Program: 1.1 Date: 05-08-2024

Even/Odd Check

Aim: Ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user.

Algorithm

1. Start.
2. Take a number as input from the user
3. If the given number is divisible by 2, then print the number followed by " is Even"
   else, print the number followed by " is Odd".
4. End.

Program

```
num = int(input("Enter a number: "))
if num%2 == 0:
   print(str(num) + " is even")
else:
   print(str(num) + " is odd")
```

Result

Determined if the number is even or odd.

Output-1.1

Enter a number: 37
37 is odd

Enter a number: 144
144 is even

Program: 1.2 Date: 05-08-2024

Divisors of a number

Aim: To create a program that asks the user for a number and then prints out a list of all the divisors of that number

Algorithm

1. Start
2. Prompt the user to enter an integer number and store it in `num`.
3. Initialize an empty list `A` to hold the divisors.
4. Loop through each integer `i` from 1 to `num` (inclusive):
   - If `num` is divisible by `i` (i.e., `num % i == 0`), append `i` to the list `A`.
5. After the loop, print the message "The divisors are :" followed by the contents of list `A`.
6. End

Program

```
num = int(input("Enter the number: "))
A = []
for i in range(1,num+1):
   if num % i == 0:
      A.append(i)
print("The divisors are :", A)
```

Result

Identified all divisors of the number.

Output-1.2

Enter the number: 10
The divisors are : [1, 2, 5, 10]

Program: 1.3 Date: 05-08-2024

Decimal to other bases conversion

Aim: To read a number in decimal and convert it into different bases.

Algorithm

1. Start
2. Prompt the user to enter a decimal number and store it in `num`.
3. Convert `num` to binary, octal, and hexadecimal:
   - Store the binary representation in `binary` using `bin(num)`.
   - Store the octal representation in `octal` using `oct(num)`.
   - Store the hexadecimal representation in `hexadecimal` using `hex(num)`.
4. Print the binary form by slicing `binary` to exclude the '0b' prefix.
5. Print the octal form by slicing `octal` to exclude the '0o' prefix.
6. Print the hexadecimal form by slicing `hexadecimal` to exclude the '0x' prefix.
7. End

Program

```
num = int(input("Enter a decimal number : "))
binary = bin(num)
octal = oct(num)
hexadecimal = hex(num)
print("Binary form: ", binary[2:])
print("Octal form: ", octal[2:])
print("Hexadecimal form: ", hexadecimal[2:])
```

Result

Converted the decimal number into different bases.

Output-1.3

Enter a number : 37
Binary form: 100101
Octal form: 45
Hexadecimal form: 25

Program: 1.4 Date: 05-08-2024

Multiplication table

Aim: To print the multiplication table of a given number. (use while loop)

Algorithm

1. Start
2. Take a number as input from the user
3. Initialize i to 1
4. While i is less than or equal to 10:
   - Calculate product as i multiplied by num
   - Print the string representation of the multiplication in the format "{num} x {i} = {d}"
   - Increment i by 1
5. End

Program

```
num = int(input("Enter a number: "))
i = 1
print("Multiplication table of ",num)
while i<=10:
   product = i * num
   print(num," x ",i," = ",product)
   i+=1
```

Result

Generated the multiplication table.

Output-1.4

```
Enter a number: 8
Multiplication table of  8
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

Program: 1.5 Date: 05-08-2024

Nth Fibonacci

Aim: To Find nth Fibonacci number using recursion.

Algorithm

1. Start
2. Define a function fibonacci(n) to calculate the Fibonacci number:
3. If n is equal to 1, return 0
   else if n is equal to 2 or 3, return 1
   else, return the sum of fibonacci(n-1) and fibonacci(n-2)
4. Take a number as input from the user
5. Print the string representation of the number followed by "th Fibonacci number is: " and the result of fibonacci(number)
6. End

Program

```
num = int(input("Enter a number: "))
def fibonacci(a):
    if a==1:
        return 0
    elif a==2:
        return 1
    else:
        return fibonacci(a-1) + fibonacci(a-2)

result = fibonacci(num)
print(str(num)+"th fibonacci number is "+str(result))
```

Result

Found the Nth Fibonacci number.

Output-1.5

Enter a number: 5
5th fibonacci number is 3

Program: 1.6 Date: 05-08-2024

Sparse matrix

Aim: To read and display a sparse matrix using dictionary

Algorithm

1. Start
2. Function read_matrix:
   - Initialize matrix as an empty dictionary.
   - Input num_rows, num_cols, and num_entries.
   - For each entry from 1 to num_entries:
     - Input row, col, and value.
     - Store value in matrix using key (row, col).
   - Return matrix, num_rows, num_cols, and num_entries.
3. Function display_matrix:
   - For each i from 0 to rows - 1:
     - For each j from 0 to cols - 1:
       - Print matrix.get((i, j), 0) followed by a tab.
     - Print a newline.
4. Main Program:
   - Call read_matrix and store results.
   - Print matrix dimensions and number of non-zero entries.
   - Call display_matrix to show the sparse matrix.
5. End

Program

```
def read_matrix():
    matrix = {}
    num_rows = int(input("Enter the number of rows: "))
    num_cols = int(input("Enter the number of columns: "))
    num_entries = int(input("Enter the number of non-zero entries: "))
    for i in range(num_entries):
        row = int(input("Enter the row index: "))
        col = int(input("Enter the column index: "))
        value = int(input("Enter the value: "))
        matrix[(row, col)] = value
    return matrix, num_rows, num_cols, num_entries

def display_matrix(matrix, rows, cols):
    for i in range(rows):
        for j in range(cols):
            print(f"{matrix.get((i, j), 0)}\t", end=" ")
        print()
```

Output-1.6

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the number of non-zero entries: 2
Enter the row index: 1
Enter the column index: 1
Enter the value: 5
Enter the row index: 2
Enter the column index: 2
Enter the value: 7
Matrix dimensions: 3 rows x 3 columns
Number of non-zero entries: 2
Sparse Matrix:
3       3       2
0       0       0
0       5       0
0       0       7
```

```
sparse_matrix, num_rows, num_cols, num_entries = read_matrix()
print(f"Matrix dimensions: {num_rows} rows x {num_cols} columns")
print(f"Number of non-zero entries: {num_entries}")
print("Sparse Matrix:")
print(num_rows,"\t",num_cols,"\t",num_entries)
display_matrix(sparse_matrix, num_rows, num_cols)
```

Result

Successfully displayed the sparse matrix.

Program: 1.7 Date: 05-08-2024

Average grade

Aim: To Create a Python dictionary representing a student's grades in different subjects (e.g., Math, Science, History). Write a function to calculate the average grade of the student across all subjects.

