

Code:

```
import numpy as np
import random

class Chromosome:
    def __init__(self, genes):
        self.genes = genes
        self.fitness = 0
        self.calculate_fitness()

    def calculate_fitness(self):
        self.fitness = 0
        for i, gene in enumerate(self.genes):
            self.fitness += (i+1)*(gene**2)

class GeneticAlgorithm:
    def __init__(self):
        self.n_chromosomes = 0
        self.n_genes = 0
        self.population = []
        self.max_fitness = 0
        # self.fitness_graph = []
        self.generate_population()
        self.calculate_max_fitness()

    def generate_population(self):
        self.n_chromosomes = int(input("Enter number of chromosomes: "))
        self.n_genes = int(input("Enter number of genes: "))
        print("Chromosomes: {} \n Genes: {} \n".format(self.n_chromosomes, self.n_genes))
        self.population = []
        for i in range(self.n_chromosomes):
            genes = np.random.standard_normal(size=self.n_genes).tolist()
            self.population.append(Chromosome(genes))

    def calculate_max_fitness(self):
        self.max_fitness = 0
        for i in range(self.n_genes):
            self.max_fitness += (i+1)*100

    def selection(self):
        population_size = (self.n_chromosomes*2)//2
        selected_population = []
        while len(selected_population) != self.n_chromosomes:
            # Creating tournament
```

```

        tournament_size = population_size//2 if population_size//10 < 5 else size//10
        tournament_population = random.sample(self.population, tournament_size)
        # Appending winner in selected population to generate offsprings
        winner = max(tournament_population, key = lambda chromosome :
chromosome.fitness)
        #print(winner.genes)
        selected_population.append(winner)
        return selected_population

def cross_over(self, selected_population):
    offsprings = []
    while len(offsprings) != self.n_chromosomes:
        # Randomly generating crossover point
        crossover_point = random.randint(0,self.n_genes-2)
        # Randomly selecting two parents
        parent_a, parent_b = random.sample(selected_population, 2)
        # Generating offsprings by swapping genes
        parent_a.genes[crossover_point:self.n_genes],
parent_b.genes[crossover_point:self.n_genes] = parent_b.genes[crossover_point:self.n_genes],
parent_a.genes[crossover_point:self.n_genes]
        offsprings += [parent_a, parent_b]
    return offsprings

def mutation(self, offsprings):
    mutation_range = random.randint(1,len(offsprings)//3)
    random_offsprings = random.sample(range(0, len(offsprings)-1), mutation_range)
    for i in random_offsprings:
        index = random.randint(0,self.n_genes-1)
        offsprings[i].genes[index] = random.uniform(0, 10)
    return offsprings

def replacement(self, offsprings):
    self.population = offsprings
    for i in range(self.n_chromosomes):
        self.population[i].calculate_fitness()

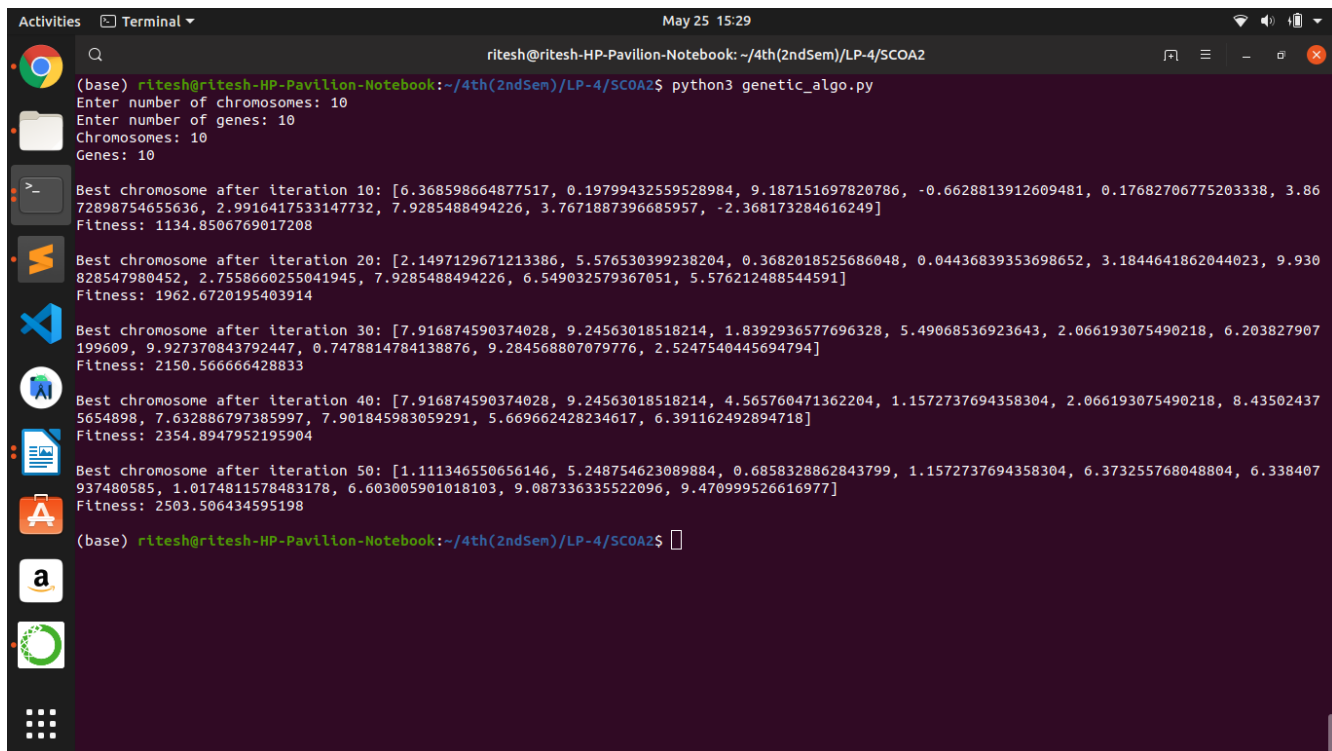
def check_termination(self, i):
    best_chromosome = max(self.population, key = lambda chromosome :
chromosome.fitness)
    if i%10==0:
        print("Best chromosome after iteration {}: {} \nFitness: {} \n".format(i,best_chromosome.genes,best_chromosome.fitness))
        # self.fitness_graph.append(best_chromosome.fitness)
    if best_chromosome.fitness >= self.max_fitness:
        print("Termination criteria reached")

```

```
    return True
return False
```

```
genetic_algorithm = GeneticAlgorithm()
for i in range(50):
    selected_population = genetic_algorithm.selection()
    offsprings = genetic_algorithm.cross_over(selected_population)
    offsprings = genetic_algorithm.mutation(offsprings)
    genetic_algorithm.replacement(offsprings)
    if genetic_algorithm.check_termination(i+1):
        break
```

Output:



```
ritesh@ritesh-HP-Pavillon-Notebook: ~/4th(2ndSem)/LP-4/SCOA2
(base) ritesh@ritesh-HP-Pavillon-Notebook:~/4th(2ndSem)/LP-4/SCOA2$ python3 genetic_algo.py
Enter number of chromosomes: 10
Enter number of genes: 10
Chromosomes: 10
Genes: 10

Best chromosome after iteration 10: [6.368598664877517, 0.19799432559528984, 9.187151697820786, -0.6628813912609481, 0.17682706775203338, 3.86
72898754655636, 2.9916417533147732, 7.9285488494226, 3.7671887396685957, -2.368173284616249]
Fitness: 1134.8506769017208

Best chromosome after iteration 20: [2.1497129671213386, 5.576530399238204, 0.3682018525686048, 0.04436839353698652, 3.1844641862044023, 9.930
828547980452, 2.7558660255041945, 7.9285488494226, 6.549032579367051, 5.576212488544591]
Fitness: 1962.6720195403914

Best chromosome after iteration 30: [7.916874590374028, 9.24563018518214, 1.8392936577696328, 5.49068536923643, 2.066193075490218, 6.203827907
199609, 9.927370843792447, 0.7478814784138876, 9.284568807079776, 2.5247540445694794]
Fitness: 2150.566666428833

Best chromosome after iteration 40: [7.916874590374028, 9.24563018518214, 4.565760471362204, 1.1572737694358304, 2.066193075490218, 8.43502437
5654898, 7.632886797385997, 7.901845983059291, 5.669662428234617, 6.391162492894718]
Fitness: 2354.8947952195904

Best chromosome after iteration 50: [1.111346550656146, 5.248754623089884, 0.6858328862843799, 1.1572737694358304, 6.373255768048804, 6.338407
937480585, 1.0174811578483178, 6.603005901018103, 9.087336335522096, 9.470999526616977]
Fitness: 2503.506434595198

(base) ritesh@ritesh-HP-Pavillon-Notebook:~/4th(2ndSem)/LP-4/SCOA2$
```