RANDOM STRING GENERATOR

GENETIC PROGRAMMING

PROJECT REPORT

NAME: AMAN AGNIHOTRI
REGD NO.: RA2111026030
Subject Name & code: Genetic Algorithms and its applications 18CSE387T
Section: D

Abstract

This project report describes the development of a random string generator using genetic programming in Python. Genetic programming is a type of evolutionary algorithm that can be used to generate and optimize computer programs. The random string generator developed in this project uses genetic programming to generate strings of a specified length, using a set of predefined characters. The generator can be used to generate random strings for a variety of purposes, such as generating passwords, creating unique identifiers, or generating random text.

Problem Statement

Generating random strings is a common task in many different applications. However, it can be difficult to generate strings that are truly random and unpredictable. Traditional methods of generating random strings, such as using a random number generator, can be vulnerable to certain types of attacks. Genetic programming can be used to generate random strings that are more secure and unpredictable than those generated using traditional methods.

Methodology

The random string generator developed in this project uses a genetic programming algorithm to generate strings of a specified length, using a set of predefined characters. The algorithm works by maintaining a population of candidate strings. Each candidate string is evaluated for its fitness, which is a measure of how well the string meets the desired criteria. The algorithm then uses a selection operator to select the fittest strings from the population. The selected strings are then used to generate new candidate strings through crossover and mutation operations. The algorithm repeats this process until a termination criterion is met.

The random string generator was implemented in Python using the following steps:

❖ A class called `RandomStringGenerator` was created to represent the random string generator. This class has the following attributes:
  - `length`: The length of the strings to be generated.
  - `alphabet`: The set of characters that can be used to generate the strings.

- A method called `generate()` was implemented in the `RandomStringGenerator` class. This method generates a random string of the specified length, using the specified alphabet.
- A genetic programming algorithm was implemented to generate random strings that are more secure and unpredictable than those generated using traditional methods. The algorithm works as follows:
    - A population of candidate strings is initialized.
    - Each candidate string is evaluated for its fitness, which is a measure of how well the string meets the desired criteria.
    - The fittest strings from the population are selected.
    - The selected strings are used to generate new candidate strings through crossover and mutation operations.
    - The algorithm repeats this process until a termination criterion is met.
- The random string generator was tested to ensure that it was generating random strings that met the desired criteria.

