PLAGIARISM CHECKER REPORT

Ahmet Hüzeyfe Demir

HTTPS://GİTHUB.COM/KOYUBOY?TAB=REPOSİTORİES

https://www.linkedin.com/in/ahmet-h%C3%BCzeyfe-demir-80515b157/

INTRODUCTION

A Line or word based algorithms, deep learning based methods etc. detects and reveals a plagiarized content, whether intentional or not.

There are many approaches and methods to detect plagiarized contents. Line or word based algorithms, deep learning based methods etc. Usually, plagiarism checkers work online, which means they search and detect content online.

We developed an algorithm based on row analysis in our project and we made use of the scipy library while doing this. We used Cosine Similarity Function from Scipy library.

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Another definition is Cosine similarity measures the similarity between two vectors of an inner product space.

The function is -> cosine(u, v)

u and v are one dimensional arrays. Function computes the Cosine distance between two arrays. After returns a double value.

All program based on row_similarity(row1, row2) funciton. So we will understand how this works. This function takes two arguments. They are row1 and row2 and they are in string type. Function take this rows and splitis. After that uses two dictionary to reveal frequency of splitted elements. After that, three things are important to us. Two lists vector1 , vector2 and a set of all_words_set. We used a set to eliminate duplicate elements. We have filled vector1 and vector2 with this set in a loop with a rule. After that cosine similarity function called with vector1 and vector2.

There is an example of this operation:

row1 = “I have an apple”

row2 = “She has an apple and apple juice”

all_words_set = [“I”, “have”, “an”, “apple”, “She”, “has”, “and”, “juice”]

vector1 = [1, 1, 1, 1, 0, 0, 0, 0]

vector2 = [0, 0, 1, 2, 1, 1, 1, 1]

1 – spatial.distance.cosine(vector1, vector2)

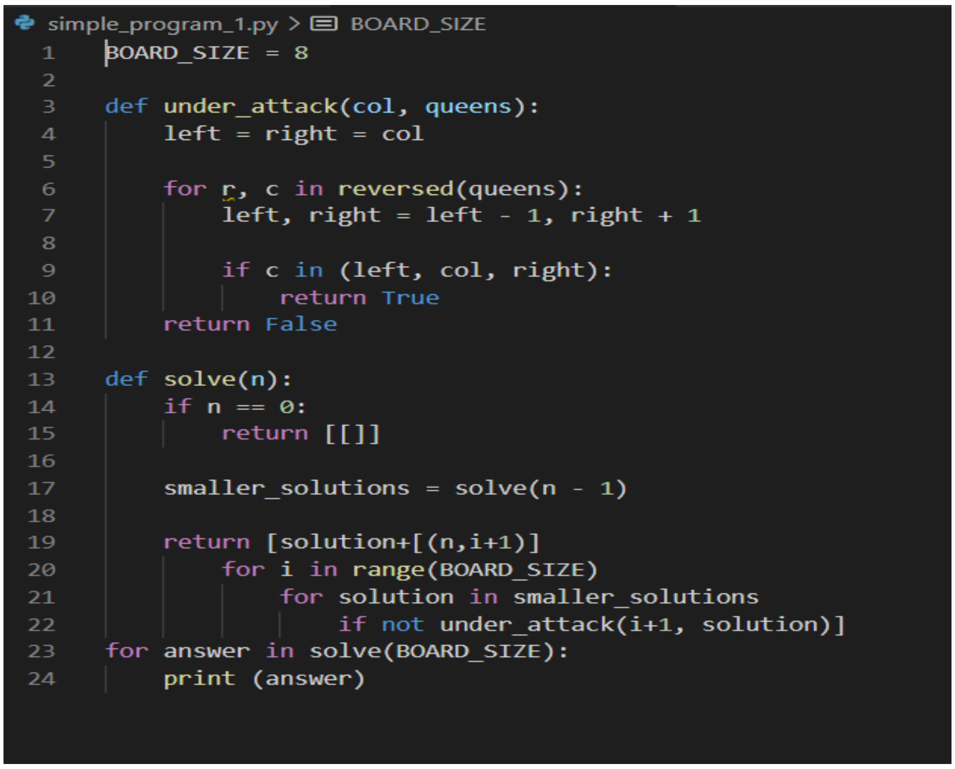
This operation returns a similarity ratio. We can manually choose a threshold for accepting these two contents as duplicates. We finished explaining what this function does. Let's see what the whole program does.

Other jobs of the program is quite simple. The program and the files to be compared must be in the same directory. It reads files with Python extension, splits them into lines and puts them into some preprocessing. Lastly it sends the lines in pairs to the row_similarity function. Each line in a file is compared with lines in other files.

The number of times it has been detected in other files is written next to each line in the files. At the end, the similarity ratios of the files are printed on the screen and the program ends.

