LAB-1

Code:

import random

def fitness(individual):

    x = int(individual, 2)

    return x \* x

def generate\_individual(length=5):

    return ''.join(random.choice('01') for \_ in range(length))

def generate\_population(size=4):

    return [generate\_individual() for \_ in range(size)]

def select\_parents(population, k=3):

    selected = random.sample(population, k)

    selected.sort(key=fitness, reverse=True)

    return selected[0], selected[1]

def crossover(parent1, parent2):

    point = random.randint(1, len(parent1) - 1)

    child1 = parent1[:point] + parent2[point:]

    child2 = parent2[:point] + parent1[point:]

    return child1, child2

def mutate(individual, mutation\_rate=0.01):

    mutated = ''.join(

        bit if random.random() > mutation\_rate else '1' if bit == '0' else '0'

        for bit in individual

    )

    return mutated

def print\_population(population, generation=None):

    if generation is None:

        print("Initial Population:")

    else:

        print(f"\nGeneration {generation} Population:")

    for ind in population:

        x = int(ind, 2)

        fit = fitness(ind)

        print(f"  Individual: {ind}, x = {x}, fitness = {fit}")

def genetic\_algorithm(generations=6, population\_size=4, mutation\_rate=0.01):

    population = generate\_population(population\_size)

    # Print initial population

    print\_population(population)

    for generation in range(1, generations + 1):

        new\_population = []

        while len(new\_population) < population\_size:

            parent1, parent2 = select\_parents(population)

            child1, child2 = crossover(parent1, parent2)

            new\_population.append(mutate(child1, mutation\_rate))

            if len(new\_population) < population\_size:

                new\_population.append(mutate(child2, mutation\_rate))

        population = new\_population

        # Print generation population

        print\_population(population, generation)

        # Print best in generation

        best = max(population, key=fitness)

        print(f"Best individual: {best}, x = {int(best, 2)}, fitness = {fitness(best)}")

    # Return best overall

    best = max(population, key=fitness)

    return best

best\_solution = genetic\_algorithm()

print("\nBest solution found:")

print(f"Binary: {best\_solution}, x = {int(best\_solution, 2)}, f(x) = {fitness(best\_solution)}")

Output:

