

Wroclaw University Of Science And Technology

Caner Olcay 276715

# **Selected Topics in Artificial Intelligence**

06.10.2025 (17:30 - Tuesday)

## **Exercise 1 – NumPy: Vectors and Matrix Operations**

**Description:**

In this task, two vectors containing 10 elements each were generated using NumPy. The first vector contained values from the range [50, 150], and the second vector from [1.5, 2.5]. Both vectors were concatenated into a 10×2 matrix. The goal was to find the maximum value in the second column and return the corresponding value from the first column.

**Essential Code:**

import numpy as np

vector1 = np.random.uniform(50, 150, 10)

vector2 = np.random.uniform(1.5, 2.5, 10)

matrix = np.column\_stack((vector1, vector2))

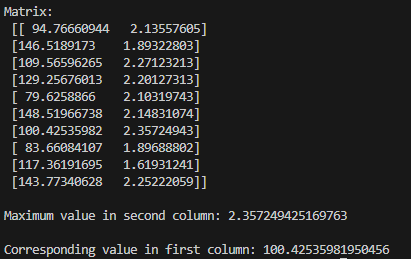
max\_index = np.argmax(matrix[:, 1])

corresponding\_value = matrix[max\_index, 0]

print("\nMatrix:\n", matrix)

print("\nMaximum value in second column:", matrix[max\_index, 1])

print("\nCorresponding value in first column:", corresponding\_value)

**Result :**

### 

### **Explanation:**

### The program correctly finds the row with the maximum value in the second column and retrieves the paired value from the first column. Random generation ensures that results vary with each run.

### **Exercise 2 – Pandas: DataFrame and Conditional Column**

**Description:**

A Pandas DataFrame was created with two columns (length\_1, length\_2), each containing 40 integer values between 800 and 1600. A new Boolean column, long, was added, which takes the value True if length\_1 > 1200 and False otherwise.

## 

## **Essential Code:**

import pandas as pd

import numpy as np

length\_1 = np.random.randint(800, 1600, 40)

length\_2 = np.random.randint(800, 1600, 40)

df = pd.DataFrame({

"length\_1": length\_1,

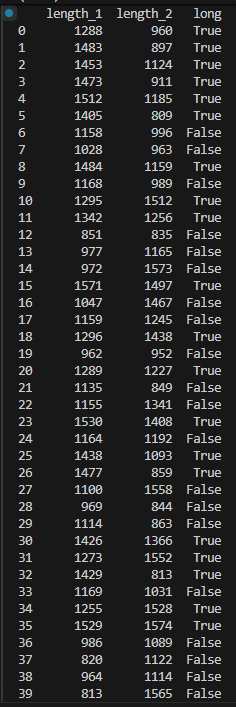
"length\_2": length\_2

})

df["long"] = df["length\_1"] > 1200

print(df)

**Result :**

****

## 

**Explanation:**

The DataFrame structure allows easy filtering and logical evaluation. The generated data correctly produces a mix True/False values depending on the threshold.

### 

### **Exercise 3 – SciPy & Matplotlib: Polynomial Interpolation**

**Description:**

A quadratic function f(x)=x^2+x−4 . Values were generated for x∈[−5,5] with a step of 0.25. SciPy was used to create linear and cubic interpolations, and both were plotted together with the original function using Matplotlib.

## **Essential Code:**

import numpy as np

import matplotlib.pyplot as plt

from scipy.interpolate import interp1d

def f(x):

return x\*\*2 + x - 4

step = 0.25

x = np.arange(-5, 5 + 1e-12, step)

y = f(x)

lin = interp1d(x, y, kind="linear")

cub = interp1d(x, y, kind="cubic")

x\_s = np.linspace(x.min(), x.max(), 400)

y\_lin = lin(x\_s)

y\_cub = cub(x\_s)

plt.figure(figsize=(8,5))

plt.plot(x, y, "o", label="initial points")

plt.plot(x\_s, y\_lin, "-", label="linear interp.")

plt.plot(x\_s, y\_cub, "--", label="cubic interp.")

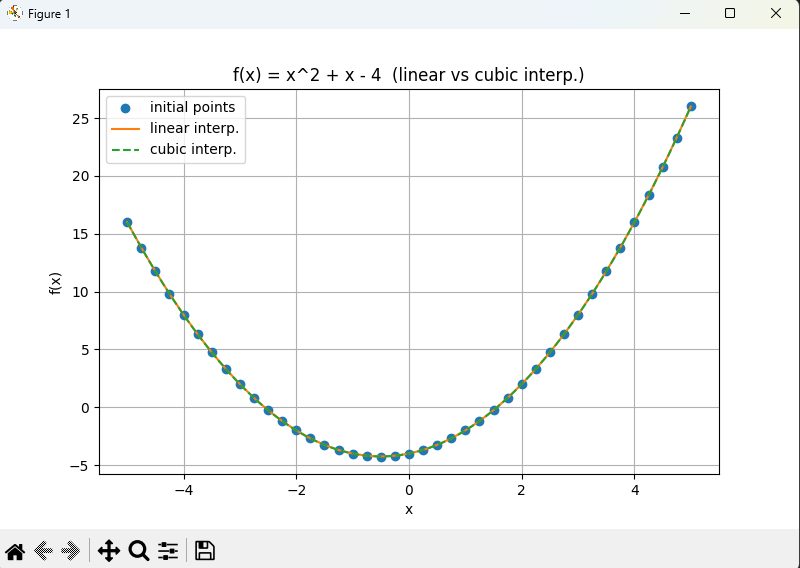
plt.title("f(x) = x^2 + x - 4 (linear vs cubic interp.)")

plt.xlabel("x"); plt.ylabel("f(x)")

plt.legend(); plt.grid(True)

plt.show()

**Result:**

****

**Explanation of the graph:**

The plotted graph shows the original polynomial points (blue), the linear interpolation (orange line), and the cubic interpolation (green dashed line). Both interpolations closely match the original quadratic curve, with the cubic interpolation providing smoother transitions.