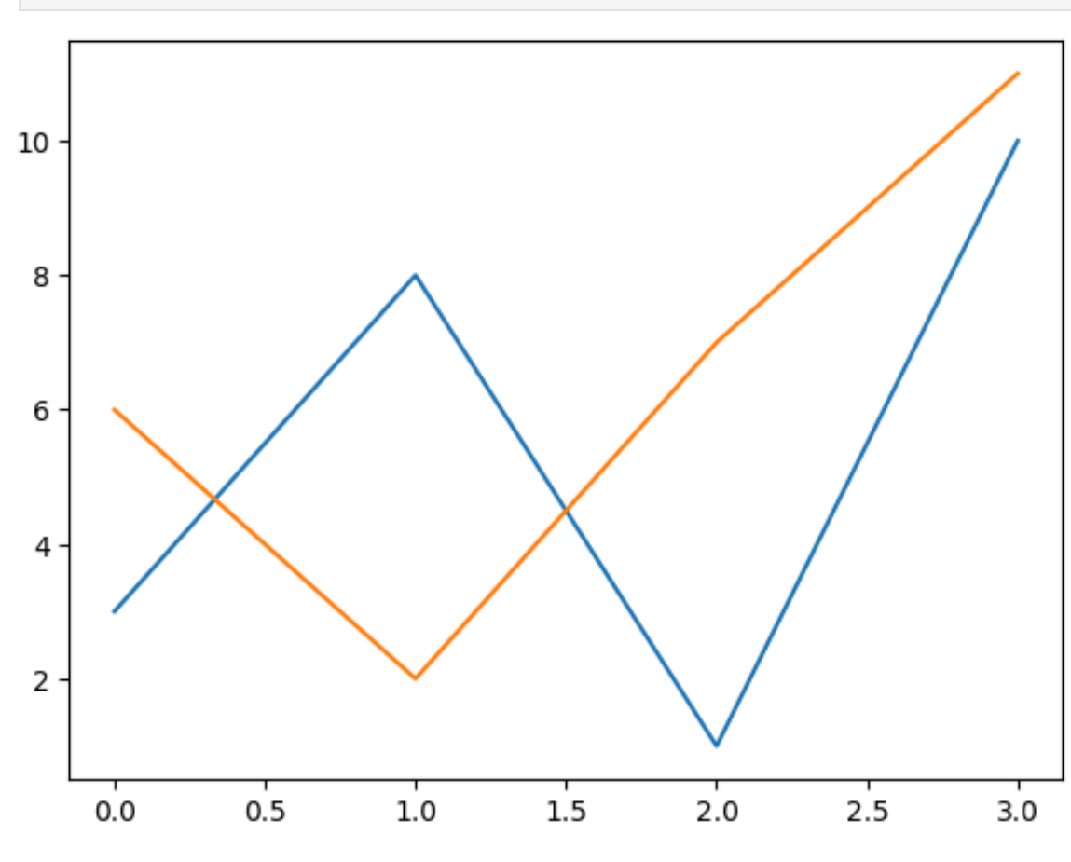
```python
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])
ypoints=np.sort(ypoints)
plt.plot(ypoints)
plt.show()
```



Multiple Lines

In [20]:

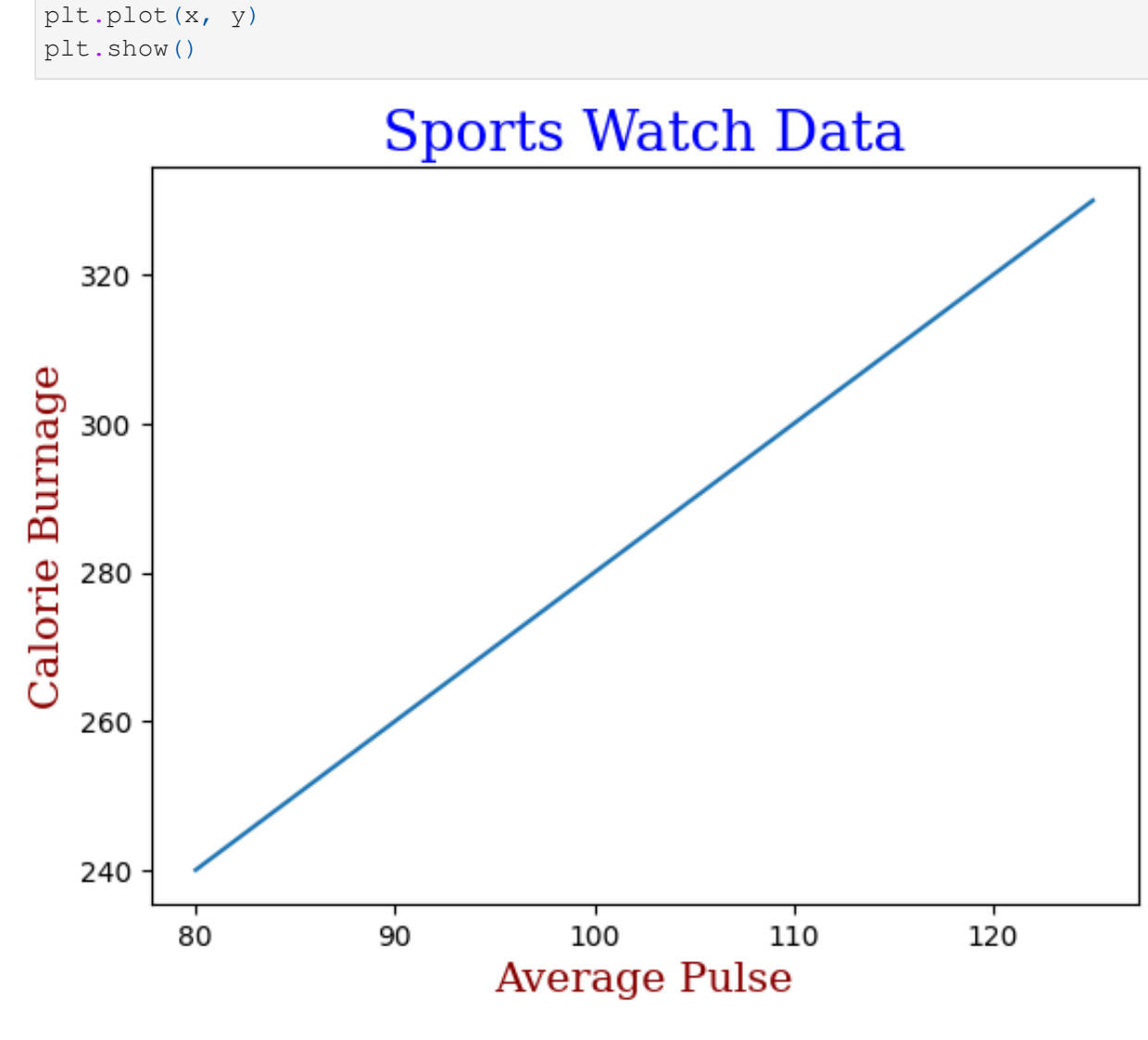
```python
import matplotlib.pyplot as plt
import numpy as np

y1 = np.array([3, 8, 1, 10])

y2 = np.array([6, 2, 7, 11])

plt.plot(y1)


plt.plot(y2)

plt.show()
```



Set Font Properties for Title and Labels

In [6]:

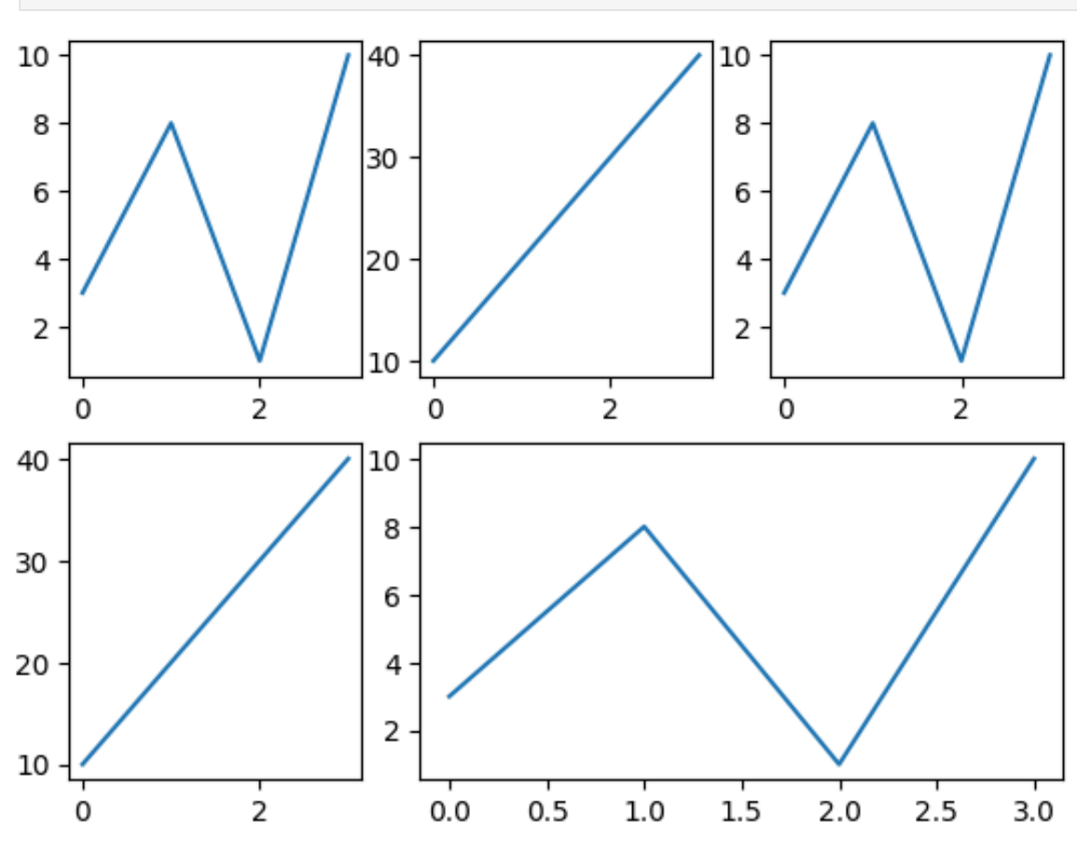
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x, y)
plt.show()
```



The subplot() Function

In [22]:

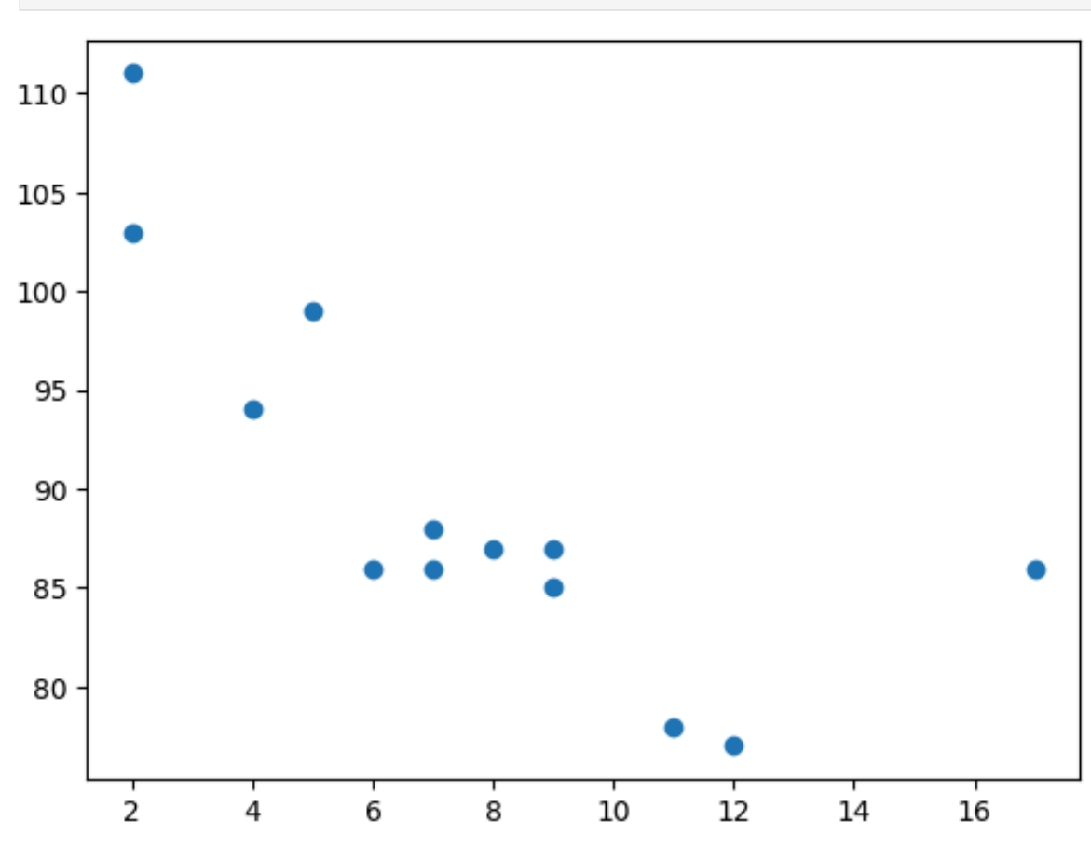
```python
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, (5,6))
plt.plot(x,y)