Algorithm

1. Start
2. Initialize a dictionary grades with subjects as keys and their corresponding grades as values
3. Define a function findavg(grades) to calculate the average:
   - Calculate the average as the sum of the values in grades divided by the number of grades
   - Return the calculated average
4. Print "Average grade is: " followed by the rounded result of findavg(grades)
5. End

Program

```
grades = {"Math":70,
          "English":84,
          "Science":90}

def findavg(grades):
   average = sum(grades.values())/len(grades)
   return average

print("Average grade is",round(findavg(grades),2))
```

Result

Computed the average grade.

Output-1.7

Average grade is 81.33

Program: 2.1 Date: 12-08-2024

Valid parenthesis

Aim: Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: a. Open brackets must be closed by the same type of brackets., b. Open brackets must be closed in the correct order. c. Every close bracket has a corresponding open bracket of the same type.

Algorithm

1. Start
2. Define function ‘checker(s)’:
   - Initialize an empty list `stack`.
   - Create a dictionary `checker` with matching pairs of parentheses, brackets, and braces.
3. For each character `char` in the string `s`:
   - If `char` is an opening bracket, append it to `stack`.
   - If `char` is a closing bracket:
     - If `stack` is empty or the last item in `stack` doesn't match `char`, return False.
     - Otherwise, pop the last item from `stack`.
4. After the loop:
   - Return `True` if `stack` is empty (valid string), otherwise return `False`.
5. In the main program:
   - Input a string from the user.
   - Call `checker` with the input string.
   - Print "Valid String" if `checker` returns True, otherwise print "Invalid String".
6. End

Program

```
def checker(s):
    stack = []
    checker = {'(': ')', '{': '}', '[': ']'}

    for char in s:
        if char in checker:
            stack.append(char)
        elif char in checker.values():
            if not stack or checker[stack.pop()] != char:
                return False

    return not stack

string = str(input("Enter a String: "))
if(checker(string)):
    print("Valid String")
else:
    print("Invalid String")
```

Result

Checked the validity of the bracket string.

Output-2.1

Enter a String: {}[]()
Valid String

Enter a String: [][[]]
Valid String

Enter a String: ({)}
Invalid String

Program: 2.2 Date: 12-08-2024

Fruit presence check

Aim: Define a tuple containing the names of different fruits. Write a Python program that prompts the user to enter a fruit name. If the entered fruit exists in the tuple, display a message confirming its presence; otherwise, display a message indicating its absence.

Algorithm

1. Start
2. Initialize a tuple fruits with the values ("apple", "banana", "orange", "grapes", "mango", "pineapple", "strawberry")
3. Take a fruit name as input from the user and convert it to lowercase, storing it in fruit_name
4. If fruit_name is present in fruits:
   - Print ‘fruit_name’ is present in the tuple."

   Else:
   - Print ‘fruit_name’ not present in the tuple."
5. End

Program

```
fruits = ("apple","mango","grapes","pineapple",”strawberry”,”orange”,”banana”)
fruit_name = str(input("Enter a fruit name: ")).lower()
if fruit_name in fruits:
   print(fruit_name," is in the tuple")
else:
   print(fruit_name," is not in the tuple")
```

Result

Checked for the presence of the fruit in the tuple.

Output-2.2

Enter a fruit name: mango
mango is in the tuple

Enter a fruit name: watermelon
watermelon is not in the tuple

Program: 2.3 Date: 12-08-2024

Count Vowels, Consonants, Words and "?"

Aim: Count the number of vowels, consonants, words and question marks in a given string.

Algorithm

1. Start
2. Input a string and store it in the variable `name`.
3. Initialize a string `vowels` containing all lowercase and uppercase vowels.
4. Initialize counters `vowelcount`, `qmarkcount`, and `consonentcount` to 0.
5. Split the input string `name` into words and store the number of words in `wordscount`.
6. For each character `char` in the string `name`:
   - If `char` is a vowel, increment `vowelcount`.
   - Else, if `char` is alphabetic, increment `consonentcount`.
   - Else, if `char` is a question mark (`?`), increment `qmarkcount`.
7. Print the counts of vowels, consonants, question marks, and words.
8. End

Program

```
name = input("Enter a string: ")
vowels = 'aeiouAEIOU'
vowelcount = qmarkcount = consonentcount = 0
words = name.split()
wordscount = len(words)

for char in name:
    if char in vowels:
        vowelcount += 1
    elif char.isalpha():
        consonentcount += 1
    elif char=='?':
        qmarkcount +=1

print("Counts of")
print("Vowels = ",vowelcount)
print("Consonents = ",consonentcount)
print("Question marks = ",qmarkcount)
print("Words = ",wordscount))
```

Result

Counted vowels, consonants, words, and question marks in the string.

Output

Enter a string: What if what we know is just an illusion?
Counts of
Vowels = 12
Consonents = 20
Question marks = 1
Words = 9

Program: 2.4 Date: 12-08-2024

Pivot Tables and Cross Tabulations

Aim: To create Pivot Tables and Cross Tabulations

Algorithm

1. Start
2. Import pandas as `pd`.
3. Create a DataFrame `data` with columns: 'Student ID', 'Name', 'Subject', 'Score', 'Semester'.
4. Create a pivot table `pivot_table`:
   - Set 'Subject' as the index.
   - Set 'Semester' as the columns.
   - Use 'Score' as the values.
   - Apply the mean function to aggregate scores.
   - Fill any missing values with 0.
5. Print the pivot table.
6. Create a cross-tabulation `cross_tab`:
   - Set 'Subject' as the row index.
   - Set 'Semester' as the column index.
   - Count the number of students for each subject per semester.
   - Include margins to show total counts for each subject and semester.
7. Print the cross-tabulation.
8. End

Program

```
import pandas as pd
data = pd.DataFrame({
    'Student ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Helen', 'Ivan', 'Judy'],
    'Subject': ['Math', 'English', 'Math', 'English', 'Math', 'Science', 'Science', 'Math', 'English', 'Science'],
    'Score': [85, 78, 92, 88, 76, 90, 82, 95, 70, 85],
    'Semester': [1, 1, 2, 2, 1, 1, 2, 2, 1, 2]
})

pivot_table = pd.pivot_table(data, values='Score', index='Subject', columns='Semester', aggfunc='mean', fill_value=0)
print("Pivot Table:")
print(pivot_table)

cross_tab = pd.crosstab(data['Subject'], data['Semester'], values=data['Student ID'], aggfunc='count', margins=True)
print("\nCross Tabulation:")
print(cross_tab)
```

Result

Created pivot table and cross tabulations.

Output

Pivot Table:

Semester	1	2
Subject		
English	74.0	88.0
Math	80.5	93.5
Science	90.0	83.5

Cross Tabulation:

Semester	1	2	All
Subject			
English	2	1	3
Math	2	2	4
Science	1	2	3
All	5	5	10

Program: 2.5 Date: 12-08-2024

Bar Chart of Continents

Aim: The areas of the various continents of the world (in millions of square miles) are as follows:11.7 for Africa; 10.4 for Asia; 1.9 for Europe; 9.4 for North America; 3.3 Oceania; 6.9 South America; 7.9 Soviet Union. Draw a bar chart representing the given data.

Algorithm

1. Start
2. Import the matplotlib.pyplot library as plt
3. Initialize a list continents with given continent names
4. Initialize a list areas with the given areas.
5. Create a bar chart using plt.bar() with:
   - continents as the x-axis
   - areas as the y-axis
6. Set the label for the x-axis to "Continents" using plt.xlabel()
7. Set the label for the y-axis to "Area (in millions of square miles)" using plt.ylabel()
8. Set the title of the chart to "Areas of Continents" using plt.title()
9. Display the plot using plt.show()
10. End

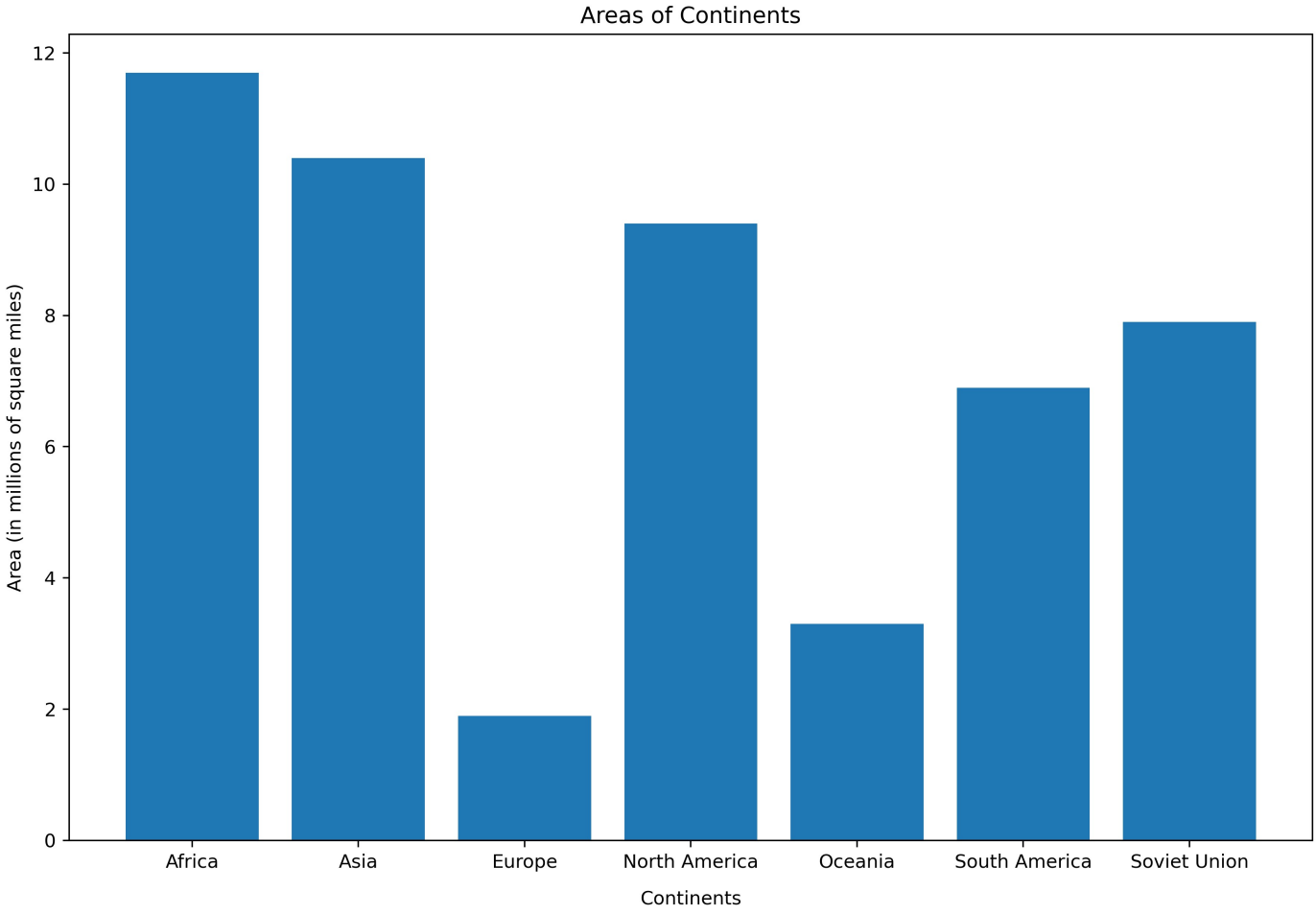
Program

```
import matplotlib.pyplot as plt

continents = ["Africa", "Asia", "Europe", "North America", "Oceania", "South America", "Soviet Union"]
areas = [11.7, 10.4, 1.9, 9.4, 3.3, 6.9, 7.9]

plt.bar(continents, areas)
plt.xlabel("Continents")
plt.ylabel("Area (in millions of square miles)")
plt.title("Areas of Continents")
plt.show()
```

Result

Displayed a bar chart representing the areas of the continents.

Output-2.5


Areas of Continents
Area (in millions of square miles)
0
2
4
6
8
10
12
Africa
Asia
Europe
North America
Oceania
South America
Soviet Union
Continents

Program: 3.1 Date: 22-08-2024

Plot y = f(x)

Aim: To plot a graph y = f(x)

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Import the matplotlib.pyplot library as `plt`.
4. Generate x values:
   - Use `np.linspace` to create 100 points from 0 to 10.
5. Compute y values:
   - Calculate y as the square of each x value (y = x^2).
6. Create the plot:
   - Use `plt.plot` to plot x against y.
7. Set the labels and title:
   - Label the x-axis as "x".
   - Label the y-axis as "y".
   - Set the title as "Graph of y = x^2".
8. Display the plot using `plt.show()`.
9. End

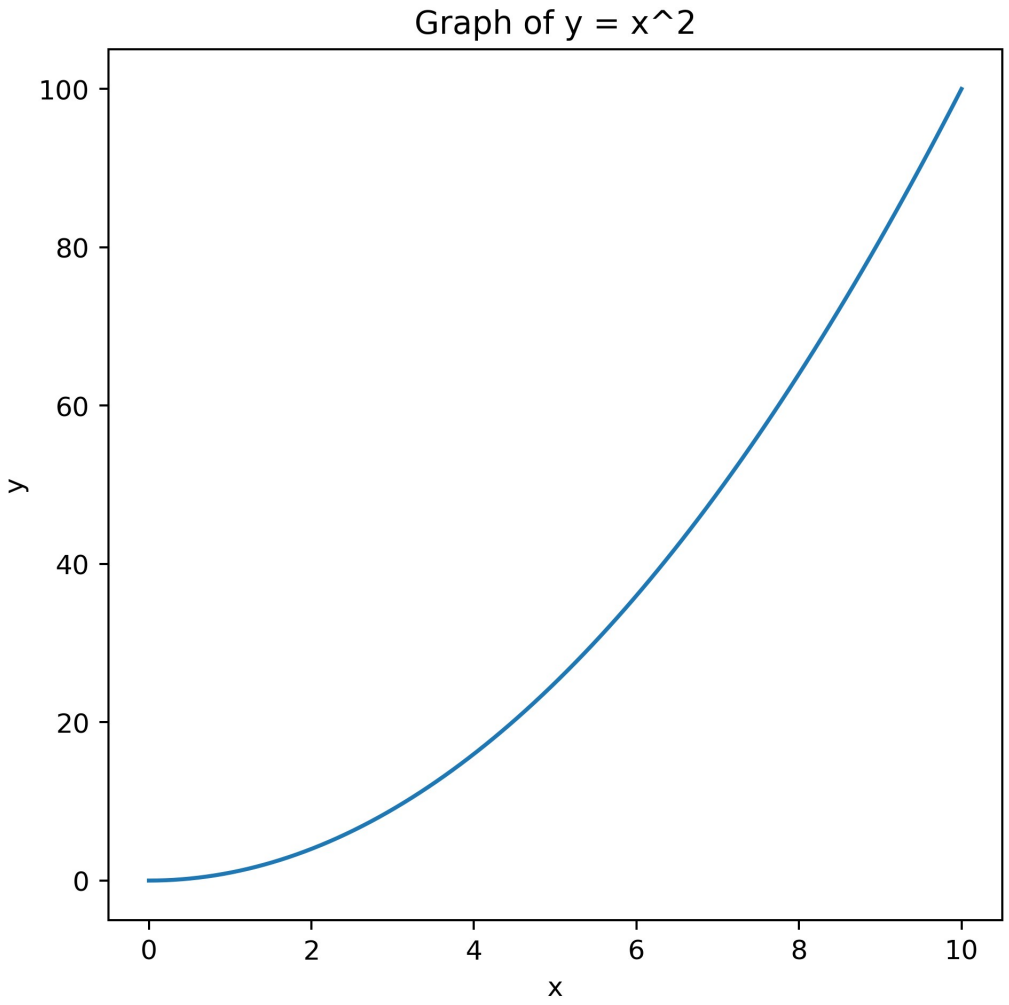
Program

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

x = np.linspace(0,10,100)
y = x**2

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Graph of y = x^2")
plt.show()
```

Result

Plotted y = f(x) showing the relationship between x and y.

Output-3.1



Program: 3.2 Date: 22-08-2024

Get 2D Diagonals from 3D Array

Aim: To get all 2D diagonals of a 3D NumPy array.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Create a 3D array `arr`:
   - Use `np.arange(27)` to generate values from 0 to 26.
   - Reshape the array into shape (3, 3, 3).
4. Compute diagonals:
   - Calculate `diagonals_1_2` along axes 1 and 2 using `np.diagonal(arr, axis1=1, axis2=2)`.
   - Calculate `diagonals_1_3` along axes 1 and 3 using `np.diagonal(arr, axis1=0, axis2=2)`.
   - Calculate `diagonals_2_3` along axes 2 and 3 using `np.diagonal(arr, axis1=0, axis2=1)`.
5. Print the results:
   - Print "Diagonals along axis 1 and 2:" followed by `diagonals_1_2`.
   - Print "Diagonals along axis 1 and 3:" followed by `diagonals_1_3`.
   - Print "Diagonals along axis 2 and 3:" followed by `diagonals_2_3`.
6. End

Program

```
import numpy as np
arr = np.arange(27).reshape(3, 3, 3)

diagonals_1_2 = np.diagonal(arr, axis1=1, axis2=2)
diagonals_1_3 = np.diagonal(arr, axis1=0, axis2=2)
diagonals_2_3 = np.diagonal(arr, axis1=0, axis2=1)

print("Diagonals along axis 1 and 2:")
print(diagonals_1_2)
print("Diagonals along axis 1 and 3:")
print(diagonals_1_3)
print("Diagonals along axis 2 and 3:")
print(diagonals_2_3)
```

Result

Extracted 2D diagonals from the 3D array.

Output-3.2

Diagonals along axis 1 and 2:
[[0 4 8]
 [9 13 17]
 [18 22 26]]

Diagonals along axis 1 and 3:
[[0 10 20]
 [3 13 23]
 [6 16 26]]

Diagonals along axis 2 and 3:
[[0 12 24]
 [1 13 25]
 [2 14 26]]

Program: 3.3 Date: 22-08-2024

Flatten 2D Array to 1D

Aim: To flatten a 2d numpy array into 1d array

Algorithm

1. Start
2. Import the numpy library as np
3. Create a 2D array array_2d with the values [[1, 2, 3], [4, 5, 6]]
4. Flatten the 2D array using flatten() and store the result in array_1d
5. Print "Flattened array: " followed by array_1d
6. End

Program

```
import numpy as np

array_2d = np.array([[1, 2, 3], [4, 5, 6]])
array_1d = array_2d.flatten()

print("Flattened array:", array_1d)
```

Result

Flattened 2D array to 1D.

Output-3.3

Flattened array: [1 2 3 4 5 6]

Program: 3.4 Date: 22-08-2024

Fibonacci Series Using Binet's Formula

Aim: To compute Fibonacci numbers efficiently using Binet's formula with NumPy.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define the function `fibonacci_binet(n)`:
   - Calculate `phi` as (1 + sqrt(5)) / 2.
   - Calculate `psi` as (1 - sqrt(5)) / 2.
   - Return the Fibonacci number as int((phi^n - psi^n) / sqrt(5)).
4. Input the number of Fibonacci numbers to display (`n_terms`).
5. Generate a list `fibonacci_numbers`:
   - Use a list comprehension to compute Fibonacci numbers for `n` from 0 to `n_terms - 1` by calling `fibonacci_binet(n)`.
6. Print the list of Fibonacci numbers.
7. End

Program

