1. Write a program “PrintFriendsNames” that takes 3 friends’ names as input arguments and prints out “Hi” to these friends in the reverse of the order given. It also adds “and “ before the last name.

For example:

Input Arguments : Mahesh, Suresh, Devesh

Output : “Hi Devesh, Suresh and Mahesh”

```
def PrintFriendsNames():
    # Get names from user input
    name1 = input("Enter the first friend's name: ")
    name2 = input("Enter the second friend's name: ")
    name3 = input("Enter the third friend's name: ")

    # Print the greeting with names in reverse order
    print(f"Hi {name3}, {name2} and {name1}")

# Call the function to execute the program
PrintFriendsNames()
```

2. Write a program “PrintInitials” that takes initials as input and prints the initials using nine rows of asterisks like the one below.

```
def PrintInitials(initials):
    patterns = {
        'A': ["  **  ","  *  *  ","*    *","******","*    *","*    *","*    *","*    *","*    *"],
        'B': ["***** ","*    *","*    *","***** ","*    *","*    *","*    *","***** ","     "],
        'C': ["  *** ","*    *","*     ","*     ","*     ","*     ","*    *","  *** ","     "],
        # Add patterns for other characters as required...
        '.': ["******    ","*    *   ","     ","     ","     ","     ","     ","     ","  **  "]
    }
```

```
    for row in range(9):
        for initial in initials:
            print(patterns.get(initial, patterns['.'])[row], end="  ")  # Using get with a default value for '.'
        print()  # Move to the next line for the next row

# Example usage:
if __name__ == "__main__":
    initials = input("Enter initials: ").upper()
    PrintInitials(initials)
```

3. Write a program CheckTheSeason if the month number is entered. In this program check needs to be done if the month number is correct data type and numbers are between 1 to 12. If not, the program should throw an error and should continue till it is not correctly entered.

Following table indicates Month numbers and seasons:

2 & 3 : Vasanta

4 & 5: Grishma / Summer

6 & 7: Monsoon / Rainy

8 & 9: Sharada

10 & 11: Hemanta

12 & 1 : Shishira / Winter

12 & 1 : Winter

```
def CheckTheSeason():
    seasons = {
        (2, 3): "Vasanta",
        (4, 5): "Grishma / Summer",
        (6, 7): "Monsoon / Rainy",
        (8, 9): "Sharada",
        (10, 11): "Hemanta",
```

```
    (12, 1): "Shishira / Winter"
  }

  while True:
    try:
      month = int(input("Enter the month number (1-12): "))
      if 1 <= month <= 12:
        # Determine the season based on the month
        for (start, end), season in seasons.items():
          if month in (start, end):
            print(f"The season for month {month} is {season}.")
            return
      else:
        print("Month number should be between 1 and 12. Try again.")

    except ValueError:
      print("Invalid input. Please enter a valid integer between 1 and 12.")

# Call the function to execute the program
CheckTheSeason()
```

4. Write a program Quadratic to find the roots of the equation a*x*x + b*x + c. Since the equation is x*x, hence there are 2 roots. The 2 roots of the equation can be found using a formula

delta = b*b - 4*a*c

Root 1 of x = (-b + sqrt(delta))/(2*a)

Root 2 of x = (-b - sqrt(delta))/(2*a)

Take a, b, and c as input values to find the roots of x.

```
import math
```

```
def Quadratic():
    # Taking input for coefficients a, b, and c
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))

    # Calculate the discriminant (delta)
    delta = b**2 - 4*a*c

    # Check if roots are real or complex
    if delta > 0:
        root1 = (-b + math.sqrt(delta)) / (2*a)
        root2 = (-b - math.sqrt(delta)) / (2*a)
        print(f"The roots are real and distinct: {root1} and {root2}")
    elif delta == 0:
        root = -b / (2*a)
        print(f"The roots are real and equal: {root}")
    else:
        real_part = -b / (2*a)
        imaginary_part = math.sqrt(abs(delta)) / (2*a)
        print(f"The roots are complex: {real_part} + {imaginary_part}i and {real_part} -
{imaginary_part}i")

# Call the function to execute the program
Quadratic()
```

5. Write a TemperatureConversion program, given the temperature in Fahrenheit as input outputs the temperature in Celsius or vice versa using the formula

