Subject : Python Programming Lab

Code : LC–CSE-215

Department of Computer Science Engineering
RPS College of Engineering and Technology
Mahendergarh, Haryana - 123029
## LIST OF PROGRAMS

<table>
<thead>
<tr>
<th>PROGRAM NO.</th>
<th>NAME OF PROGRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WAP to compute the GCD of two numbers.</td>
</tr>
<tr>
<td>2</td>
<td>WAP to find the square root of a number using Newton's method</td>
</tr>
<tr>
<td>3</td>
<td>WAP to find Exponentiation (Power of a number)</td>
</tr>
<tr>
<td>4</td>
<td>WAP to find the maximum of a list of numbers</td>
</tr>
<tr>
<td>5</td>
<td>WAP to implement Linear search method</td>
</tr>
<tr>
<td>6</td>
<td>WAP to implement Binary search method</td>
</tr>
<tr>
<td>7</td>
<td>WAP to sort numbers using Selection sort method</td>
</tr>
<tr>
<td>8</td>
<td>WAP to sort numbers using Insertion sort method</td>
</tr>
<tr>
<td>9</td>
<td>WAP to sort numbers using Merge sort method</td>
</tr>
<tr>
<td>10</td>
<td>WAP to find first n prime numbers</td>
</tr>
<tr>
<td>11</td>
<td>WAP to Multiply matrices.</td>
</tr>
<tr>
<td>12</td>
<td>WAP that take command line arguments for word count in python</td>
</tr>
<tr>
<td>13</td>
<td>WAP to find the most frequent words in a text read from a file</td>
</tr>
<tr>
<td>14</td>
<td>WAP to simulate elliptical orbits in Pygame</td>
</tr>
<tr>
<td>15</td>
<td>WAP to simulate bouncing ball using Pygame</td>
</tr>
</tbody>
</table>

## OBJECTIVES

- To write, test, and debug simple Python programs.
- To implement Python programs with conditionals and loops.
- Use functions for structuring Python programs.
- Represent compound data using Python lists, tuples, and dictionaries.
- Read and write data from/to files in Python.
PROGRAM 1: WAP to compute GCD of two numbers

CODE

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Taking input from the user
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

# Computing GCD
result = gcd(num1, num2)
print("The GCD of", num1, "and", num2, "is", result)
PROGRAM 2: WAP to find the square root of a number using Newton's method

CODE

def square_root(n):
    # Initial guess for the square root
    x = n / 2
    # Define the precision
    epsilon = 0.0001
    # Loop until the difference between x and n/x is less than epsilon
    while True:
        # Newton's method iteration
        root = (x + n / x) / 2
        # If the difference between the current root and previous root is less than epsilon, break
        if abs(root - x) < epsilon:
            break
        x = root
    return root

# Taking input from the user
number = float(input("Enter a number: "))
# Computing square root using Newton's method
result = square_root(number)
print("Square root of", number, "is approximately", result)
PROGRAM 3 : WAP to find Exponentiation (Power of a number)

CODE

def power(base, exponent):
    result = 1
    for _ in range(exponent):
        result *= base
    return result

    # Taking input from the user
base = float(input("Enter the base: "))
exponent = int(input("Enter the exponent: "))
    # Calculating exponentiation using the custom function
result = power(base, exponent)
print(base, "raised to the power of", exponent, "is", result)
PROGRAM 4 : WAP to find the maximum of a list of numbers

CODE

# List of numbers
numbers = [10, 25, 7, 42, 15, 3]

# Initializing maximum with the first element of the list
maximum = numbers[0]

# Iterate through the list to find the maximum
for num in numbers:
    if num > maximum:
        maximum = num

print("The maximum number in the list is:", maximum)
**PROGRAM 5**: WAP to implement Linear search method

**CODE**

def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

    # Example usage:
arr = [5, 10, 3, 8, 12, 6]
target = 8
result = linear_search(arr, target)
if result != -1:
    print("Element", target, "found at index", result)
else:
    print("Element", target, "not found in the list.")
PROGRAM 6 : WAP to implement Binary search method

CODE

def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        mid_val = arr[mid]
        if mid_val == target:
            return mid
        elif mid_val < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

# Example usage:
arr = [3, 6, 8, 10, 12, 15]
target = 10
result = binary_search(arr, target)
if result != -1:
    print("Element", target, "found at index", result)
else:
    print("Element", target, "not found in the list.")
**PROGRAM 7**: WAP to sort numbers using Selection sort method

**CODE**

```python
def selection_sort(arr):
    # Traverse through all array elements
    for i in range(len(arr)):
        # Find the minimum element in the remaining unsorted array
        min_index = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_index]:
                min_index = j
                # Swap the found minimum element with the first element
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

# Example usage:
numbers = [64, 25, 12, 22, 11]
sorted_numbers = selection_sort(numbers)
print("Sorted numbers using Selection Sort:", sorted_numbers)
```

```
PROGRAM 8 : WAP to sort numbers using Insertion sort method

CODE

def insertion_sort(arr):
    # Traverse through all array elements starting from the second element
    for i in range(1, len(arr)):
        key = arr[i]  # Current element to be compared
        # Move elements of arr[0..i-1], that are greater than key,
        # to one position ahead of their current position
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

# Example usage:
numbers = [64, 25, 12, 22, 11]
sorted_numbers = insertion_sort(numbers)
print("Sorted numbers using Insertion Sort:", sorted_numbers)
PROGRAM 9: WAP to sort numbers using Merge sort method

CODE
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        # Recursively sort the left and right halves
        merge_sort(left_half)
        merge_sort(right_half)

        # Merge the sorted halves
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
    return arr

# Example usage:
numbers = [64, 25, 12, 22, 11]
sorted_numbers = merge_sort(numbers)
print("Sorted numbers using Merge Sort:", sorted_numbers)
PROGRAM 10: WAP to find first n prime numbers

CODE

def is_prime(num):
    if num <= 1:
        return False
    elif num == 2:
        return True
    elif num % 2 == 0:
        return False
    else:
        # Check divisibility from 3 to the square root of the number
        for i in range(3, int(num**0.5) + 1, 2):
            if num % i == 0:
                return False
        return True

def first_n_primes(n):
    primes = []
    num = 2
    while len(primes) < n:
        if is_prime(num):
            primes.append(num)
        num += 1
    return primes

# Example usage:
n = int(input("Enter the value of n: "))
prime_numbers = first_n_primes(n)
print("The first", n, "prime numbers are:", prime_numbers)
PROGRAM 11 : WAP to Multiply matrices.

CODE

def multiply_matrices(matrix1, matrix2):
    rows1 = len(matrix1)
    cols1 = len(matrix1[0])
    rows2 = len(matrix2)
    cols2 = len(matrix2[0])
    if cols1 != rows2:
        print("Error: Number of columns in the first matrix must be equal to the number of rows in the second matrix.")
        return None
    result = [[0 for _ in range(cols2)] for _ in range(rows1)]
    for i in range(rows1):
        for j in range(cols2):
            for k in range(cols1):
                result[i][j] += matrix1[i][k] * matrix2[k][j]
    return result

# Example usage:
matrix1 = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
matrix2 = [[9, 8, 7],
          [6, 5, 4],
          [3, 2, 1]]
result_matrix = multiply_matrices(matrix1, matrix2)
if result_matrix:
    print("Result of matrix multiplication:")
    for row in result_matrix:
        print(row)
**PROGRAM 12**: WAP that take command line arguments for word count in python

**CODE**

```python
import sys

def word_count(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
            words = content.split()
            return len(words)
    except FileNotFoundError:
        print("Error: File not found.")
        return -1

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python word_count.py <filename>")
    else:
        filename = sys.argv[1]
        count = word_count(filename)
        if count != -1:
            print("Number of words in", filename + ":", count)
```

- Save the code in a file named `word_count.py`.
- Run the script from the command line, passing the file name as an argument `python word_count.py input.txt`
- Replace `input.txt` with the name of the file you want to count words in.
- This script will read the contents of the file, split it into words, and return the count of words in the file
PROGRAM 13: WAP to find the most frequent words in a text read from a file

CODE
import sys
import re
from collections import Counter
def most_frequent_words(filename, num_words=10):
    try:
        with open(filename, 'r') as file:
            content = file.read()
            words = re.findall(r'\b\w+\b', content.lower())  # Tokenize words and convert to lowercase
            word_counts = Counter(words)
            return word_counts.most_common(num_words)
    except FileNotFoundError:
        print("Error: File not found.")
        return None
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python most_frequent_words.py <filename>")
    else:
        filename = sys.argv[1]
        frequent_words = most_frequent_words(filename)
        if frequent_words:
            print("Most frequent words in", filename + ":")
            for word, count in frequent_words:
                print(word, ",", count)

• Save this code in a file named **most_frequent_words.py**.
• Run the script from the command line, passing the file name as an argument: `python most_frequent_words.py input.txt`
• Replace **input.txt** with the name of the file you want to analyze.
• This script will read the contents of the file, tokenize it into words (ignoring punctuation and converting to lowercase), count the frequency of each word, and then return the specified number of most frequent words along with their counts.
• Adjust the **num_words** parameter to get a different number of most frequent words.
PROGRAM 14: WAP to simulate elliptical orbits in Pygame

- Creating a simulation of elliptical orbits in Pygame involves several steps, including defining the gravitational forces between the objects, calculating their positions over time, and rendering them on the screen.

CODE
Install Pygame using pip install pygame

Python script to simulate elliptical orbits

```python
import pygame
import math

# Constants
G = 0.1  # Gravitational constant
FPS = 60
WIDTH, HEIGHT = 800, 600
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)

# Define the class for celestial bodies
class Body:
    def __init__(self, mass, position, velocity):
        self.mass = mass
        self.position = position
        self.velocity = velocity

    def update_velocity(self, bodies):
        for body in bodies:
            if body != self:
                dx = body.position[0] - self.position[0]
                distance = math.sqrt(dx**2 + dy**2)
                force_magnitude = G * self.mass * body.mass / distance**3
                self.velocity[0] += force_magnitude * dx
                self.velocity[1] += force_magnitude * dy

    def update_position(self):
        self.position[0] += self.velocity[0]

# Initialize Pygame
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Elliptical Orbits Simulation")
```
clock = pygame.time.Clock()

# Create celestial bodies
sun = Body(100000, [WIDTH // 2, HEIGHT // 2], [0, 0])
planet = Body(10, [WIDTH // 4, HEIGHT // 2], [0, -1])

running = True

while running:
    screen.fill(BLACK)

    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update celestial bodies
    planet.update_velocity([sun])
    planet.update_position()

    # Draw celestial bodies
    pygame.draw.circle(screen, WHITE, (int(sun.position[0]), int(sun.position[1])), 20)
    pygame.draw.circle(screen, WHITE, (int(planet.position[0]), int(planet.position[1])), 5)
    pygame.display.flip()
    clock.tick(FPS)

pygame.quit()
PROGRAM 15 : WAP to simulate bouncing ball using Pygame

CODE

```python
import pygame
import random

# Constants
WIDTH, HEIGHT = 800, 600
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
BALL_RADIUS = 20
BALL_COLOR = (255, 0, 0)

# Initialize Pygame
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Bouncing Ball")
clock = pygame.time.Clock()

# Ball properties
ball_pos = [WIDTH // 2, HEIGHT // 2]
ball_vel = [random.randint(-5, 5), random.randint(-5, 5)]

running = True
while running:
    screen.fill(BLACK)
    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # Update ball position
    ...# ball_pos[0] = max(min(ball_pos[0] + ball_vel[0], WIDTH - BALL_RADIUS), 0)
    ...# ball_pos[1] = max(min(ball_pos[1] + ball_vel[1], HEIGHT - BALL_RADIUS), 0)

    pygame.draw.circle(screen, BALL_COLOR, ball_pos, BALL_RADIUS)
    pygame.display.flip()

    clock.tick(60)
```

```
ball_pos[0] += ball_vel[0]

# Check for collisions with walls
if ball_pos[0] <= BALL_RADIUS or ball_pos[0] >= WIDTH - BALL_RADIUS:
    ball_vel[0] = -ball_vel[0]
if ball_pos[1] <= BALL_RADIUS or ball_pos[1] >= HEIGHT - BALL_RADIUS:

# Draw the ball
pygame.draw.circle(screen, BALL_COLOR, (int(ball_pos[0]), int(ball_pos[1])), BALL_RADIUS)
pygame.display.flip()

pygame.quit()