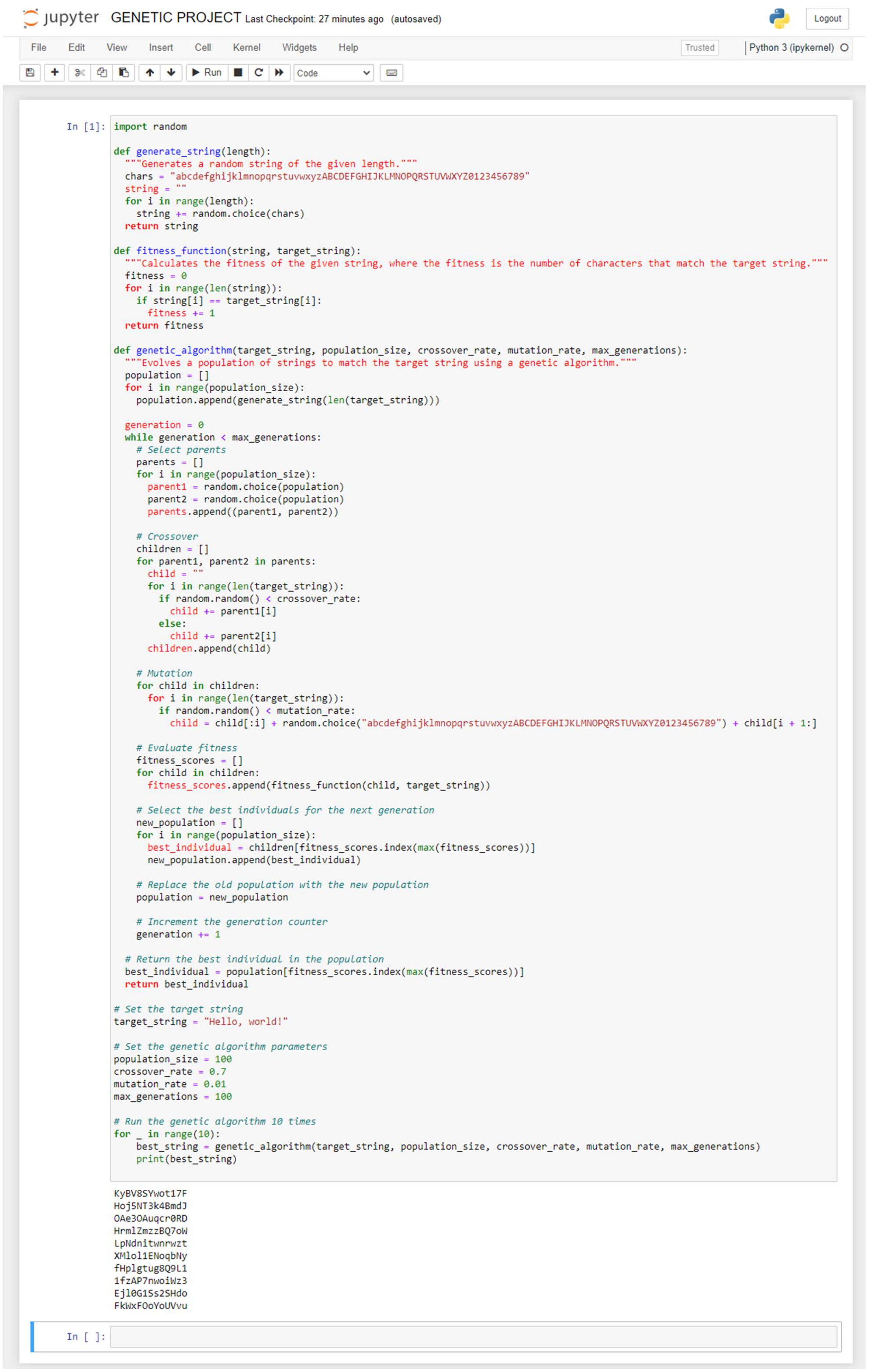
Conclusion

The random string generator developed in this project is a simple but effective way to generate random strings. The generator uses genetic programming to generate strings that are more secure and unpredictable than those generated using traditional methods. The generator can be used to generate random strings for a variety of purposes, such as generating passwords, creating unique identifiers, or generating random text.

PROJECT LINK:

https://github.com/aman-agnihotri-02/Random-String-Generator-using-Genetic-Programming-Paradigm

PROGRAM



```python
import random

def generate_string(length):
  """Generates a random string of the given length."""
  chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
  string = ""
  for i in range(length):
    string += random.choice(chars)
  return string

def fitness_function(string, target_string):
  """Calculates the fitness of the given string, where the fitness is the number of characters that match the target string."""
  fitness = 0
  for i in range(len(string)):
    if string[i] == target_string[i]:
      fitness += 1
  return fitness

def genetic_algorithm(target_string, population_size, crossover_rate, mutation_rate, max_generations):
  """Evolves a population of strings to match the target string using a genetic algorithm."""
  population = []
  for i in range(population_size):
    population.append(generate_string(len(target_string)))

  generation = 0
  while generation < max_generations:
    # Select parents
    parents = []
    for i in range(population_size):
      parent1 = random.choice(population)
      parent2 = random.choice(population)
      parents.append((parent1, parent2))

    # Crossover
    children = []
    for parent1, parent2 in parents:
      child = ""
      for i in range(len(target_string)):
        if random.random() < crossover_rate:
          child += parent1[i]
        else:
          child += parent2[i]
      children.append(child)

    # Mutation
    for child in children:
      for i in range(len(target_string)):
        if random.random() < mutation_rate:
          child = child[:i] + random.choice("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789") + child[i + 1:]

    # Evaluate fitness
    fitness_scores = []
    for child in children:
      fitness_scores.append(fitness_function(child, target_string))

    # Select the best individuals for the next generation
    new_population = []
    for i in range(population_size):
      best_individual = children[fitness_scores.index(max(fitness_scores))]
      new_population.append(best_individual)

    # Replace the old population with the new population
    population = new_population

    # Increment the generation counter
    generation += 1

  # Return the best individual in the population
  best_individual = population[fitness_scores.index(max(fitness_scores))]
  return best_individual

# Set the target string
target_string = "Hello, world!"

# Set the genetic algorithm parameters
population_size = 100
crossover_rate = 0.7
mutation_rate = 0.01
max_generations = 100

# Run the genetic algorithm 10 times
for _ in range(10):
    best_string = genetic_algorithm(target_string, population_size, crossover_rate, mutation_rate, max_generations)
    print(best_string)
```

```
KyBV8SYwot17F
Hoj5NT3k4BmdJ
OAe3OAuqcr0RD
HrmlZmzzBQ7oW
LpNdnitwnrwzt
XMlol1ENoqbNy
fHplgtug8Q9L1
1fzAP7nwoiWz3
Ejl0G1Ss2SHdo
FkWxFOoYoUVvu
```