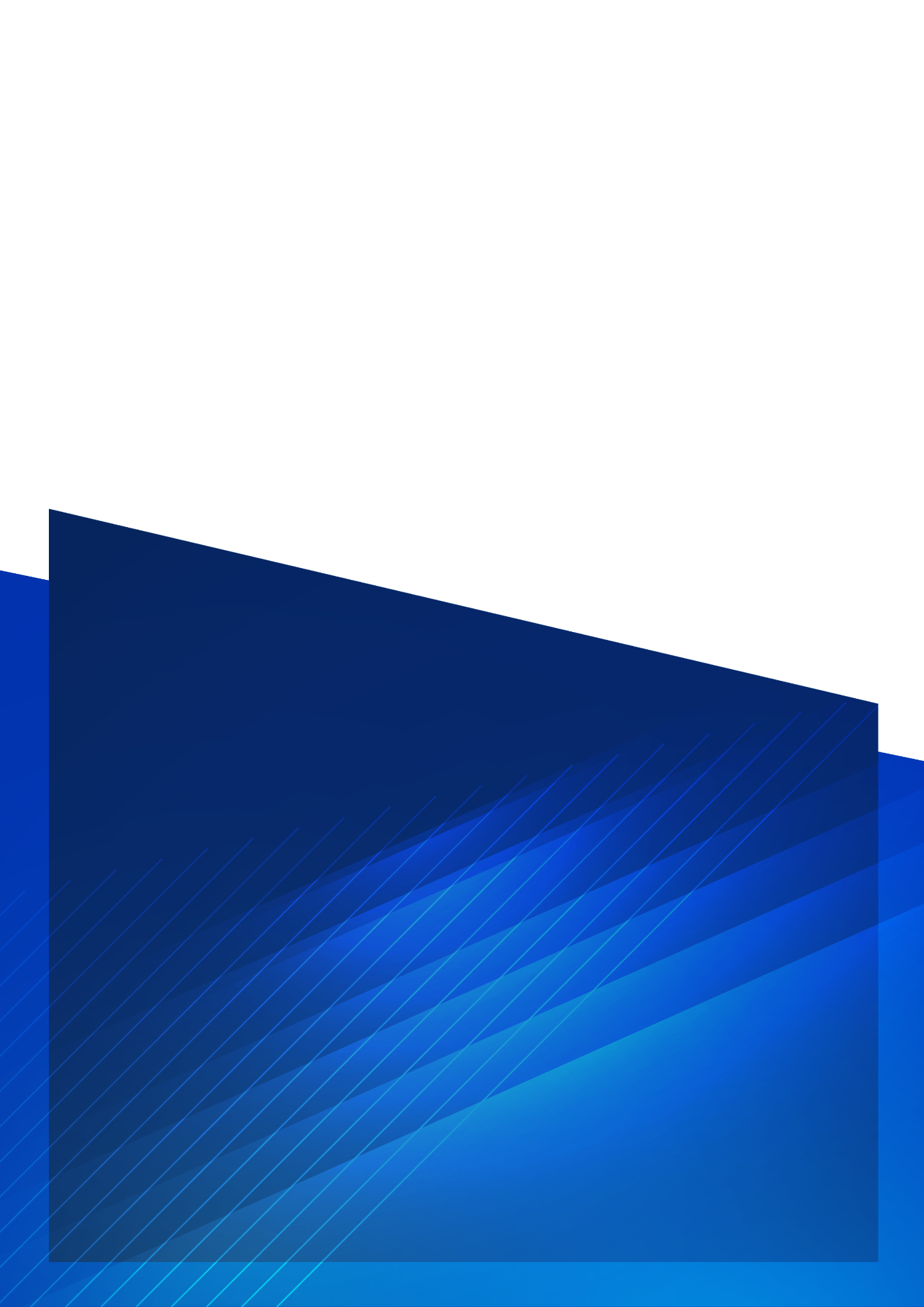
Artificial Intelligence Project

**Smart Cracker**



**Faisal Mukhtar** (51722)

BSE 7B

**Abstract**

Genetic algorithm is one of a class of algorithms that searches a solution space for the optimal solution to a problem. This search is done in a fashion that mimics the operation of evolution – a "population" of possible solutions is formed, and new solutions are formed by "breeding" the best solutions from the population's members to form a new generation. The population evolves for many generations; when the algorithm finishes the best solution is returned. Genetic algorithms are particularly useful for problems where it is extremely difficult or impossible to get an exact solution, or for difficult problems where an exact solution may not be required. They offer an interesting alternative to the typical algorithmic methods, and are highly customizable.

Here is a project: <https://github.com/iFaisalMukhtar/SmartCracker>

**Introduction**

Smart Cracker is a project in which we reveal or decode the password. This project is built using the Genetic algorithm technique to see how long it takes the algorithm to crack the password. Through this project, we can easily crack alpha-numeric Password. Also, we can get a Fitness and Generation Graph of each attempt where we will see the status of fitness as we move toward next generation and so on.

**Packages**

We have used following Packages for this project:

* Random
* String
* Numpy
* Pyplot
* Getpass

**Code**

import random

import numpy as np

import matplotlib.pyplot as plt

import time

import string

import matplotlib.animation as animation

from getpass import getpass

character\_list = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '!', '#', '$', '&', '\*', '+', '-', '/', '<', '=', '>', '?', '@', '[', ']', '^', '\_', '`']

secret\_pass=getpass("Type your Password Here: ")

secret\_password=list(secret\_pass)

password\_length = len(secret\_password)

population\_size = 100

num\_parents = 20

elite\_size = 2

population = []

for i in range(population\_size):

chromosome = []

for x in range(password\_length):

chromosome.append(random.choice(character\_list))

population.append(chromosome)

# fitness scoring

def fitness(population):

fitness\_scores = []

for chromosome in population:

matches = 0

for index in range(password\_length):

if secret\_password[index] == chromosome[index]:

matches += 1

result = [chromosome,matches]

fitness\_scores.append(result)

return fitness\_scores

# parent selection

def select\_parents(fitness\_scores):

parents\_list = []

for chromosome in sorted(fitness\_scores, key=lambda x: x[1], reverse = True)[:num\_parents]:

parents\_list.append(chromosome[0])

return(parents\_list)

# breeding logic

def breed(parent1,parent2):

child = []

parent1 = parents[0]

parent2 = parents[1]

geneA = int(random.random() \* password\_length)

geneB = int(random.random() \* password\_length)

startGene = min(geneA, geneB)

endGene = max(geneA, geneB)

for i in range(0,password\_length):

if (i < startGene) or (i > endGene):

child.append(parent1[i])

else:

child.append(parent2[i])

return child

# breeding and elitism

def create\_children(parents\_pool):

children = []

num\_new\_children = len(population) - elite\_size

for i in range(0,elite\_size):

children.append(parents\_pool[i])

for i in range(0,num\_new\_children):

parent1 = parents\_pool[int(random.random() \* len(parents\_pool))]

parent2 = parents\_pool[int(random.random() \* len(parents\_pool))]

children.append(breed(parent1,parent2))

return children

# mutation

def mutation(children\_set):

for i in range(len(children\_set)):

if random.random() > 0.1:

continue

else:

mutated\_position = int(random.random() \* password\_length)

mutation = random.choice(character\_list)

children\_set[i][mutated\_position] = mutation

return children\_set

# run Algorithm

fitness\_tracker = []

solutions = []

generations = 0

t0 = time.time()

while True:

fitness\_scores = fitness(population)

fitness\_tracker.append(max([i[1] for i in fitness\_scores]))

solutions.append(''.join([i[0] for i in fitness\_scores if i[1] == max([i[1] for i in fitness\_scores])][0]))

print(''.join([i[0] for i in fitness\_scores if i[1] == max([i[1] for i in fitness\_scores])][0]))

if max([i[1] for i in fitness\_scores]) == password\_length:

print("Cracked in {} generations, and {} seconds! \nSecret passcode = {} \nDiscovered passcode = {}".format(generations,time.time() - t0,''.join(secret\_password),''.join([i[0] for i in fitness\_scores if i[1] == password\_length][0])))

break

parents = select\_parents(fitness\_scores)

children = create\_children(parents)

population = mutation(children)

generations += 1

fig = plt.figure()

plt.plot(list(range(generations+1)), fitness\_tracker)

fig.suptitle('Fitness Score by Generation', fontsize=14, fontweight='bold')

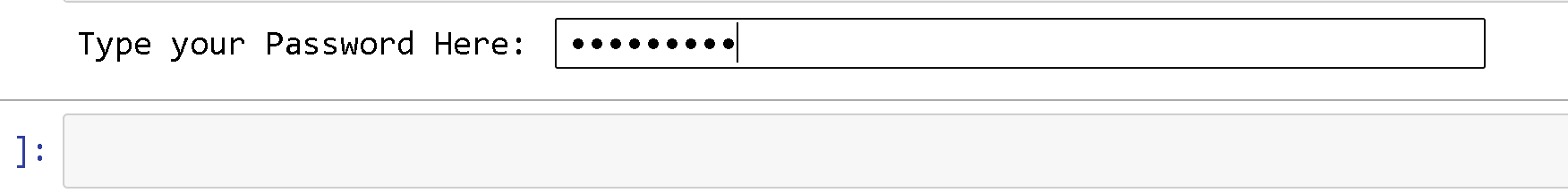
ax = fig.add\_subplot(111)

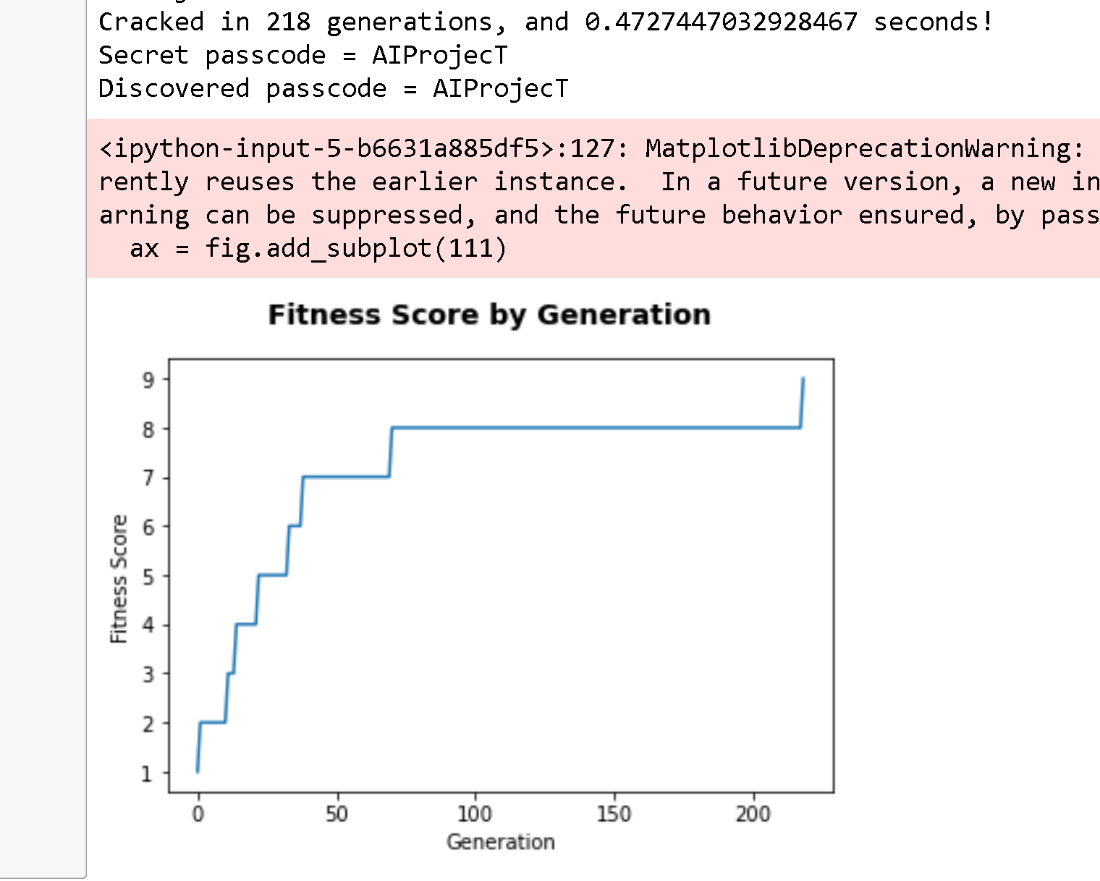
ax.set\_xlabel('Generation')

ax.set\_ylabel('Fitness Score')

plt.show()

**Result**





**Thank You ☺**