EXP #1 : Genetic Algorithm

Code:

```
import random

POP_SIZE = 500
MUT_RATE = 0.1
TARGET = 'rayan ali'
GENES = ' abcdefghijklmnopqrstuvwxyz'

def initialize_pop(TARGET):
    population = list()
    tar_len = len(TARGET)

    for i in range(POP_SIZE):
        temp = list()
        for j in range(tar_len):
            temp.append(random.choice(GENES))
        population.append(temp)

    return population

def crossover(selected_chromo, CHROMO_LEN, population):
    offspring_cross = []
    for i in range(int(POP_SIZE)):
        parent1 = random.choice(selected_chromo)
        parent2 = random.choice(population[:int(POP_SIZE*50)])

        p1 = parent1[0]
        p2 = parent2[0]

        crossover_point = random.randint(1, CHROMO_LEN-1)
        child =  p1[:crossover_point] + p2[crossover_point:]
        offspring_cross.extend([child])
    return offspring_cross

def mutate(offspring, MUT_RATE):
    mutated_offspring = []

    for arr in offspring:
        for i in range(len(arr)):
            if random.random() < MUT_RATE:
                arr[i] = random.choice(GENES)
```

```
            mutated_offspring.append(arr)
    return mutated_offspring

def selection(population, TARGET):
    sorted_chromo_pop = sorted(population, key= lambda x: x[1])
    return sorted_chromo_pop[:int(0.5*POP_SIZE)]

def fitness_cal(TARGET, chromo_from_pop):
    difference = 0
    for tar_char, chromo_char in zip(TARGET, chromo_from_pop):
        if tar_char != chromo_char:
            difference+=1

    return [chromo_from_pop, difference]

def replace(new_gen, population):
    for _ in range(len(population)):
        if population[_][1] > new_gen[_][1]:
          population[_][0] = new_gen[_][0]
          population[_][1] = new_gen[_][1]
    return population

def main(POP_SIZE, MUT_RATE, TARGET, GENES):
    # 1) initialize population
    initial_population = initialize_pop(TARGET)
    found = False
    population = []
    generation = 1

    # 2) Calculating the fitness for the current population
    for _ in range(len(initial_population)):
        population.append(fitness_cal(TARGET, initial_population[_]))

    # now population has 2 things, [chromosome, fitness]
    # 3) now we loop until TARGET is found
    while not found:

      # 3.1) select best people from current population
      selected = selection(population, TARGET)

      # 3.2) mate parents to make new generation
      population = sorted(population, key= lambda x:x[1])
      crossovered = crossover(selected, len(TARGET), population)
```

```
    # 3.3) mutating the childeren to diversfy the new generation
    mutated = mutate(crossovered, MUT_RATE)

    new_gen = []
    for _ in mutated:
        new_gen.append(fitness_cal(TARGET, _))

    # 3.4) replacement of bad population with new generation
    # we sort here first to compare the least fit population with the most fit new_gen

    population = replace(new_gen, population)


    if (population[0][1] == 0):
      print('Target found')
      print('String: ' + str(population[0][0]) + ' Generation: ' + str(generation) + ' Fitness: ' +
str(population[0][1]))
      break
    print('String: ' + str(population[0][0]) + ' Generation: ' + str(generation) + ' Fitness: ' +
str(population[0][1]))
    generation+=1

main(POP_SIZE, MUT_RATE, TARGET, GENES)
```

Output:

```
String: ['r', 'a', 'r', 'j', 'f', 't', 'g', 'f', 'i'] Generation: 1 Fitness: 6
String: ['d', 'l', 'd', 'a', 'n', ' ', 'r', 'j', 'i'] Generation: 2 Fitness: 5
String: ['r', 'i', 'm', 'a', 'n', ' ', 'j', 'j', 'i'] Generation: 3 Fitness: 4
String: ['r', 'i', 'm', 'a', 'n', ' ', 'j', 'l', 'i'] Generation: 4 Fitness: 3
String: ['r', 'i', 'm', 'a', 'n', ' ', 'j', 'l', 'i'] Generation: 5 Fitness: 3
String: ['r', 'i', 'm', 'a', 'n', ' ', 'j', 'l', 'i'] Generation: 6 Fitness: 3
String: ['r', 'i', 'm', 'a', 'n', ' ', 'j', 'l', 'i'] Generation: 7 Fitness: 3
String: ['r', 'a', 'j', 'a', 'n', ' ', 'a', 't', 'i'] Generation: 8 Fitness: 2
String: ['r', 'a', 'y', 'a', 'n', ' ', 'a', 't', 'i'] Generation: 9 Fitness: 1
String: ['r', 'a', 'y', 'a', 'n', ' ', 'a', 't', 'i'] Generation: 10 Fitness: 1
String: ['r', 'a', 'y', 'a', 'n', ' ', 'a', 't', 'i'] Generation: 11 Fitness: 1
Target found
String: ['r', 'a', 'y', 'a', 'n', ' ', 'a', 'l', 'i'] Generation: 12 Fitness: 0
```