Celsius to Fahrenheit: (°C × 9/5) + 32 = °F

Fahrenheit to Celsius: (°F – 32) x 5/9 = °C.

```
def convert_to_celsius(fahrenheit):
    """Converts Fahrenheit to Celsius."""
    return (fahrenheit - 32) * 5/9

def convert_to_fahrenheit(celsius):
    """Converts Celsius to Fahrenheit."""
    return (celsius * 9/5) + 32

def TemperatureConversion():
    while True:
        choice = input("Choose an option:\n1. Convert Fahrenheit to Celsius\n2. Convert Celsius to Fahrenheit\n3. Exit\nEnter your choice (1/2/3): ")

        if choice == '1':
            fahrenheit = float(input("Enter temperature in Fahrenheit: "))
            celsius = convert_to_celsius(fahrenheit)
            print(f"{fahrenheit}°F is {celsius:.2f}°C.\n")
        elif choice == '2':
            celsius = float(input("Enter temperature in Celsius: "))
            fahrenheit = convert_to_fahrenheit(celsius)
            print(f"{celsius}°C is {fahrenheit:.2f}°F.\n")
        elif choice == '3':
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please choose a valid option.\n")

# Call the function to execute the program
if __name__ == "__main__":
    TemperatureConversion()
```

6. Write a program which converts each character of an input string as an ASCII value and stores each of these numbers in an array. Print that array

```
def ConvertToASCIIArray(input_string):
    ascii_array = []  # Initialize an empty list to store ASCII values

    for char in input_string:
        ascii_value = ord(char)  # Convert the character to its ASCII value
        ascii_array.append(ascii_value)  # Add the ASCII value to the list

    return ascii_array

# Get input from the user
input_string = input("Enter a string: ")

# Convert each character to its ASCII value and store in an array
ascii_values = ConvertToASCIIArray(input_string)

# Print the array of ASCII values
print("ASCII values of the characters in the input string:")
print(ascii_values)
```

Logical Programming:

1. **Write a program that takes a range of numbers as input and outputs the Prime Numbers in that range.**

```
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
```

```
            return False
    return True

# Get input for the range
start_num = int(input("Enter the start number: "))
end_num = int(input("Enter the end number: "))

# Find and print prime numbers in the range
# print(f"Prime numbers between {start_num} and {end_num} are:")
for num in range(start_num, end_num + 1):
    if is_prime(num):
        print(num, end=" ")
```

2. Write a program PowerOf2 that takes a command-line argument n and prints a table of the powers of 2 that are less than or equal to 2^n.

```
import sys

def PowerOf2(n):
    """Prints powers of 2 up to 2^n."""
    for i in range(n + 1):
        print(f"2^{i} = {2 ** i}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python PowerOf2.py <value_of_n>")
    else:
        try:
            n = int(sys.argv[1])
            PowerOf2(n)
        except ValueError:
```

```
        print("Please provide a valid integer value for n.")
```

1. Save the above code in a file named `PowerOf2.py 5`.
2. Run the program from the command line as follows:

Copy code

3. Write two programs Sin and Cos that compute sin x and cos x using the Taylor series expansions as shown below...

Note - Convert angle x to an angle between -2 PI and 2 PI using the following logic

x = x % (2 * Math.PI);

```
import math

def compute_sin(x):
    """Compute sin(x) using Taylor series expansion."""
    x = x % (2 * math.pi)
    result = x
    sign = -1
    for n in range(3, 21, 2):  # Consider only odd powers up to n = 19
        result += (sign * (x ** n)) / math.factorial(n)
        sign *= -1
    return result

if __name__ == "__main__":
    x = float(input("Enter the angle x in radians: "))
    print(f"sin({x}) = {compute_sin(x)}")
```

```
import math

def compute_cos(x):
    """Compute cos(x) using Taylor series expansion."""
```

```
    x = x % (2 * math.pi)
    result = 1
    sign = -1
    for n in range(2, 21, 2):  # Consider only even powers up to n = 20
        result += (sign * (x ** n)) / math.factorial(n)
        sign *= -1
    return result

if __name__ == "__main__":
    x = float(input("Enter the angle x in radians: "))
    print(f"cos({x}) = {compute_cos(x)}")
```

4. Write a program to find common elements between two arrays (string values). Save the common values in another new array.

```
def find_common_elements(arr1, arr2):
    """Find common elements between two arrays."""
    common_elements = []

    for element in arr1:
        if element in arr2 and element not in common_elements:
            common_elements.append(element)

    return common_elements

if __name__ == "__main__":
    # Input arrays (lists of strings)
    arr1 = input("Enter elements of the first array (comma-separated): ").split(',')
    arr2 = input("Enter elements of the second array (comma-separated): ").split(',')

    # Find common elements
```

```
    common_values = find_common_elements(arr1, arr2)

    # Display common elements
    if common_values:
        print(f"Common elements: {', '.join(common_values)}")
    else:
        print("No common elements found.")
```

5. Write a program to find the second smallest element in an integer array.

```
def find_second_smallest(arr):
    if len(arr) < 2:
        return "Array must have at least two elements."

    arr.sort()  # Sort the array in ascending order
    return arr[1]  # Return the second element (index 1) after sorting

if __name__ == "__main__":
    arr = list(map(int, input("Enter elements of the array (space-separated): ").split()))
    result = find_second_smallest(arr)
    print(f"The second smallest element in the array is: {result}")
```

6. Write a method to check if two strings are equal or not. If the second string's size is more than the first it returns -1. Else it compares character by character. If equals it returns 0. Else it returns the position wrt the first character where the difference is found.

```
def compare_strings(str1, str2):
    """Compare two strings character by character."""

    # Check if the size of str2 is greater than str1
```

```
    if len(str2) > len(str1):
        return -1

    # Compare each character of str1 and str2
    for i in range(len(str1)):
        if i >= len(str2):  # If str2 is shorter than str1
            return i
        if str1[i] != str2[i]:
            return i

    # If all characters match and str2 is shorter or equal to str1
    if len(str2) == len(str1):
        return 0

    # If str1 and str2 are equal up to len(str2) characters, but str2 is shorter
    return len(str2)

# Example usage:
str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")

result = compare_strings(str1, str2)

if result == -1:
    print("The size of the second string is greater than the first string.")
elif result == 0:
    print("Both strings are equal.")
else:
    print(f"The first difference between the strings is at position {result}.")
```

7. Write a program SumOfTwoDice that prints the sum of two random integers

between 1 and 6 (such as you might get when rolling dice).

```
import random

def roll_dice():
    """Simulate rolling a dice and return the result."""
    return random.randint(1, 6)

if __name__ == "__main__":
    # Roll two dice
    dice1 = roll_dice()
    dice2 = roll_dice()

    # Calculate the sum
    total_sum = dice1 + dice2

    # Print the result
    print(f"The sum of the two dice rolls is: {total_sum}")
```

oops

1). Write a program to compute the area of a rectangle by creating a class named 'Rectangle' having two methods. First method named as 'setDim' takes length and breadth of the rectangle as parameters and the second method named as 'getArea' returns the area of the rectangle. Length and breadth of rectangle are entered through the keyboard.

Plan creating 2 rectangle objects, provide dimensions of them and then check which one has the higher area.

```
class Rectangle:
    def __init__(self):
        self.length = 0
```

```
        self.breadth = 0

    def setDim(self, length, breadth):
        """Set the dimensions of the rectangle."""
        self.length = length
        self.breadth = breadth

    def getArea(self):
        """Compute and return the area of the rectangle."""
        return self.length * self.breadth


if __name__ == "__main__":
    # Create two rectangle objects
    rect1 = Rectangle()
    rect2 = Rectangle()

    # Set dimensions for the first rectangle
    length1 = float(input("Enter length for Rectangle 1: "))
    breadth1 = float(input("Enter breadth for Rectangle 1: "))
    rect1.setDim(length1, breadth1)

    # Set dimensions for the second rectangle
    length2 = float(input("Enter length for Rectangle 2: "))
    breadth2 = float(input("Enter breadth for Rectangle 2: "))
    rect2.setDim(length2, breadth2)

    # Compute areas of the rectangles
    area1 = rect1.getArea()
    area2 = rect2.getArea()
```

```
# Determine and print which rectangle has a higher area
if area1 > area2:
    print(f"Rectangle 1 has a larger area: {area1}")
elif area2 > area1:
    print(f"Rectangle 2 has a larger area: {area2}")
else:
    print("Both rectangles have the same area.")
```

2. Stock Report

a. Desc -> Write a program to read in Stock Names, Number of Share, Share Price. Print a Stock Report with total value of each Stock and the total value of Stock.

b. I/P -> N number of Stocks, for Each Stock Read In the Share Name, Number of Share, and Share Price

c. Logic -> Calculate the value of each stock and the total value

d. O/P -> Print the Stock Report.

