CS6002 - SOFT COMPUTING

IMPLEMENTATION OF FUZZY LOGIC AND GENETIC ALGORITHM

TABLE OF CONTENTS:

KIRAN SEKHAR C
(2018103553)

KARTHICKRAJA S
(2018103547)

JAYA SURYA V
(2018103541)

IMPLEMENTATION OF FUZZY LOGIC

PROBLEM: Evaluation of Academic Performance of Students using fuzzy logic

ABSTRACT:

Students' academic success is evaluated by their performance in exams conducted by the institutes or Universities. Considering the high demand of IT professionals and the gap between academia and IT Industry it is important that we must explore the possibilities of automated system which can effectively evaluate the performance of students in computer science and IT related courses.

FUZZY LOGIC :

Fuzzy logic is an approach to variable processing that allows for multiple possible truth values to be processed through the same variable. Fuzzy logic attempts to solve problems with an open, imprecise spectrum of data and heuristics that makes it possible to obtain an array of accurate conclusions. Fuzzy logic is designed to solve problems by considering all available information and making the best possible decision given the input.

Fuzzy logic stems from the mathematical study of multivalued logic. Whereas ordinary logic deals with statements of absolute truth (such as, "Is this object green?"), fuzzy logic addresses sets with subjective or relative definitions, such as "tall," "large," or "beautiful." This attempts to mimic the way humans analyze problems and make decisions, in a way that relies on vague or imprecise values rather than absolute truth or falsehood.

In practice, these constructs all allow for partial values of the "true" condition. Instead of requiring all statements to be absolutely true or absolutely false, as in classical logic, the truth values in fuzzy logic can be any value between zero and one. This creates an opportunity for algorithms to make decisions based on ranges of data as opposed to one discrete data point.

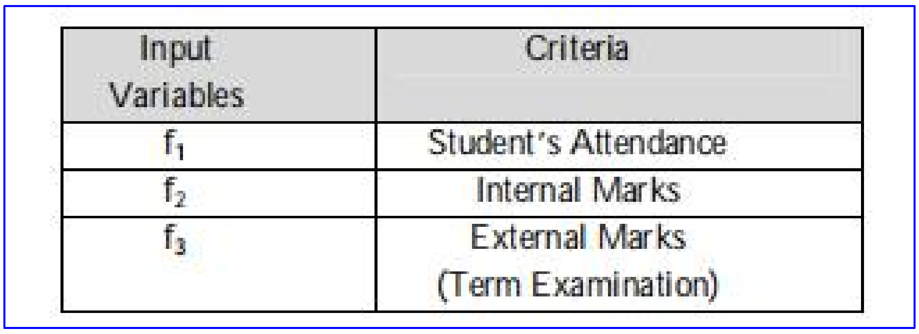
The scaling for the fuzzy sets is very important because the fuzzy system can be renewed with other devices or have different ranges of operation by just trying to change the scaling of the input and output. The decision-making idea figure out how the fuzzy logic operations are performed, and with the knowledge base figure out the outputs of each fuzzy IF-THEN rules that was created. Those at the end are joint and converted to crispy values with the defuzzification block. The output crisp value can then be calculated by the centre of gravity or the weighted average or any other defuzzification method.

METHODOLOGY:

A. Crisp Value (Data)

The values for input variables may be collected from the records of the students' end term result with internal assessment (f1), external Assessment (f2) and overall attendance of the Semester (f3)

TABLE I.INPUT VARIABLES (ELEMENTS) OF THE PROPOSED EVALUATION MODEL



Input Variables	Criteria
f_1	Student's Attendance
f_2	Internal Marks
f_3	External Marks (Term Examination)

B. Fuzzification (Fuzzy Input Value)

Fuzzification of three input variables (elements) is done by using variable which are similar to verbal human language such as average, good, very good, excellent etc.. Then each input variable is assigned a trapezoidal Membership function, defined by a lower limit 'a', an upper limit 'd', a lower support limit 'b', and an upper support limit 'c', where $a < b < c < d$, for the degree of association for respective linguistic variables.

$$\mu_A(x) = \begin{cases} 0, & (x < a) \text{ or } (x > d) \\ \dfrac{x-a}{b-a}, & a \le x \le b \\ 1, & b \le x \le c \\ \dfrac{d-x}{d-c}, & c \le x \le d \end{cases}$$

C. <u>Development of Fuzzy Rule and Inference Mechanism</u>

To relate the inputs and output membership functions, fuzzy inference rules are used in inference process. These linguistics rules use “IF-THEN” statements. These rule are flexible and can be formulated depending upon the importance to be given to a particular input with the discussion with the academic experts.

D. <u>Defuzzification Of Fuzzy Output (To Find Out The Final Result With The Help Of Suitable Defuzzification Method)</u>

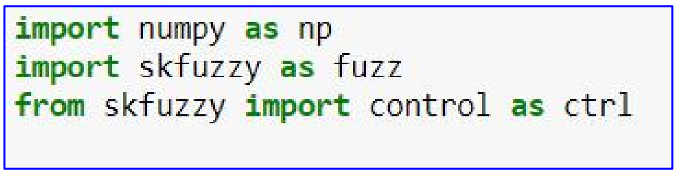
The output variable F is the students’ final performance. If the three input variables are expressed as f1, f2, f3 and membership functions of the three input variables are μ(f1), μ(f2), μ(f3.) respectively for rule k=1,2,3,4,........r, then
The membership function of the output variable F is given by equation,

$$\mu F(x)=Maxk\ [\min[\mu(f1),\mu(f2),\mu(f3)]],\ k=1,2,3,4,.....r$$

This expression expresses the value of membership function for output variable overall performance for active rules for each input.

IMPLEMENTATION :

Using the python functions below, we implement membership functions of fuzzy sets.



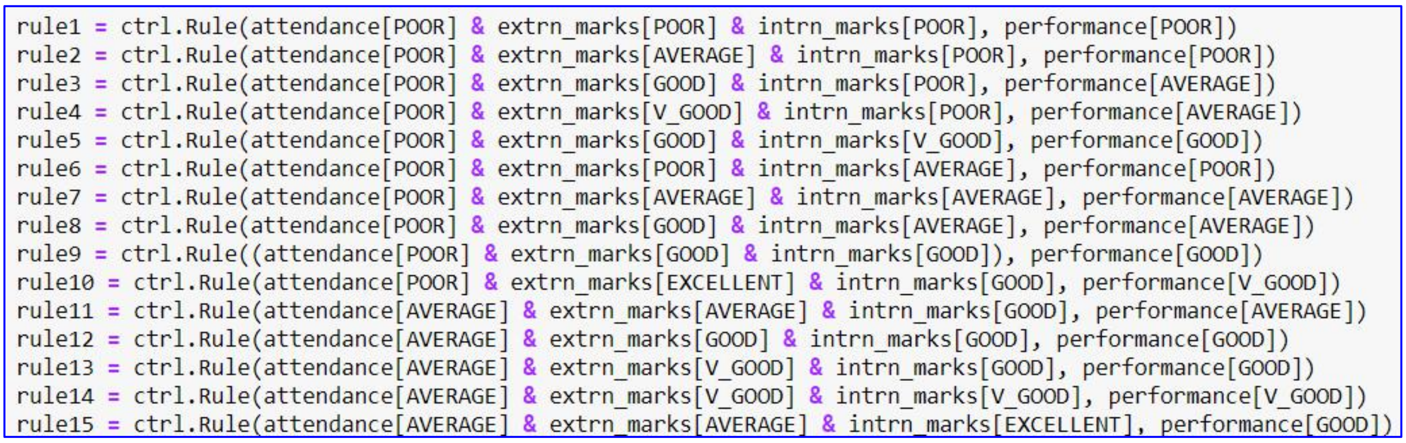
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

```python
ATTENDANCE = 'Attendance'
PERFORMANCE = 'Performance'
INTERNAL_MARKS = 'Internal_Marks'
EXTERNAL_MARKS = 'External_Marks'
```

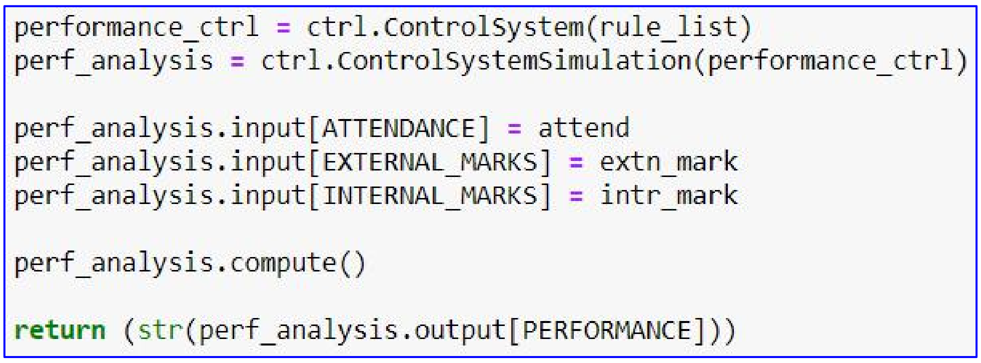
```python
POOR = 'Poor'
AVERAGE = 'Average'
GOOD = 'Good'
V_GOOD = 'V.Good'
EXCELLENT = 'Excellent'
```

```python
low_parameter = [0,0,40,50]
average_parameter = [30,40,50,60]
good_parameter = [40,50,60,70]
v_good_parameter = [50,60,70,80]
excellent_parameter = [65,80,100,100]
```

```python
def compute_fuzzy(attend, intr_mark, extn_mark):

    intrn_marks = ctrl.Antecedent(np.arange(0,105,5), INTERNAL_MARKS)
    attendance = ctrl.Antecedent(np.arange(0,105,5), ATTENDANCE)
    extrn_marks = ctrl.Antecedent(np.arange(0,105,5), EXTERNAL_MARKS)
    performance = ctrl.Consequent(np.arange(0,105,5), PERFORMANCE)

    intrn_marks[POOR] = fuzz.trapmf(intrn_marks.universe, low_parameter)
    intrn_marks[AVERAGE] = fuzz.trapmf(intrn_marks.universe, average_parameter)
    intrn_marks[GOOD] = fuzz.trapmf(intrn_marks.universe, good_parameter)
    intrn_marks[V_GOOD] = fuzz.trapmf(intrn_marks.universe, v_good_parameter)
    intrn_marks[EXCELLENT] = fuzz.trapmf(intrn_marks.universe, excellent_parameter)

    attendance[POOR] = fuzz.trapmf(attendance.universe, [0,0,45,55])
    attendance[AVERAGE] = fuzz.trapmf(attendance.universe, [35,45,55,65])
    attendance[GOOD] = fuzz.trapmf(attendance.universe, [45,55,65,75])
    attendance[V_GOOD] = fuzz.trapmf(attendance.universe, [55,65,75,85])
    attendance[EXCELLENT] = fuzz.trapmf(attendance.universe, [65,75,100,100])

    extrn_marks[POOR] = fuzz.trapmf(extrn_marks.universe, low_parameter)
    extrn_marks[AVERAGE] = fuzz.trapmf(extrn_marks.universe, average_parameter)
    extrn_marks[GOOD] = fuzz.trapmf(extrn_marks.universe, good_parameter)
    extrn_marks[V_GOOD] = fuzz.trapmf(extrn_marks.universe, v_good_parameter)
    extrn_marks[EXCELLENT] = fuzz.trapmf(extrn_marks.universe, excellent_parameter)

    performance[POOR] = fuzz.trapmf(performance.universe, low_parameter)
    performance[AVERAGE] = fuzz.trapmf(performance.universe, average_parameter)
    performance[GOOD] = fuzz.trapmf(performance.universe, good_parameter)
    performance[V_GOOD] = fuzz.trapmf(performance.universe, v_good_parameter)
    performance[EXCELLENT] = fuzz.trapmf(performance.universe, excellent_parameter)
```



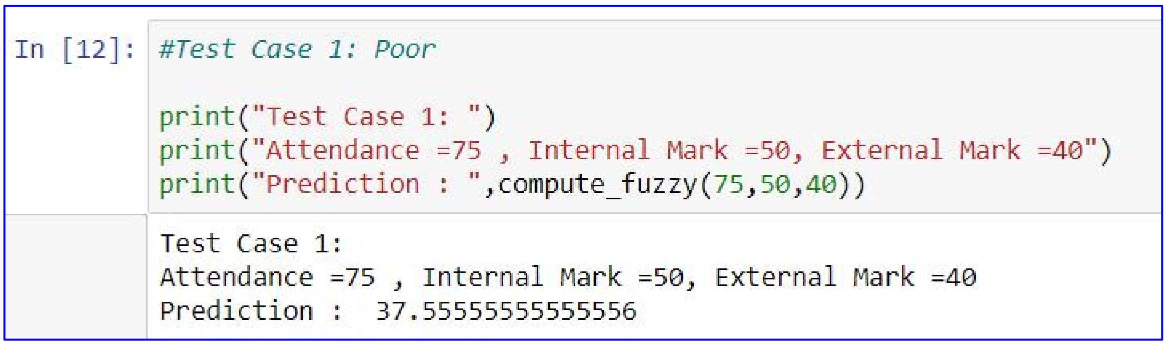
```
rule1 = ctrl.Rule(attendance[POOR] & extrn_marks[POOR] & intrn_marks[POOR], performance[POOR])
rule2 = ctrl.Rule(attendance[POOR] & extrn_marks[AVERAGE] & intrn_marks[POOR], performance[POOR])
rule3 = ctrl.Rule(attendance[POOR] & extrn_marks[GOOD] & intrn_marks[POOR], performance[AVERAGE])
rule4 = ctrl.Rule(attendance[POOR] & extrn_marks[V_GOOD] & intrn_marks[POOR], performance[AVERAGE])
rule5 = ctrl.Rule(attendance[POOR] & extrn_marks[GOOD] & intrn_marks[V_GOOD], performance[GOOD])
rule6 = ctrl.Rule(attendance[POOR] & extrn_marks[POOR] & intrn_marks[AVERAGE], performance[POOR])
rule7 = ctrl.Rule(attendance[POOR] & extrn_marks[AVERAGE] & intrn_marks[AVERAGE], performance[AVERAGE])
rule8 = ctrl.Rule(attendance[POOR] & extrn_marks[GOOD] & intrn_marks[AVERAGE], performance[AVERAGE])
rule9 = ctrl.Rule((attendance[POOR] & extrn_marks[GOOD] & intrn_marks[GOOD]), performance[GOOD])
rule10 = ctrl.Rule(attendance[POOR] & extrn_marks[EXCELLENT] & intrn_marks[GOOD], performance[V_GOOD])
rule11 = ctrl.Rule(attendance[AVERAGE] & extrn_marks[AVERAGE] & intrn_marks[GOOD], performance[AVERAGE])
rule12 = ctrl.Rule(attendance[AVERAGE] & extrn_marks[GOOD] & intrn_marks[GOOD], performance[GOOD])
rule13 = ctrl.Rule(attendance[AVERAGE] & extrn_marks[V_GOOD] & intrn_marks[GOOD], performance[GOOD])
rule14 = ctrl.Rule(attendance[AVERAGE] & extrn_marks[V_GOOD] & intrn_marks[V_GOOD], performance[V_GOOD])
rule15 = ctrl.Rule(attendance[AVERAGE] & extrn_marks[AVERAGE] & intrn_marks[EXCELLENT], performance[GOOD])
```



```
performance_ctrl = ctrl.ControlSystem(rule_list)
perf_analysis = ctrl.ControlSystemSimulation(performance_ctrl)

perf_analysis.input[ATTENDANCE] = attend
perf_analysis.input[EXTERNAL_MARKS] = extn_mark
perf_analysis.input[INTERNAL_MARKS] = intr_mark

perf_analysis.compute()

return (str(perf_analysis.output[PERFORMANCE]))
```

TESTCASES:

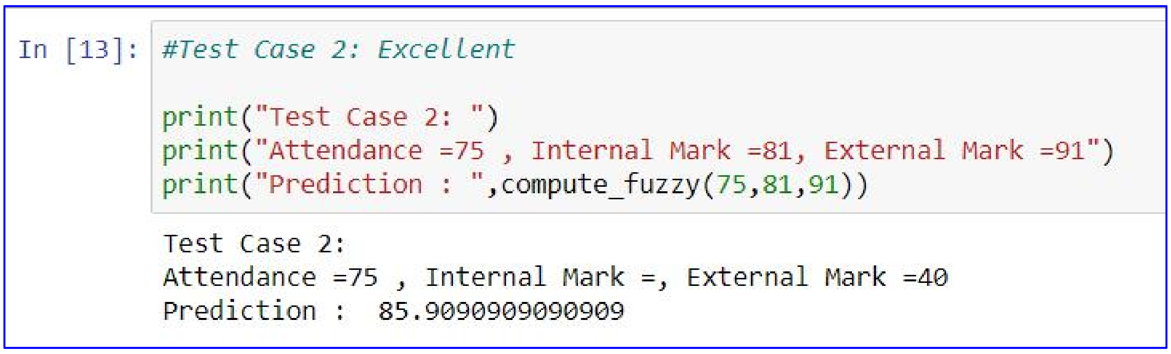
TC1:



```
In [12]: #Test Case 1: Poor

         print("Test Case 1: ")
         print("Attendance =75 , Internal Mark =50, External Mark =40")
         print("Prediction : ",compute_fuzzy(75,50,40))
```

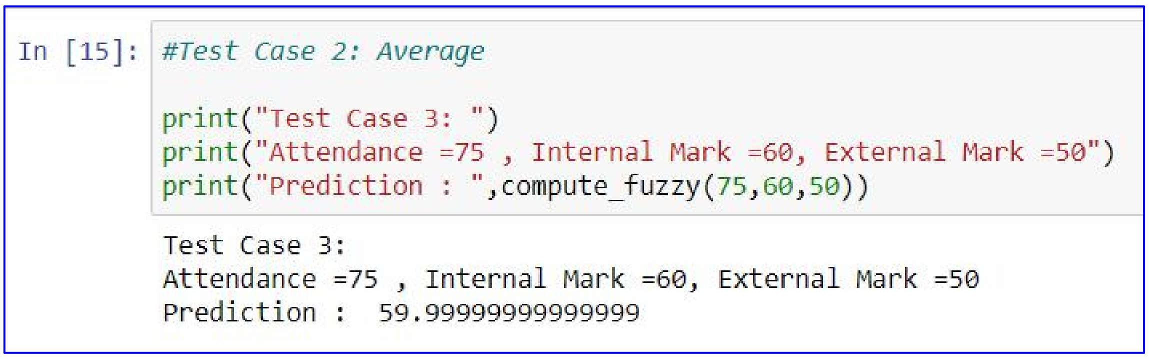
```
Test Case 1:
Attendance =75 , Internal Mark =50, External Mark =40
Prediction :  37.5555555555556
```

TC2:



```
In [13]: #Test Case 2: Excellent

         print("Test Case 2: ")
         print("Attendance =75 , Internal Mark =81, External Mark =91")
         print("Prediction : ",compute_fuzzy(75,81,91))

         Test Case 2:
         Attendance =75 , Internal Mark =, External Mark =40
         Prediction :  85.9090909090909
```

TC3:



```
In [15]: #Test Case 2: Average

         print("Test Case 3: ")
         print("Attendance =75 , Internal Mark =60, External Mark =50")
         print("Prediction : ",compute_fuzzy(75,60,50))

         Test Case 3:
         Attendance =75 , Internal Mark =60, External Mark =50
         Prediction :  59.99999999999999
```

CONCLUSION:

Therefore one can apply computer based Fuzzy System Approach in plane of time consuming conventional method. However, in some cases, the variations in results from fuzzy system have been observed for some students who have same result through conventional method. It was due the difference in their attendance which shows that expert system incorporates input attendance effectively.

On the contrary in the conventional system, for a regular course, a student must have mandatory attendance failing to which the student may not be allowed to appear in exams. This shows that the expert system provides flexibility to the inflexible conventional system which is greatly required in present age of technology.

IMPLEMENTATION OF GENETIC ALGORITHM

PROBLEM : TRAVELLING SALESMAN PROBLEM USING GENETIC ALGORITHM

Genetic Algorithm:

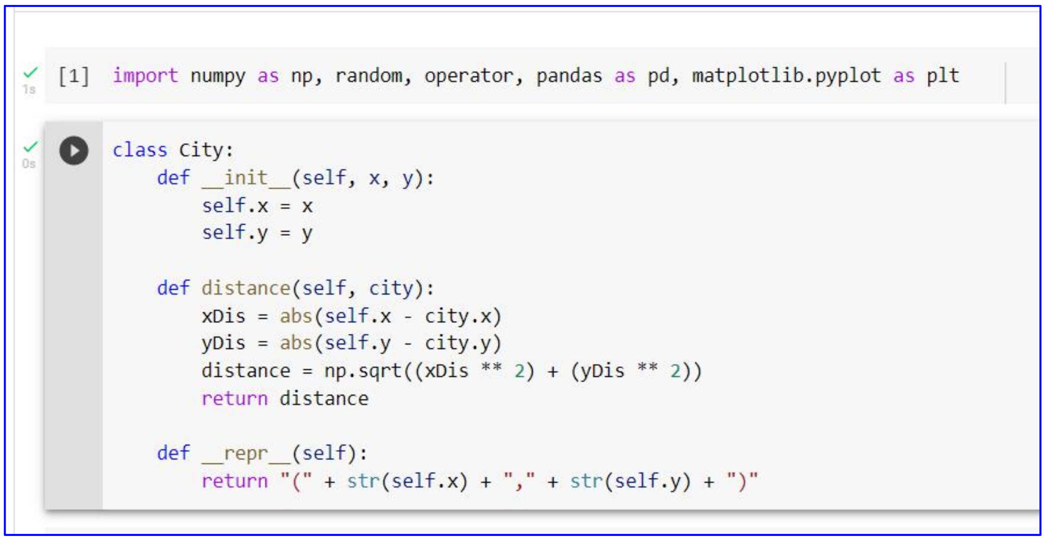
Genetic algorithms (GAs) are stochastic search algorithms that mimic the biological process of evolution enabling thereby users to solve complex optimization problems. They operate based on a population of chromosomes, where a chromosome represents a candidate solution. *Genetic operators* allow the population to evolve over time and to converge to the optimal solution. In the spirit of "*survival of the fittest*," GAs are naturally designed to solve maximization problems in which chromosomes with high fitness are more likely to survive and the objective function is termed fitness function.

Travelling Salesman Problem:

The traveling salesman problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that must be visited. In the problem statement the points are the cities a salesperson might visit. TSP has many important applications, such as in logistics, planning, and scheduling. The problem has been studied for decades, and many traditional optimization algorithms have been proposed to solve it, such as dynamic programming and branch-and-bound. Although these optimization algorithms are capable of solving TSP with dozens of nodes, it is usually intractable to use these algorithms to solve optimally above thousands of nodes on modern computers due to their exponential execution times.

IMPLEMENTATION:

Initially, we import required modules of numpy,pandas and matplot lib.

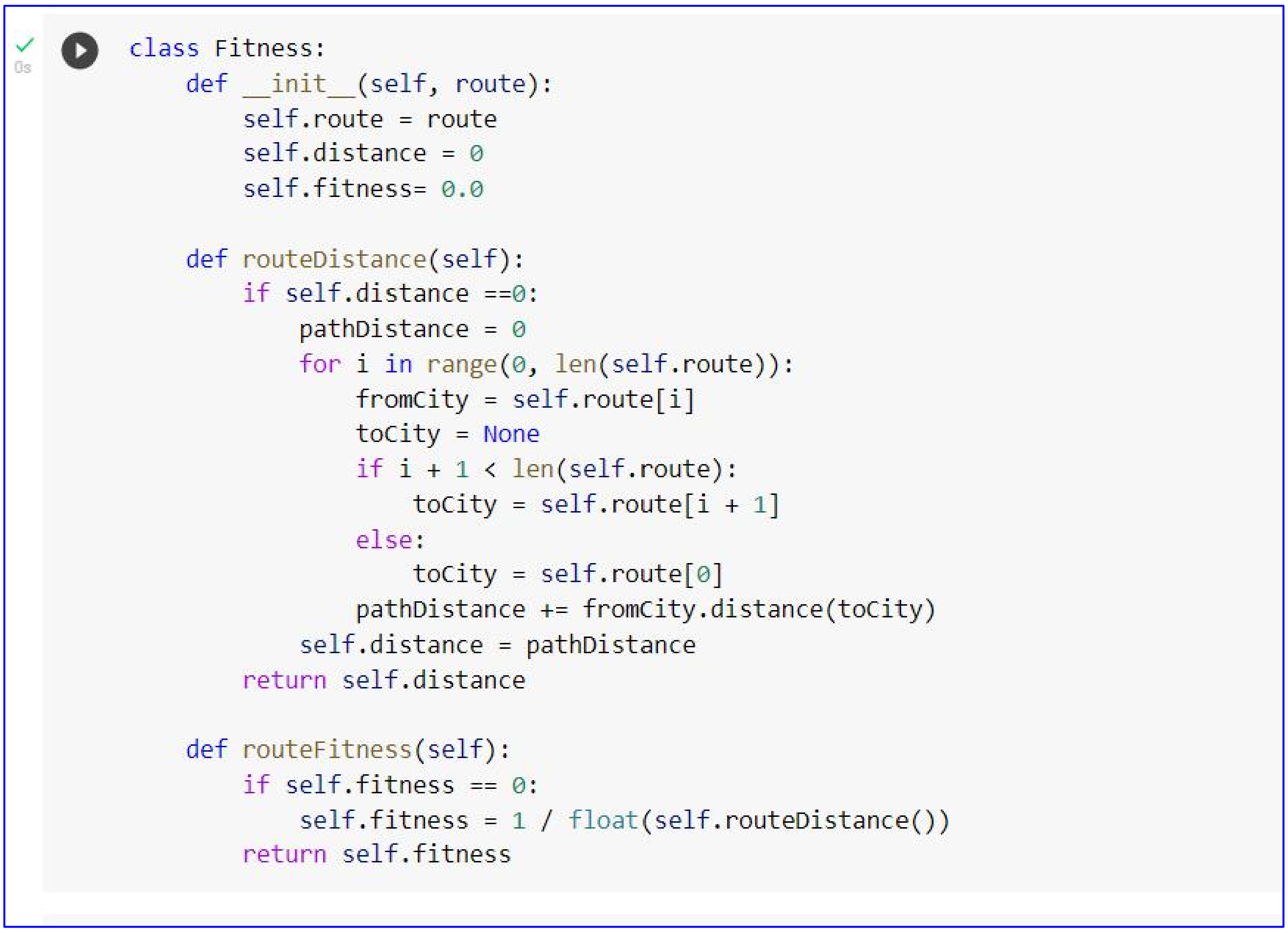


```python
import numpy as np, random, operator, pandas as pd, matplotlib.pyplot as plt
```

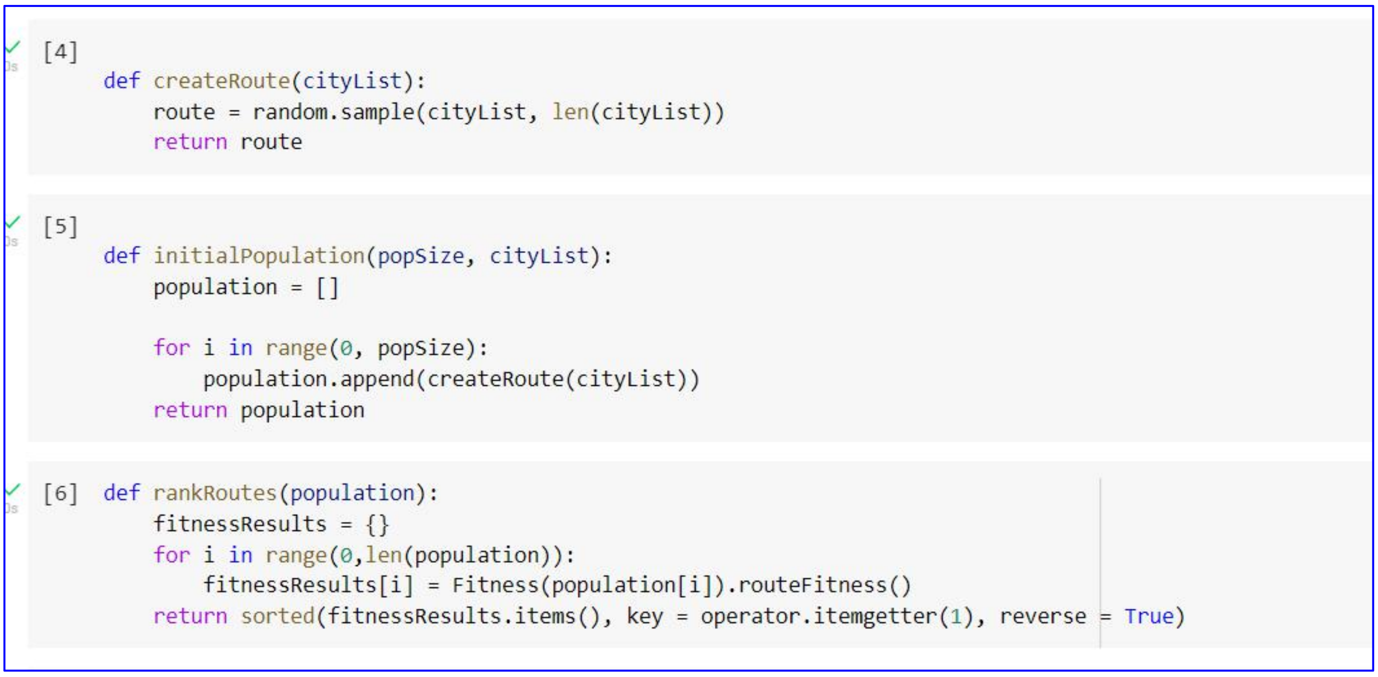
```python
class City:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, city):
        xDis = abs(self.x - city.x)
        yDis = abs(self.y - city.y)
        distance = np.sqrt((xDis ** 2) + (yDis ** 2))
        return distance

    def __repr__(self):
        return "(" + str(self.x) + "," + str(self.y) + ")"
```

Fitness Function:

The fitness function simply defined as a function which takes a candidate solution to the problem as input and produces as output how “fit” or how “good” the solution is with respect to the problem in consideration.Calculation of fitness value is done repeatedly in genetic algorithm and is sufficiently fast.



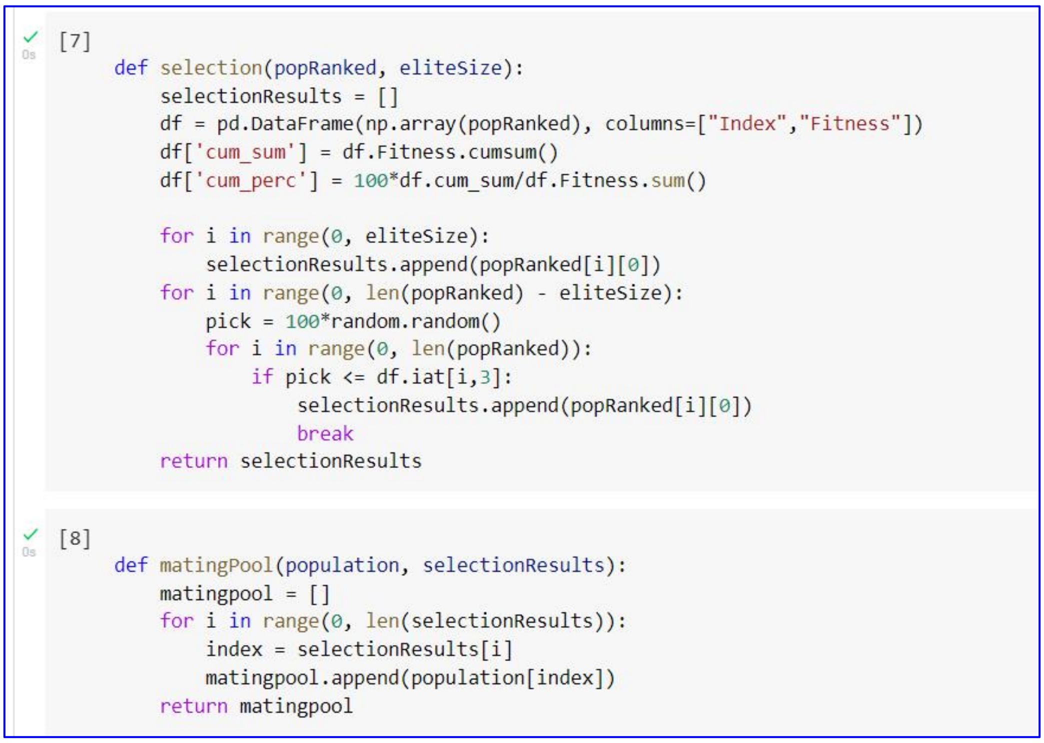
```python
class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0

    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromCity = self.route[i]
                toCity = None
                if i + 1 < len(self.route):
                    toCity = self.route[i + 1]
                else:
                    toCity = self.route[0]
                pathDistance += fromCity.distance(toCity)
            self.distance = pathDistance
        return self.distance

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
```



```
[4]
def createRoute(cityList):
    route = random.sample(cityList, len(cityList))
    return route
```

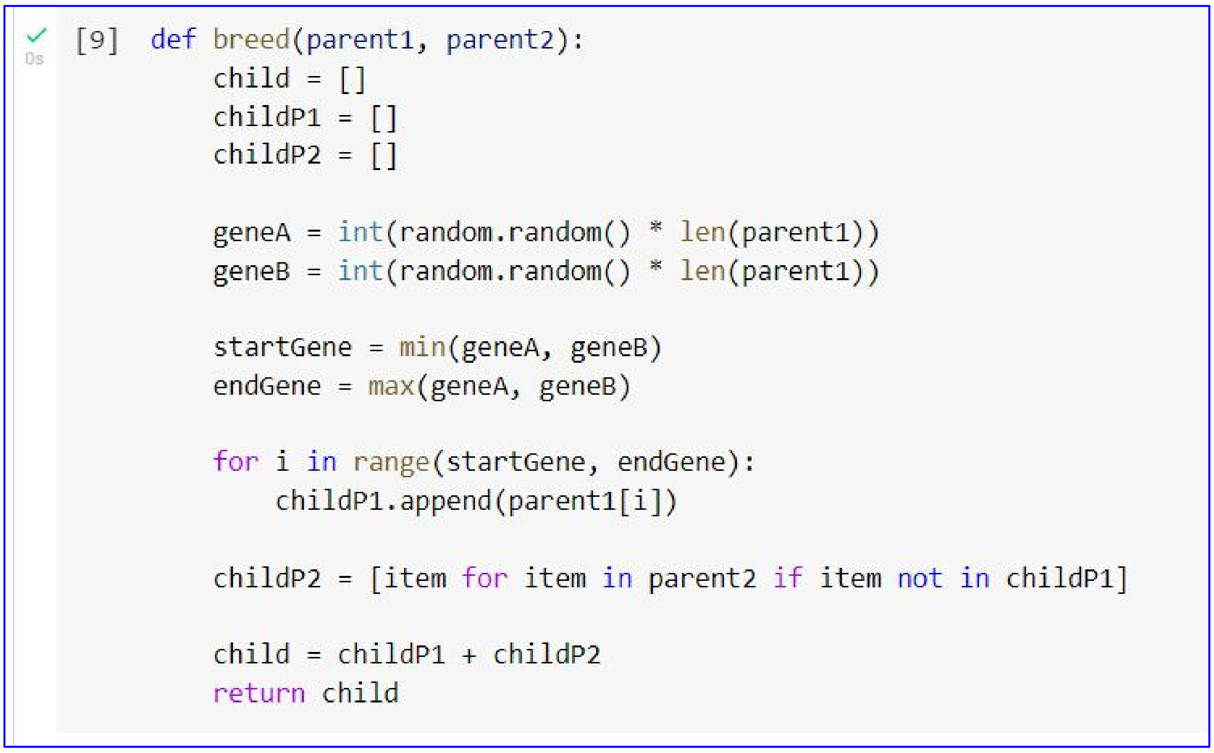
```
[5]
def initialPopulation(popSize, cityList):
    population = []

    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
```

```
[6] def rankRoutes(population):
        fitnessResults = {}
        for i in range(0,len(population)):
            fitnessResults[i] = Fitness(population[i]).routeFitness()
        return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)
```

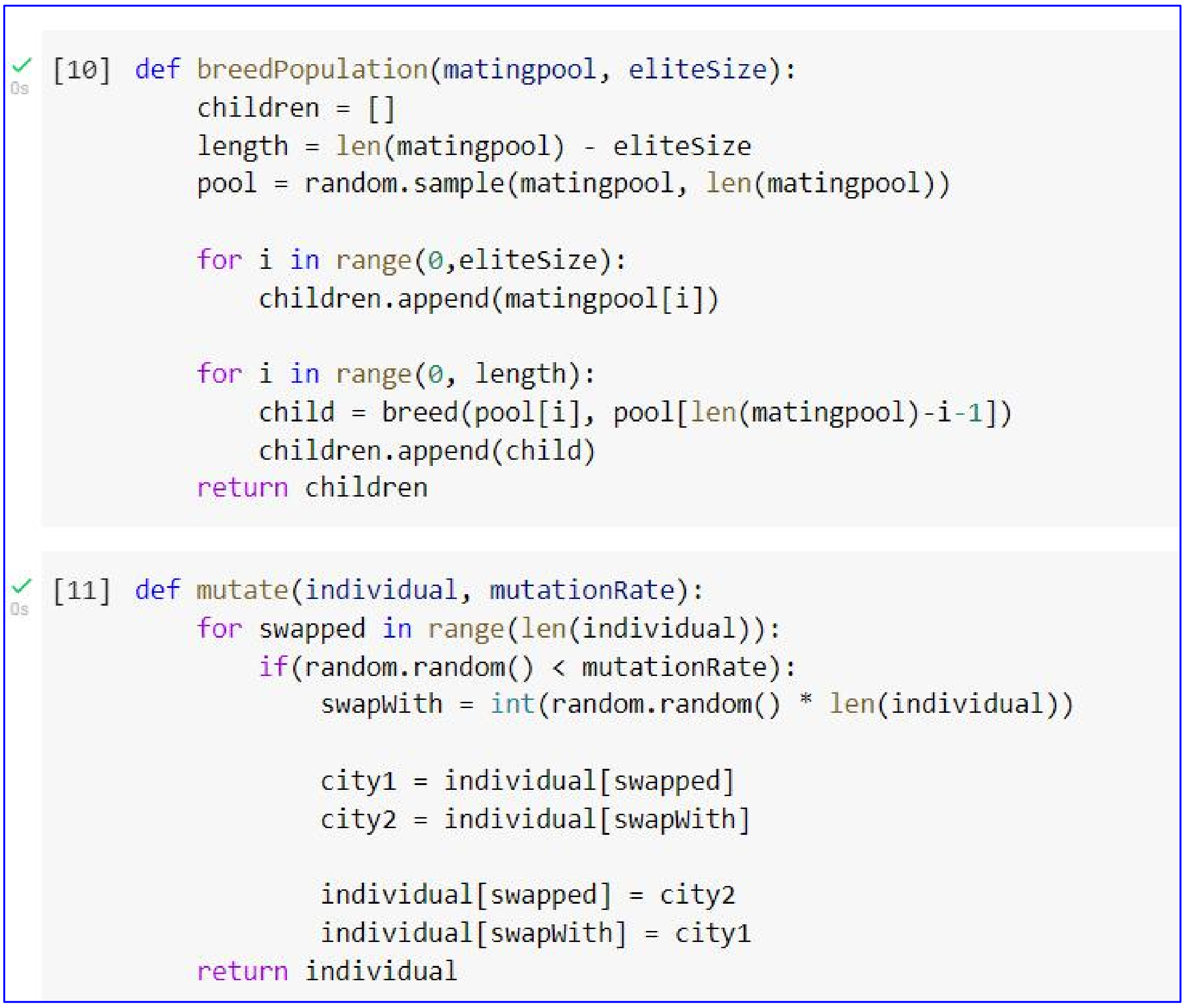


```
[7]
def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults
```

```
[8]
def matingPool(population, selectionResults):
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool
```

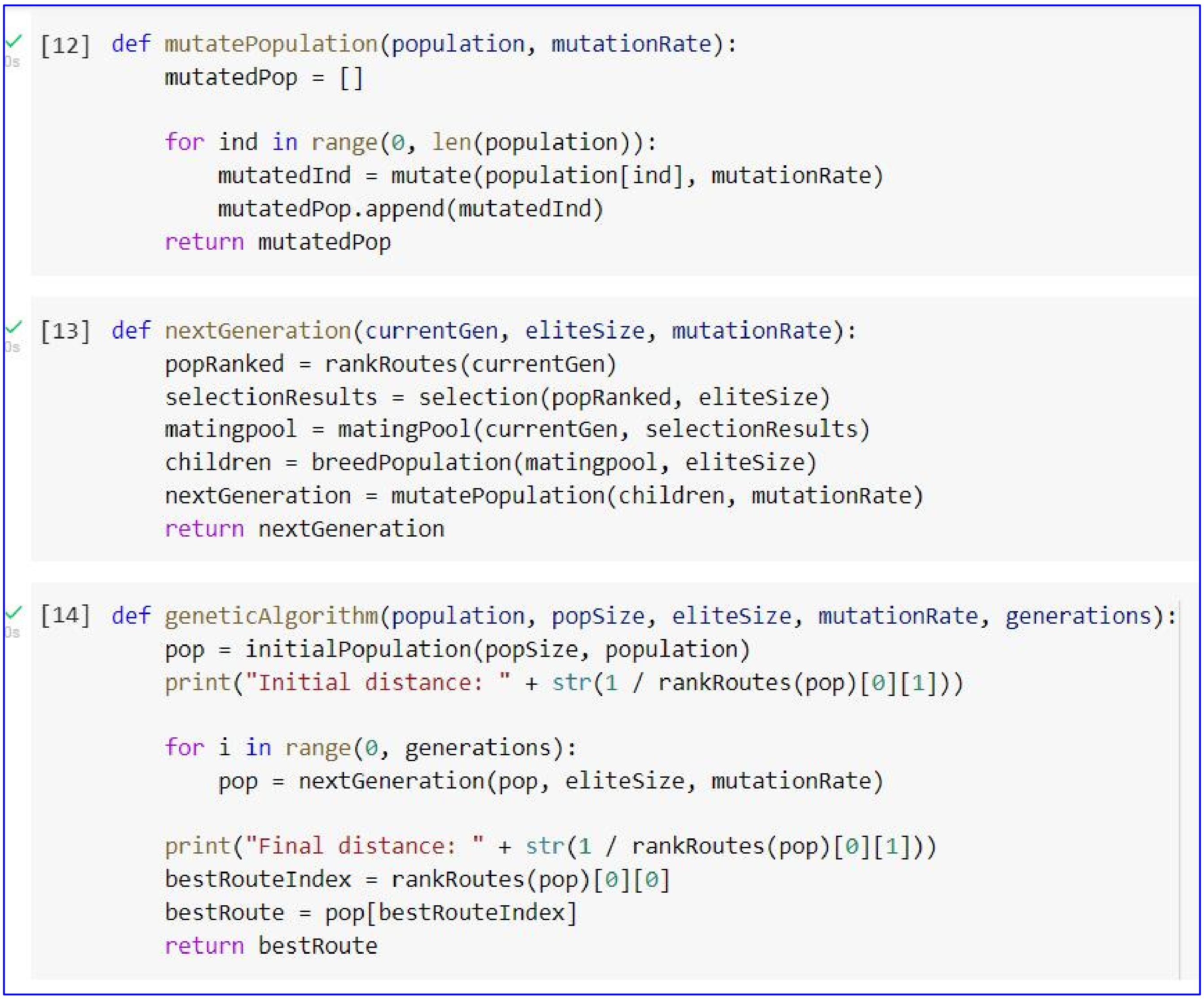


```
✓   [9]  def breed(parent1, parent2):
0s           child = []
             childP1 = []
             childP2 = []

             geneA = int(random.random() * len(parent1))
             geneB = int(random.random() * len(parent1))

             startGene = min(geneA, geneB)
             endGene = max(geneA, geneB)

             for i in range(startGene, endGene):
                 childP1.append(parent1[i])

             childP2 = [item for item in parent2 if item not in childP1]

             child = childP1 + childP2
             return child
```



```
✓   [10]  def breedPopulation(matingpool, eliteSize):
0s            children = []
              length = len(matingpool) - eliteSize
              pool = random.sample(matingpool, len(matingpool))

              for i in range(0,eliteSize):
                  children.append(matingpool[i])

              for i in range(0, length):
                  child = breed(pool[i], pool[len(matingpool)-i-1])
                  children.append(child)
              return children
```

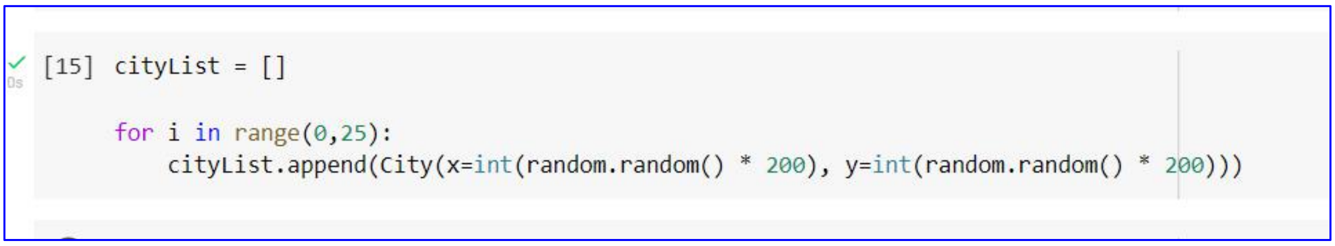
```
✓   [11]  def mutate(individual, mutationRate):
0s            for swapped in range(len(individual)):
                  if(random.random() < mutationRate):
                      swapWith = int(random.random() * len(individual))

                      city1 = individual[swapped]
                      city2 = individual[swapWith]

                      individual[swapped] = city2
                      individual[swapWith] = city1
              return individual
```



```
[12] def mutatePopulation(population, mutationRate):
         mutatedPop = []

         for ind in range(0, len(population)):
             mutatedInd = mutate(population[ind], mutationRate)
             mutatedPop.append(mutatedInd)
         return mutatedPop
```

```
[13] def nextGeneration(currentGen, eliteSize, mutationRate):
         popRanked = rankRoutes(currentGen)
         selectionResults = selection(popRanked, eliteSize)
         matingpool = matingPool(currentGen, selectionResults)
         children = breedPopulation(matingpool, eliteSize)
         nextGeneration = mutatePopulation(children, mutationRate)
         return nextGeneration
```

```
[14] def geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations):
         pop = initialPopulation(popSize, population)
         print("Initial distance: " + str(1 / rankRoutes(pop)[0][1]))

         for i in range(0, generations):
             pop = nextGeneration(pop, eliteSize, mutationRate)

         print("Final distance: " + str(1 / rankRoutes(pop)[0][1]))
         bestRouteIndex = rankRoutes(pop)[0][0]
         bestRoute = pop[bestRouteIndex]
         return bestRoute
```

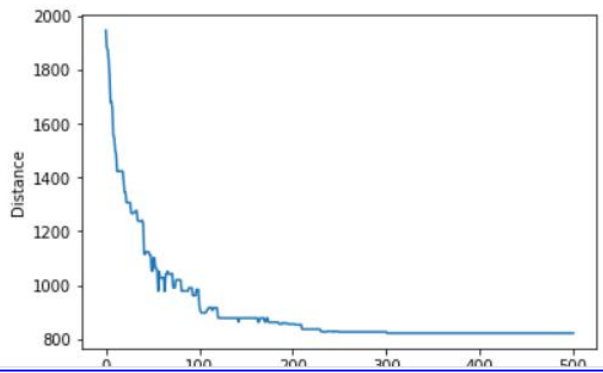


```
[15] cityList = []

     for i in range(0,25):
         cityList.append(City(x=int(random.random() * 200), y=int(random.random() * 200)))
```

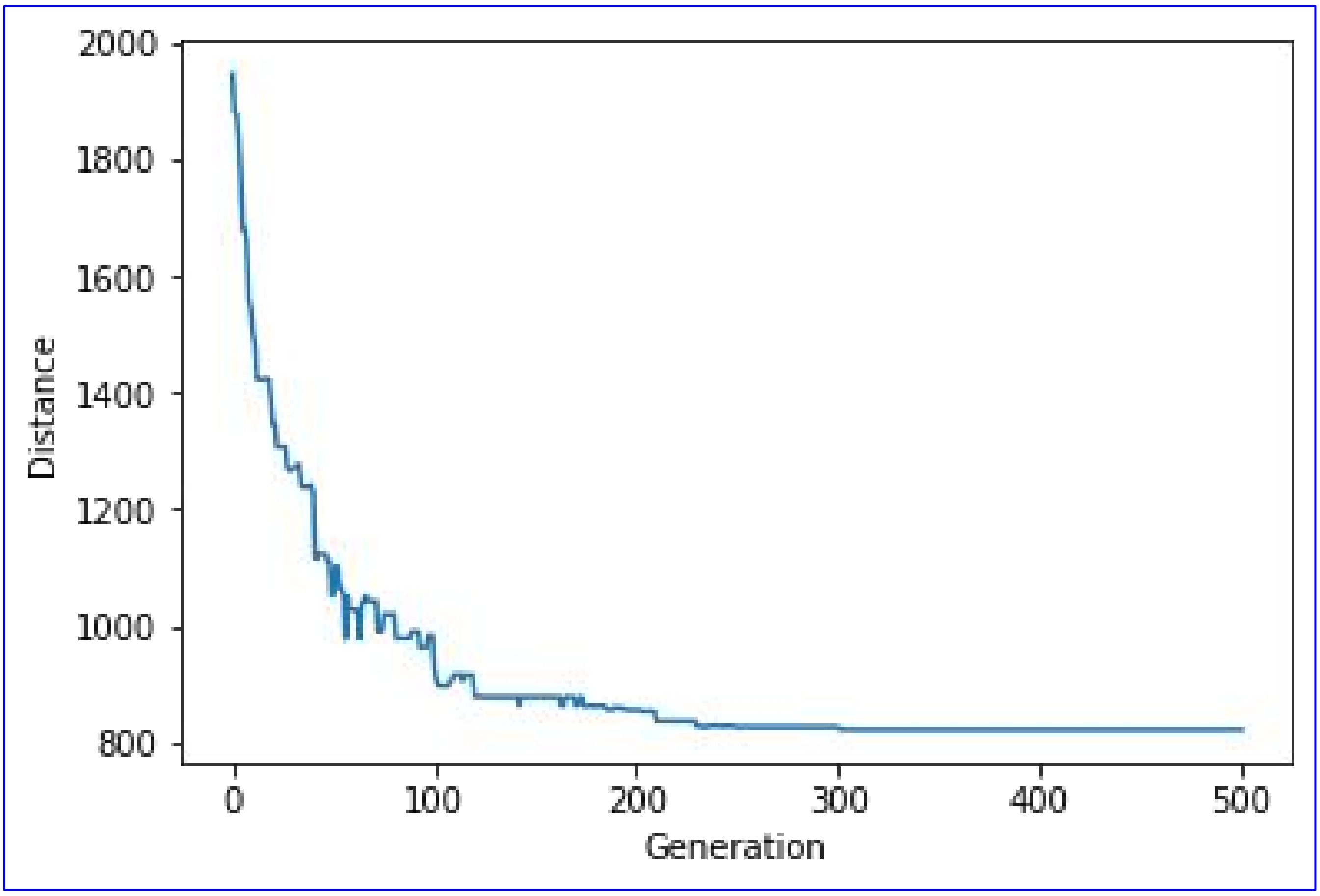
```
[16] geneticAlgorithm(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=500)

     Initial distance: 1795.7170471822537
     Final distance: 863.9610632296734
     [(186,67),
      (158,4),
      (121,21),
      (122,36),
      (89,47),
      (71,32),
      (74,53),
      (27,43),
      (23,54),
      (26,63),
      (12,90),
      (8,94),
      (35,99),
      (27,139),
      (68,188),
      (78,196),
      (123,196),
      (117,168),
      (155,144),
      (99,142),
      (89,149),
      (100,124),
      (63,110),
      (90,72),
      (122,95)]
```

```
[17] def geneticAlgorithmPlot(population, popSize, eliteSize, mutationRate, generations):
         pop = initialPopulation(popSize, population)
         progress = []
         progress.append(1 / rankRoutes(pop)[0][1])

         for i in range(0, generations):
             pop = nextGeneration(pop, eliteSize, mutationRate)
             progress.append(1 / rankRoutes(pop)[0][1])

         plt.plot(progress)
         plt.ylabel('Distance')
         plt.xlabel('Generation')
         plt.show()
```

```
[18] geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=500)
```



CONCLUSION:



From the above graph we can conclude that the distance reduces along increasing value of generation,thus finding an optimal solution.