plt.show()
```



Creating Scatter Plots

In [8]:

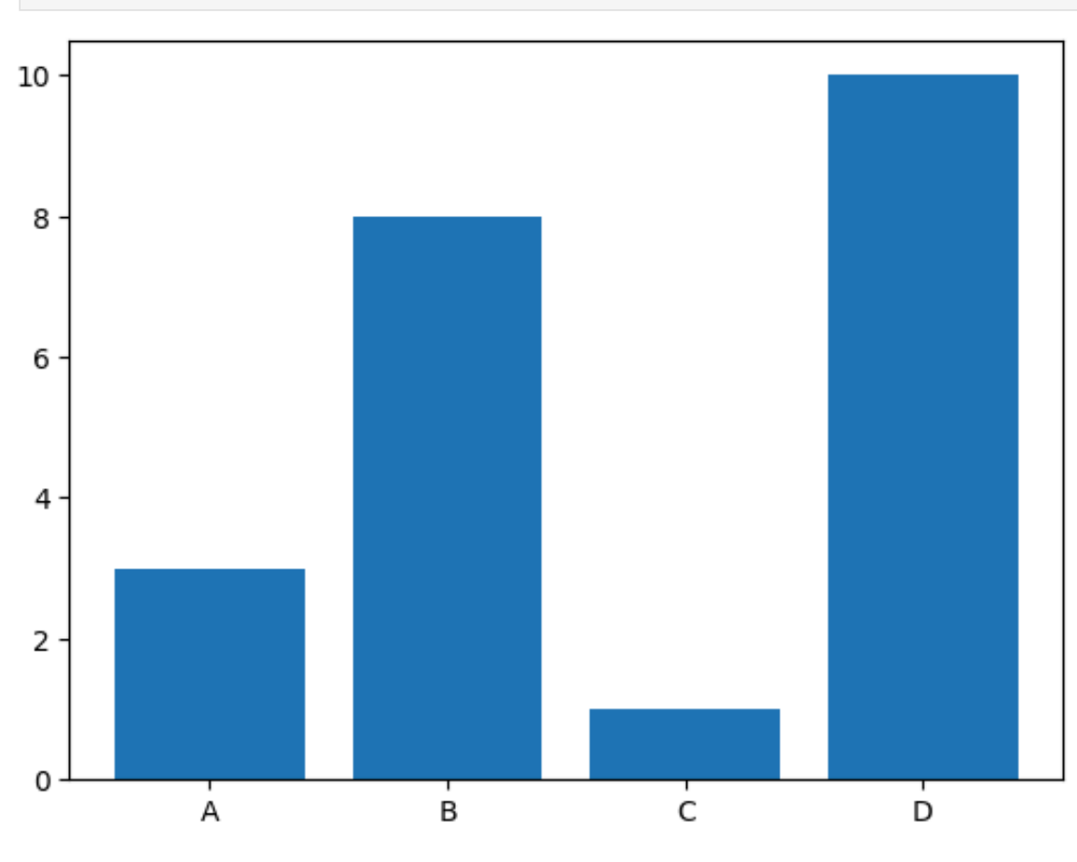
```python
import matplotlib.pyplot as plt
import numpy as np