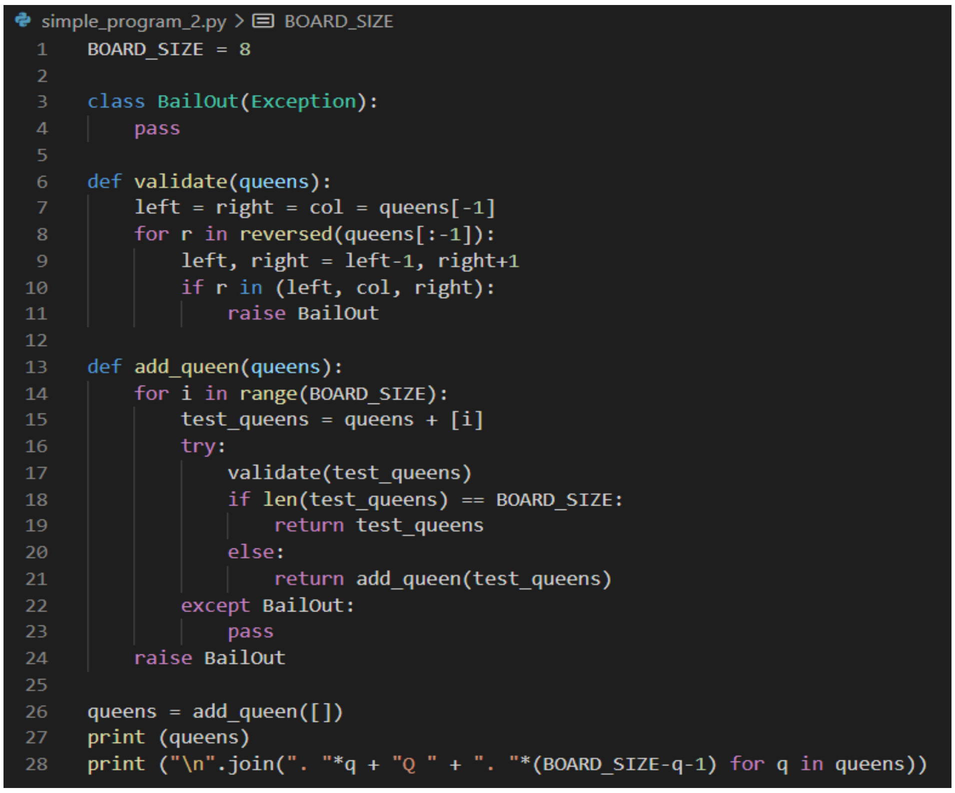
SAMPLE RUN -1

We have 4 python files here: simple_program_1.py, simple_program_2.py, simple_program_3.py, simple_program_4.py



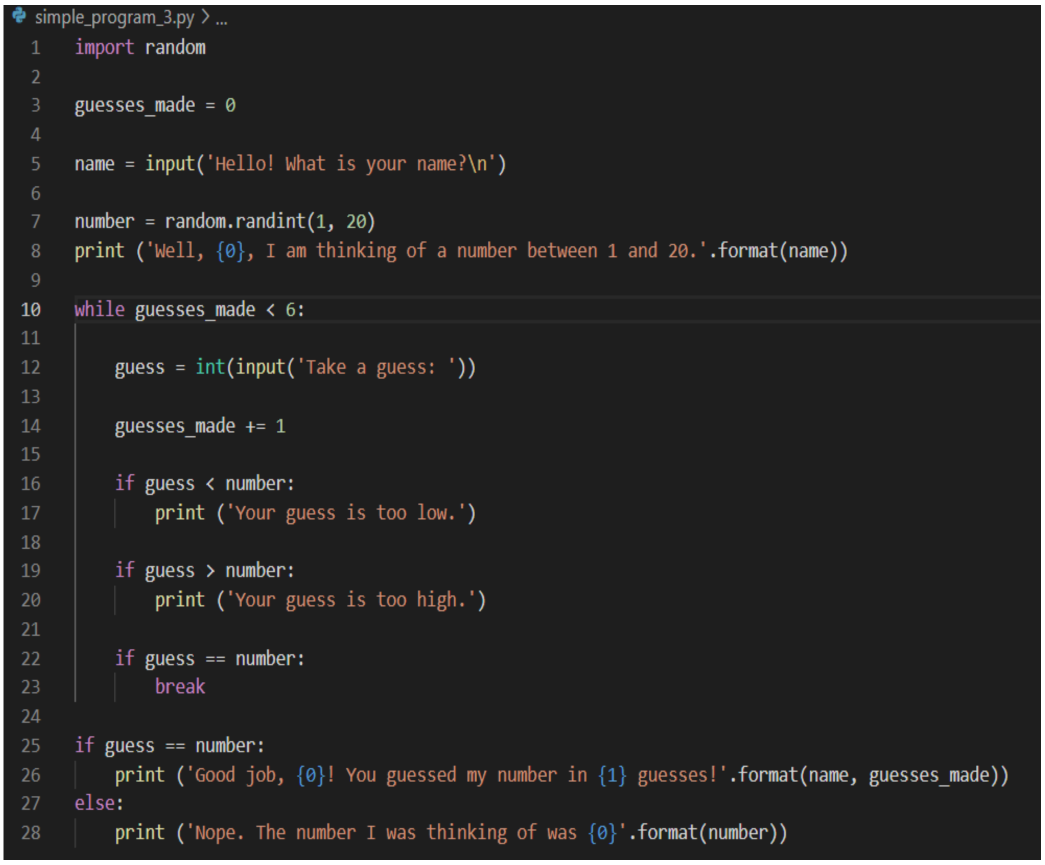
simple_program_1.py > BOARD_SIZE

```python
BOARD_SIZE = 8

def under_attack(col, queens):
    left = right = col

    for r, c in reversed(queens):
        left, right = left - 1, right + 1

        if c in (left, col, right):
            return True
    return False

def solve(n):
    if n == 0:
        return [[]]

    smaller_solutions = solve(n - 1)

    return [solution+[(n,i+1)]
        for i in range(BOARD_SIZE)
            for solution in smaller_solutions
                if not under_attack(i+1, solution)]
for answer in solve(BOARD_SIZE):
    print (answer)
```



