```
import math
N = int(input("Enter the number of elements: "))
numbers = []
for i in range(N):
    num = float(input(f"Enter number {i + 1}: "))
    numbers.append(num)

mean = sum(numbers) / N
variance = sum((x - mean) ** 2 for x in numbers) /N
std_dev = math.sqrt(variance)

print(f"Mean: {mean:.2f}")
print(f"Variance: {variance:.2f}")
print(f"Standard Deviation: {std_dev:.2f}")
```

```
Enter the number of elements: 2
Enter number 1: 22
Enter number 2: 13
Mean: 17.50
Variance: 20.25
Standard Deviation: 4.50
```

```
def determine_grade(marks):
    if marks >= 91:
        return 'S Grade'
    elif marks >= 81:
        return 'A Grade'
    elif marks >= 71:
        return 'B Grade'
    elif marks >= 61:
        return 'C Grade'
    elif marks >= 51:
        return 'D Grade'
    elif marks >= 41:
        return 'E Grade'
    else:
        return 'F Grade'

for i in range(1, 5):
    mark = int(input(f"Enter marks for subject {i}: "))

    print(f"Subject {i} - Marks: {mark}, Grade: {determine_grade(mark)}")
```

```
number_str = input("Enter a multi-digit number: ")

digit_counts = {}

for digit in number_str:
    if digit in digit_counts:
        digit_counts[digit] += 1
    else:
        digit_counts[digit] = 1

print("Frequency of each digit:")
for digit in sorted(digit_counts):
    print(f"Digit {digit}: {digit_counts[digit]} times")
```

```
n=input("enter a binary number")
count_0=n.count('0')
count_1=n.count('1')
if count_0==1 or count_1==1:
    print("yes")
else:
    print("no")
```

```
l=[4353, 2314, 2956, 3382, 9362, 3900]
l.remove(3382)
print(l)
index=l.index(9362)
print(index)
l.insert(index+1,4499)
print(l)
l.extend([5566,1830])
print(l)
l.reverse()
print(l)
l.sort()
print(l)
```

```
import re
pattern=r'(\b\w+\b)\s(\d{3}-\d{8})'
with open('1.txt',"r") as f:
    for line in f:
        match =re.search(pattern,line)
        if match:
            name=match.group(1)
            print(f"name :{name}")
```

```
a=input("enter the name of the text file ")
f=open(a+".txt","r")
n=int(input("enter the number of lines to be read"))
for i in f:
    print(f.readline(),end="")
f.seek(0)
word=input("enter the word")
c=0
for i in f.read().split():
    if i==word:
        c+=1
print(f"the word {word} occured {c} times")
```

```
import re

def is_valid_password(password):
    if len(password) < 6 or len(password) > 12:
        return "Password must be between 6 and 12 characters long."
    if not re.search(r'[a-z]', password):
        return "Must include a lowercase letter (a-z)."
    if not re.search(r'[A-Z]', password):
        return "Must include an uppercase letter (A-Z)."
    if not re.search(r'[0-9]', password):
        return "Must include a digit (0-9)."
    if not re.search(r'[$#@]', password):
        return "Must include a special character ($, #, @)."
    return "Password is valid."