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```



Creating Bars

In [9]:

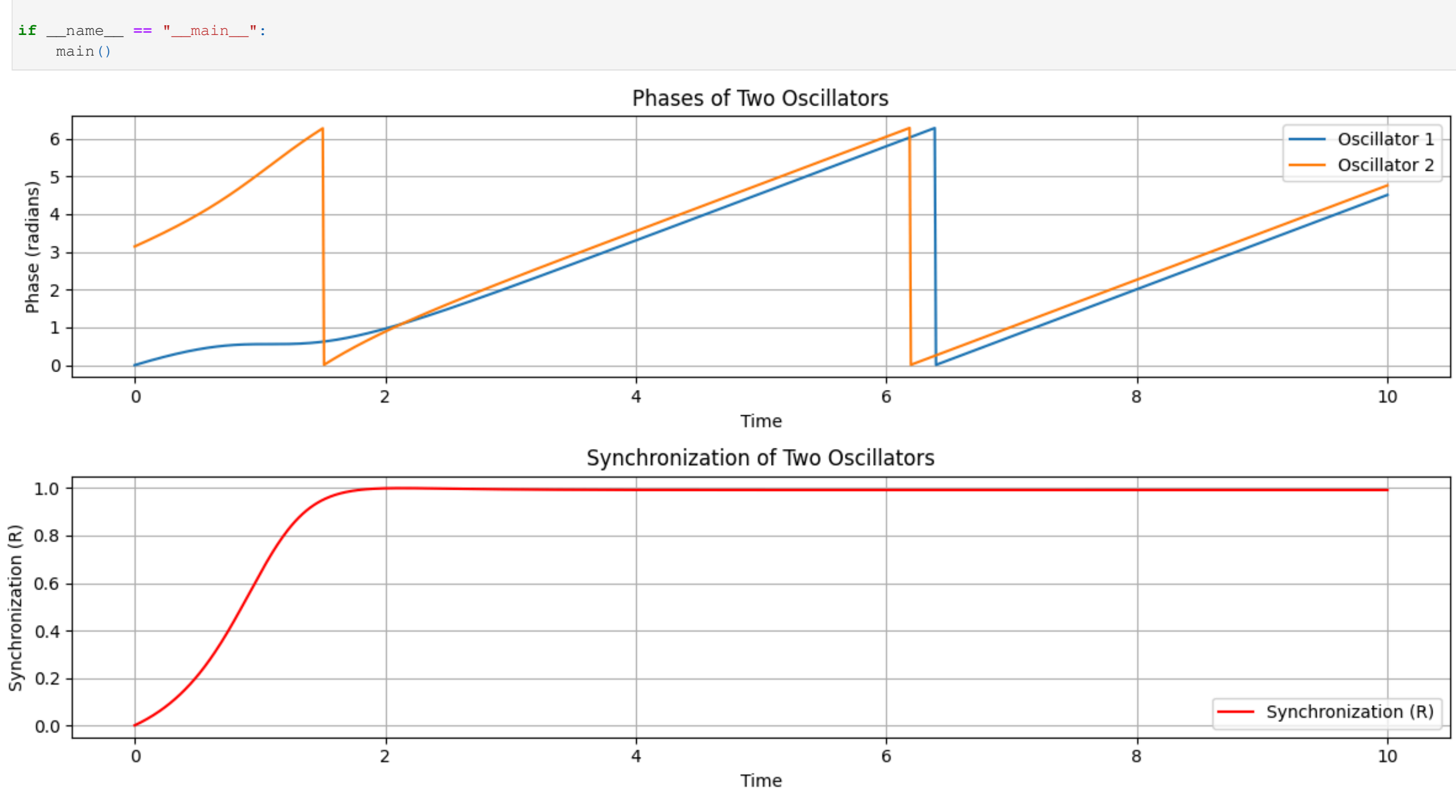
```python
import matplotlib.pyplot as plt
import numpy as np

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```



Kuramoto model

In [2]:

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

def kuramoto_euler(theta1, theta2, omega1, omega2, K, dt, T):
    """
    Simulates the Kuramoto model for two oscillators using the Euler method.

    Parameters:
        theta1, theta2: Initial phases of the oscillators (radians).
        omega1, omega2: Natural frequencies of the oscillators.
        K: Coupling strength.
        dt: Time step for the simulation.
        T: Total simulation time.

    Returns:
        time: Array of time points.
        theta1_vals, theta2_vals: Arrays of phase values for the oscillators.
        sync_vals: Array of synchronization values over time.
    """
    steps = int(T / dt)  # Number of time steps
    time = np.linspace(0, T, steps)  # Time array

    # Arrays to store the phases
    theta1_vals = np.zeros(steps)
    theta2_vals = np.zeros(steps)
    sync_vals = np.zeros(steps)  # Array to store synchronization values

    # Set initial conditions
    theta1_vals[0] = theta1
    theta2_vals[0] = theta2

    # Euler integration
    for i in range(1, steps):
        dtheta1_dt = omega1 + K * np.sin(theta2_vals[i-1] - theta1_vals[i-1])
        dtheta2_dt = omega2 + K * np.sin(theta1_vals[i-1] - theta2_vals[i-1])

        theta1_vals[i] = theta1_vals[i-1] + dtheta1_dt * dt
        theta2_vals[i] = theta2_vals[i-1] + dtheta2_dt * dt

        if theta1_vals[i]>=2*np.pi:
            theta1_vals[i]=theta1_vals[i]-2*np.pi
        if theta2_vals[i]>=2*np.pi:
            theta2_vals[i]=theta2_vals[i]-2*np.pi
        if theta1_vals[i]<=0:
            theta1_vals[i]=theta1_vals[i]+2*np.pi
        if theta2_vals[i]<=0:
            theta2_vals[i]=theta2_vals[i]+2*np.pi

        # Calculate synchronization (order parameter)
