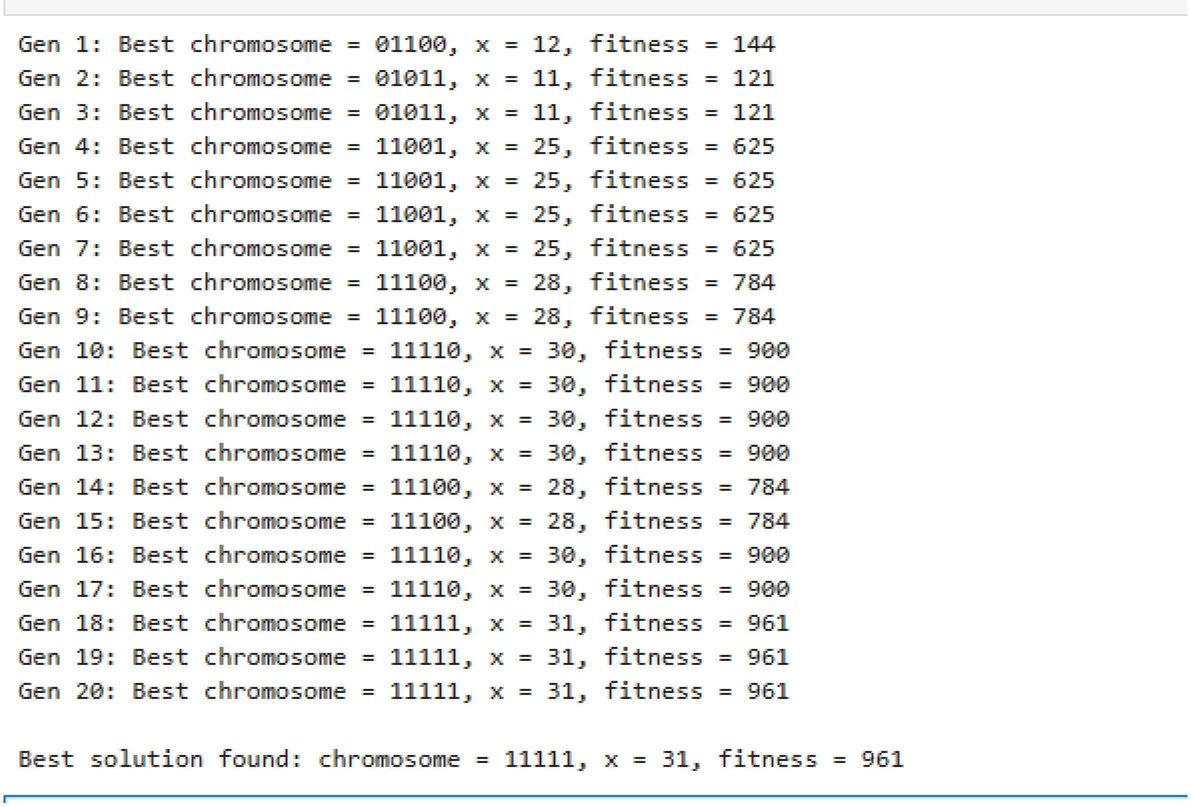
Genetic Algorithm

```
import random

# Parameters
POP_SIZE = 6
CHROM_LENGTH = 5  # 5 bits -> numbers 0-31
MAX_GEN = 20
CROSSOVER_RATE = 0.8
MUTATION_RATE = 0.1

# Fitness function: f(x) = x^2
def fitness(chromosome):
    # Convert binary string to decimal
    x = int(chromosome, 2)
    return x ** 2

# Generate initial population
def init_population():
    population = []
    for _ in range(POP_SIZE):
        chromosome = ''.join(random.choice('01') for _ in range(CHROM_LENGTH))
        population.append(chromosome)
    return population

# Selection: roulette wheel selection
def select(population):
    max_fitness = sum(fitness(ch) for ch in population)
    pick = random.uniform(0, max_fitness)
    current = 0
    for ch in population:
        current += fitness(ch)
        if current > pick:
            return ch
    return population[-1]

# Crossover: single point
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_RATE:
        point = random.randint(1, CHROM_LENGTH - 1)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]
        return child1, child2
    else:
        return parent1, parent2

# Mutation: bit flip
def mutate(chromosome):
    chromosome = list(chromosome)
    for i in range(CHROM_LENGTH):
        if random.random() < MUTATION_RATE:
            chromosome[i] = '1' if chromosome[i] == '0' else '0'
    return ''.join(chromosome)

# Genetic Algorithm main function
def genetic_algorithm():
    population = init_population()

    for generation in range(MAX_GEN):
        new_population = []

        while len(new_population) < POP_SIZE:
            parent1 = select(population)
            parent2 = select(population)

            child1, child2 = crossover(parent1, parent2)

            child1 = mutate(child1)
            child2 = mutate(child2)

            new_population.extend([child1, child2])

        population = new_population[:POP_SIZE]

        # Best solution in current generation
        best = max(population, key=fitness)
        print(f"Gen {generation+1}: Best chromosome = {best}, x = {int(best, 2)}, fitness =
{fitness(best)}")

    best = max(population, key=fitness)
    print(f"\nBest solution found: chromosome = {best}, x = {int(best, 2)}, fitness = {fitness(best)}")

# Run the GA
if __name__ == "__main__":
    genetic_algorithm()
```

Output:



```
Gen 1: Best chromosome = 01100, x = 12, fitness = 144
Gen 2: Best chromosome = 01011, x = 11, fitness = 121
Gen 3: Best chromosome = 01011, x = 11, fitness = 121
Gen 4: Best chromosome = 11001, x = 25, fitness = 625
Gen 5: Best chromosome = 11001, x = 25, fitness = 625
Gen 6: Best chromosome = 11001, x = 25, fitness = 625
Gen 7: Best chromosome = 11001, x = 25, fitness = 625
Gen 8: Best chromosome = 11100, x = 28, fitness = 784
Gen 9: Best chromosome = 11100, x = 28, fitness = 784
Gen 10: Best chromosome = 11110, x = 30, fitness = 900
Gen 11: Best chromosome = 11110, x = 30, fitness = 900
Gen 12: Best chromosome = 11110, x = 30, fitness = 900
Gen 13: Best chromosome = 11110, x = 30, fitness = 900
Gen 14: Best chromosome = 11100, x = 28, fitness = 784
Gen 15: Best chromosome = 11100, x = 28, fitness = 784
Gen 16: Best chromosome = 11110, x = 30, fitness = 900
Gen 17: Best chromosome = 11110, x = 30, fitness = 900
Gen 18: Best chromosome = 11111, x = 31, fitness = 961
Gen 19: Best chromosome = 11111, x = 31, fitness = 961
Gen 20: Best chromosome = 11111, x = 31, fitness = 961

Best solution found: chromosome = 11111, x = 31, fitness = 961
```