Project Report

on

Programming in Python

Submitted By:

Kalyan Sarmah(CS23BCAGN056)

BCA 4th SEM

School of Computing Sciences

The Assam Kaziranga University ,Jorhat, Assam

Table of Contents

Project-1

1. Write a program using python showing implementation of any arithmetic and quadratic operation.
Ans:- This program performs basic arithmetic operations like addition, subtraction, multiplication, division and also solves quadratic equations using the quadratic formula.

```
import math

# Perform selected Arithmetic Operation
def arithmetic_operations(a, b, operation):
    print("\nArithmetic Operation:")
    if operation == '1':
        print(f"Addition: {a} + {b} = {a + b}")
    elif operation == '2':
        print(f"Subtraction: {a} - {b} = {a - b}")
    elif operation == '3':
        print(f"Multiplication: {a} * {b} = {a * b}")
    elif operation == '4':
        if b != 0:
            print(f"Division: {a} / {b} = {a / b}")
        else:
            print("Division: Undefined (division by zero)")
    else:
        print("Invalid operation choice!")

# Quadratic Equation Solver
# Equation format: ax^2 + bx + c = 0
def solve_quadratic(a, b, c):
    print("\nSolving Quadratic Equation:")
    print(f"Equation: {a}x² + {b}x + {c} = 0")

    discriminant = b**2 - 4*a*c

    if discriminant > 0:
        root1 = (-b + math.sqrt(discriminant)) / (2*a)
        root2 = (-b - math.sqrt(discriminant)) / (2*a)
        print(f"Two real roots: {root1:.2f} and {root2:.2f}")
    elif discriminant == 0:
        root = -b / (2*a)
        print(f"One real root: {root:.2f}")
    else:
        real_part = -b / (2*a)
        imag_part = math.sqrt(-discriminant) / (2*a)
        print(f"Two complex roots: {real_part:.2f} + {imag_part:.2f}i and
{real_part:.2f} - {imag_part:.2f}i")

# Main Code

# Arithmetic operation input
print("Arithmetic Operations Menu:")
print("1. Addition")
print("2. Subtraction")
print("3. Multiplication")
print("4. Division")
choice = input("Choose an operation (1-4): ")

a1 = float(input("Enter first number (a): "))
b1 = float(input("Enter second number (b): "))
```

```python
arithmetic_operations(a1, b1, choice)

# Quadratic equation input
print("\nEnter coefficients for quadratic equation ax² + bx + c = 0:")
a2 = float(input("Enter coefficient a: "))
b2 = float(input("Enter coefficient b: "))
c2 = float(input("Enter coefficient c: "))

solve_quadratic(a2, b2, c2)
```

Output:-

```
Arithmetic Operations Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Division
Choose an operation (1-4):
1
Enter first number (a):
12
Enter second number (b):
15

Arithmetic Operation:
Addition: 12.0 + 15.0 = 27.0

Enter coefficients for quadratic equation ax² + bx + c = 0:
Enter coefficient a:
1
Enter coefficient b:
6
Enter coefficient c:
5

Solving Quadratic Equation:
Equation: 1.0x² + 6.0x + 5.0 = 0
Two real roots: -1.00 and -5.00


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Project-2

2.Write a Python program showing implementation of linear equation.
Ans:- Solves a system of two linear equations with two variables using **NumPy**. The program uses matrix representation and applies **numpy.linalg.solve** to find the values of x and y that satisfy both equations.

```
import numpy as np

# Linear Equations in Two Variables
# Equations: a1x + b1y = c1 and a2x + b2y = c2
def solve_two_variable_linear(a1, b1, c1, a2, b2, c2):
    print("\nSolving Linear Equations (Two Variables):")
    print(f"Equation 1: {a1}x + {b1}y = {c1}")
    print(f"Equation 2: {a2}x + {b2}y = {c2}")

    # Matrix representation: AX = B
    A = np.array([[a1, b1], [a2, b2]])
    B = np.array([c1, c2])

    # Check if determinant is non-zero
    det = np.linalg.det(A)

    if det != 0:
        solution = np.linalg.solve(A, B)
        x, y = solution
        print(f"Solution: x = {x:.2f}, y = {y:.2f}")
    else:
        print("No unique solution (Determinant is zero)")

# Main Program
print("Enter coefficients for the system of equations:")
print("Equation format: a1x + b1y = c1 and a2x + b2y = c2")

# User input
a1 = float(input("Enter a1: "))
b1 = float(input("Enter b1: "))
c1 = float(input("Enter c1: "))

a2 = float(input("Enter a2: "))
b2 = float(input("Enter b2: "))
c2 = float(input("Enter c2: "))

# Solve the system
solve_two_variable_linear(a1, b1, c1, a2, b2, c2)
```

Output:-

```
Enter coefficients for the system of equations:
Equation format: a1x + b1y = c1 and a2x + b2y = c2
Enter a1:
4
Enter b1:
2
Enter c1:
1
Enter a2:
3
Enter b2:
2
Enter c2:
1

Solving Linear Equations (Two Variables):
Equation 1: 4.0x + 2.0y = 1.0
Equation 2: 3.0x + 2.0y = 1.0
Solution: x = 0.00, y = 0.50


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

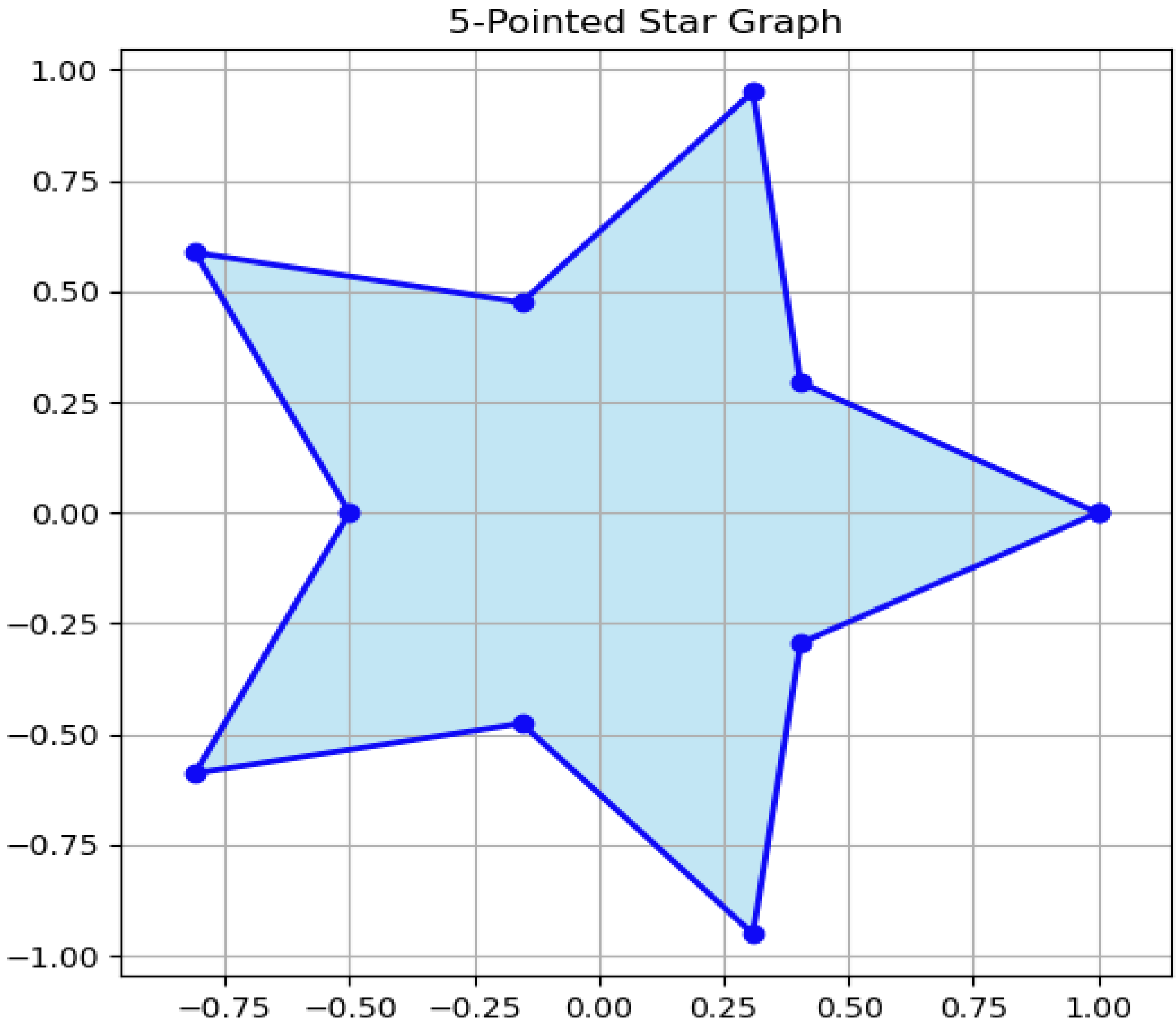
Project-3

3. Write a python program using any mathematical function or equation to give graphical representation like star graph.

Ans:- This Python script uses **matplotlib** to generate a star-shaped polar graph. It demonstrates the use of mathematical equations for plotting complex visual patterns. Ideal for learning how to represent equations graphically.

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

def draw_star(n_points=5, inner_radius=0.5, outer_radius=1):
    """
    Draw a star with n_points using polar coordinates.
    inner_radius: radius of inner vertices
    outer_radius: radius of outer vertices
    """

    print(f"Drawing a {n_points}-pointed star...")

    angles = np.linspace(0, 2 * np.pi, num=2 * n_points, endpoint=False)
    radii = np.empty(2 * n_points)

    # Alternate between outer and inner radius
    radii[::2] = outer_radius
    radii[1::2] = inner_radius

    # Convert polar to Cartesian coordinates
    x = radii * np.cos(angles)
    y = radii * np.sin(angles)

    # Close the star shape by repeating the first point
    x = np.append(x, x[0])
    y = np.append(y, y[0])

    # Plotting
    plt.figure(figsize=(6, 6))
    plt.plot(x, y, marker='o', color='blue', linestyle='-', linewidth=2)
    plt.fill(x, y, color='skyblue', alpha=0.5)
    plt.title(f"{n_points}-Pointed Star Graph")
    plt.axis('equal')
    plt.grid(True)
    plt.show()

# Run star shapes

draw_star(n_points=5)  # 5-point star
```

Output:-


5-Pointed Star Graph
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Project-4

4. Write a python program showing the implementation of a function.
Ans:- This Python program demonstrates the use of simple functions to perform basic tasks: addition, squaring a number, and checking if a number is even or odd.

```
# Function to add two numbers
def add(a, b):
    return a + b

# Function to find the square of a number
def square(n):
    return n * n

# Function to check if a number is even or odd
def is_even(n):
    return n % 2 == 0

# Main Program
print("Function Implementation Example:\n")

# Using add function with user input
x = int(input("Enter first number for addition: "))
y = int(input("Enter second number for addition: "))
print(f"Addition of {x} and {y} is: {add(x, y)}\n")

# Using square function with user input
num = int(input("Enter a number to find its square: "))
print(f"Square of {num} is: {square(num)}\n")

# Using is_even function with user input
check_num = int(input("Enter a number to check even or odd: "))
if is_even(check_num):
    print(f"{check_num} is Even")
else:
    print(f"{check_num} is Odd")
```

Output:-

```
Function Implementation Example:

Enter first number for addition:
10
Enter second number for addition:
2
Addition of 10 and 2 is: 12

Enter a number to find its square:
4
Square of 4 is: 16

Enter a number to check even or odd:
3
3 is Odd


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Project-5

5.Write a python program using tinker make any formatted application according to our ideas (Tetris,Snake,Card-block).

Ans:- A mini version of the Tetris game. Blocks fall from the top, and the player rotates/moves them using keyboard controls to fit them at the bottom. Full rows disappear, and the score increases. Built using `Tkinter`.

```
import tkinter as tk
import random

# Game constants
CELL_SIZE = 30
COLUMNS = 10
ROWS = 20
DELAY = 300  # milliseconds

# Shapes (tetrominoes)
SHAPES = {
    "I": [[1, 1, 1, 1]],
    "O": [[1, 1],
          [1, 1]],
    "T": [[0, 1, 0],
          [1, 1, 1]],
    "S": [[0, 1, 1],
          [1, 1, 0]],
    "Z": [[1, 1, 0],
          [0, 1, 1]],
    "J": [[1, 0, 0],
          [1, 1, 1]],
    "L": [[0, 0, 1],
          [1, 1, 1]]
}

COLORS = {
    "I": "cyan",
    "O": "yellow",
    "T": "purple",
    "S": "green",
    "Z": "red",
    "J": "blue",
    "L": "orange"
}

class Tetris:
    def __init__(self, root):
        self.root = root
        self.canvas = tk.Canvas(root, width=CELL_SIZE * COLUMNS, height=CELL_SIZE *
ROWS, bg="black")
        self.canvas.pack()
```

```
        self.board = [[None for _ in range(COLUMNS)] for _ in range(ROWS)]
        self.score = 0

        self.current_shape = None
        self.shape_type = None
        self.x = 0
        self.y = 0

        self.running = True
        self.paused = False

        # Key bindings
        self.root.bind("<Left>", lambda e: self.move(-1))
        self.root.bind("<Right>", lambda e: self.move(1))
        self.root.bind("<Down>", lambda e: self.drop())
        self.root.bind("<Up>", lambda e: self.rotate())
        self.root.bind("<space>", lambda e: self.hard_drop())
        self.root.bind("p", lambda e: self.toggle_pause())
        self.root.bind("r", lambda e: self.restart())

        self.spawn_new_shape()
        self.update()

    def draw_board(self):
        self.canvas.delete("all")

        for r in range(ROWS):
            for c in range(COLUMNS):
                color = self.board[r][c]
                if color:
                    self.draw_cell(c, r, color)

        for r, row in enumerate(self.current_shape):
            for c, val in enumerate(row):
                if val:
                    self.draw_cell(self.x + c, self.y + r, COLORS[self.shape_type])

        # Draw Score
        self.canvas.create_text(CELL_SIZE * COLUMNS - 60, 20,
                                text=f"Score: {self.score}", fill="white", font=("Arial", 14))

        # Draw Pause message
        if self.paused:
            self.canvas.create_text(CELL_SIZE * COLUMNS // 2, CELL_SIZE * ROWS // 2,
                                    text="PAUSED", fill="yellow", font=("Arial", 32, "bold"))

    def draw_cell(self, x, y, color):
        x1 = x * CELL_SIZE
        y1 = y * CELL_SIZE
        x2 = x1 + CELL_SIZE
        y2 = y1 + CELL_SIZE
```

```
        self.canvas.create_rectangle(x1, y1, x2, y2, fill=color, outline="gray")

    def spawn_new_shape(self):
        self.shape_type = random.choice(list(SHAPES.keys()))
        self.current_shape = [row[:] for row in SHAPES[self.shape_type]]  # deep copy
        self.x = COLUMNS // 2 - len(self.current_shape[0]) // 2
        self.y = 0

        if not self.valid_position():
            self.game_over()

    def valid_position(self, dx=0, dy=0, shape=None):
        if shape is None:
            shape = self.current_shape
        for r, row in enumerate(shape):
            for c, val in enumerate(row):
                if val:
                    new_x = self.x + c + dx
                    new_y = self.y + r + dy
                    if new_x < 0 or new_x >= COLUMNS or new_y >= ROWS:
                        return False
                    if new_y >= 0 and self.board[new_y][new_x]:
                        return False
        return True

    def move(self, dx):
        if self.running and not self.paused and self.valid_position(dx=dx):
            self.x += dx
            self.draw_board()

    def drop(self):
        if self.running and not self.paused:
            if self.valid_position(dy=1):
                self.y += 1
            else:
                self.lock_shape()
                self.clear_lines()
                self.spawn_new_shape()
            self.draw_board()

    def hard_drop(self):
        if self.running and not self.paused:
            while self.valid_position(dy=1):
                self.y += 1
            self.lock_shape()
            self.clear_lines()
            self.spawn_new_shape()
            self.draw_board()

    def rotate(self):
        if not self.running or self.paused:
            return
```

```
        rotated = list(zip(*self.current_shape[::-1]))
        rotated = [list(row) for row in rotated]  # convert tuples to lists

        # Wall kick tries
        for dx in (0, -1, 1, -2, 2):
            if self.valid_position(dx=dx, shape=rotated):
                self.current_shape = rotated
                self.x += dx
                break
        self.draw_board()

    def lock_shape(self):
        for r, row in enumerate(self.current_shape):
            for c, val in enumerate(row):
                if val:
                    self.board[self.y + r][self.x + c] = COLORS[self.shape_type]

    def clear_lines(self):
        new_board = [row for row in self.board if not all(row)]
        cleared = ROWS - len(new_board)
        if cleared > 0:
            self.score += cleared * 100
            new_board = [[None for _ in range(COLUMNS)] for _ in range(cleared)] + new_board
            self.board = new_board

    def update(self):
        if self.running and not self.paused:
            self.drop()
        self.draw_board()
        if self.running:
            self.root.after(DELAY, self.update)

    def game_over(self):
        self.canvas.create_text(CELL_SIZE * COLUMNS // 2, CELL_SIZE * ROWS // 2,
                                text="GAME OVER", fill="red", font=("Arial", 36, "bold"))
        self.running = False

    def toggle_pause(self):
        if not self.running:
            return
        self.paused = not self.paused
        self.draw_board()

    def restart(self):
        self.board = [[None for _ in range(COLUMNS)] for _ in range(ROWS)]
        self.score = 0
        self.running = True
        self.paused = False
        self.spawn_new_shape()
        self.draw_board()
```

```
# Run the Game
if __name__ == "__main__":
    root = tk.Tk()
    root.title("🧱 Tetris Game using Tkinter")
    game = Tetris(root)
    root.mainloop()
```

Output:-