        cos_sum = np.cos(theta1_vals[i]) + np.cos(theta2_vals[i])
        sin_sum = np.sin(theta1_vals[i]) + np.sin(theta2_vals[i])
        sync_vals[i] = 0.5 * np.sqrt(cos_sum**2 + sin_sum**2)

    return time, theta1_vals, theta2_vals, sync_vals

def main():
    # Parameters
    theta1 = 0.0  # Initial phase of oscillator 1
    theta2 = np.pi  # Initial phase of oscillator 2
    omega1 = 1.0  # Natural frequency of oscillator 1
    omega2 = 1.5  # Natural frequency of oscillator 2
    K = 1.0  # Coupling strength
    dt = 0.01  # Time step
    T = 10  # Total simulation time

    # Simulate the Kuramoto model
    time, theta1_vals, theta2_vals, sync_vals = kuramoto_euler(theta1, theta2, omega1, omega2, K, dt, T)

    # Plot the phases
    plt.figure(figsize=(12, 6))
    plt.subplot(2, 1, 1)
    plt.plot(time, theta1_vals, label='Oscillator 1')
    plt.plot(time, theta2_vals, label='Oscillator 2')
    plt.xlabel('Time')
    plt.ylabel('Phase (radians)')
    plt.title('Phases of Two Oscillators')
    plt.legend()
    plt.grid()

    # Plot the synchronization
    plt.subplot(2, 1, 2)
    plt.plot(time, sync_vals, label='Synchronization (R)', color='red')
    plt.xlabel('Time')
    plt.ylabel('Synchronization (R)')
    plt.title('Synchronization of Two Oscillators')
    plt.legend()
    plt.grid()

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```



Complexity Explorables

Ride my Kuramotocycle!

The Kuramoto model Synchronization of Phase-Coupled Oscillators

This explorable illustrates the Kuramoto model for phase coupled oscillators. This model is used to describe synchronization phenomena in natural systems, e.g. the flash synchronization of fire flies or wall-mounted clocks.

https://www.complexity-explorables.org/explorables/ride-my-kuramotocycle/

Python Tutorial

Learn Python with `w3schools`

Learn to Code

With the world's largest web developer site.

https://www.w3schools.com/python/default.asp

In [ ]:

In [ ]:

In [ ]: