Genetic Algorithm - code adapted from https://www.youtube.com/watch?v=zumC_C0C25c&t=12s , main functionality reflects body of original, but input and output adapted to problem.

Developed using PyCharm CE on a mac using Python 3.5.

Aim of genetic algorithm:

- Set the target
- Set the list of characters used for random selection

- Randomly generate initial population
    - compare the individuals within the initial population against the target
    - score the individuals on how many characters are in the right position against the target

- Only allow a select number of individuals into the next generation/randomly generated population
    - The individuals with the highest scores are the only ones to stay.

- Take the highest scored individuals
    - depending on the letters that are incorrect, take another high achiever and mutate that randomly with the current individual

- repeat entire process until the target and one of the individuals in one of the generations have the same fitness score.

Aim of random algorithm:

- Take in the target
- Take in the character set

- Randomly generate population from the character set
- Compare individuals in population against target
    - If the individual is same as target, end program with statement
    - Else continue to generate new individuals until one matches the target.

GA program discussion

- Reducing the characters to choose from reduce the number of iterations the program needs to do. However, it makes the program only applicable for that one scenario. But increasing the character set from just the required letters to all letters in the alphabet with a few other sentence constructs, the number of generations required to get the target can go from 100-500 to 1000-3000 approximately.

```
Based on:
Characters = varying

popsize=10
NoOfEliteIndividuals=1
tournamentselectionsize=4
RateOfMutation = 0.25
```

- Increasing the NoOfEliteIndividuals, the number of individuals with the highest scores to go onto the next generation, from 1 to 5 can reduce the number of generations to 50-500 approximately. However, if the number of elite individuals going onto the next generation is the same or equal to the population size, then an infinite loop appears, since there's no space for new individuals to integrate into the population.

Based on:

```
characters=['H','e','l','o','W','r','d','!',' ']
popsize=10
NoOfEliteIndividuals=5
tournamentselectionsize=4
RateOfMutation = 0.25
```

- Increasing the number of tournament selections requires approximately 50-350 generations to get to the target. Since what this does is force higher scorers to mix, potentially creating higher fitness individuals. Increasing the tournament selection size has decreased the overall number of generations the most out of the three parameters changed so far.

Based on:

```
popsize=10
NoOfEliteIndividuals=1
tournamentselectionsize=15
RateOfMutation = 0.25
characters=['H','e','l','o','W','r','d','!',' ']
```

- The parameter having the largest impact is the population size, popsize. It drastically decreases the number of iterations to approximately 20-100

Based on:

```
popsize=100
NoOfEliteIndividuals=1
tournamentselectionsize=4
RateOfMutation = 0.25
characters=['H','e','l','o','W','r','d','!',' ']
```

- Another way to reduce the number of iterations is by reducing the probability of a mutation occurring. What this does is only allow for 10% of the individuals in the population to mutate which reduces the variability of the population. So individuals do not change more letters than they need to in one iteration. Instead if the score is high, then only a couple of letters may have an opportunity to change.

Based on:

```
popsize=10
NoOfEliteIndividuals=1
tournamentselectionsize=4
RateOfMutation = 0.1
characters=['H','e','l','o','W','r','d','!',' ']
```

Therefore, the best way to reduce the number of iterations to achieve the target is to keep the mutation rate very low, reduce the character set to make individuals from, have a large population and increase the number of tournament selections.

e.g.

```
popsize=100
NoOfEliteIndividuals=1
tournamentselectionsize=10
RateOfMutation = 0.1
characters=['H','e','l','o','W','r','d','!',' ']
```

GA code

```
#runs on python 3.5
#body of code reflecting working found from
https://www.youtube.com/watch?v=zumC_C0C25c&t=12s
#altered using PyCharm CE

import random

popsize=100
#needs to be larger than number of elite individuals that are passed through to
next generation
NoOfEliteIndividuals=1
#infinitly loops if equal to or larger than population size
tournamentselectionsize=10
RateOfMutation = 0.1
#probability rate that a letter does mutation
characters=['H','e','l','o','W','r','d','!',' ']
#biased selection based on target

#characters='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ !,'
#takes a lot longer to search through
lengthofcharacters=len(characters)

target = 'Hello World!'
lengthoftarget=len(target)




class individual:
#self can be any variable name
    def __init__(self):
        self.letters = []
        self.fitness=0
        i=0
        while i <target.__len__():
#while the counter is less than length of the target
            if random.random()>=0.1:
#randomly populate array holding initial population
                self.letters.append(random.choice(characters))
            else:
                self.letters.append(random.choice(characters))
            i+=1
    def getletters(self):
        return self.letters
    def getfitness(self):
        self.fitness=0
        for i in range(self.letters.__len__()):
# for loop over all genes
            if self.letters[i] == target[i]:
                self.fitness+=1
#add 1 to the fitness if the gene is the same as the target
        return self.fitness

    def __str__(self):
        return self.letters.__str__()
#returns the contents of the gene array


class Population:
    def __init__(self, size):
#takes in size of population of Individuals
        self.individual=[]
        i=0
        while i< size:
```

```
            self.individual.append(individual())
#put new individuals into an array
            i+=1

    def getindividual(self): return self.individual
#returns the list of individuals in population


class genalgo:
    @staticmethod
#public, static evolve method?
    def evolve(pop):
        return genalgo.mutatepop(genalgo.crossoverpopulation(pop))

    @staticmethod
    def crossoverpopulation(pop):
        crossoverpop=Population(0)
        for i in range(NoOfEliteIndividuals):
            crossoverpop.getindividual().append(pop.getindividual()[i])
#leaves the elite individuals as is, no mutation
        i=NoOfEliteIndividuals
        while i<popsize:
            individual1 = genalgo.tournamentpopselect(pop).getindividual()[0]
            individual2 = genalgo.tournamentpopselect(pop).getindividual()[0]

crossoverpop.getindividual().append(genalgo.crossoverindividual(individual1,
individual2))
            i+=1
        return crossoverpop


    @staticmethod
    def mutatepop(pop):
        for i in range(NoOfEliteIndividuals,popsize):
            genalgo.mutateindividual(pop.getindividual()[i])
        return pop

    @staticmethod
    def crossoverindividual(individual1,individual2):
        crossoverindividual=individual()
        for i in range(target.__len__()):
            if random.random()>=0.5:
                crossoverindividual.getletters()[i] = individual1.getletters()[i]
            else:
                crossoverindividual.getletters()[i] = individual2.getletters()[i]
        return crossoverindividual

    @staticmethod
    def mutateindividual(individual):
#for loop over each passed letter
        for i in range(target.__len__()):
            if random.random()<RateOfMutation:
#if random number is smaller than the mutation rate, then mutate
                if random.random()<0.5:
                    individual.getletters()[i]=random.choice(characters)
                else:
                    individual.getletters()[i]=random.choice(characters)

    @staticmethod
    def tournamentpopselect(pop):
#randomly chooses individuals from past and present population
        tournamentpop=Population(0)
        i=0
        while i < tournamentselectionsize:

tournamentpop.getindividual().append(pop.getindividual()[random.randrange(0,popsize
```

```
)])
            i+=1
        tournamentpop.getindividual().sort(key=lambda x: x.getfitness(),
reverse=True)
        return tournamentpop


def printpopulation(pop, gennumber):
#print population function
    print('\n-----------------------------------------------------------')
    print( 'Generation number', gennumber, ', Fitness of highest ranked individual
in population: ', pop.getindividual()[0].getfitness())
    print('Target Individual: ', target, 'Highest possible fitness value: ',
lengthoftarget)
    print("-----------------------------------------------------------")
    i=0
    for x in pop.getindividual():
#goes through all the individual in the past pop
        print("Randomly generated individual number:", i, ":", x, "Associated
fitness score: ", x.getfitness())
        i +=1


population=Population(popsize)
population.getindividual().sort(key=lambda x: x.getfitness(), reverse=True)
#sorts the chromosomes with the individual with fitness in descending order
printpopulation(population,0)

generationnumber = 1
while population.getindividual()[0].getfitness()<target.__len__():
#whilst the fitness of each individual in each population is less than target
length
    population = genalgo.evolve(population)
    population.getindividual().sort(key=lambda x: x.getfitness(), reverse=True)
    printpopulation(population,generationnumber)
    generationnumber+=1
#go to next generation
```

Random search

Random Search Code

```
import random

characters=['H','e','l','o','W','r','d','!',',',' ']
lengthofcharacters=len(characters)

target = 'Hello World!'
lengthoftarget=len(target)

popsize=100

emptystoragelocation2=[]    #random letter
emptystoragelocation3=[]    #split up target
targetsplit=[]

#######################################CREATING INITIAL
POPULATION#############################################################

class individuals:
#self can be any variable name
    def __init__(self):
        self.letter = []
#create empty list to hold letters
        self.fitness=0
#initial fitness is set to 0
        counter=0
        while counter <target.__len__():
#while the counter is less than length of the target
            if random.random()>=0.5:
#randomly populate array holding initial population
                self.letter.append(random.choice(characters))
#this is where you put in  the letters
            else:
                self.letter.append(random.choice(characters))
#this is where you put in  the letters
            counter+=1

    def getletter(self):
#get new letters
        return self.letter


    def __str__(self):
        return self.letter.__str__()
#returns the contents of the letters array




class Population:
    def __init__(self, size):
#takes in size of population of individuals
        self.individuals=[]
        counter=0
        while counter< size:
            self.individuals.append(individuals())
#put new individuals into an array
            counter+=1

    def getindividuals(self): return self.individuals
#returns the list of individuals



def printpopulation(pop, gennumber):
```

```
#print population function
    print(' ')
    print('Generation:', gennumber)
    print('Target: ', target)
    print(' ')
    counter=0
    for x in pop.getindividuals():
#goes through all the individuals in the past pop
        print("Randomly generated individual:", counter, ":", x)
        counter+=1
    print(' ')


#def getphrasetargetcomparison(self): return self.individuals


population=Population(popsize)
#population.getindividuals().sort(key=lambda x: , reverse=True)
#sorts the individuals with the individuals with fitness in descending order
printpopulation(population,0)


##################################is the population holding a target
matcher?############################################


def phrasetotarget(pop):
    i = 0
    for i in individuals:
        if individuals(i) == target(i):
            print('goal achieved')
    else:
        print('goal not achieved')
        i += 1


answer=0

class SplitUpTarget(Population):
#splits target into separate words!!!
    words=target.split()
    emptystoragelocation3.append(words)
    print ("Target split into separate words: " + str(emptystoragelocation3))
    for i in emptystoragelocation3:
        targetsplit.append(list(target))
        print ('The components of the target are: '+str(targetsplit))
        #print("target length split: "+str(len(targetsplit)))
        print(' ')


class getnextgen(Population):

    def evolvepop(Population):
        return Population

generationnumber=1
while answer<1:
    if Population==targetsplit:
        print('Target found')
        answer+=1
        #print('answer', answer)
    else:
        print('Target not found, need to run again :-(')
        answer=0
        #print('answer',answer)
```

```
    Population=getnextgen.evolvepop(Population)
    printpopulation(population, generationnumber)
    generationnumber +=1

    if generationnumber>5000:
        print('Doubt the target will get found, you\'re at generation',
generationnumber-1, ', try running it again if you like until you\'re satisfied.
\nOr you could go to the Genetric Algorithm version :-)')
        exit()
```