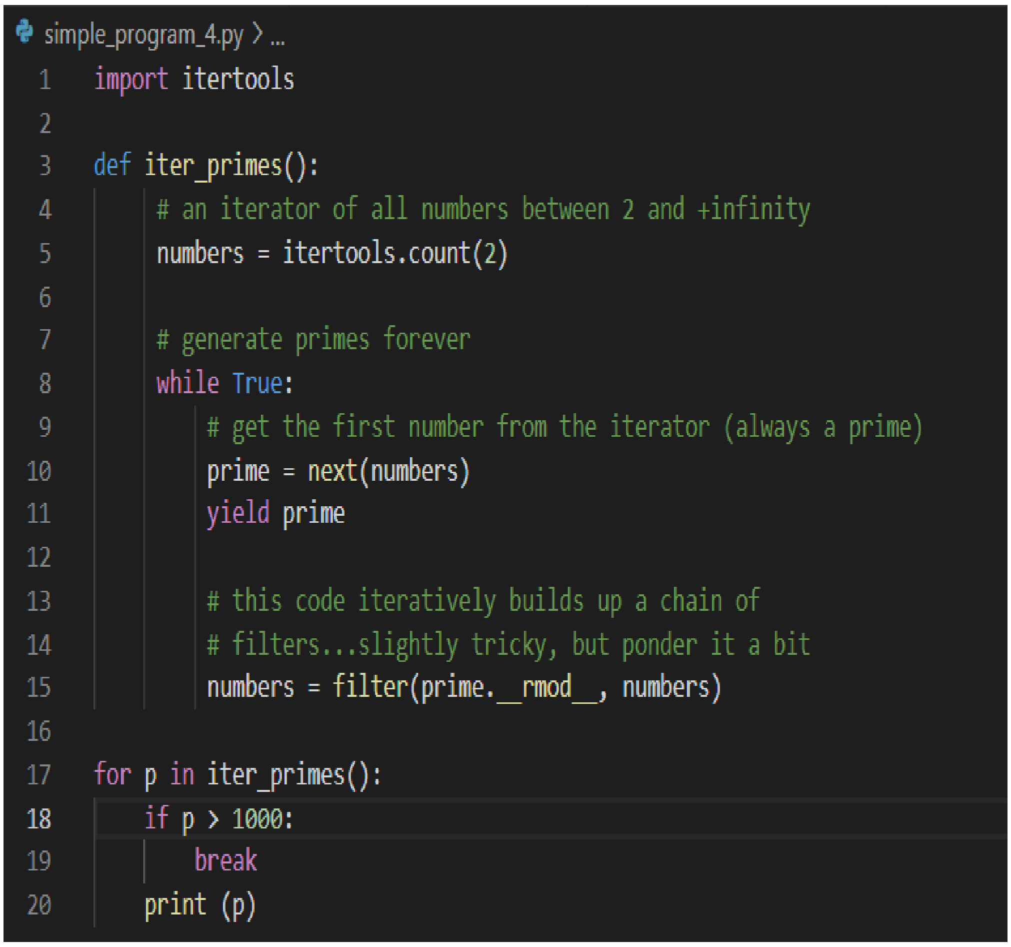
simple_program_2.py > BOARD_SIZE

```python
BOARD_SIZE = 8

class BailOut(Exception):
    pass

def validate(queens):
    left = right = col = queens[-1]
    for r in reversed(queens[:-1]):
        left, right = left-1, right+1
        if r in (left, col, right):
            raise BailOut

def add_queen(queens):
    for i in range(BOARD_SIZE):
        test_queens = queens + [i]
        try:
            validate(test_queens)
            if len(test_queens) == BOARD_SIZE:
                return test_queens
            else:
                return add_queen(test_queens)
        except BailOut:
            pass
    raise BailOut

queens = add_queen([])
print (queens)
print ("\n".join(". "*q + "Q " + ". "*(BOARD_SIZE-q-1) for q in queens))
```



simple_program_3.py > ...

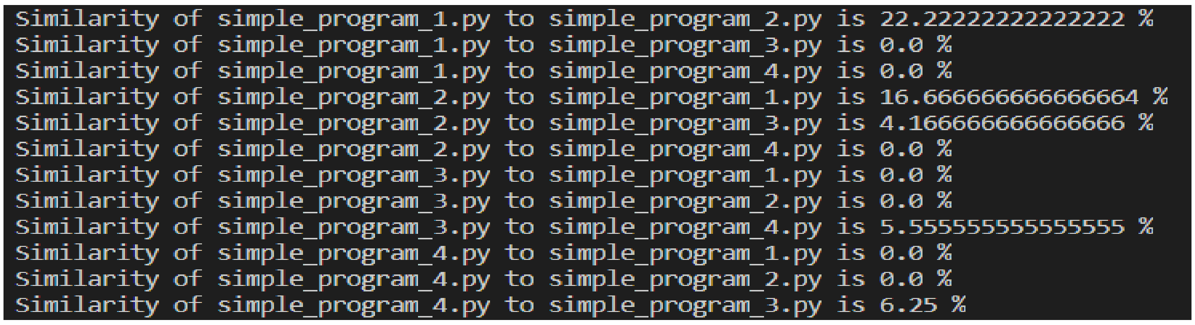
```python
import random

guesses_made = 0

name = input('Hello! What is your name?\n')

number = random.randint(1, 20)
print ('Well, {0}, I am thinking of a number between 1 and 20.'.format(name))

while guesses_made < 6:

    guess = int(input('Take a guess: '))

    guesses_made += 1

    if guess < number:
        print ('Your guess is too low.')

    if guess > number:
        print ('Your guess is too high.')

    if guess == number:
        break

if guess == number:
    print ('Good job, {0}! You guessed my number in {1} guesses!'.format(name, guesses_made))
else:
    print ('Nope. The number I was thinking of was {0}'.format(number))
```



simple_program_4.py > ...

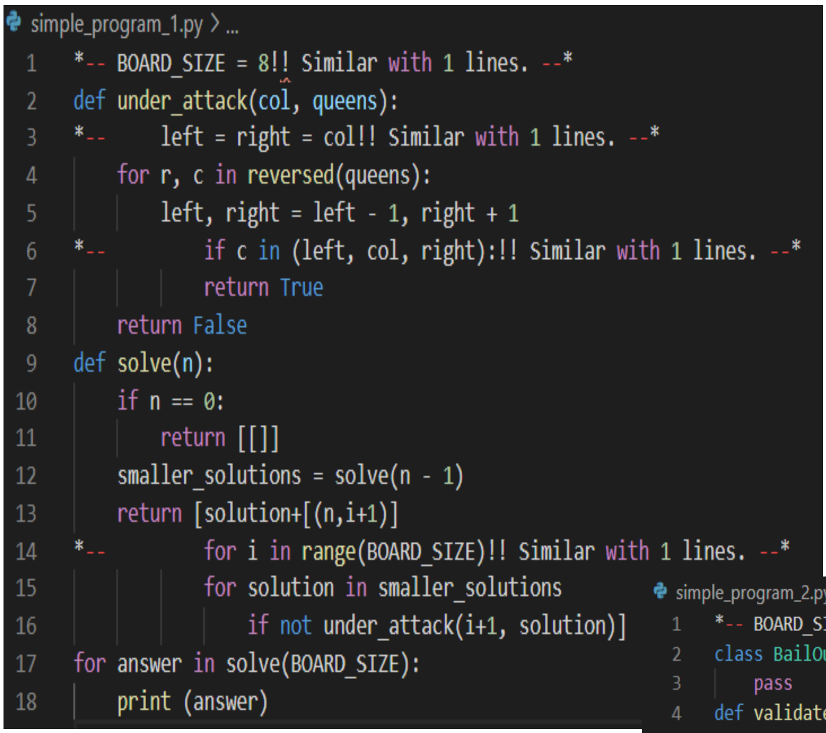
```python
import itertools

def iter_primes():
     # an iterator of all numbers between 2 and +infinity
     numbers = itertools.count(2)

     # generate primes forever
     while True:
         # get the first number from the iterator (always a prime)
         prime = next(numbers)
         yield prime

         # this code iteratively builds up a chain of
         # filters...slightly tricky, but ponder it a bit
         numbers = filter(prime.__rmod__, numbers)

for p in iter_primes():
    if p > 1000:
        break
    print (p)
```

Output of the program:



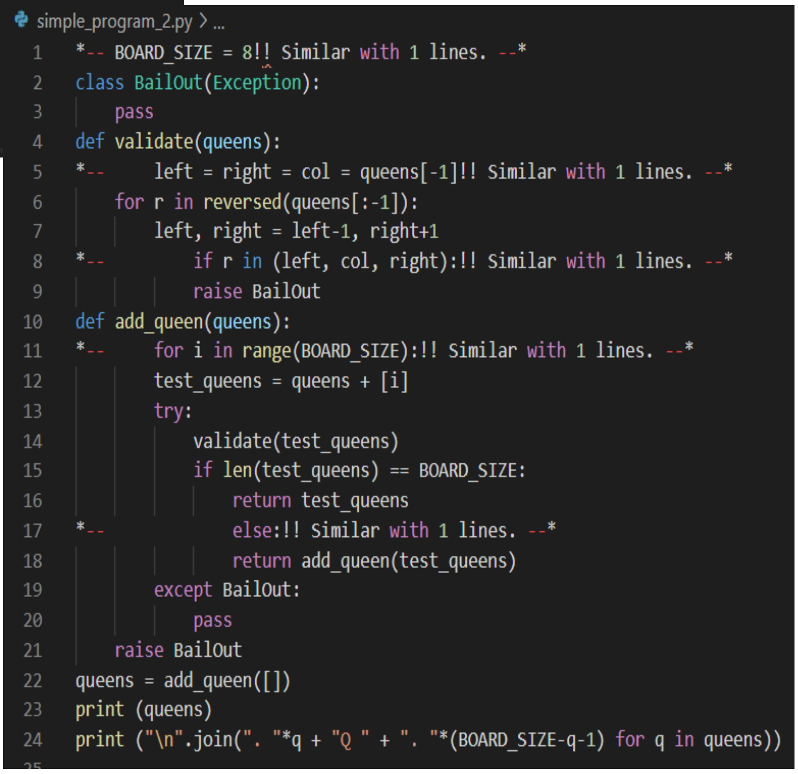
```
Similarity of simple_program_1.py to simple_program_2.py is 22.22222222222222 %
Similarity of simple_program_1.py to simple_program_3.py is 0.0 %
Similarity of simple_program_1.py to simple_program_4.py is 0.0 %
Similarity of simple_program_2.py to simple_program_1.py is 16.666666666666664 %
Similarity of simple_program_2.py to simple_program_3.py is 4.166666666666666 %
Similarity of simple_program_2.py to simple_program_4.py is 0.0 %
Similarity of simple_program_3.py to simple_program_1.py is 0.0 %
Similarity of simple_program_3.py to simple_program_2.py is 0.0 %
Similarity of simple_program_3.py to simple_program_4.py is 5.555555555555555 %
Similarity of simple_program_4.py to simple_program_1.py is 0.0 %
Similarity of simple_program_4.py to simple_program_2.py is 0.0 %
Similarity of simple_program_4.py to simple_program_3.py is 6.25 %
```

Python files after operation:



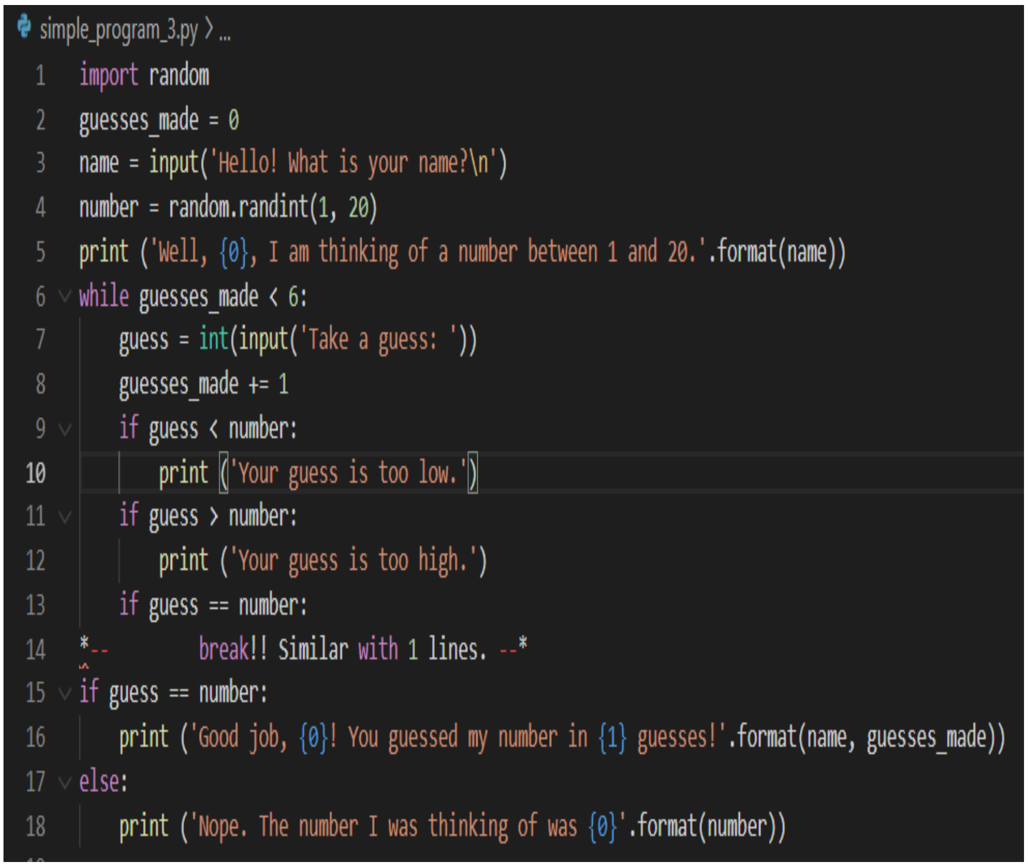
simple_program_1.py > ...

```python
*-- BOARD_SIZE = 8!! Similar with 1 lines. --*
def under_attack(col, queens):
*--     left = right = col!! Similar with 1 lines. --*
    for r, c in reversed(queens):
        left, right = left - 1, right + 1
*--         if c in (left, col, right):!! Similar with 1 lines. --*
            return True
    return False
def solve(n):
    if n == 0:
        return [[]]
    smaller_solutions = solve(n - 1)
    return [solution+[(n,i+1)]
*--         for i in range(BOARD_SIZE)!! Similar with 1 lines. --*
            for solution in smaller_solutions
                if not under_attack(i+1, solution)]
for answer in solve(BOARD_SIZE):
    print (answer)
```



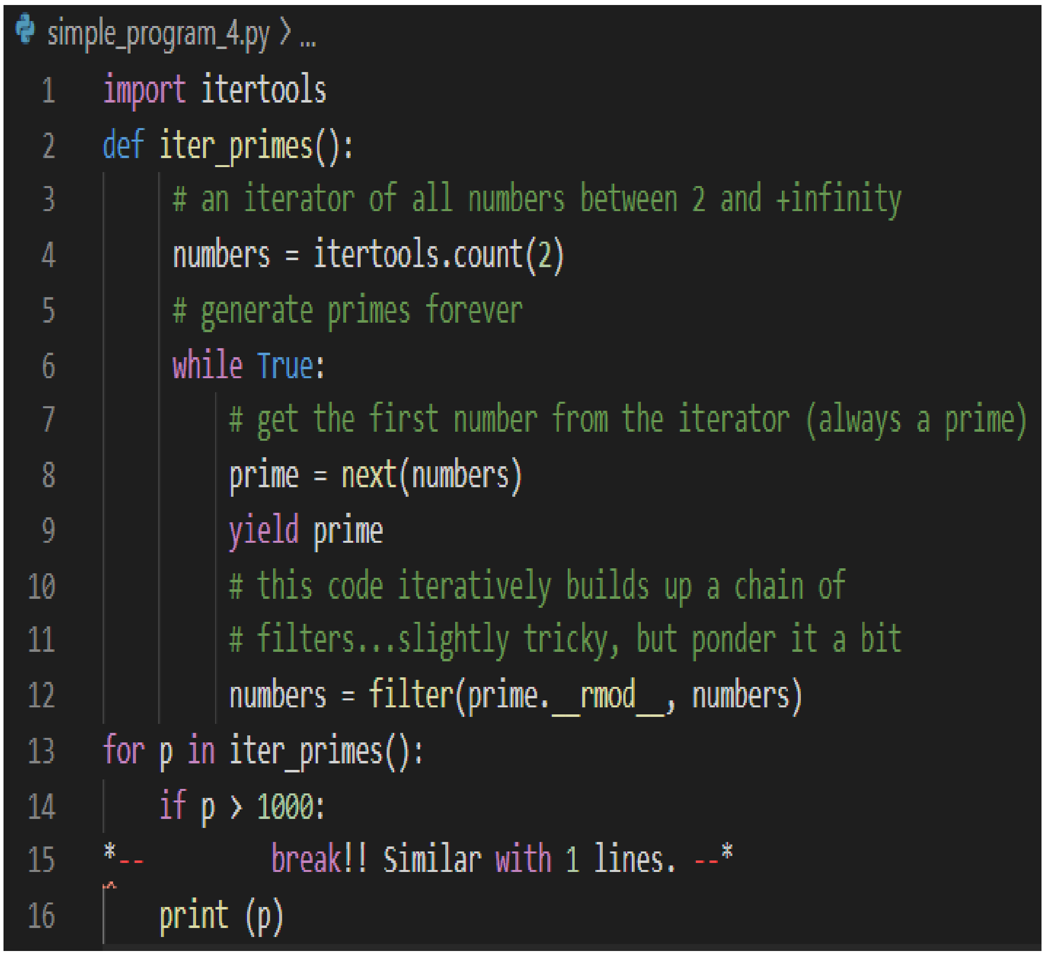
simple_program_2.py > ...

```python
*-- BOARD_SIZE = 8!! Similar with 1 lines. --*
class BailOut(Exception):
    pass
def validate(queens):
*--     left = right = col = queens[-1]!! Similar with 1 lines. --*
    for r in reversed(queens[:-1]):
        left, right = left-1, right+1
*--         if r in (left, col, right):!! Similar with 1 lines. --*
            raise BailOut
def add_queen(queens):
*--     for i in range(BOARD_SIZE):!! Similar with 1 lines. --*
        test_queens = queens + [i]
        try:
            validate(test_queens)
            if len(test_queens) == BOARD_SIZE:
                return test_queens
*--             else:!! Similar with 1 lines. --*
                return add_queen(test_queens)
        except BailOut:
            pass
    raise BailOut
queens = add_queen([])
print (queens)
print ("\n".join(". "*q + "Q " + ". "*(BOARD_SIZE-q-1) for q in queens))
```



simple_program_3.py > ...

```python
import random
guesses_made = 0
name = input('Hello! What is your name?\n')
number = random.randint(1, 20)
print ('Well, {0}, I am thinking of a number between 1 and 20.'.format(name))
while guesses_made < 6:
    guess = int(input('Take a guess: '))
    guesses_made += 1
    if guess < number:
        print ('Your guess is too low.')
    if guess > number:
        print ('Your guess is too high.')
    if guess == number:
*--         break!! Similar with 1 lines. --*
if guess == number:
    print ('Good job, {0}! You guessed my number in {1} guesses!'.format(name, guesses_made))
else:
    print ('Nope. The number I was thinking of was {0}'.format(number))
```



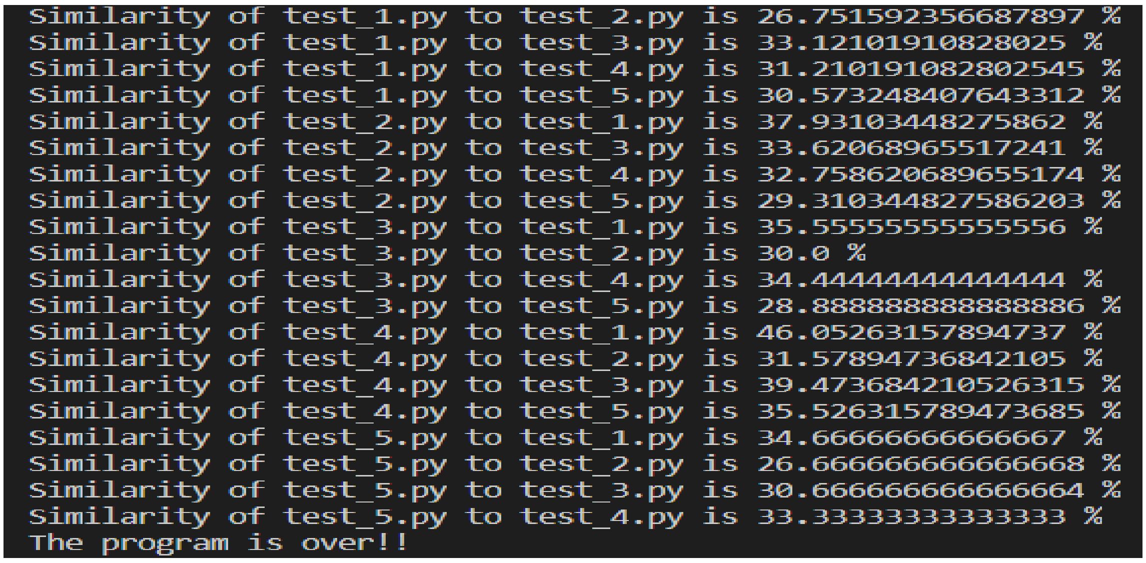
simple_program_4.py > ...

```python
import itertools
def iter_primes():
     # an iterator of all numbers between 2 and +infinity
     numbers = itertools.count(2)
     # generate primes forever
     while True:
         # get the first number from the iterator (always a prime)
         prime = next(numbers)
         yield prime
         # this code iteratively builds up a chain of
         # filters...slightly tricky, but ponder it a bit
         numbers = filter(prime.__rmod__, numbers)
for p in iter_primes():
    if p > 1000:
*--         break!! Similar with 1 lines. --*
    print (p)
```

SAMPLE RUN -2

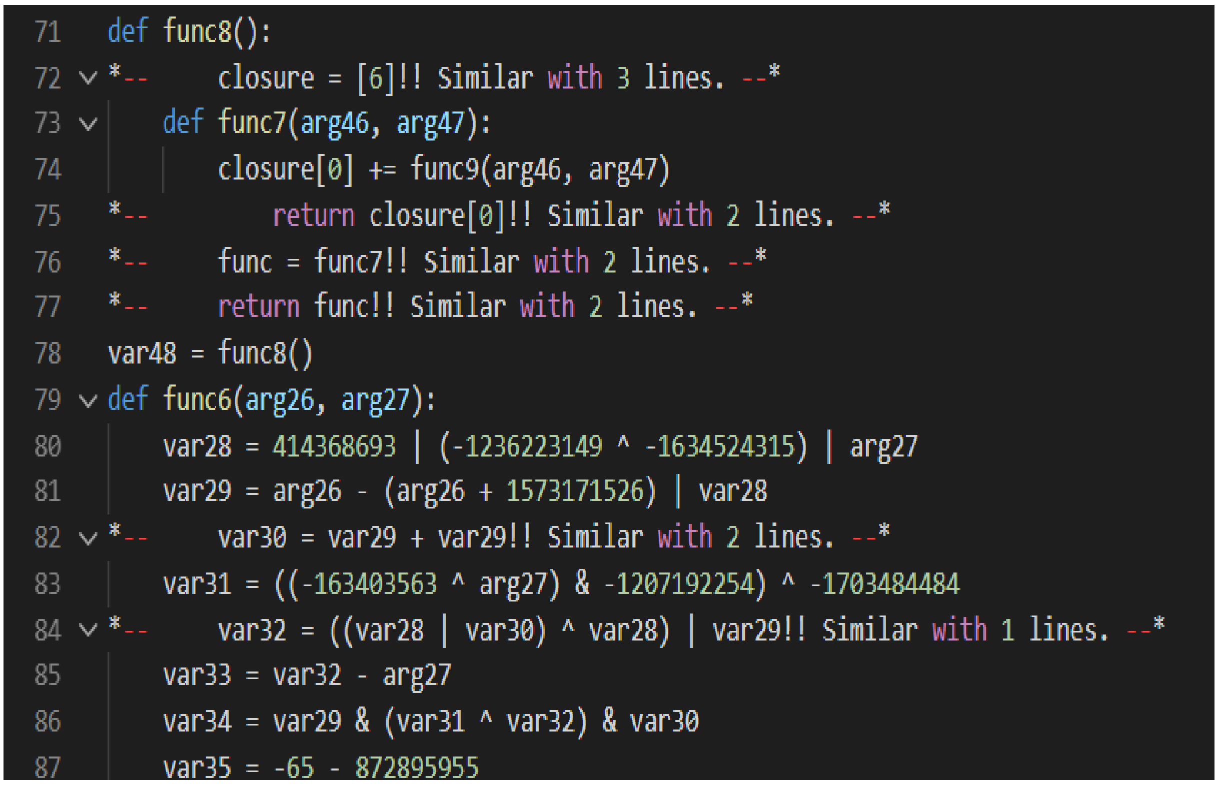
We have 5 python files here: test_1.py, test_2.py, test_3.py, test_4.py, test_5.py
These are randomly generated python codes from http://www.4geeks.de/cgi-bin/webgen.py

Output of the program:



```
Similarity of test_1.py to test_2.py is 26.751592356687897 %
Similarity of test_1.py to test_3.py is 33.12101910828025 %
Similarity of test_1.py to test_4.py is 31.210191082802545 %
Similarity of test_1.py to test_5.py is 30.573248407643312 %
Similarity of test_2.py to test_1.py is 37.93103448275862 %
Similarity of test_2.py to test_3.py is 33.62068965517241 %
Similarity of test_2.py to test_4.py is 32.758620689655174 %
Similarity of test_2.py to test_5.py is 29.310344827586203 %
Similarity of test_3.py to test_1.py is 35.55555555555556 %
Similarity of test_3.py to test_2.py is 30.0 %
Similarity of test_3.py to test_4.py is 34.44444444444444 %
Similarity of test_3.py to test_5.py is 28.888888888888886 %
Similarity of test_4.py to test_1.py is 46.05263157894737 %
Similarity of test_4.py to test_2.py is 31.57894736842105 %
Similarity of test_4.py to test_3.py is 39.473684210526315 %
Similarity of test_4.py to test_5.py is 35.526315789473685 %
Similarity of test_5.py to test_1.py is 34.66666666666667 %
Similarity of test_5.py to test_2.py is 26.666666666666668 %
Similarity of test_5.py to test_3.py is 30.666666666666664 %
Similarity of test_5.py to test_4.py is 33.33333333333333 %
The program is over!!
```

A part of test_1.py files after operation:



```
def func8():
*--     closure = [6]!! Similar with 3 lines. --*
    def func7(arg46, arg47):
        closure[0] += func9(arg46, arg47)
*--         return closure[0]!! Similar with 2 lines. --*
*--     func = func7!! Similar with 2 lines. --*
*--     return func!! Similar with 2 lines. --*
var48 = func8()
def func6(arg26, arg27):
    var28 = 414368693 | (-1236223149 ^ -1634524315) | arg27
    var29 = arg26 - (arg26 + 1573171526) | var28
*--     var30 = var29 + var29!! Similar with 2 lines. --*
    var31 = ((-163403563 ^ arg27) & -1207192254) ^ -1703484484
*--     var32 = ((var28 | var30) ^ var28) | var29!! Similar with 1 lines. --*
    var33 = var32 - arg27
    var34 = var29 & (var31 ^ var32) & var30
    var35 = -65 - 872895955
```

CONCLUSION

This program looks at every line and then every word. This is a little bad approach. Algorithm can be developed with recognize the structs like functions, loops, switch-cases etc. After that program can compares that structs with each other. This will make better accuracy. Because some code lines can be in all the programs and this lines increase our plagiarism ratio. Therefore program should look code files by part based not only line based.

Also in our program threshold is setting manually by programmer. This threshold ratio should be setted nicely. A deep learning model or AI based programs can be used to determine the threshold.

Lastly if this algorithm improves it can be detailed checker. Because we can say that this plagiarism checker algorithm is variable-free. It means is that revealing plagiarism is not affected from changing function names, if program improved to part based algorithm.