e. Hint -> Create Stock and Stock Portfolio Class holding the list of Stocks read from the input file. Have functions in the Class to calculate the value of each stock and the value of total stocks

```
class Stock:
    def __init__(self, name, num_shares, price):
        self.name = name
        self.num_shares = num_shares
        self.price = price

    def calculate_value(self):
        """Calculate the value of the stock."""
        return self.num_shares * self.price


class StockPortfolio:
```

```
    def __init__(self):
        self.stocks = []

    def add_stock(self, stock):
        """Add a stock to the portfolio."""
        self.stocks.append(stock)

    def calculate_total_value(self):
        """Calculate the total value of all stocks in the portfolio."""
        return sum(stock.calculate_value() for stock in self.stocks)


if __name__ == "__main__":
    portfolio = StockPortfolio()

    n = int(input("Enter the number of stocks: "))

    for _ in range(n):
        name = input("Enter the stock name: ")
        num_shares = int(input("Enter the number of shares: "))
        price = float(input("Enter the share price: "))

        stock = Stock(name, num_shares, price)
        portfolio.add_stock(stock)

    print("\nStock Report")
    print("============")
    for stock in portfolio.stocks:
        print(f"Stock Name: {stock.name}")
        print(f"Number of Shares: {stock.num_shares}")
        print(f"Share Price: ${stock.price:.2f}")
```

```
        print(f"Total Value: ${stock.calculate_value():.2f}\n")

    print(f"Total Portfolio Value: ${portfolio.calculate_total_value():.2f}")
```

3. Inventory Management Problem

a. Desc -> Extend the above program to Create InventoryManager to manage the Inventory. The Inventory Manager will use InventoryFactory to create Inventory Object from JSON. The InventoryManager will call each Inventory Object in its list to calculate the Inventory Price and then call the Inventory Object to return the JSON String. The main program will be with InventoryManager

b. I/P -> read in JSON File

c. Logic -> Get JSON Object in Java or NSDictionary in iOS. Create Inventory Object from JSON. Calculate the value for every Inventory.

d. O/P -> Create the JSON from Inventory Object and output the JSON String.

```
import json

class Inventory:
    def __init__(self, name, price_per_unit, quantity):
        self.name = name
        self.price_per_unit = price_per_unit
        self.quantity = quantity

    def calculate_value(self):
        """Calculate the value of the inventory item."""
        return self.price_per_unit * self.quantity

    def to_json(self):
        """Convert the inventory item to a JSON string."""
        return json.dumps(self.__dict__)
```

```
class InventoryFactory:
    @staticmethod
    def create_inventory_from_json(json_data):
        """Create an Inventory object from JSON data."""
        name = json_data['name']
        price_per_unit = json_data['price_per_unit']
        quantity = json_data['quantity']
        return Inventory(name, price_per_unit, quantity)


class InventoryManager:
    def __init__(self):
        self.inventory_list = []

    def add_inventory(self, inventory):
        """Add an inventory item to the manager."""
        self.inventory_list.append(inventory)

    def calculate_total_value(self):
        """Calculate the total value of all inventory items."""
        return sum(item.calculate_value() for item in self.inventory_list)

    def to_json(self):
        """Convert the inventory manager to a JSON string."""
        inventory_data = [item.__dict__ for item in self.inventory_list]
        return json.dumps(inventory_data)


if __name__ == "__main__":
    # Read JSON data from a file (assuming data.json contains the inventory details)
```

```
    with open('data.json', 'r') as file:
        data = json.load(file)

    # Create Inventory objects from JSON data
    manager = InventoryManager()
    for item_data in data:
        inventory_item = InventoryFactory.create_inventory_from_json(item_data)
        manager.add_inventory(inventory_item)

    # Calculate total inventory value and convert to JSON
    total_value = manager.calculate_total_value()
    json_output = manager.to_json()

    # Output the results
    print(f"Total Inventory Value: ${total_value:.2f}")
    print("\nInventory Details (JSON Format):")
    print(json_output)
```

Data Structures and Algorithms

1. Write a program to where it takes 10 integer numbers from a user, puts them in an array and then through Bubble sort, sorts this array. Input taking and forming an array should be one method and sorting should be other method.

```
def take_input_and_form_array():
    """Take input from the user and form an array."""
    arr = []
    for _ in range(10):
        num = int(input("Enter an integer number: "))
        arr.append(num)
    return arr
```

```
def bubble_sort(arr):
    """Sort the array using Bubble Sort."""
    n = len(arr)
    for i in range(n - 1):
        # Flag to check if any swapping is done in the current pass
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Swap arr[j] and arr[j+1]
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # If no two elements were swapped in the inner loop, the array is already sorted
        if not swapped:
            break

def print_array(arr):
    """Print the array."""
    print("Sorted Array:", arr)

if __name__ == "__main__":
    # Take input from the user and form an array
    array = take_input_and_form_array()

    # Sort the array using Bubble Sort
    bubble_sort(array)

    # Print the sorted array
    print_array(array)
```

2. Write a program to sort data using two algorithms and compare the performance.

a. Generate 100,000 random floating point numbers 1.0 to 100.0 Keep these

numbers in an array.

b. Sort this array using Selection Sort and

c. Sort with Quick Sort.

d. Compute the time of execution for both the algorithms in mili-seconds.

e. Check which one is faster and by what percentage?

f. Number generation, Sorting algos and performance comparisons should be different methods.

```
import random
import time

def generate_numbers(size):
    """Generate random floating-point numbers between 1.0 and 100.0."""
    return [random.uniform(1.0, 100.0) for _ in range(size)]

def selection_sort(arr):
    """Implement Selection Sort."""
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def quick_sort(arr):
    """Implement Quick Sort."""
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
```

```
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

if __name__ == "__main__":
    size = 100000
    numbers = generate_numbers(size)

    # Measure the time for Selection Sort
    start_time = time.time()
    selection_sort(numbers.copy())
    selection_time = (time.time() - start_time) * 1000  # Convert to milliseconds

    # Measure the time for Quick Sort
    start_time = time.time()
    quick_sort(numbers.copy())
    quick_time = (time.time() - start_time) * 1000  # Convert to milliseconds

    print(f"Time taken by Selection Sort: {selection_time:.2f} ms")
    print(f"Time taken by Quick Sort: {quick_time:.2f} ms")

    # Calculate and print the performance comparison
    if selection_time < quick_time:
        faster_algo = "Selection Sort"
        percentage = ((quick_time - selection_time) / selection_time) * 100
    else:
        faster_algo = "Quick Sort"
        percentage = ((selection_time - quick_time) / quick_time) * 100

    print(f"\n{faster_algo} is faster by {percentage:.2f}%")
```

3. Given a sorted array of n integers and a target value, determine if the target exists in the array using the binary search algorithm. If the target exists in the array, print the index of it.

eg.

Input array : 2, 3, 5, 7, 9

Target : 7

Output : Element found at Index 3

Input array : 1, 4, 5, 8, 9

Target : 7

Output : Element Not Found

```
def binary_search(arr, target):
    """Search for the target in the array using binary search."""
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

if __name__ == "__main__":
    # Example arrays
    array1 = [2, 3, 5, 7, 9]
```

```
array2 = [1, 4, 5, 8, 9]

# Example targets
target1 = 7
target2 = 7

# Perform binary search and print the result
result1 = binary_search(array1, target1)
result2 = binary_search(array2, target2)

if result1 != -1:
    print(f"Element found at Index {result1}")
else:
    print("Element Not Found")

if result2 != -1:
    print(f"Element found at Index {result2}")
else:
    print("Element Not Found")
```

4. List Handling:

a. Desc -> Read the Text from a file, split it into words and arrange it as Linked List. Take a user input to search a Word in the List. If the Word is not found then add it to the list, and if it found then remove the word from the List. In the end save the list into a file

b. I/P -> Read from file the list of Words and take user input to search a Text

c. Logic -> Create a Unordered Linked List. The Basic Building Block is the Node Object. Each node object must hold at least two pieces of information. One ref to the data field and second the ref to the next node object.

d. O/P -> The List of Words to a File.

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def add_word(self, word):
        """Add a word to the linked list."""
        new_node = Node(word)
        if not self.head:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def remove_word(self, word):
        """Remove a word from the linked list."""
        if not self.head:
            return
        if self.head.data == word:
            self.head = self.head.next
            return
        prev = None
        current = self.head
        while current and current.data != word:
```

```
            prev = current
            current = current.next
        if current:
            prev.next = current.next

    def search_word(self, word):
        """Search for a word in the linked list."""
        current = self.head
        while current:
            if current.data == word:
                return True
            current = current.next
        return False

    def save_to_file(self, filename):
        """Save the list of words to a file."""
        with open(filename, 'w') as file:
            current = self.head
            while current:
                file.write(current.data + '\n')
                current = current.next

if __name__ == "__main__":
    # Read words from a file and create a linked list
    filename = "words.txt"
    word_list = LinkedList()
    with open(filename, 'r') as file:
        for line in file:
            word_list.add_word(line.strip())

    # Take user input to search a word
```

```
search_word = input("Enter a word to search: ")

# Search for the word and modify the linked list
if word_list.search_word(search_word):
    word_list.remove_word(search_word)
    print(f"'{search_word}' found and removed from the list.")
else:
    word_list.add_word(search_word)
    print(f"'{search_word}' added to the list.")

# Save the modified list back to the file
word_list.save_to_file(filename)
print(f"The list of words has been saved to {filename}.")
```