Genetic algorithm

Step 1: Identify the objective function to optimize. In this case maximize $f(x) = x^2$

Step 2: Set the following parameters: population size as 100, mutation rate as 0.1, crossover rate as 0.8, number of generations as 50, lower bound as -10, upper bound as 10

Step 3: Generate an initial population of 100 random individuals within the range of -10 to 10

Step 4: For each individual in the population compute its fitness using fitness function $f(x) = x^2$

Step 5: Use Roulette wheel selection to select two parents from the population

Step 6: For the selected parents, perform crossover with a probability of 0.8

Step 7: For each offspring apply mutation with a probability of 0.1

Step 8: Collect the newly created offspring until the new population reaches the original population size

Step 9: Replace old population with new generation of individuals

Step 10: After final generation evaluate fitness of the last population

Step 11: Print best solution found and its fitness

```
import random
def fitness_function(x):
    return x**2
population_size = 100
mutation_rate = 0.1
crossover_rate = 0.8
num_generations = 50
lower_bound = -10
upper_bound = 10

def generate_population(size, lower_bound, upper_bound):
    return [random.uniform(lower_bound, upper_bound)
    for _ in range(size)]

def evaluate_population(population):
    return [fitness_function(ind) for ind in population]

def select_parents(population, fitness):
    total_fitness = sum(fitness)
    selection_probs = [f/total_fitness for f in fitnesses]
    parent1 = random.choices(population, weights =
              selection_probs, k=1)[0]
    parent2 = random.choices(population, weights =
              selection_probs, k=1)[0]
    return parent1, parent2

def crossover(parent1, parent2, crossover_rate):
    if random.random() < crossover_rate:
        crossover_point = random.random()
        child1 = crossover_point * parent1 + (1-crossover_point) * pare
        child2 = crossover_point * parent2 + (1-crossover_point) * pare
        return child1, child2
```

```
else:
return parent1, parent2

def mutate(i, m_rate, lower_bound, upper_bound):
    if random.random() < m_rate:
        i = random.uniform(lower_bound, upper_bound)
    return individual

def genetic_algorithm():
    population = generate_population(population_size,
    lower_bound, upper_bound)
for generation in range(num_generations):
    fitness = evaluate_population(population)
    new_p = []
    while len(new_p) < population_size:
    parent1, parent2 = select_parents(population, fitness)
    child1, child2 = crossover(parent1, parent2, crossover_
                                                    rate)
    child1 = mutate(child1, m_rate, lower_bound, upper_
    child2 = mutate(child2, m_rate, lower_bound, upper_
                                                    bound)
    new_population.extend(child1, child2)
    population = new_p[:population_size]
final_fitness = evaluate_population(population)
best_individual = population[final_fitness.index
(max(final_fitness))]
best_fitness = max(final_fitness)
return best_individual, best_fitness
best_solution, best_fitness = genetic_algorithm()
print("Best solution: {best_solution}")
print("Fitness of solution: {best_fitness}")
```

Output :

Best solution found : +9.956292750107881

Fitness of the best solution : 99.12776532585076