

Project Report on
Programming in Python



Submitted By:

Sumon Das(CS23BCAGN008)

BCA 4th SEM

School of Computing Sciences

The Assam Kaziranga University ,Jorhat, Assam Table of Contents

1. Arithmetic and Quadratic Operation (Project 1)
2. Linear Equation Solver (Project 2)
3. Mathematical Graph (Star Pattern) (Project 3)
4. Function Implementation (Project 4)
5. Tkinter Game: Snake(Project 5)

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':
        b != 0:
            print(f"Division: {a} / {b} = {a / b}")
        else:
            print("Division: b 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}x^2 + {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): "))

arithmetic_operations(a1, b1, choice)
```

```
# Quadratic equation input
print("\nEnter coefficients for quadratic equation  $ax^2 + bx + c = 0$ :") 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  $ax^2 + bx + c = 0$ :
Enter coefficient a:
1
Enter coefficient b:
6
Enter coefficient c:
5

Solving Quadratic Equation:
Equation:  $1.0x^2 + 6.0x + 5.0 = 0$ 
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:  $a_1x + b_1y = c_1$  and  $a_2x + b_2y = c_2$ 
def solve_two_variable_linear(a1, b1, c1, a2, b2, c2):
    print("\nSolving Linear Equations\n(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\n(Determinant is zero)")

# Main Program
print("Enter coefficients for the system of equations:")
print("format:  $a_1x + b_1y = c_1$  and  $a_2x + b_2y = c_2$ ")

# 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

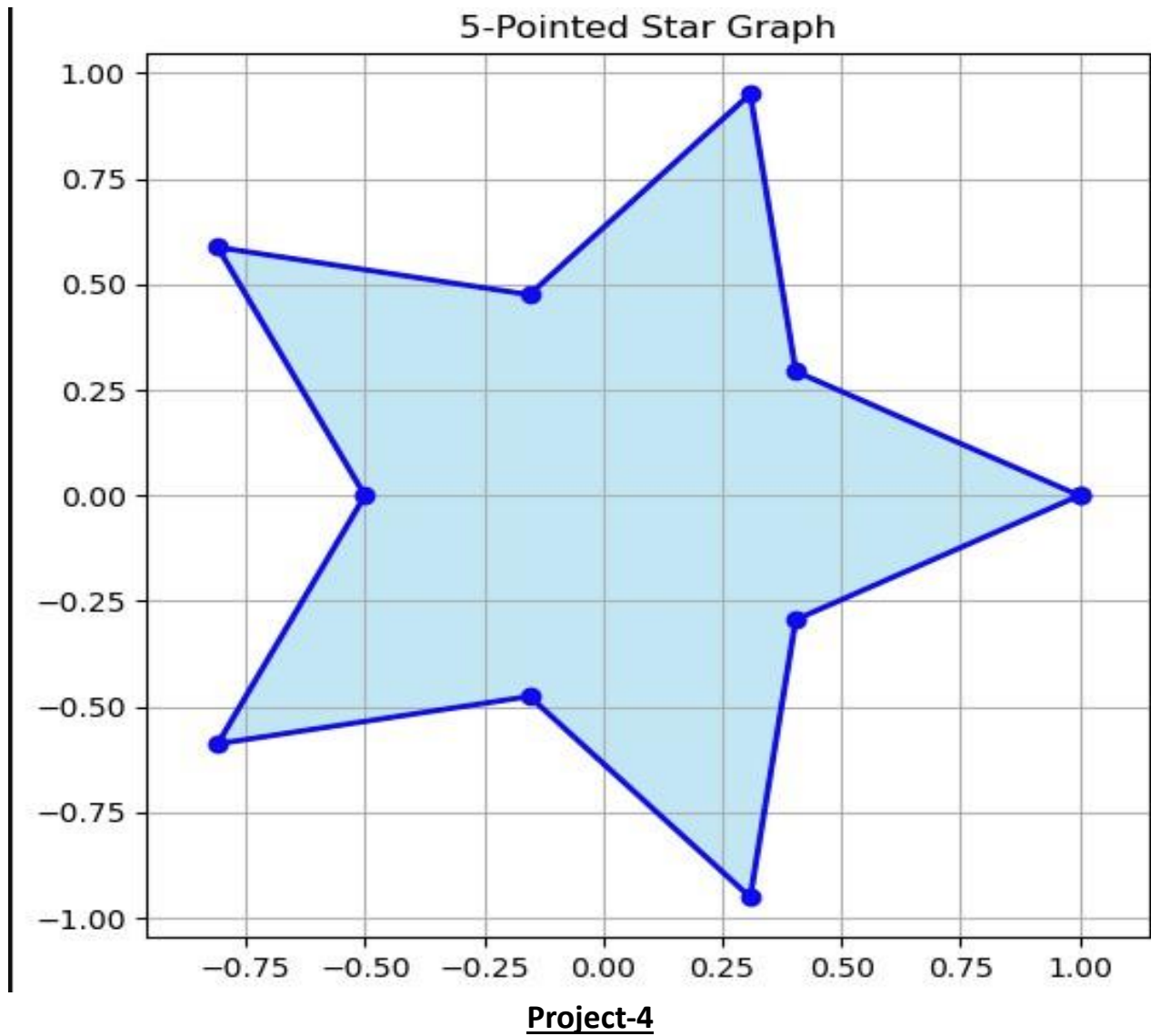
    # Convert polar to Cartesian
    coordinates x = radii * np.cos(angles)
    y = radii * np.sin(angles)

    # 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:-



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 Snake game made using Tkinter. The snake moves with arrow keys, grows on eating food, and the game ends if the snake hits the wall or itself. Real-time movement, score tracking, and collision detection are implemented.

```

import tkinter as tk
import random

# Constants
GAME_WIDTH = 600
GAME_HEIGHT = 400
SNAKE_ITEM_SIZE = 20
INITIAL_SPEED = 100 # milliseconds
FOOD_COLOR = "red"
SNAKE_COLOR = "green"

DIRECTIONS = {
    "Up": (0, -1),
    "Down": (0, 1),
    "Left": (-1, 0),
    "Right": (1, 0)
}

class SnakeGame:
    def __init__(self, root):
        self.root = root
        self.root.title("Snake Game - Enhanced Version")
        self.canvas = tk.Canvas(root, width=GAME_WIDTH, height=GAME_HEIGHT, bg="black")
        self.canvas.pack()

        self.reset_game()
        self.root.bind("<Key>", self.change_direction)
        self.update()

    def reset_game(self):
        self.snake = [(100, 100), (80, 100), (60, 100)]
        self.direction = "Right"
        self.running = True
        self.paused = False
        self.score = 0
        self.speed = INITIAL_SPEED

        self.canvas.delete("all")
        self.score_text = self.canvas.create_text(50, 10, fill="white", font="Arial 14", text=f"Score: {self.score}")
        self.draw_snake()

```

```

self.create_food()

def draw_snake(self):
    self.canvas.delete("snake")
for x, y in self.snake:
    self.canvas.create_rectangle(x, y, x + SNAKE_ITEM_SIZE, y + SNAKE_ITEM_SIZE,
fill=SNAKE_COLOR, tags="snake")

def create_food(self):
self.canvas.delete("food")
x = random.randint(0, (GAME_WIDTH - SNAKE_ITEM_SIZE) // SNAKE_ITEM_SIZE) *
SNAKE_ITEM_SIZE
y = random.randint(0, (GAME_HEIGHT - SNAKE_ITEM_SIZE) // SNAKE_ITEM_SIZE) *
SNAKE_ITEM_SIZE
self.food = (x, y)
self.canvas.create_oval(x, y, x + SNAKE_ITEM_SIZE, y + SNAKE_ITEM_SIZE,
fill=FOOD_COLOR, tags="food")

def change_direction(self, event):
key = event.keysym    if key ==
"p":
    self.paused = not self.paused
elif key == "r":
    self.reset_game()    elif key in DIRECTIONS:        opposite = {"Up":
"Down", "Down": "Up", "Left": "Right", "Right": "Left"}    if key !=
opposite.get(self.direction):        self.direction = key

def move_snake(self):    dx, dy = DIRECTIONS[self.direction]    head_x, head_y =
self.snake[0]    new_head = (head_x + dx * SNAKE_ITEM_SIZE, head_y + dy *
SNAKE_ITEM_SIZE)

    # Collision Check
    if (new_head in self.snake or        not 0 <=
new_head[0] < GAME_WIDTH or        not 0 <=
new_head[1] < GAME_HEIGHT):        self.running
= False
    self.canvas.create_text(GAME_WIDTH // 2, GAME_HEIGHT // 2, fill="white",
font="Arial 24 bold", text="Game Over!\nPress 'R' to Restart")    return
self.snake.insert(0, new_head)

    if new_head == self.food:
        self.score += 1
        self.speed = max(50, INITIAL_SPEED - (self.score * 2))
self.canvas.itemconfig(self.score_text, text=f"Score:        {self.score
}")    self.create_food()    else:
self.snake.pop()

```

```

self.draw_snake()

def update(self):    if self.running
and not self.paused:
self.move_snake()
    self.root.after(self.speed, self.update)

# ----- Run the Game -----if
__name__ == "__main__":  root =
tk.Tk()
    game = SnakeGame(root)
root.mainloop()

```

Output:-

