

Kaziranga University  
Department of Computer Science

**Programming in Python Report**

**Name**: Akash Baruah  
**ID**: CS23BCAGN041  
**Faculty**: Nawab Wasim Rahman

**Python Program Report**

This report includes various Python programs showcasing arithmetic operations, linear and quadratic equations, graphical representations, mathematical functions, and a graphical user interface application using tkinter.

INDEX

|  |  |  |
| --- | --- | --- |
| **Task No.** | **Title** | **Description** |
| **1** | **WAP using Python Implementation of Arithmetic and Quadratic Operations** | Performs basic arithmetic operations and solves quadratic equations using standard formula |
| **2** | **Implementation of Linear Equation Solver** | Solves linear equations of the form *ax + b = 0* |
| **3** | **Graphical Representation Using Mathematical Functions** | Plots graphs like *y = x²* and *y = sin(x)* using matplotlib and numpy |
| **4** | **WAP to Implement Mathematical Functions** | Implements **Factorial**, **Area of Circle**, and **Fibonacci Series** as reusable functions |
| **5** | **Tkinter Application Tkinter Game** | A game where you move a paddle to catch a falling ball, demonstrating basic animation and user interaction. |

**Task 1: Arithmetic and Quadratic Operations**

**Program Logic and Purpose**

**This Python program performs three main functions. First, it handles basic arithmetic operations by taking two numbers as input and computing their addition, subtraction, multiplication, and division—ensuring division by zero is avoided. Second, it performs bitwise operations on the same two numbers, including AND, OR, XOR, left shift, and right shift. Third, it solves quadratic equations by accepting three coefficients aaa, bbb, and ccc for an equation of the form ax2+bx+c=0ax^2 + bx + c = 0ax2+bx+c=0, using the quadratic formula. Based on the discriminant, it determines whether the roots are two real and distinct numbers, one real repeated root, or a pair of complex roots.**

**Code**

import math

def arithmetic\_ops(a, b):

print("\nArithmetic:")

print(f"{a} + {b} = {a + b}")

print(f"{a} - {b} = {a - b}")

print(f"{a} \* {b} = {a \* b}")

print(f"{a} / {b} = {a / b}" if b != 0 else "Division by zero error")

def bitwise\_ops(x, y):

print("\nBitwise:")

print(f"{x} & {y} = {x & y}")

print(f"{x} | {y} = {x | y}")

print(f"{x} ^ {y} = {x ^ y}")

print(f"{x} << 1 = {x << 1}")

print(f"{y} >> 1 = {y >> 1}")

def quadratic\_solver(a, b, c):

print("\nQuadratic Solver:")

d = b\*\*2 - 4\*a\*c

if d > 0:

r1 = (-b + math.sqrt(d)) / (2\*a)

r2 = (-b - math.sqrt(d)) / (2\*a)

print(f"Real & Distinct Roots: {r1}, {r2}")

elif d == 0:

r = -b / (2\*a)

print(f"Real & Equal Root: {r}")

else:

rp = -b / (2\*a)

ip = math.sqrt(-d) / (2\*a)

print(f"Complex Roots: {rp}+{ip}i, {rp}-{ip}i")

# Input

a = float(input("Enter 1st number: "))

b = float(input("Enter 2nd number: "))

arithmetic\_ops(a, b)

bitwise\_ops(int(a), int(b))

print("\n-- Quadratic Equation --")

a = float(input("a: "))

b = float(input("b: "))

c = float(input("c: "))

quadratic\_solver(a, b, c) (f

"{x} ^ {y} = {x ^ y}")

b, c)

**User Manual**

To use the program, simply run it in any Python environment such as IDLE, PyCharm, or VS Code. When prompted, enter two numbers to perform arithmetic operations. The program will display the results of addition, subtraction, multiplication, and division (with a check to prevent division by zero). It will also perform bitwise operations—AND, OR, XOR, left shift, and right shift—on the integer values of the input numbers. After that, the program will ask you to input the coefficients aa, bb, and cc of a quadratic equation in the form ax2+bx+c=0ax^2 + bx + c = 0. Based on the discriminant, it will compute and display the roots of the equation, indicating whether they are real and distinct, real and equal, or complex.

**Sample Input/Output**

* **Input:**

Enter 1st number: 10

Enter 2nd number: 4

* **Output:**

Arithmetic:

10.0 + 4.0 = 14.0

10.0 - 4.0 = 6.0

10.0 \* 4.0 = 40.0

10.0 / 4.0 = 2.5

Bitwise:

10 & 4 = 0

10 | 4 = 14

10 ^ 4 = 14

10 << 1 = 20

4 >> 1 = 2

**Task 2: Linear Equation Solver**

**Purpose**

This program solves a linear equation of the standard form ax+b=0ax + b = 0ax+b=0 by calculating the value of xxx using the formula x=−b/ax = -b/ax=−b/a. It intelligently handles special cases as well: if a=0a = 0a=0 and b=0b = 0b=0, the equation has **infinite solutions**, indicating it is true for all values of xxx; if a=0a = 0a=0 and b≠0b \neq 0b=0, the equation becomes a contradiction with **no solution**.

**Code**

def solve\_linear\_equation(a, b):

print("\nSolving Linear Equation: ax + b = 0")

if a == 0:

if b == 0:

print("Infinite solutions (every x satisfies the equation).")

else:

print("No solution (this is a contradiction).")

else:

x = -b / a

print(f"Solution: x = {-b} / {a} = {x}")

a = float(input("Enter coefficient a: "))

b = float(input("Enter coefficient b: "))

solve\_linear\_equation(a, b)

**User Manual**

**To use the program, simply run the script in a Python IDE or terminal. When prompted, enter values for the coefficients aa and bb of the linear equation ax+b=0ax + b = 0. Based on the inputs, the program will output either the calculated solution for xx, a message indicating that there are infinite solutions (if both aa and bb are zero), or a message stating that there is no solution (if a=0a = 0 but b≠0b \neq 0).**

**Sample Input and Output**

* **Input:**

Enter coefficient a: 5

Enter coefficient b: -10

* **Output:**

Solving Linear Equation: ax + b = 0

Solution: x = 10.0 / 5.0 = 2.0

**Task 3: Graphical Representation**

**Purpose**

This program demonstrates the use of the **matplotlib** and **NumPy** libraries in Python to plot mathematical graphs. Specifically, it visualizes two functions: y=x2y = x^2, which forms a **parabola**, and y=sin⁡(x)y = \sin(x), which produces a **sine wave**. Both graphs are displayed **side by side in a single window** using the subplot feature of matplotlib, showcasing multiple plots in a coordinated layout.

**Code**

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(-10, 10, 400)

y1 = x\*\*2

y2 = np.sin(x)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.plot(x, y1, color='blue')

plt.title("Graph of y = x²")

plt.xlabel("x")

plt.ylabel("y")

plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(x, y2, color='green')

plt.title("Graph of y = sin(x)")

plt.xlabel("x")

plt.ylabel("y")

plt.grid(True)

plt.tight\_layout()

plt.show()

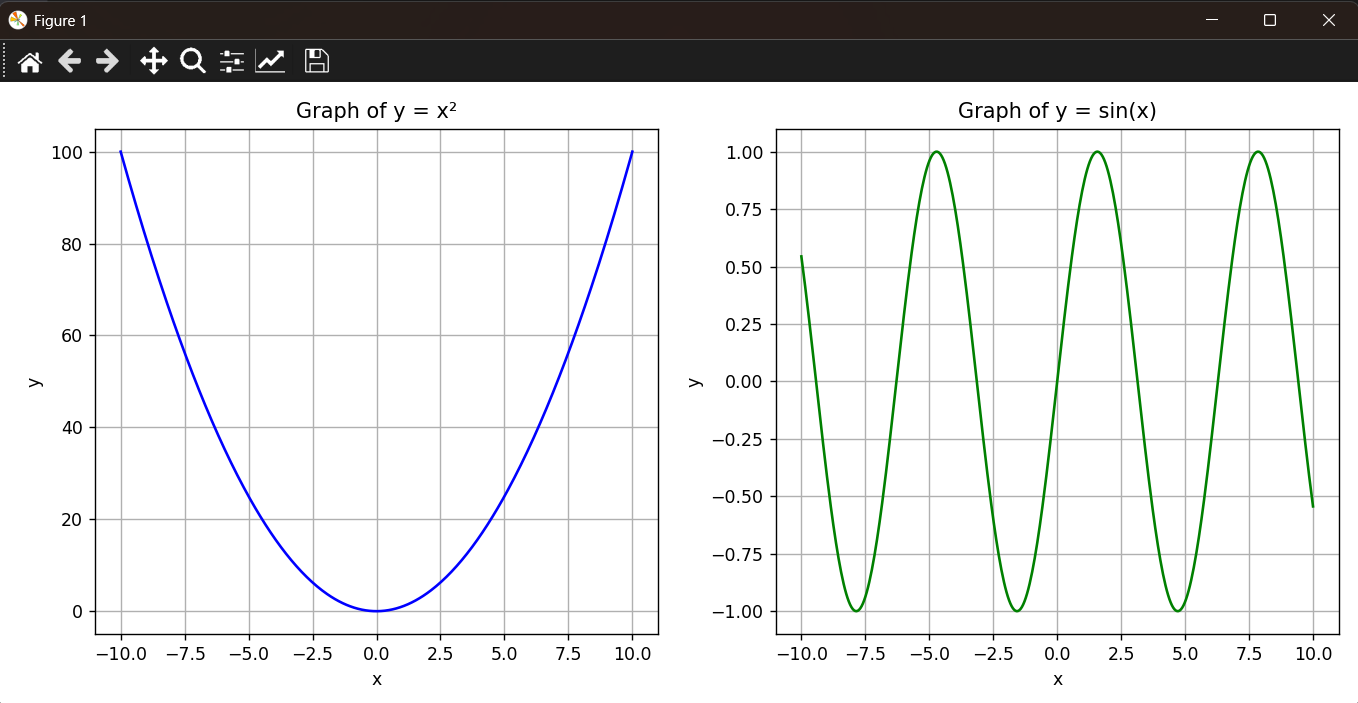
**User Manual**

1. **Install Required Libraries (Only Once):**  
   Run this in your terminal or command prompt:

pip install matplotlib numpy

1. **Run the Program:**

You can use any Python IDE or script runner such as IDLE, VS Code, Jupyter Notebook, or others to run the program. When executed, the program will display **two side-by-side plots in a single window**, allowing for a clear visual comparison or analysis of the data or functions being plotted.

**Sample Output**

1. **Left Plot: y=x2**
   * A **parabola** opening upwards.
   * Symmetric around the y-axis.
2. **Right Plot: y=sin(x)**
   * A **wave pattern**.
   * Periodic and oscillating between -1 and 1.

Both graphs have:

* **Axes labels**: x and y
* **Titles**
* **Grid lines** for clarity

**Task 4: Function Implementations**

**Purpose**

This program implements three commonly used mathematical functions. It includes a **factorial** function that computes the factorial of a number using recursion, an **area of a circle** function that calculates the area based on a given radius, and a **Fibonacci series** function that generates the first *n* terms of the Fibonacci sequence. These functions demonstrate basic mathematical computations using fundamental programming techniques.

