1. Write a Python script that imports the math module and calculates the square root of a given number. Print the result

Answer:

import math

number = float(input("Enter a number: "))

sqrt\_result = math.sqrt(number)

print(f"The square root of {number} is {sqrt\_result}")

2. Create a Python module named calculator that includes functions for addition, subtraction, multiplication, and division. Import this module into another script and perform arithmetic operations using the functions.

# calculator.py

def add(a, b):

return a + b

def subtract(a, b):

return a - b

def multiply(a, b):

return a \* b

def divide(a, b):

if b == 0:

return "Division by zero is not allowed"

return a / b

# main.py

import calculator

print("Addition:", calculator.add(5, 3))

print("Subtraction:", calculator.subtract(10, 4))

print("Multiplication:", calculator.multiply(6, 7))

print("Division:", calculator.divide(20, 4))

3. Develop a Python program that uses the random module to generate a random number between 1 and 10. Prompt the user to guess the number and provide feedback on whether the guess is correct.

import random

number = random.randint(1, 10)

guess = int(input("Guess the number (between 1 and 10): "))

if guess == number:

print("Correct! You guessed it!")

else:

print(f"Wrong guess. The number was {number}.")

4. Create a 3x4 NumPy array and write a program to find the maximum element from each row.

import numpy as np

array = np.array([[5, 2, 9, 1],

[8, 7, 6, 3],

[4, 0, 11, 10]])

max\_in\_rows = np.max(array, axis=1)

print("3x4 Array:")

print(array)

print("Maximum elements from each row:", max\_in\_rows)

5. Briefly describe the role of Pandas in data analysis.

Answer:

Pandas is a powerful Python library used for data analysis and manipulation. It provides two main data structures — Series and DataFrame — which make it easy to load, analyze, clean, and visualize structured data. With Pandas, users can:

* Read and write data from various formats (CSV, Excel, SQL, etc.)
* Handle missing data
* Perform filtering, grouping, and aggregation
* Merge and join datasets
* Easily manipulate time series data

Pandas is essential in data science and machine learning workflows due to its simplicity and versatility.

6. You are given a CSV file named 'student.csv', whose first few records are as below

Write the python code / command/ express for following questions

1. ﻿﻿﻿Load this file into a python DataFrame
2. ﻿﻿﻿Find the number of rows and columns in it.
3. ﻿﻿﻿See last 5 rows from the bottom
4. ﻿﻿﻿Find the details of student with lowest marks
5. ﻿﻿﻿Find sum total marks of all female students
6. ﻿﻿﻿List names of all the male students
7. ﻿﻿﻿Find mean age of the class

import pandas as pd

# Load the CSV file into a DataFrame

df = pd.read\_csv("student.csv")

# 1. Find the number of rows and columns

rows, cols = df.shape

print("Number of rows:", rows)

print("Number of columns:", cols)

# 2. See last 5 rows from the bottom

print("\nLast 5 rows:")

print(df.tail())

# 3. Find the details of student with lowest marks

lowest\_marks = df[df['Marks'] == df['Marks'].min()]

print("\nStudent(s) with lowest marks:")

print(lowest\_marks)

# 4. Find sum total marks of all female students

female\_total\_marks = df[df['Sex'] == 'F']['Marks'].sum()

print("\nTotal marks of all female students:", female\_total\_marks)

# 5. List names of all the male students

male\_students = df[df['Sex'] == 'M']['Name'].tolist()

print("\nMale students:")

print(male\_students)

# 6. Find mean age of the class

mean\_age = df['Age'].mean()

print("\nMean age of the class:", mean\_age)

7. Plot a comparison plot of x² vs x³ for a set of numbers using matplotlib.

import matplotlib.pyplot as plt

x = list(range(1, 11))

y1 = [i\*\*2 for i in x]

y2 = [i\*\*3 for i in x]

plt.plot(x, y1, label='x^2', marker='o')

plt.plot(x, y2, label='x^3', marker='x')

plt.title('Comparison Plot: x^2 vs x^3')

plt.xlabel('x')

plt.ylabel('Values')

plt.legend()

plt.grid(True)

plt.show()

8. Implement a Python function for linear search that takes a list and a target element as parameters. Return the index of the target element if found, or -1 if not found.

def linear\_search(lst, target):

for i in range(len(lst)):

if lst[i] == target:

return i

return -1

# Example usage

numbers = [10, 20, 30, 40, 50]

print("Index found at:", linear\_search(numbers, 30))

9. Write a Python function for binary search that takes a sorted list and a target element as parameters. Return the index of the target element if found, or -1 if not found.

def binary\_search(lst, target):

low = 0

high = len(lst) - 1

while low <= high:

mid = (low + high) // 2

if lst[mid] == target:

return mid

elif lst[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1

# Example usage

sorted\_list = [5, 10, 15, 20, 25, 30]

print("Index found at:", binary\_search(sorted\_list, 20))

10. Develop a Python function to perform the Bubble Sort algorithm on a list of integers. Print the sorted list and discuss the time complexity of Bubble Sort.

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

return arr

# Example usage

nums = [64, 34, 25, 12, 22, 11, 90]

sorted\_nums = bubble\_sort(nums)

print("Sorted list:", sorted\_nums)

Time Complexity of Bubble Sort:

* Worst Case: O(n²)
* Best Case (already sorted): O(n)
* Average Case: O(n²)
* Bubble Sort is simple but inefficient for large datasets.

11. Utilize NumPy to sort a NumPy array of integers in ascending order. Print the sorted array and discuss the advantages of using NumPy for sorting.

import numpy as np

arr = np.array([10, 5, 2, 8, 7])

sorted\_arr = np.sort(arr)

print("Sorted NumPy array:", sorted\_arr)

Advantages of using NumPy for sorting:

* Faster execution due to optimized C backend.
* Handles large arrays more efficiently.
* Easy to apply sorting on specific axes of multi-dimensional arrays.

12. What advantages does NumPy offer over lists?

Answer:

* Performance: NumPy is much faster than lists, especially for large datasets due to its implementation in C.
* Memory Efficient: Consumes less memory than lists by using fixed-type arrays.
* Functionality: Supports advanced mathematical operations, broadcasting, and vectorization.
* Convenience: Offers built-in functions for linear algebra, statistics, random number generation, and more.
* Multidimensional Support: Efficient handling of multi-dimensional arrays and matrices, unlike native lists.