# Get password from the user
password = input("Enter your password: ")
print(is_valid_password(password))
```

```
def remove_vowels(s):
    vowels = 'aeiouAEIOU'
    return ''.join(char for char in s if char not in vowels or not char.isalpha())
input_string = input("Enter a string: ")

output_string = remove_vowels(input_string)
print("String with consonants removed:", output_string)
```

```
import re
f=open('date.txt','r')
text=f.read()
numbers=re.findall(r'\b\d+\.?\d*\b',text)
total_sum=sum(map(float,numbers))
dates=re.findall(r'\b\d{2}[-/]\d{2}[-/]\d{4}\b',text)
print(total_sum)
for date in dates:
    print(date)
```

```
class Employee:
    def add_employee(self):
        self.emp_id = input("Enter Employee ID: ")
        self.emp_name = input("Enter Employee Name: ")
        self.emp_designation = input("Enter Employee Designation: ")
        self.experience = int(input("Enter Employee Experience (in years): "))
        self.age = int(input("Enter Employee Age: "))

    def display_details(self):
        print(f"Employee ID: {self.emp_id}")
        print(f"Employee Name: {self.emp_name}")
        print(f"Employee Designation: {self.emp_designation}")
        print(f"Experience: {self.experience} years")
        print(f"Age: {self.age} years")

    def calculate_salary(self, basic):
        if self.age < 30 and self.experience > 5:
            final_salary = 1.5 * basic
        elif self.age < 40 and self.experience > 5:
            final_salary = 1.75 * basic
        elif self.age < 40 and self.experience > 10:
            final_salary = 2 * basic
        elif self.age < 50 and self.experience > 20:
            final_salary = 2.25 * basic
        elif self.age < 50 and self.experience > 25:
            final_salary = 2.5 * basic
        elif self.age < 58 and self.experience > 30:
            final_salary = 3 * basic
        else:
            final_salary = basic  # No criteria met, basic salary only

        print(f"Calculated Salary: {final_salary:.2f}")

# Example usage
emp = Employee()
emp.add_employee()
emp.display_details()
basic_salary = float(input("Enter basic salary: "))
emp.calculate_salary(basic_salary)
```

```
class FourthSem:
    def __init__(self, roll_nums, test1_marks, test2_marks, test3_marks):
        self.RollNum = roll_nums
        self.Test1Marks = test1_marks
        self.Test2Marks = test2_marks
        self.Test3Marks = test3_marks

    def calculate_class_average(self):
        avg_test1 = sum(self.Test1Marks) / len(self.Test1Marks)
        avg_test2 = sum(self.Test2Marks) / len(self.Test2Marks)
        avg_test3 = sum(self.Test3Marks) / len(self.Test3Marks)
        return avg_test1, avg_test2, avg_test3

    def calculate_student_averages(self):
        student_averages = []
        for i in range(len(self.RollNum)):
            avg = (self.Test1Marks[i] + self.Test2Marks[i] + self.Test3Marks[i]) / 3
            student_averages.append(avg)
        return student_averages

    def display_top_and_last_scores(self, test_scores):
        sorted_scores = sorted(test_scores)
        top_5 = sorted_scores[-5:]
        last_5 = sorted_scores[:5]
        return top_5, last_5

    def display_all_info(self):
        avg_test1, avg_test2, avg_test3 = self.calculate_class_average()
        print(f"Class average for Test 1: {avg_test1}")
        print(f"Class average for Test 2: {avg_test2}")
        print(f"Class average for Test 3: {avg_test3}")

        student_averages = self.calculate_student_averages()
        for i in range(len(self.RollNum)):
            print(f"Student {self.RollNum[i]} average: {student_averages[i]}")

        top_5_test1, last_5_test1 = self.display_top_and_last_scores(self.Test1Marks)
        top_5_test2, last_5_test2 = self.display_top_and_last_scores(self.Test2Marks)
        top_5_test3, last_5_test3 = self.display_top_and_last_scores(self.Test3Marks)

        print(f"Top 5 scores for Test 1: {top_5_test1}")
        print(f"Last 5 scores for Test 1: {last_5_test1}")
        print(f"Top 5 scores for Test 2: {top_5_test2}")
        print(f"Last 5 scores for Test 2: {last_5_test2}")
        print(f"Top 5 scores for Test 3: {top_5_test3}")
        print(f"Last 5 scores for Test 3: {last_5_test3}")

# Example usage:
roll_nums = [i for i in range(1, 21)]
test1_marks = [75, 85, 95, 65, 55, 70, 80, 90, 60, 50, 77, 87, 97, 67, 57, 72, 82, 92, 62, 52]
test2_marks = [78, 88, 98, 68, 58, 73, 83, 93, 63, 53, 79, 89, 99, 69, 59, 74, 84, 94, 64, 54]
test3_marks = [80, 90, 100, 70, 60, 75, 85, 95, 65, 55, 82, 92, 102, 72, 62, 77, 87, 97, 67, 57]

fourth_sem = FourthSem(roll_nums, test1_marks, test2_marks, test3_marks)
fourth_sem.display_all_info()
```

```
# Original marks data
marks = [78, 92, 36, 64, 89]

# Calculate the sum of all marks
total_marks = sum(marks)
print(f"Sum of all marks: {total_marks}")

# Award extra marks
extra_marks = [2, 2, 5, 10, 2]
adjusted_marks = [marks[i] + extra_marks[i] for i in range(len(marks))]

# Display adjusted marks
print("\nAdjusted Marks:")
subjects = ["English", "Mathematics", "Physics", "Chemistry", "Biology"]
for i in range(len(subjects)):
    print(f"{subjects[i]}: {adjusted_marks[i]}")

# Calculate the new sum of all marks after awarding extra marks
new_total_marks = sum(adjusted_marks)
print(f"\nNew sum of all marks after awarding extra marks: {new_total_marks}")
```

```
import numpy as np

# Given horsepower data
horsepower = np.array([130, 165, 150, 150, 140])

# Calculate the mean
mean_hp = np.mean(horsepower)
print(f"Mean of horsepower values: {mean_hp}")

# Calculate the median
median_hp = np.median(horsepower)
print(f"Median of horsepower values: {median_hp}")
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('student.csv')

# 1. Compare the distribution of grades across different subjects using a box plot
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['marks1', 'marks2', 'marks3', 'marks4', 'marks5']])
plt.title('Distribution of Grades Across Different SubjePcts')
plt.xlabel('Subjects')
plt.ylabel('Grades')
plt.savefig('boxplot_grades.png')


# 2. Show the relationship between grades in two specific subjects using a scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(df['marks1'], df['marks2'])
plt.title('Relationship between Grades in Subject 1 and Subject 2')
plt.xlabel('Subject 1 Grades')
plt.ylabel('Subject 2 Grades')
plt.savefig('scatterplot_grades.png')




# 3. Display the distribution of overall grades using a histogram
plt.figure(figsize=(10, 6))
plt.hist(df['sum_value1_value2'], bins=10, edgecolor='black')
plt.title('Distribution of Overall Grades')
plt.xlabel('Total Grades')
plt.ylabel('Frequency')

print("Analysis complete. Visualizations have been saved as PNG files.")
```

```
import matplotlib.pyplot as plt
import pandas as pd

# Sample data for daily visitors
daily_data = {
    'Date': pd.date_range(start='2023-07-01', periods=14, freq='D'),
    'Visitors': [100, 120, 130, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250]
}

# Create a DataFrame
df_daily = pd.DataFrame(daily_data)

# 1. Line Plot: Daily Number of Visitors
plt.figure(figsize=(10, 5))
plt.plot(df_daily['Date'], df_daily['Visitors'], marker='o', linestyle='-', color='b')
plt.title('Daily Number of Visitors')
plt.xlabel('Date')
plt.ylabel('Number of Visitors')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Sample data for visitors on weekdays vs weekends
weekday_visitors = [110, 130, 120, 140, 150]
weekend_visitors = [180, 170]
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

# 2. Bar Plot: Weekdays vs Weekends
plt.figure(figsize=(10, 5))
plt.bar(days[:5], weekday_visitors, color='blue', label='Weekdays')
plt.bar(days[5:], weekend_visitors, color='green', label='Weekends')
plt.title('Number of Visitors: Weekdays vs Weekends')
plt.xlabel('Day')
plt.ylabel('Number of Visitors')
plt.legend()
plt.show()

# Sample data for traffic sources
traffic_sources = ['Direct', 'Search Engines', 'Social Media', 'Other']
traffic_values = [300, 500, 200, 100]


# 3. Pie Chart: Traffic Sources
plt.figure(figsize=(7, 7))
plt.pie(traffic_values, labels=traffic_sources, autopct='%1.1f%%', startangle=140, colors=['blue', 'green', 'red', 'purple'])
plt.title('Traffic Sources')
```

```
import pandas as pd
import numpy as np

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('rainfall.csv')
```