**Code**

import math

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

def area\_of\_circle(radius):

return math.pi \* radius \* radius

def fibonacci(n):

fib\_series = []

a, b = 0, 1

for \_ in range(n):

fib\_series.append(a)

a, b = b, a + b

return fib\_series

num = int(input("Enter a number for factorial: "))

print(f"Factorial of {num} = {factorial(num)}")

radius = float(input("\nEnter radius of circle: "))

print(f"Area of circle with radius {radius} = {area\_of\_circle(radius):.2f}")

terms = int(input("\nEnter number of terms for Fibonacci series: "))

print(f"Fibonacci series with {terms} terms: {fibonacci(terms)}")

**User Manual**

**To use the program, run it in a Python environment and provide the required inputs when prompted. First, enter a non-negative integer to calculate its factorial. Next, input a positive float value to compute the area of a circle using that radius. Finally, enter a positive integer to generate the specified number of terms in the Fibonacci series. Once all inputs are given, the program will display the results of all three computations in sequence.**

**Sample Input and Output**

* **Input:**

Enter a number for factorial: 5

* **Output:**

Enter a number for factorial: 5

**Task 5: Tkinter Game - Catch the Ball**

**Purpose**

The purpose of this project is to design and implement a simple interactive game using **Python's Tkinter library** to enhance understanding of **GUI programming, event handling, and basic game logic**. This project helps students and beginners learn how to create graphical applications by combining fundamental programming concepts with user interaction.

**Code**

import tkinter as tk

import random

# Window setup

root = tk.Tk()

root.title("Catch the Ball")

root.geometry("400x500")

canvas = tk.Canvas(root, width=400, height=500, bg="lightblue")

canvas.pack()

# Paddle setup

paddle = canvas.create\_rectangle(160, 480, 240, 490, fill="blue")

# Ball setup

ball = canvas.create\_oval(0, 0, 20, 20, fill="red")

ball\_speed = 5

# Score

score = 0

score\_text = canvas.create\_text(10, 10, anchor='nw', text=f"Score: {score}", font=("Arial", 14, "bold"))

# Move Paddle

def move\_left(event):

canvas.move(paddle, -20, 0)

def move\_right(event):

canvas.move(paddle, 20, 0)

root.bind("<Left>", move\_left)

root.bind("<Right>", move\_right)

# Ball drop logic

def drop\_ball():

global score

x = random.randint(0, 380)

canvas.coords(ball, x, 0, x + 20, 20)

def fall():

global score # This line fixes the UnboundLocalError

pos = canvas.coords(ball)

if pos[3] < 500:

canvas.move(ball, 0, ball\_speed)

root.after(20, fall)

else:

paddle\_pos = canvas.coords(paddle)

# Check if ball overlaps paddle horizontally

if paddle\_pos[0] < pos[2] and paddle\_pos[2] > pos[0]:

score += 1

canvas.itemconfig(score\_text, text=f"Score: {score}")

drop\_ball()

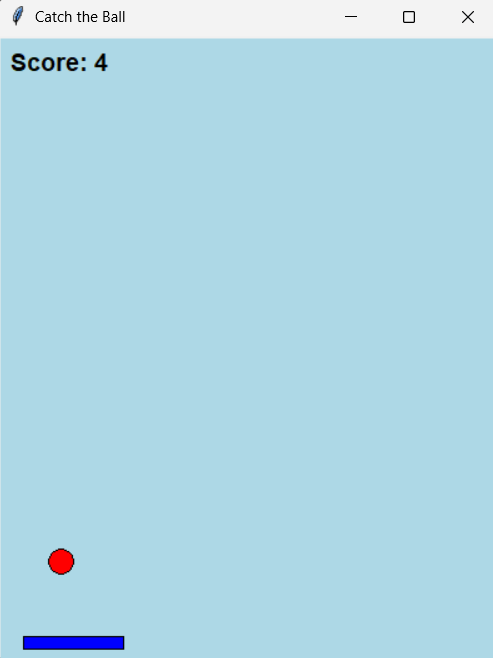
fall()

drop\_ball()

root.mainloop()

**User Manual**

The user manual for the "Catch the Ball" game explains how to run the program in a Python environment with Tkinter. Players use the left and right arrow keys to move a paddle and catch a falling ball. Each catch increases the score shown on the screen. The game helps users practice basic game controls, collision detection, and scorekeeping in a simple graphical interface.

**Game Output Examples**