```
import numpy as np

def fibonacci_binet(n):
    phi = (1 + np.sqrt(5)) / 2
    psi = (1 - np.sqrt(5)) / 2
    return int((phi**n - psi**n) / np.sqrt(5))

n_terms = int(input("Enter number of fibonacci numbers to display: "))
fibonacci_numbers = [fibonacci_binet(n) for n in range(n_terms)]
print("Fibonacci numbers:", fibonacci_numbers)
```

Result

Generated Fibonacci series using Binet's formula.

Output-3.4

```
Enter number of fibonacci numbers to display: 7
Fibonacci numbers: [0, 1, 1, 2, 3, 5, 8]
```

Program: 3.5 Date: 22-08-2024

Inverse a Matrix

Aim: To inverse a matrix using NumPy.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define a matrix.
4. Calculate the determinant of the matrix using `np.linalg.det(matrix)`.
5. Check if the determinant is zero:
   - If true, print "The matrix is singular and does not have an inverse."
   - If false, calculate the inverse using `np.linalg.inv(matrix)` and print the inverted matrix.
6. End

Program

```
import numpy as np
matrix = np.array([[1, 2],[3, 4]])
determinant = np.linalg.det(matrix)
if determinant == 0:
    print("The matrix is singular and does not have an inverse.")
else:
    inverse_matrix = np.linalg.inv(matrix)
    print("Inverse of the matrix:")
    print(inverse_matrix)
```

Result

Calculated the matrix inverse.

Output-3.5

Inverse of the matrix:
[[-2. 1.]
 [1.5 -0.5]]

Program: 3.6 Date: 22-08-2024

Inner, Outer, and Cross Products

Aim: To calculate inner, outer, and cross products of matrices and vectors using NumPy.

Algorithm

1. Start
2. Define two 1D NumPy arrays `vector_a` and `vector_b` with given values.
3. Calculate the inner product of `vector_a` and `vector_b` using `np.inner()` and store it in `inner_product`.
4. Calculate the outer product of `vector_a` and `vector_b` using `np.outer()` and store it in `outer_product`.
5. Calculate the cross product of `vector_a` and `vector_b` using `np.cross()` and store it in `cross_product`.
6. Print the results:
   - Display the `inner_product`.
   - Display the `outer_product`.
   - Display the `cross_product`.
7. End

Program

```
import numpy as np
vector_a = np.array([1, 2, 3])
vector_b = np.array([4, 5, 6])

inner_product = np.inner(vector_a,vector_b)
outer_product = np.outer(vector_a,vector_b)
cross_product = np.cross(vector_a,vector_b)

print("Inner product: ", inner_product)
print("Outer product: ", outer_product)
print("Cross product: ", cross_product)
```

Result

Computed inner, outer, and cross products.

Output-3.6

Inner Product:
32

Outer Product:
[[4 5 6]
 [8 10 12]
 [12 15 18]]

Cross Product:
[-3 6 -3]

Program: 3.7 Date: 22-08-2024

Kronecker Product of Arrays

Aim: To calculate inner, outer, and cross products of matrices and vectors using NumPy.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define two 2D arrays `array_a` and `array_b`.
4. Calculate the Kronecker product:
   - Use `np.kron(array_a, array_b)` and store the result in `kronecker_product`.
5. Print the Kronecker product.
6. End

Program

```
import numpy as np
array_a = np.array([[1, 2], [3, 4]])
array_b = np.array([[0, 5], [6, 7]])

kronecker_product = np.kron(array_a, array_b)

print("Kronecker Product:")
print(kronecker_product)
```

Result

Calculated the Kronecker product of arrays.

Output-3.7

Kronecker Product:

```
[[ 0  5  0 10]
 [ 6  7 12 14]
 [ 0 15  0 20]
 [18 21 24 28]]
```

Program: 3.8 Date: 22-08-2024

Convert Matrix to List

Aim: To convert the matrix into a list.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define a 2D NumPy array `matrix`.
4. Convert the matrix to a list:
   - Use `matrix.tolist()` and store the result in `matrix_list`.
5. Print the matrix as a list.
6. End

Program

```
import numpy as np
matrix = np.array([[1,2], [3,4]])

matrix_list = matrix.tolist()
print("Matrix as a list: ",matrix_list)
```

Result

Converted matrix to list.

Output-3.8

Original matrix:
[[1 2 3]
 [4 5 6]]

Matrix as list:
[[1, 2, 3], [4, 5, 6]]

Program: 4.1 Date: 23-09-2024

QR Decomposition of Matrix

Aim: To calculate the QR decomposition of a given matrix using NumPy.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define a 2D array `A`.
4. Perform QR decomposition:
   - Use `np.linalg.qr(A)` to decompose `A` into `Q` (orthogonal matrix) and `R` (upper triangular matrix).
5. Print matrices `Q` and `R`.
6. End

Program

```
import numpy as np
A = np.array([[1,2],[3,4],[5,6]])

Q,R = np.linalg.qr(A)

print("Matrix Q (Orthogonal):")
print("Q:",Q)
print("\nMatrix R (Upper Triangular):")
print("R:",R)
```

Result

Computed QR decomposition using NumPy.

Output-4.1

```
Matrix Q (Orthogonal):
Q: [[-0.16903085  0.89708523]
    [-0.50709255  0.27602622]
    [-0.84515425 -0.34503278]]

Matrix R (Upper Triangular):
R: [[-5.91607978 -7.43735744]
    [ 0.          0.82807867]]
```

Program: 4.2 Date: 23-09-2024

Eigenvalues and Eigenvectors

Aim: To compute the eigenvalues and right eigenvectors of a given square array using NumPy.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define a 2D array `A`.
4. Compute the eigenvalues and eigenvectors:
   - Use `np.linalg.eig(A)` to calculate `eigenvalues` and `eigenvectors`.
5. Print the eigenvalues and eigenvectors.
6. End

Program

```
import numpy as np
A = np.array([[1,2],[0,3]])

eigenvalues,eigenvectors = np.linalg.eig(A)

print("Eigenvalues:",eigenvalues)
print("Eigenvectors:",eigenvectors)
```

Result

Calculated eigenvalues and right eigenvectors of the square array.

Output-4.2

```
Eigenvalues: [1. 3.]
Eigenvectors: [[1.         0.70710678]
               [0.         0.70710678]]
```

Program: 4.3 Date: 23-09-2024

n-th Order Discrete Difference

Aim: To calculate the n-th order discrete difference along the given axis

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define a 1D array `A`.
4. Input the n-value for the order of difference.
5. Calculate the n-th order difference:
   - Use `np.diff(A, n)` and store the result in `diff`.
6. Print the n-th order difference.
7. End

Program

```
import numpy as np
A = np.array([1,2,4,7,0])
n=int(input("n-value: "))

diff = np.diff(A,n)
print(f"Order {n} Difference:",diff)
```

Result

Calculated n-th order discrete difference along the given axis.

Output-4.3

n-value: 1
Order 1 Difference: [1 2 3 -7]

n-value: 2
Order 2 Difference: [1   1 -10]

Program: 4.4 Date: 23-09-2024

Einstein's Summation Convention

Aim: To evaluate Einstein's summation convention of two multidimensional NumPy arrays.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define two 2D arrays `A` and `B`.
4. Compute the matrix multiplication using Einstein summation:
   - Use `np.einsum('ij,jk->ik', A, B)` and store the result in `result`.
5. Print the result.
6. End

Program

```
import numpy as np
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])

result = np.einsum('ij,jk->ik',A,B)
print("Result:",result)
```

Result

Evaluated Einstein's summation for two multidimensional arrays.

Output-4.4

```
Result: [[19 22]
         [43 50]]
```

Program: 4.5 Date: 23-09-2024

Pearson Correlation Coefficients

Aim: To compute pearson product-moment correlation coefficients of two given NumPy arrays

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define two 1D arrays `x` and `y`.
4. Compute the Pearson correlation coefficient:
   - Use `np.corrcoef(x, y)` and store the result in `correlation_matrix`.
5. Print the Pearson correlation coefficient.
6. End

Program

```
import numpy as np
x = np.array([1,2,3,4])
y = np.array([4,5,6,7])

correlation_matrix = np.corrcoef(x,y)
print("Pearson Correlation Coefficient:", correlation_matrix)
```

Result

Computed Pearson correlation coefficients between two arrays.

Output-4.5

Pearson Correlation Coefficient: [[1. 1.]
[1. 1.]]

Program: 4.6 Date: 23-09-2024

Average, Variance, and Standard Deviation

Aim: To compute pearson product-moment correlation coefficients of two given NumPy arrays

Algorithm

1. Start
2. Define a 1D NumPy array `A` with the given values `[1, 2, 3, 4, 5]`.
3. Calculate the average of array `A` using `np.mean()` and store it in `average`.
4. Calculate the variance of array `A` using `np.var()` and store it in `variance`.
5. Calculate the standard deviation of array `A` using `np.std()` and store it in `sd`.
6. Print the calculated values:
   - Display `average`.
   - Display `variance`.
   - Display `sd` rounded to 2 decimal places using `round()`.
7. End

Program

```
import numpy as np
A = np.array([1,2,3,4,5])

average = np.mean(A)
variance = np.var(A)
sd = np.std(A)

print("Average:",average)
print("Variance:",variance)
print("Standard Deviation:",round(sd,2))
```

Result

Calculated average, variance, and standard deviation.

Output-4.6

Average: 3.0
Variance: 2.0
Standard Deviation: 1.41

Program: 4.7 Date: 23-09-2024

Line Graph

Aim: To plot line graph from NumPy array.

Algorithm

1. Start
2. Import the numpy library as `np` and the matplotlib.pyplot library as `plt`.
3. Create a NumPy array `x` for the x-axis:
   - Use `np.linspace(0, 2 * np.pi, 500)` to generate 500 points from 0 to 2π radians.
4. Create a NumPy array `y` for the y-axis:
   - Calculate the sine values using `np.sin(x)`.
5. Create the line graph:
   - Use `plt.plot(x, y, color='blue')`.
6. Add labels and title:
   - Set the x-axis label using `plt.xlabel('X-axis (radians)')`.
   - Set the y-axis label using `plt.ylabel('Y-axis (sin values)')`.
   - Set the title using `plt.title('Line graph of Sine Function from NumPy array')`.
7. Customize the x-ticks:
   - Use `plt.xticks()` to set custom tick positions and labels.
8. Display the plot:
   - Use `plt.grid()` for better readability.
   - Use `plt.show()` to display the graph.
9. End

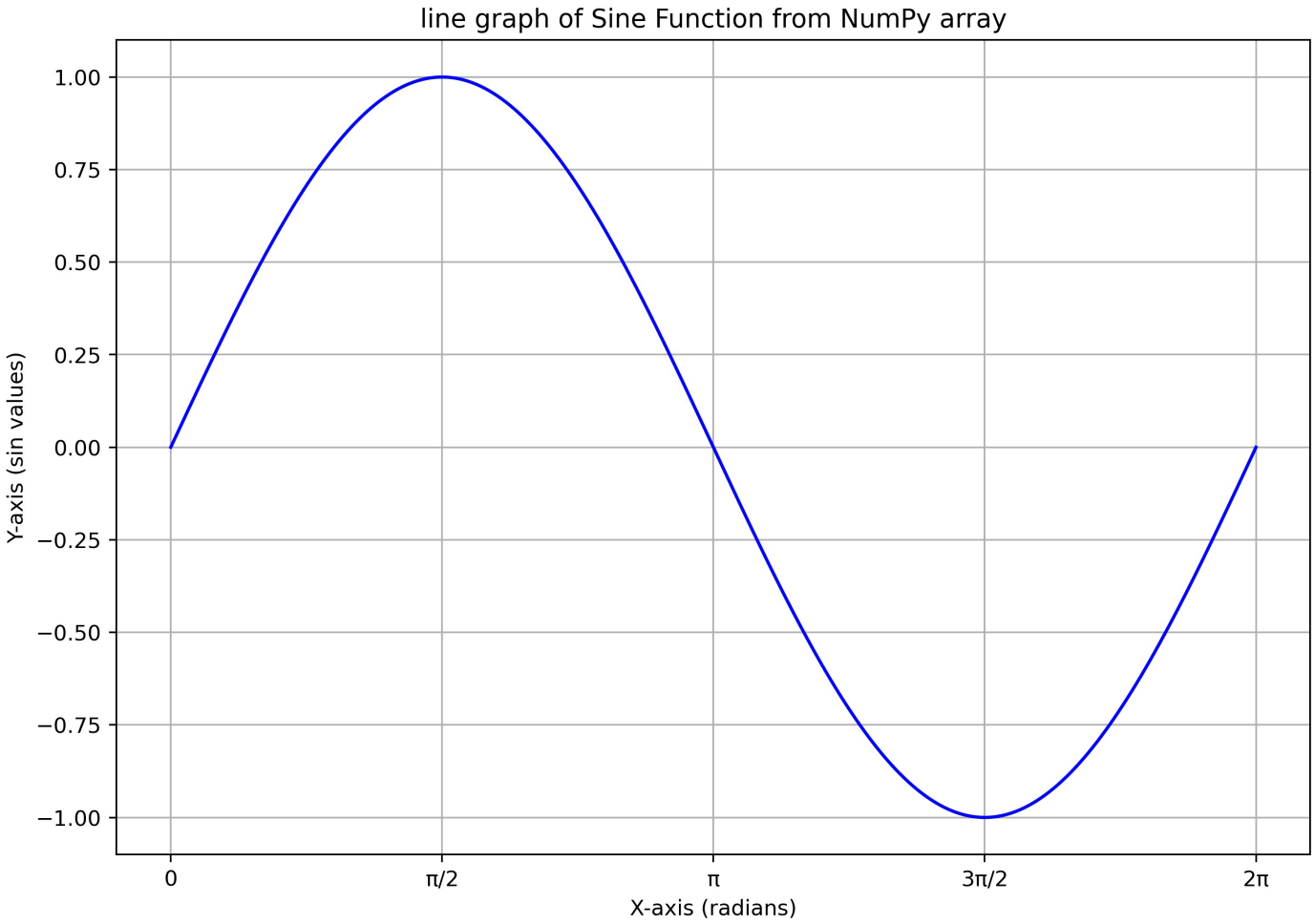
Program

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

x = np.linspace(0, 2 * np.pi, 500)
y = np.sin(x)

plt.plot(x, y, color='blue')
plt.xlabel('X-axis (radians)')
plt.ylabel('Y-axis (sin values)')
plt.title('line graph of Sine Function from NumPy array')
plt.grid()
plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi],
           ['0', 'π/2', 'π', '3π/2', '2π'])
plt.show()
```

Result

Calculated average, variance, and standard deviation.

Output-4.7


line graph of Sine Function from NumPy array
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y-axis (sin values)
0
π/2
π
3π/2
2π
X-axis (radians)

Program: 4.8 Date: 23-09-2024

Convert NumPy Array to CSV

Aim: To plot line graph from NumPy array.

Algorithm

1. Start
2. Import the numpy library as `np`.
3. Define a 2D NumPy array `data`.
4. Save the array to a CSV file:
   - Use `np.savetxt('output.csv', data, delimiter=',', fmt='%d')` to save the array as "output.csv" with comma-separated values and integer formatting.
5. Print a success message:
   - Print "CSV file created successfully!".
6. End

Program

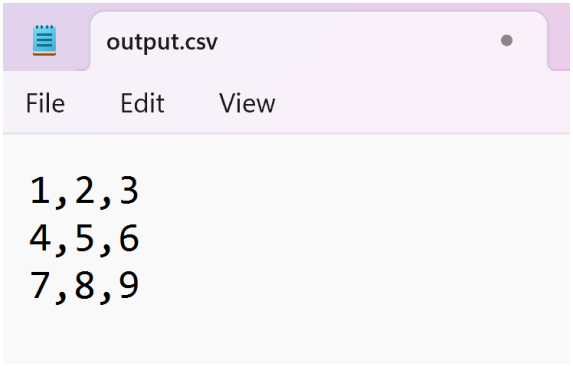
```
import numpy as np
data = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])

np.savetxt('output.csv', data, delimiter=',', fmt='%d')
print("CSV file created successfully!")
```

Result

Converted the NumPy array into a CSV file.

Output-4.8

CSV file created successfully!



Program: 5.1 Date: 30-09-2024

Count Words, Sentences, and Characters in File

Aim: To write a program to count the number of words, sentences, upper case letters, lowercase letters and special symbols in a text stored in file.

Algorithm

1. Start
2. Import the `string` module.
3. Define a function `text_analysis(file_path)` that takes the path of a text file as input.
4. Open the specified file in read mode using `with open(file_path, 'r') as file:`.
5. Read the content of the file and store it in the variable `text`.
6. Count the number of words with `word_count = len(text.split())`.
7. Count the number of sentences with `sentence_count = text.count('.') + text.count('!') + text.count('?')`.
8. Count uppercase letters with `uppercase_count = sum(1 for char in text if char.isupper())`.
9. Count lowercase letters with `lowercase_count = sum(1 for char in text if char.islower())`.
10. Count special symbols with `special_symbols_count = sum(1 for char in text if char in string.punctuation)`.
11. Print the counts of words, sentences, uppercase letters, lowercase letters, and special symbols.
12. Set `file_path = 'sampletext.txt'` to specify the file to analyze.
13. Call the `text_analysis(file_path)` function to execute the analysis.
14. End

Program

```
import string
def text_analysis(file_path):
    with open(file_path, 'r') as file:
        text = file.read()

    word_count = len(text.split())
    sentence_count = text.count('.') + text.count('!') + text.count('?')
    uppercase_count = sum(1 for char in text if char.isupper())
    lowercase_count = sum(1 for char in text if char.islower())
    special_symbols_count = sum(1 for char in text if char in string.punctuation)

    print(f"Words: {word_count}")
    print(f"Sentences: {sentence_count}")
    print(f"Uppercase Letters: {uppercase_count}")
    print(f"Lowercase Letters: {lowercase_count}")
    print(f"Special Symbols: {special_symbols_count}")

file_path = 'sampletext.txt'
text_analysis(file_path)
```

Result

Converted the NumPy array into a CSV file.

Output-5.1

Words: 44
Sentences: 3
Uppercase Letters: 6
Lowercase Letters: 219
Special Symbols: 6

<- Sampletext.txt ->

Launched in 1977, Voyager 1 is the farthest human-made object from Earth. It has provided valuable data about the outer planets and is now in interstellar space. Voyager 1 carries a golden record containing sounds and images of Earth, intended for potential extraterrestrial life.

Program: 5.2 Date: 30-09-2024

Histogram

Aim: To measure your classmate's height and draw the histogram.

Algorithm

1. Start
2. Import the `matplotlib.pyplot` module as `plt`.
3. Define a list named `heights` containing the height data of classmates in centimeters.
4. Create a histogram using `plt.hist(heights, bins=6, edgecolor='#000000', color='green')` to visualize the distribution of heights:
   - Specify `bins=6` to divide the data into 6 intervals.
   - Set `edgecolor='#000000'` to color the edges of the bars in black.
   - Set `color='green'` to fill the bars with green color.
5. Set the x-axis label to 'Heights (cm)' with font size 14 using `plt.xlabel()`.
6. Set the y-axis label to 'Frequency' with font size 14 using `plt.ylabel()`.
7. Set the title of the plot to 'Height Distribution of Classmates' with font size 16 using `plt.title()`.
8. Display the plot using `plt.show()`.
9. End

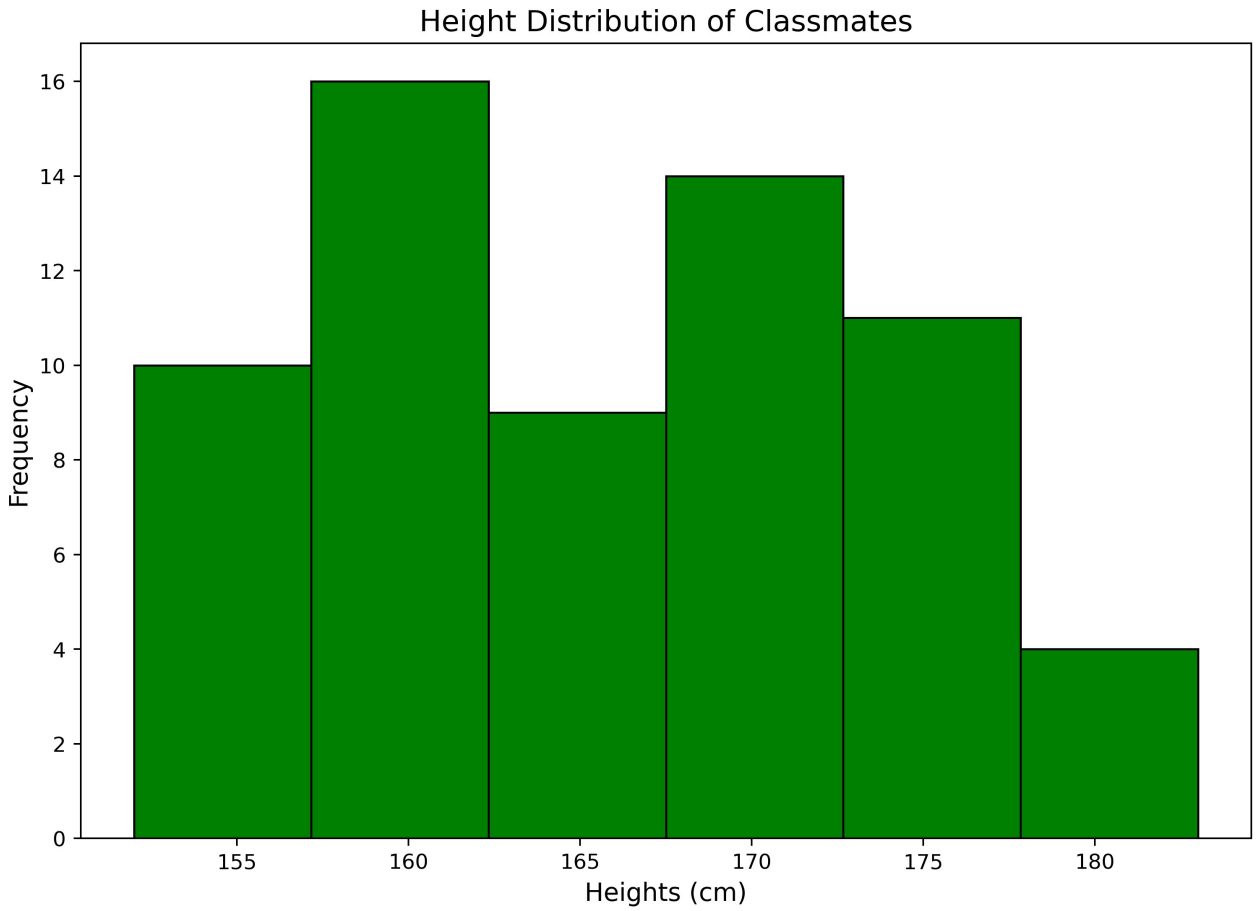
Program

```
import matplotlib.pyplot as plt
heights = [
   165, 170, 160, 168, 172, 166, 179, 174, 169, 177,
   175, 167, 173, 158, 171, 159, 176, 162, 165, 164,
   170, 178, 167, 175, 169, 171, 174, 172, 166, 177,
   173, 163, 165, 168, 174, 176, 170, 172, 169, 171,
   178, 155, 160, 152, 158, 162, 157, 154, 156, 159,
   155, 158, 160, 161, 157, 153, 162, 156, 155, 159,
   183, 158, 161, 160
]

plt.hist(heights, bins=6, edgecolor='#000000', color='green')
plt.xlabel('Heights (cm)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Height Distribution of Classmates', fontsize=14)
plt.show()
```

Result

Measured classmate's height and created a histogram.

Output-5.2


Height Distribution of Classmates
Frequency
Heights (cm)
0
2
4
6
8
10
12
14
16
155
160
165
170
175
180

Program: 5.3 Date: 30-09-2024

Mean, Median, and Variance

Aim: Table contains population and murder rates (in units of murders per 100,000 people per year) for different states. Compute the mean, median and variance for the population.

State	Population	Murder
Alabama	4,779,736	5.7
Alaska	710,231	5.6
Arizona	6,392,017	4.7
Arkansas	2,915,918	5.6
California	37,253,956	4.4
Colorado	5,029,196	2.8
Connecticut	3,574,097	2.4
Delaware	897,934	5.8

Algorithm

1. Start
2. Import the NumPy library as `np`.
3. Define a NumPy array named `population` containing the population data of various states.
4. Calculate the mean of the population data using `np.mean(population)` and store the result in `mean_population`.
5. Calculate the median of the population data using `np.median(population)` and store the result in `median_population`.
6. Calculate the variance of the population data using `np.var(population)` and store the result in `variance_population`.
7. Print the mean population using `print(f"Population Mean: {mean_population}")`.
8. Print the median population using `print(f"Population Median: {median_population}")`.
9. Print the variance of the population using `print(f"Population Variance: {variance_population}")`.
10. End

Program

```
import numpy as np
population = np.array([4779736,710231,6392017,2915918,37253956,5029196,3574097,897934])

mean_population = np.mean(population)
median_population = np.median(population)
variance_population = np.var(population)
```

Output-5.3

Population Mean: 7694135.625
Population Median: 4176916.5
Population Variance: 128230435522129.22

```
print(f"Population Mean: {mean_population}")
print(f"Population Median: {median_population}")
print(f"Population Variance: {variance_population}")
```

Result

Calculated the mean, median, and variance of the population.