To implement a genetic algorithm for finding the maximum or minimum of a function, we'll follow these steps:

1. Representation (Chromosome): Define how solutions are represented.
2. Initialization: Generate an initial population.
3. Selection: Decide how to select individuals for reproduction.
4. Crossover: Combine parts of two parents to create offspring.
5. Mutation: Introduce random changes to individuals.
6. Evaluation: Assess the fitness of individuals.
7. Termination: Determine when to stop the algorithm.

In [4]:
```python
import random

def f(x):
    return x**2

def initialize_population(size=10):
    return [random.uniform(-10, 10) for _ in range(size)]

def select_parent(population):
    individual1, individual2 = random.sample(population, 2)
    return individual1 if f(individual1) > f(individual2) else individual2

def crossover(parent1, parent2):
    return (parent1 + parent2) / 2

def mutate(individual):
    mutation_chance = 0.1
    if random.random() < mutation_chance:
        individual += random.uniform(-1, 1)
    return individual

def genetic_algorithm(generations=100, top_solutions=3):
    population = initialize_population()

    for generation in range(generations):
        print(f"Generation {generation + 1}:")
        population_with_fitness = [(individual, f(individual)) for individual in po
        for individual, fitness in sorted(population_with_fitness, key=lambda x: x[
            print(f"  Individual: {individual:.4f}, Fitness: {fitness:.4f}")

        new_population = []
        for _ in range(len(population)):
            parent1 = select_parent(population)
            parent2 = select_parent(population)
            child = crossover(parent1, parent2)
            child = mutate(child)
            new_population.append(child)
        population = new_population

    # Find the best solutions
    final_population_with_fitness = sorted([(individual, f(individual)) for individ
```

```
    best_solutions = final_population_with_fitness[:top_solutions]
    return best_solutions

best_solutions = genetic_algorithm(generations=10, top_solutions=3)  # Reduced gene
print("\nTop 3 Solutions:")
for individual, fitness in best_solutions:
    print(f"Individual: {individual:.4f}, Fitness: {fitness:.4f}")
```

```
Generation 1:
  Individual: 6.7358, Fitness: 45.3715
  Individual: -6.4451, Fitness: 41.5394
  Individual: 4.8959, Fitness: 23.9700
  Individual: 2.8527, Fitness: 8.1378
  Individual: 2.7299, Fitness: 7.4523
  Individual: 2.1706, Fitness: 4.7116
  Individual: -1.4586, Fitness: 2.1274
  Individual: 1.4123, Fitness: 1.9946
  Individual: 0.4976, Fitness: 0.2476
  Individual: 0.4194, Fitness: 0.1759
Generation 2:
  Individual: 4.7943, Fitness: 22.9849
  Individual: 4.7329, Fitness: 22.4000
  Individual: 4.7329, Fitness: 22.4000
  Individual: 4.2645, Fitness: 18.1856
  Individual: 4.0741, Fitness: 16.5981
  Individual: 3.8129, Fitness: 14.5382
  Individual: 3.3753, Fitness: 11.3926
  Individual: 2.8527, Fitness: 8.1378
  Individual: 2.6386, Fitness: 6.9624
  Individual: -1.7962, Fitness: 3.2264
Generation 3:
  Individual: 4.7636, Fitness: 22.6915
  Individual: 4.5294, Fitness: 20.5151
  Individual: 4.4987, Fitness: 20.2380
  Individual: 4.4342, Fitness: 19.6619
  Individual: 4.4035, Fitness: 19.3906
  Individual: 4.2645, Fitness: 18.1856
  Individual: 3.8235, Fitness: 14.6190
  Individual: 3.8199, Fitness: 14.5915
  Individual: 3.7164, Fitness: 13.8120
  Individual: 3.7164, Fitness: 13.8120
Generation 4:
  Individual: 4.7636, Fitness: 22.6915
  Individual: 4.6465, Fitness: 21.5896
  Individual: 4.6311, Fitness: 21.4472
  Individual: 4.5835, Fitness: 21.0086
  Individual: 4.4697, Fitness: 19.9779
  Individual: 4.4664, Fitness: 19.9489
  Individual: 4.4664, Fitness: 19.9489
  Individual: 4.4664, Fitness: 19.9489
  Individual: 4.4342, Fitness: 19.6619
  Individual: 4.1611, Fitness: 17.3145
Generation 5:
  Individual: 4.7636, Fitness: 22.6915
  Individual: 4.7050, Fitness: 22.1371
  Individual: 4.6166, Fitness: 21.3131
  Individual: 4.6150, Fitness: 21.2981
  Individual: 4.5835, Fitness: 21.0086
  Individual: 4.5488, Fitness: 20.6913
  Individual: 4.5266, Fitness: 20.4900
  Individual: 4.4680, Fitness: 19.9634
  Individual: 4.4664, Fitness: 19.9489
  Individual: 4.4503, Fitness: 19.8051
Generation 6:
```

```
  Individual: 4.6893, Fitness: 21.9893
  Individual: 4.6893, Fitness: 21.9893
  Individual: 4.6600, Fitness: 21.7156
  Individual: 4.6269, Fitness: 21.4081
  Individual: 4.6269, Fitness: 21.4081
  Individual: 4.6158, Fitness: 21.3056
  Individual: 4.6001, Fitness: 21.1606
  Individual: 4.5819, Fitness: 20.9936
  Individual: 4.5415, Fitness: 20.6254
  Individual: 4.2286, Fitness: 17.8807
Generation 7:
  Individual: 4.7117, Fitness: 22.2005
  Individual: 4.6893, Fitness: 21.9893
  Individual: 4.6893, Fitness: 21.9893
  Individual: 4.6746, Fitness: 21.8522
  Individual: 4.6581, Fitness: 21.6977
  Individual: 4.6525, Fitness: 21.6461
  Individual: 4.6447, Fitness: 21.5730
  Individual: 4.6434, Fitness: 21.5616
  Individual: 4.6356, Fitness: 21.4886
  Individual: 4.6044, Fitness: 21.2003
Generation 8:
  Individual: 4.7117, Fitness: 22.2005
  Individual: 4.7005, Fitness: 22.0948
  Individual: 4.6849, Fitness: 21.9484
  Individual: 4.6821, Fitness: 21.9224
  Individual: 4.6820, Fitness: 21.9207
  Individual: 4.6820, Fitness: 21.9207
  Individual: 4.6670, Fitness: 21.7806
  Individual: 4.6525, Fitness: 21.6461
  Individual: 4.6434, Fitness: 21.5616
  Individual: 4.5074, Fitness: 20.3166
Generation 9:
  Individual: 4.7061, Fitness: 22.1476
  Individual: 4.6983, Fitness: 22.0743
  Individual: 4.6969, Fitness: 22.0612
  Individual: 4.6894, Fitness: 21.9901
  Individual: 4.6849, Fitness: 21.9484
  Individual: 4.6834, Fitness: 21.9346
  Individual: 4.6834, Fitness: 21.9346
  Individual: 4.6834, Fitness: 21.9346
  Individual: 4.6820, Fitness: 21.9216
  Individual: 4.5976, Fitness: 21.1375
Generation 10:
  Individual: 4.7022, Fitness: 22.1109
  Individual: 4.6977, Fitness: 22.0688
  Individual: 4.6969, Fitness: 22.0612
  Individual: 4.6955, Fitness: 22.0479
  Individual: 4.6948, Fitness: 22.0409
  Individual: 4.6948, Fitness: 22.0409
  Individual: 4.6909, Fitness: 22.0044
  Individual: 4.6902, Fitness: 21.9979
  Individual: 4.6902, Fitness: 21.9979
  Individual: 4.6902, Fitness: 21.9979

Top 3 Solutions:
```

```
Individual: 4.7322, Fitness: 22.3941
Individual: 4.6996, Fitness: 22.0861
Individual: 4.6977, Fitness: 22.0688
```

This code will print the population and their fitness values at each generation. After the final generation, it will print the top 3 solutions based on their fitness values. Note that the number of generations has been reduced to 10 for brevity, and you can adjust it back to 100 or any other number to see more evolution stages.

Keep in mind that due to the randomness in the genetic algorithm (in selection, mutation, and initial population), the output will vary each time you run the script. The algorithm is designed to maximize the fitness function f(x)=x^2, so the top solutions will be those with x values close to the boundaries of the defined range [-10,10], as these yield the highest f(x) values.

In [ ]: