```
#python 3
#author: Nikhil Yadav

import random

POPULATION_SIZE = 6
MUTATION_RATE = 0.1
CROSSOVER_RATE = 0.25
GENERATIONS = 50
LOWER_LIMIT = 0
UPPER_LIMIT = 30

class Chromosomes:
    def initialize(self):       #initializes the population
        return [random.randint(LOWER_LIMIT,UPPER_LIMIT) for i in range(4)]

class Population:
    def __init__(self):
        self._size = POPULATION_SIZE
        self._chromosomes = []
        for i in range(self._size) : self._chromosomes.append(Chromosomes().initialize())

    def get_size(self):
        return self._size

    def get_chromosomes(self):
        return self._chromosomes

    def set_chromosomes(self,chromosomes):
        self._chromosomes = chromosomes

    def objective_fitness(self):        #returns objective fitness of the population
        obj_fitness = [abs(x[0] + (2*x[1]) + (3*x[2]) + (4*x[3]) - 30) for x in self._chromosomes]
        return obj_fitness

class GeneticAlgo:
    def evolve(self,population):        #evolving the population consists of the steps of
        self.selection(population)      #selection (roulette wheel selection based on fitness)
        self.crossover(population)      #crossover (between randomly selected chromosomes)
        self.mutation(population)       #mutation  (of randomly selection chromosomes)
        return population

    def selection(self,population):     #selection
        size = population.get_size()
        chromosomes = population.get_chromosomes()
        obj_fitness = population.objective_fitness()
        obj_fitness = [x+1 for x in obj_fitness]
        fitness = [1/obj_fitness[i] for i in range(size)]       #convert objective fitness to real fitness
        total_fitness = sum(fitness)
        probability = [fitness[i]/total_fitness for i in range(size)]           #assign probabilities for roulette wheel selection

        cumulative_probability = [sum(probability[:i+1]) for i in range(size)]
```

```
        roulette = [random.random() for i in range(size)]
        new_chromosomes = []
        for r in roulette:              #running the roulette wheel
            for i in range(size):
                if r <= cumulative_probability[i]:
                    new_chromosomes.append(chromosomes[i])
                    break
        population.set_chromosomes(new_chromosomes)         #update new chromosomes in the population

    def crossover(self,population):     #crossover
        size = population.get_size()
        chromosomes = population.get_chromosomes()
        n = len(chromosomes[0])
        rand = [random.random() for i in range(size)]       #generate random numbers for each chromosome
        parent = []
        for i in range(size):
            if rand[i] < CROSSOVER_RATE:            #select chromosomes for crossover
                parent.append([chromosomes[i],i])
        if(len(parent)>0):
            parent.append(parent[0])
        for i in range(len(parent)-1):              #applying crossover over the parents
            rnd = random.randint(1,n-1)
            idx = parent[i][1]
            for j in range(size):
                if j!=idx:
                    chromosomes[j] = tuple(chromosomes[j])
            chromosomes[parent[i][1]][rnd:] = parent[i+1][0][rnd:]
            for j in range(size):
                chromosomes[j] = list(chromosomes[j])
        population.set_chromosomes(chromosomes)         #update new chromosomes in the population

    def mutation(self,population):      #mutation
        size = population.get_size()
        chromosomes = population.get_chromosomes()
        chromo = chromosomes
        n = len(chromo[0])
        total_gen = n*size
        numb_mutations = round(MUTATION_RATE*total_gen)
        rand = [random.randint(1,total_gen) for i in range(numb_mutations)]
        for k in range(numb_mutations):             #applying mutation on selected chromosomes
            i = (rand[k]-1)//n
            j = (rand[k]-1)%n
            c = chromo.pop(i)
            c_ = tuple(c)
            idx = []
            for xy in chromo:
                if xy==c:
                    idx.append(chromo.index(xy))
                    chromo.remove(xy)
            c[j] = random.randint(LOWER_LIMIT,UPPER_LIMIT)          #mutate the chromosome with a randomly ge
erated number
            while(idx):
```

```
            chromo.insert(idx.pop(0),list(c_))
        chromo.insert(i,c)
    population.set_chromosomes(chromo)        #update new chromosomes in the population

class Display:
    def disp(self,population,genNumber):
        size = population.get_size()
        chromosomes = population.get_chromosomes()
        fitness = population.objective_fitness()
        print('\n> Generation '+str(genNumber)+':')
        for i in range(size):
            print('Chromosome '+str(i+1)+':',end=" ")
            print(chromosomes[i])
            print('objective fitness = ',fitness[i])
        if genNumber==GENERATIONS:
            idx = fitness.index(min(fitness))
            print("\n\nBest Possible Chromosome evolved till ",genNumber,"Generations = ",chromosomes[idx]," with o
jective fitness = ",fitness[idx])



print('POPULATION SIZE = '+str(POPULATION_SIZE))
print('CROSSOVER RATE = '+str(CROSSOVER_RATE))
print('MUTATION RATE = '+str(MUTATION_RATE))
print('LOWER_LIMIT = '+str(LOWER_LIMIT))
print('UPPER_LIMIT = '+str(UPPER_LIMIT))
genNumber = 0
population = Population()
geneticAlgorithm = GeneticAlgo()
display = Display()
display.disp(population,genNumber)
while(genNumber < GENERATIONS):     #iterating over the loop till the limit defined by GENERATIONS
    genNumber += 1
    population  = geneticAlgorithm.evolve(population)       #evolve the population in each generation
    display.disp(population,genNumber)
print('\n\n')
```