**Walk through task**

**NAME:** Abdullah Aslam Qazi

**REGNO:** BCS 203263

Genetic Algo

CODE:

import random  
  
*# Number of individuals in each generation*POPULATION\_SIZE = 100  
  
*# Valid genes*GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP  
QRSTUVWXYZ 1234567890, .-;:\_!"#%&/()=?@${[]}'''  
  
*# Target string to be generated*TARGET = "Abdullah736@cust.pk"  
  
  
class Individual(object):  
 *'''  
 Class representing individual in population  
 '''* def \_\_init\_\_(self, chromosome):  
 self.chromosome = chromosome  
 self.fitness = self.cal\_fitness()  
  
 @classmethod  
 def mutated\_genes(self):  
 *'''  
 create random genes for mutation  
 '''* global GENES  
 gene = random.choice(GENES)  
 return gene  
  
 @classmethod  
 def create\_gnome(self):  
 *'''  
 create chromosome or string of genes  
 '''* global TARGET  
 gnome\_len = len(TARGET)  
 return [self.mutated\_genes() for \_ in range(gnome\_len)]  
  
 def mate(self, par2):  
 *'''  
 Perform mating and produce new offspring  
 '''  
  
 # chromosome for offspring* child\_chromosome = []  
 for gp1, gp2 in zip(self.chromosome, par2.chromosome):  
  
 *# random probability* prob = random.random()  
  
 *# if prob is less than 0.45, insert gene  
 # from parent 1* if prob < 0.45:  
 child\_chromosome.append(gp1)  
  
 *# if prob is between 0.45 and 0.90, insert  
 # gene from parent 2* elif prob < 0.90:  
 child\_chromosome.append(gp2)  
  
 *# otherwise insert random gene(mutate),  
 # for maintaining diversity* else:  
 child\_chromosome.append(self.mutated\_genes())  
  
 *# create new Individual(offspring) using  
 # generated chromosome for offspring* return Individual(child\_chromosome)  
  
 def cal\_fitness(self):  
 *'''  
 Calculate fitness score, it is the number of  
 characters in string which differ from target  
 string.  
 '''* global TARGET  
 fitness = 0  
 for gs, gt in zip(self.chromosome, TARGET):  
 if gs != gt: fitness += 1  
 return fitness  
  
  
*# Driver code*def main():  
 global POPULATION\_SIZE  
  
 *# current generation* generation = 1  
  
 found = False  
 population = []  
  
 *# create initial population* for \_ in range(POPULATION\_SIZE):  
 gnome = Individual.create\_gnome()  
 population.append(Individual(gnome))  
  
 while not found:  
  
 *# sort the population in increasing order of fitness score* population = sorted(population, key=lambda x: x.fitness)  
  
 *# if the individual having lowest fitness score ie.  
 # 0 then we know that we have reached to the target  
 # and break the loop* if population[0].fitness <= 0:  
 found = True  
 break  
  
 *# Otherwise generate new offsprings for new generation* new\_generation = []  
  
 *# Perform Elitism, that mean 10% of fittest population  
 # goes to the next generation* s = int((10 \* POPULATION\_SIZE) / 100)  
 new\_generation.extend(population[:s])  
  
 *# From 50% of fittest population, Individuals  
 # will mate to produce offspring* s = int((90 \* POPULATION\_SIZE) / 100)  
 for \_ in range(s):  
 parent1 = random.choice(population[:50])  
 parent2 = random.choice(population[:50])  
 child = parent1.mate(parent2)  
 new\_generation.append(child)  
  
 population = new\_generation  
  
 print("Generation: {}\tString: {}\tFitness: {}". \  
 format(generation,  
 "".join(population[0].chromosome),  
 population[0].fitness))  
  
 generation += 1  
  
 print("Generation: {}\tString: {}\tFitness: {}". \  
 format(generation,  
 "".join(population[0].chromosome),  
 population[0].fitness))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

OUTPUT:



