GENETIC ALGORITHM

TRAVELLING SALESMAN PROBLEM (TSP)

FREDERICK YONATAN / 5025211121

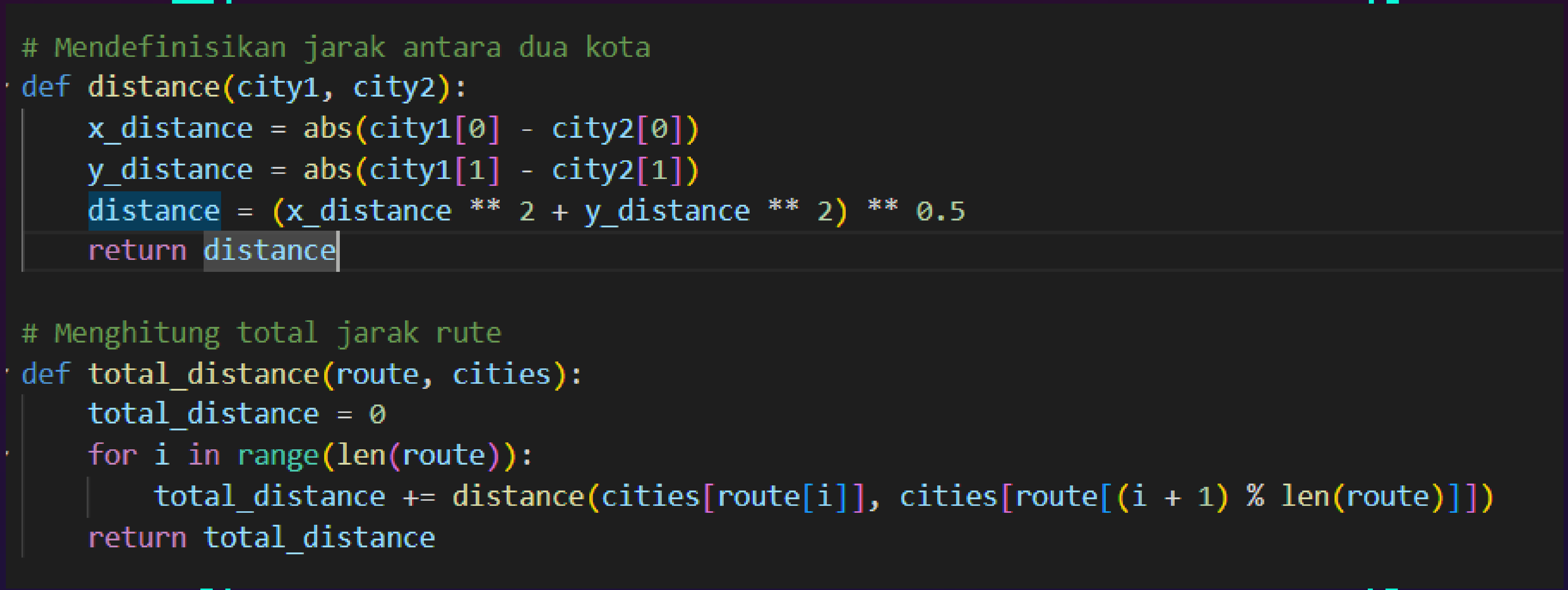
START NOW

GAME START

PLAY

MENU

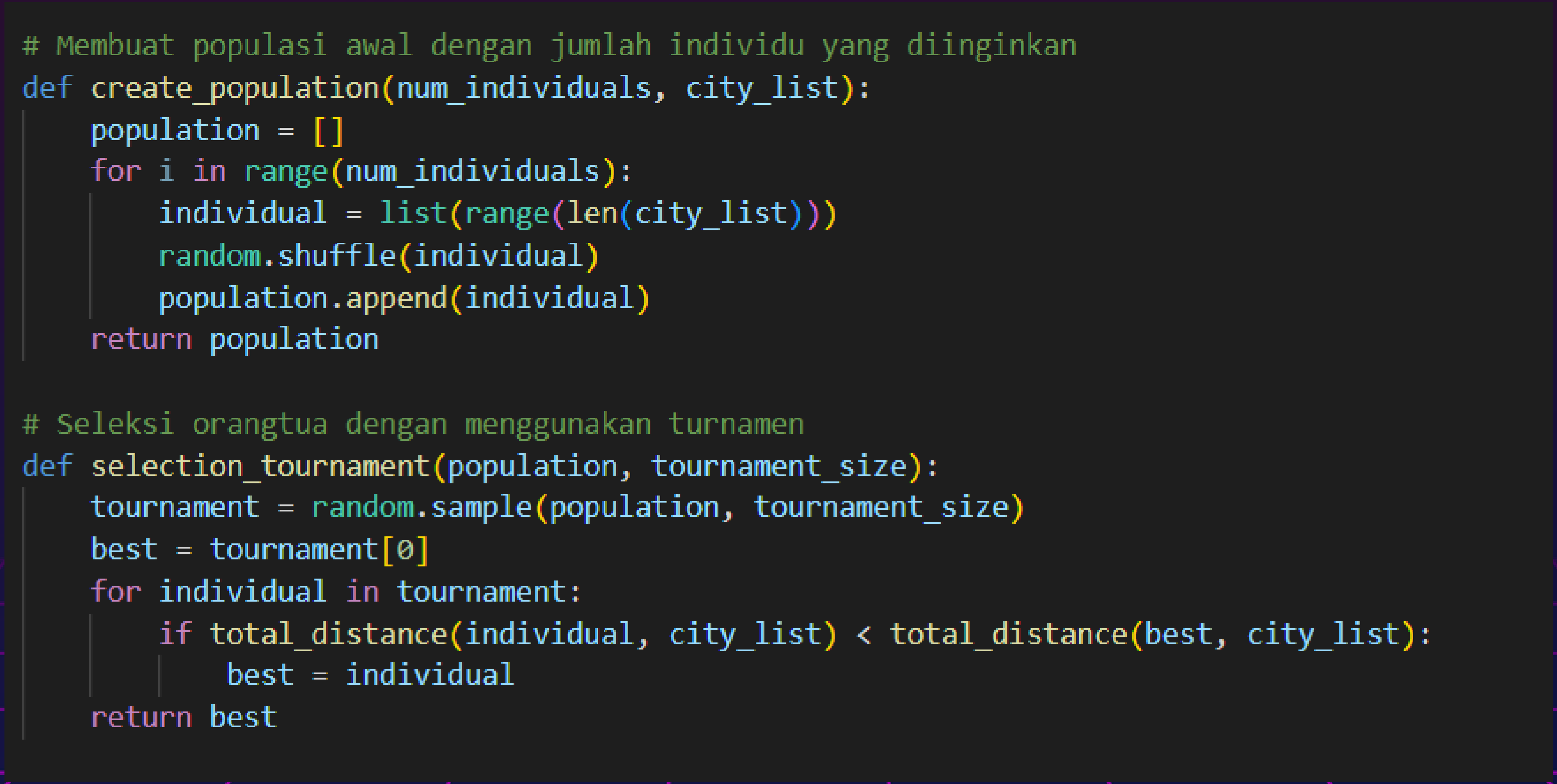
EXIT



```python
# Mendefinisikan jarak antara dua kota
def distance(city1, city2):
    x_distance = abs(city1[0] - city2[0])
    y_distance = abs(city1[1] - city2[1])
    distance = (x_distance ** 2 + y_distance ** 2) ** 0.5
    return distance

# Menghitung total jarak rute
def total_distance(route, cities):
    total_distance = 0
    for i in range(len(route)):
        total_distance += distance(cities[route[i]], cities[route[(i + 1) % len(route)]])
    return total_distance
```

FUNGSI DISTANCE(CITY1, CITY2) : MENGHITUNG JARAK ANTARA DUA KOTA MENGGUNAKAN FORMULA EUCLIDEAN DISTANCE.
FUNGSI TOTAL_DISTANCE(ROUTE, CITIES) : MENGHITUNG TOTAL JARAK YANG DITEMPUH PADA SUATU RUTE. FUNGSI INI AKAN MEMANGGIL FUNGSI DISTANCE UNTUK MENGHITUNG JARAK ANTAR KOTA-KOTA YANG ADA DI DALAM RUTE.

EXIT



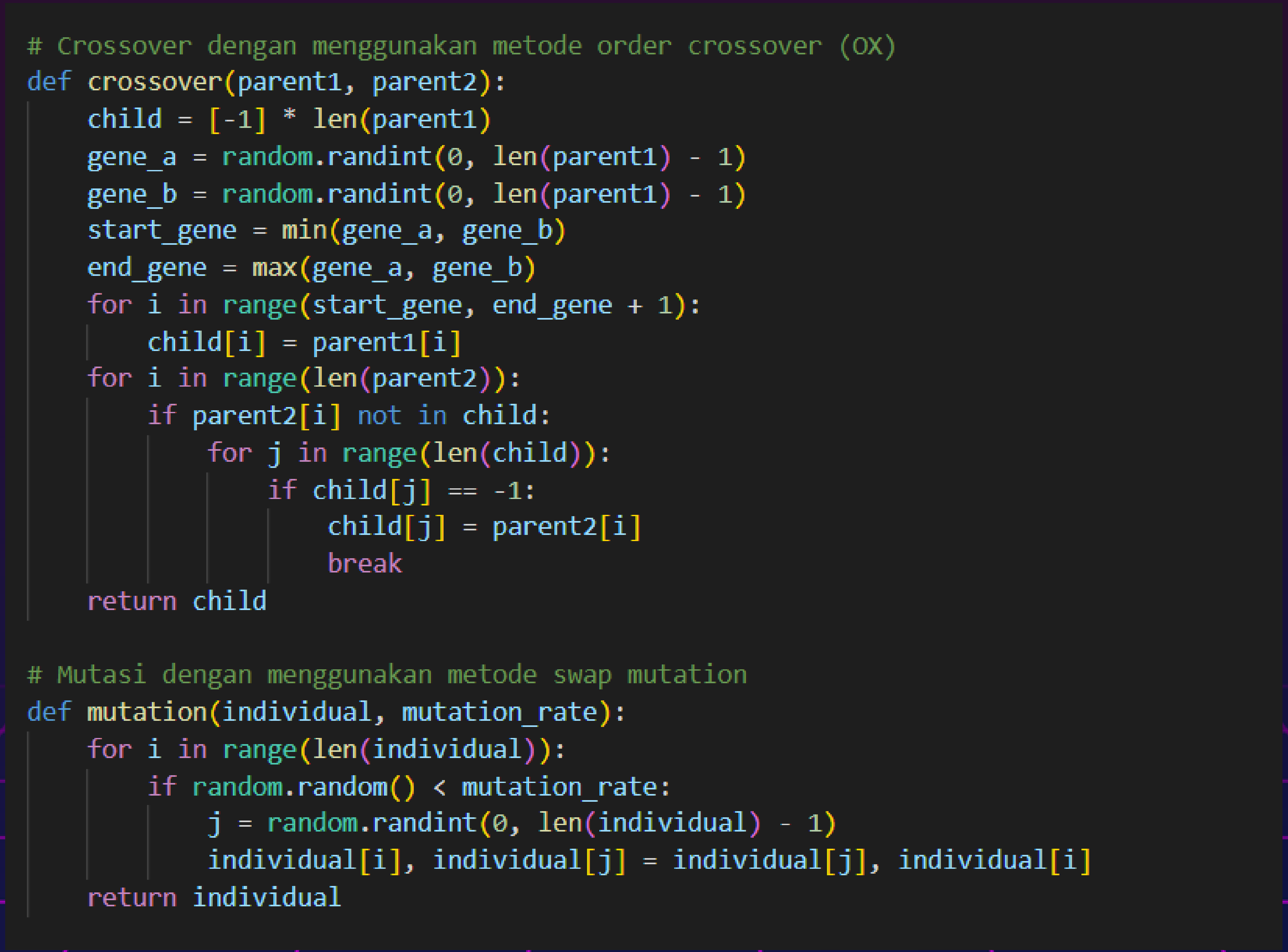
```python
# Membuat populasi awal dengan jumlah individu yang diinginkan
def create_population(num_individuals, city_list):
    population = []
    for i in range(num_individuals):
        individual = list(range(len(city_list)))
        random.shuffle(individual)
        population.append(individual)
    return population

# Seleksi orangtua dengan menggunakan turnamen
def selection_tournament(population, tournament_size):
    tournament = random.sample(population, tournament_size)
    best = tournament[0]
    for individual in tournament:
        if total_distance(individual, city_list) < total_distance(best, city_list):
            best = individual
    return best
```

EXIT

FUNGSI CREATE_POPULATION(NUM_INDIVIDUALS, CITY_LIST) : MEMBUAT POPULASI AWAL DENGAN JUMLAH INDIVIDU YANG DIINGINKAN. FUNGSI INI AKAN MEMBUAT INDIVIDU-ACAK, YAITU DAFTAR KOTA YANG DIACAK UNTUK MEMBENTUK RUTE.

FUNGSI SELECTION_TOURNAMENT(POPULATION, TOURNAMENT_SIZE) : SELEKSI ORANGTUA DENGAN MENGGUNAKAN METODE TURNAMEN. FUNGSI INI AKAN MEMILIH BEBERAPA INDIVIDU SECARA ACAK DARI POPULASI DAN MEMILIH INDIVIDU DENGAN RUTE TERPENDEK DARI KELOMPOK TERSEBUT.

EXIT



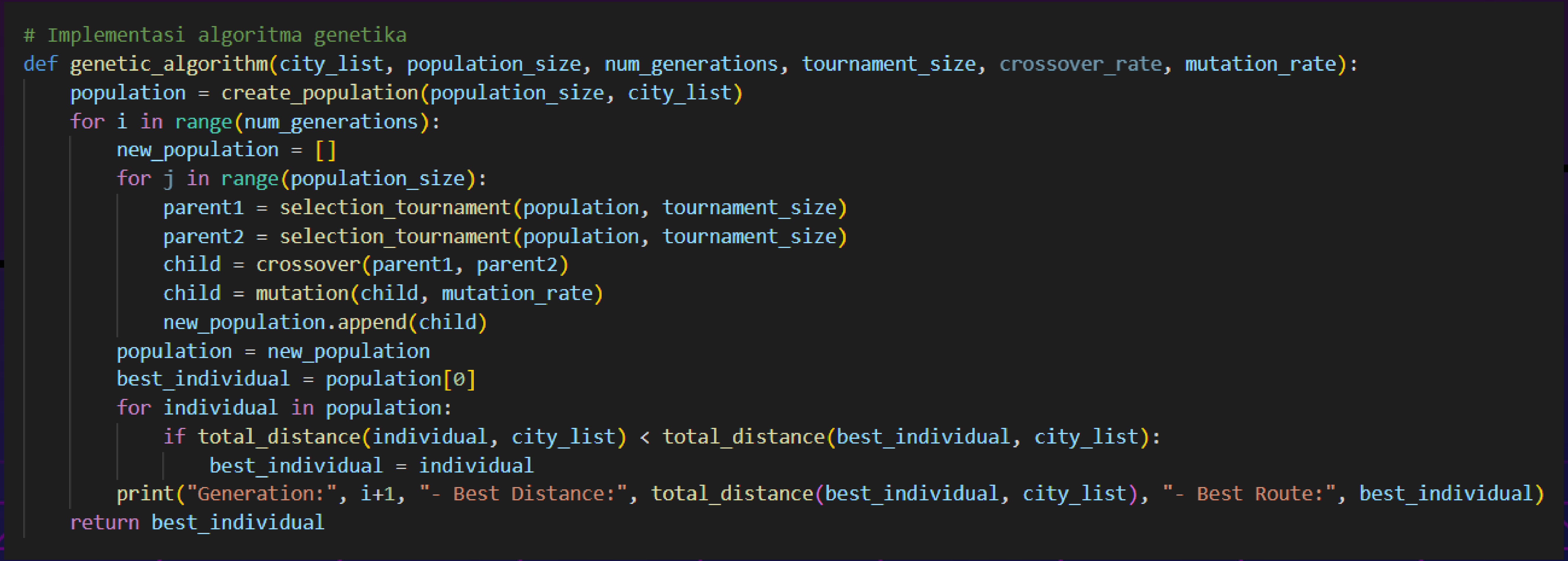
```python
# Crossover dengan menggunakan metode order crossover (OX)
def crossover(parent1, parent2):
    child = [-1] * len(parent1)
    gene_a = random.randint(0, len(parent1) - 1)
    gene_b = random.randint(0, len(parent1) - 1)
    start_gene = min(gene_a, gene_b)
    end_gene = max(gene_a, gene_b)
    for i in range(start_gene, end_gene + 1):
        child[i] = parent1[i]
    for i in range(len(parent2)):
        if parent2[i] not in child:
            for j in range(len(child)):
                if child[j] == -1:
                    child[j] = parent2[i]
                    break
    return child

# Mutasi dengan menggunakan metode swap mutation
def mutation(individual, mutation_rate):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            j = random.randint(0, len(individual) - 1)
            individual[i], individual[j] = individual[j], individual[i]
    return individual
```

EXIT

FUNGSI CROSSOVER(PARENT1, PARENT2) :
MENERAPKAN OPERASI CROSSOVER DENGAN MENGGUNAKAN METODE ORDER CROSSOVER (OX). FUNGSI INI AKAN MEMILIH DUA GEN SECARA ACAK DARI ORANGTUA, DAN MENYALIN GEN TERSEBUT KE DALAM KETURUNAN. KEMUDIAN, GEN LAINNYA DIAMBIL DARI ORANGTUA KEDUA, TETAPI GEN YANG TELAH TERPILIH TIDAK DIAMBIL.

FUNGSI MUTATION(INDIVIDUAL, MUTATION_RATE) :
MENERAPKAN OPERASI MUTASI DENGAN MENGGUNAKAN METODE SWAP MUTATION. FUNGSI INI AKAN MENUKAR POSISI DUA KOTA SECARA ACAK DALAM RUTE.

EXIT



```python
# Implementasi algoritma genetika
def genetic_algorithm(city_list, population_size, num_generations, tournament_size, crossover_rate, mutation_rate):
    population = create_population(population_size, city_list)
    for i in range(num_generations):
        new_population = []
        for j in range(population_size):
            parent1 = selection_tournament(population, tournament_size)
            parent2 = selection_tournament(population, tournament_size)
            child = crossover(parent1, parent2)
            child = mutation(child, mutation_rate)
            new_population.append(child)
        population = new_population
        best_individual = population[0]
        for individual in population:
            if total_distance(individual, city_list) < total_distance(best_individual, city_list):
                best_individual = individual
        print("Generation:", i+1, "- Best Distance:", total_distance(best_individual, city_list), "- Best Route:", best_individual)
    return best_individual
```

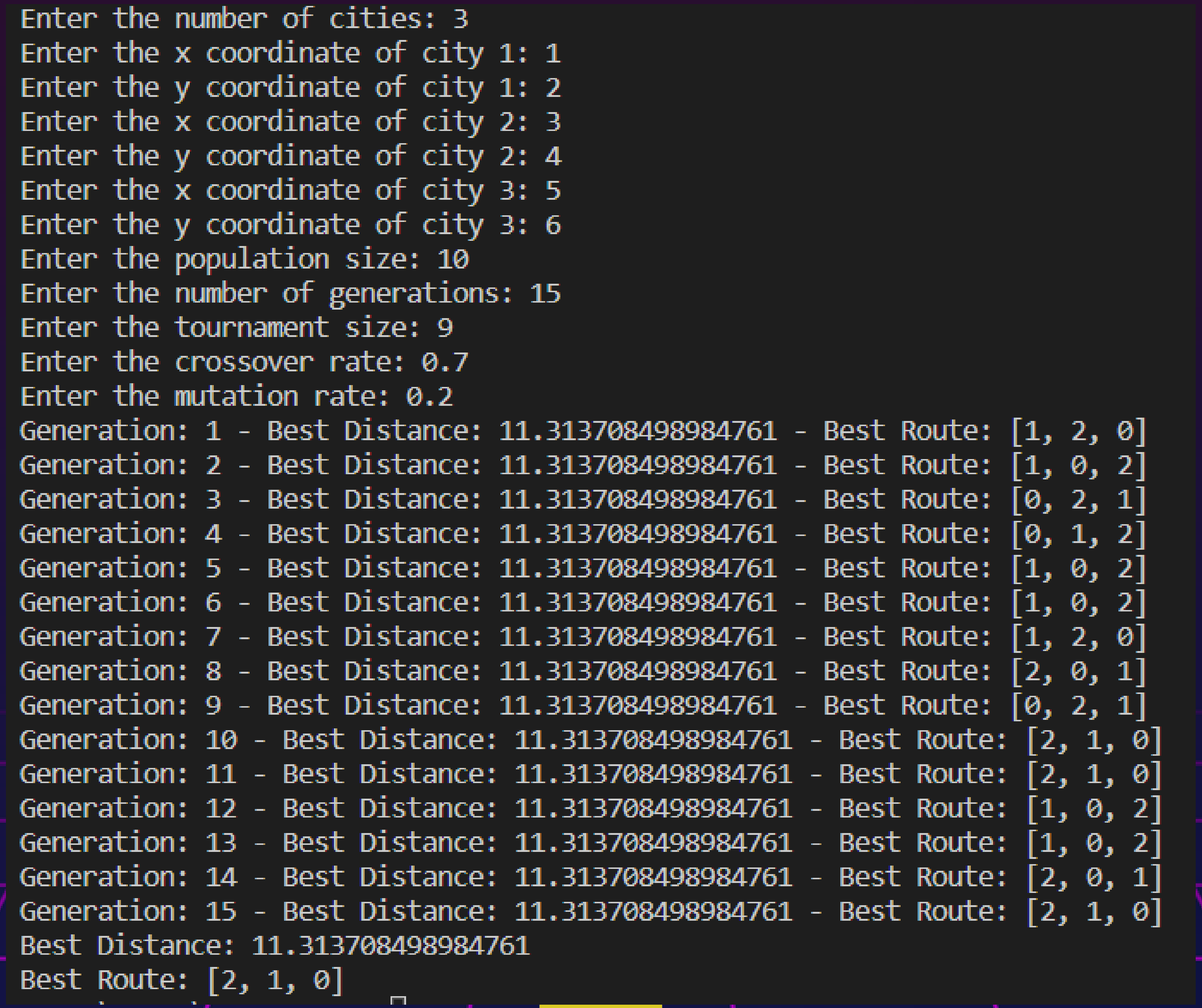
EXIT

FUNGSI GENETIC_ALGORITHM(CITY_LIST, POPULATION_SIZE, NUM_GENERATIONS, TOURNAMENT_SIZE, CROSSOVER_RATE, MUTATION_RATE) :

MENJALANKAN ALGORITMA GENETIKA UNTUK MENYELESAIKAN TSP.
FUNGSI INI AKAN MEMBUAT POPULASI AWAL,
MELAKUKAN SELEKSI ORANGTUA, MENERAPKAN OPERASI CROSSOVER DAN MUTASI, DAN MENGHASILKAN POPULASI BARU UNTUK GENERASI BERIKUTNYA.
KEMUDIAN, FUNGSI AKAN MEMILIH RUTE TERBAIK DARI POPULASI DAN MENCETAK HASILNYA.

EXIT

CONTOH OUTPUT



```
Enter the number of cities: 3
Enter the x coordinate of city 1: 1
Enter the y coordinate of city 1: 2
Enter the x coordinate of city 2: 3
Enter the y coordinate of city 2: 4
Enter the x coordinate of city 3: 5
Enter the y coordinate of city 3: 6
Enter the population size: 10
Enter the number of generations: 15
Enter the tournament size: 9
Enter the crossover rate: 0.7
Enter the mutation rate: 0.2
Generation: 1 - Best Distance: 11.313708498984761 - Best Route: [1, 2, 0]
Generation: 2 - Best Distance: 11.313708498984761 - Best Route: [1, 0, 2]
Generation: 3 - Best Distance: 11.313708498984761 - Best Route: [0, 2, 1]
Generation: 4 - Best Distance: 11.313708498984761 - Best Route: [0, 1, 2]
Generation: 5 - Best Distance: 11.313708498984761 - Best Route: [1, 0, 2]
Generation: 6 - Best Distance: 11.313708498984761 - Best Route: [1, 0, 2]
Generation: 7 - Best Distance: 11.313708498984761 - Best Route: [1, 2, 0]
Generation: 8 - Best Distance: 11.313708498984761 - Best Route: [2, 0, 1]
Generation: 9 - Best Distance: 11.313708498984761 - Best Route: [0, 2, 1]
Generation: 10 - Best Distance: 11.313708498984761 - Best Route: [2, 1, 0]
Generation: 11 - Best Distance: 11.313708498984761 - Best Route: [2, 1, 0]
Generation: 12 - Best Distance: 11.313708498984761 - Best Route: [1, 0, 2]
Generation: 13 - Best Distance: 11.313708498984761 - Best Route: [1, 0, 2]
Generation: 14 - Best Distance: 11.313708498984761 - Best Route: [2, 0, 1]
Generation: 15 - Best Distance: 11.313708498984761 - Best Route: [2, 1, 0]
Best Distance: 11.313708498984761
Best Route: [2, 1, 0]
```

EXIT

THANK YOU VERY MUCH !

EXIT