CodeCheck 2.0: An Optimized, General Purpose Textual Similarity Algorithm

Austin Daigle

Dr. Ken Nguyen

Final Term Project Report

Submitted

For

CSCI 4360

Department of Computer Science and Information Technology

Clayton State University

Spring, 2023

Abstract:

Textual similarity has many applications within the world of computer science, with general and specific applications with academic integrity, general software feature augmentation, and data analysis within specific fields. This project aims to make an open-source, lightweight textual similarity algorithm for multiple applications.

Table of Contents

Introduction:

In recent years, the need for a lightweight, efficient, and accurate textual similarity detection algorithm has grown significantly due to the increased importance of identifying plagiarism, comprehending code reuse, ensuring intellectual property protection, and analyzing similarities between niche applications of data (biometric data, filenames, etc.). This research project presents CodeCheck 2.0, an innovative similarity recognition algorithm that addresses these challenges by leveraging token pattern matching techniques using advanced pattern recognition algorithms to differentiate similarities.

The methodology of this project involves processing two input sources, tokenizing all words or shared strings between them, and examining the patterns of their usage rather than merely their frequency. These patterns are subsequently analyzed and filtered to eliminate overlaps and underlaps of data, thereby removing irrelevant data. Consequently, the refined algorithm produces a researcher-oriented, list-based data structure of the patterns for each input and a similarity percentage.

The outcome of this project is a complimentary, open-source software library and a standalone software application featuring an intuitive graphical user interface for an accurate, scalable, and adaptable text pattern recognition algorithm. This has been achieved without reliance on external software products, algorithms, machine learning, or artificial intelligence, except for default Python 3 libraries and necessary external graphical packages for standalone applications.

Background:

Assessing and establishing both relative and absolute similarities between two sets of strings is of vital importance for a wide range of applications in the field of computer science. CodeCheck 1.0 was initially developed to tackle these challenges; however, specific optimization and accuracy issues were uncovered after its development, as it primarily analyzed token frequency rather than usage patterns. As a result, CodeCheck 2.0 was conceived to address the shortcomings of the original (legacy) algorithm, providing a solution that can be effortlessly integrated into an array of computer science applications.

Project Description:

The core algorithm of CodeCheck 2.0 consists of three primary components: the tokenizer, the token pattern sequencers, and the token pattern recognition parser with a similarity percentage method. The tokenizer encompasses several processes identifying and cataloging tokens shared between two string inputs (two or more characters in matching sequential order). The token pattern sequencers arrange all identified tokens into a continuous token pattern "stream" data structure, displaying the tokens in the order they appear in their respective inputs. Subsequently, the token pattern recognition parser examines the pattern streams for input A and input B, identifying shared patterns of two or more tokens in sequential order present in both pattern streams. Irrelevancies, overlaps, and underlaps are processed and removed to ensure the accuracy of the analysis.

The final component processes the filtered streams for both inputs, calculating the relative similarity percentages for each input concerning the other. An HTML-formatted string is generated, displaying the matching text with highlights alongside the plain non-matching text for each input.

The primary distinction between the CodeCheck 2.0 library and the user application is that the application contains additional code to manage the graphical user interface. At the same time, the core algorithm remains the same.

Research Project achievements

During the research for this project, the following steps were completed to fulfill the original purpose and project proposal:

1. Proposed the project research to the CIMS department at Clayton State University.
2. Installed and configured the baseline environment for development and testing.
3. Analyzed the shortcomings of the original Java-based CodeCheck 1.0 program.
4. Adapted the processes developed for the deprecated Java-based CodeCheck 2.0 research (discontinued due to runtime limitations, incompatible development frameworks, and poor computational optimization in Java).
5. Enhanced tokenization by replacing the computationally expensive character referencing process with a more efficient string-slicing technique.
6. Optimized the pattern recognition process by focusing on sequencing tokens found in usage patterns instead of raw usage ratios.
7. Refactored all code in Python 3, ensuring minimal dependencies, computation time, and memory usage.
8. Designed a poster for the second annual CIMS symposium at Clayton State University.
9. Compiled the improved CodeCheck 2.0 and the original CodeCheck 1.0 Python code into a streamlined library.
10. Developed a graphical user interface using the PyQT5 graphical Python libraries.
11. Conducted final fine-tuning of all code and documented the development progress in a comprehensive write-up.

Pseudocode:

This pseudocode serves as a simplified conceptual abstraction of the operation of CodeCheck2 method is called within the CodeCheck library.

1. Define the **findSharedPatterns()** function.
2. Define the **CodeCheck2()** function.
3. Call **CodeCheck2()** with inputs inputA and inputB:
    1. Tokenize inputA and inputB into lists of words and punctuation.
    2. Create a list of unprocessed patterns by calling **findSharedPatterns()** with tokenized inputA and inputB.
    3. Process the list of unprocessed patterns using helper functions **processList()**, **removeAPairUnderlaps()**, **removeBPairUnderlaps()**, and **removeEmptyPatterns()**.
    4. Initialize the filteredAStream and filteredBStream lists.
    5. Iterate through the processed patterns and append the corresponding token pairs to the filteredAStream and filteredBStream.
    6. Get similarity formatted HTML for both inputs using the **getSimilarityHTML()** function.
    7. Calculate similarity percentage scores for both inputs using the **calculatePercentage()** function.
    8. Compile the results into a single list and return it

Please note that this pseudocode omits the implementations of the majority of variables, helpers functions, objects, methods, and internal structures/processes outside of broad strokes in order to keep the conceptual flow and length reasonable yet still relevant to the overall primary algorithm function of CodeCheck2 method.

This pseudocode assumes that inputA and inputB are provided when calling the CodeCheck2 function. To use the pseudocode for any arbitrary inputs, replace the inputs in step 3 with the desired input strings.

Code:

The source code for the CodeCheck library is not included directly, as it is already integrated within the application source code as part of its core analytical processes. A ported, optimized version of CodeCheck 1.0 is incorporated into the library and the application under "CodeCheck Legacy." The complete source code for the standalone CodeCheck 2.0 application, including the graphical user interface, is provided below.

The application is called CodeCheck 2.0, as this represents its primary function, along with the legacy build. However, the application is considered version 1.0, as this number denotes the current version of the code managing the graphical user interface rather than the actual version of the CodeCheck algorithm(s).

CodeCheck 2.0 Application (Version 1.0).py

```
"""
Written By:        Austin Daigle
Version:           1.0 (Stable)
Description:       This is a CodeCheck 2.0, a textual similarity
                   algorithm that analyzes two text inputs for
                   similarity based on token(s) (strings) usage
                   relative to both inputs based on how they
                   were used as opposed to raw frequency. This program
                   takes the raw functionality of the CodeCheck library
                   and adds a simple, user-friendly graphical user
                   interface.

Legacy Features:   A ported legacy build of CodeCheck 1.0 was included
                   with this program which only performs analytics based
                   on raw frequency data on tokens.

Program Structure:  This program's design structure is based on two
                   primary partitions: CodeCheck 2.0 code base &
                   graphical user interface code. The GUI code
                   has a small default main method that is included
                   the start of the program.
                   Multi-line comments are included to indicate these
                   two primary components. Internal structures or
                   components of these structures are marked with
```

```
                comments and hashtag borders with section titles.
"""

################################################################################
#  Import Dependencies
################################################################################
from PyQt5 import QtCore, QtGui, QtWidgets
import tkinter as tk
from tkinter import messagebox
import webbrowser
################################################################################

"""
Program Section:    CodeCheck Library
Description:        This is a direct copy of all of the CodeCheck
                    library code, these are all of the algorithms
                    for CodeCheck 2.0 and CodeCheck 1.0 (Legacy)
                    along with the HTML and percentage generating
                    methods.
"""
"""
Written By:     Austin Daigle
Version:        1.0
Description:    This library contains core algorithms
                for CodeCheck 2.0 and the ported legacy build of
                CodeCheck 1.0. Library offers CodeCheck 2.0
                a textual similarity analysis algorithm that
                identifies similarities between the two
                inputs and returns a relative percentage score
                and the data on which tokens are identified to
                be a positive match.
"""
################################################################################
#  PROGRAM METHODS
################################################################################
# getSimilarityHTML() METHOD
#*******************************************************************************
# the point of this method is to slide the input string
# into sections of matched and non-matched text in order
# to process it as formatted html code
def getSimilarityHTML(pattern, input):
    # create the variables for highlighting in html
    mark = "<mark>"
    endMark = "</mark>"
```

```
    # create custom mark tags since standard <mark></mark> tags don't work in PyQT5
    markFix = "<span style=\"background-color: yellow;\">"
    endMarkFix = "</span>"
    # Uncomment the two variables below if you are using a library/graphics package
    # for html that does not support mark. In this case the PyQT5 library does NOT
    # support this.
    mark = markFix
    endMark = endMarkFix


    # extract the index pair sublist from each of the pattern list
    # in the pattern superlist
    pattern = [sub[0] for sub in pattern]
    # Find the length of the input string
    maxValue= len(input)
    # Initialize an empty string to store the resulting string
    result = ""
    # Initialize a variable to keep track of the end of the previous slice
    lastRight = 0
    # Loop through each slice of the pattern in the input string
    for i in range(len(pattern)):
        # Get the left and right boundaries of the current slice
        left, right = pattern[i]
        # If the end of the previous slice does not match the beginning of the current slice
        if lastRight != left:
            # If the end of the previous slice is greater than the beginning of the current slice
            if lastRight > left:
                pass
            else:
                # update result the previous slice with a non-highlighted tag and append it to the
result string
                result += input[lastRight:left]
        # If there are more slices after the current one and the end of the current slice underlaps
with the beginning of the next slice
        if i < len(pattern) - 1 and right > pattern[i + 1][0]:
            # Set the end of the current slice to the beginning of the next slice
            right = pattern[i + 1][0]
            # update the current slice with a highlighted tag and append it to the result string
            result += f"{mark}{input[left:right]}{endMark}"
        else:
            # update the current slice with a highlighted tag and append it to the result string
            result += f"{mark}{input[left:right]}{endMark}"
        # Set the end of the previous slice to the end of the current slice
        lastRight = right
    # If the end of the last slice does not match the end of the input string
```

```
    if lastRight != maxValue:
        # Print the last slice with a non-highlighted tag and append it to the result string
        result += input[lastRight:maxValue]
    # Return the resulting string
    return result

#*********************************************************************
# calculatePercentage() METHOD
#*********************************************************************
# this method is basically getSimilarityHTML() but simplified to
# return the percent of text that is identified as matched
# to the input string
def calculatePercentage(pattern, input):
    # extract the index pair sublist from each of the pattern list
    # in the pattern superlist
    pattern = [sub[0] for sub in pattern]
    # Get the maximum value to be used later
    maxValue = len(input)
    # Set up some variables to keep track of the input and matched sizes
    textInputSize = len(input)
    matchedInputSize = 0
    # Loop through each slice of the input string that matches the pattern
    for i in range(len(pattern)):
        # Get the left and right indices of the current slice
        left, right = pattern[i]
        # If the right index of the current slice is greater than the left index of the next slice,
        # set the right index to the left index of the next slice
        if i < len(pattern) - 1 and right > pattern[i + 1][0]:
            right = pattern[i + 1][0]
        # Add the length of the input string that falls within the current slice to the matched input
size
        matchedInputSize += len(input[left:right])
    # Calculate the percentage of the input string that matched the pattern and format it to two
decimal places
    percentMatched = "{:.2f}".format((matchedInputSize / textInputSize) * 100)
    # Return the percentage of the input string that matched the pattern
    return percentMatched

#*********************************************************************
# findTokens() METHOD:
#*********************************************************************
# this method returns all of the raw, unfiltered tokens shared
# between two strings within the parameters
def findTokens(inputA, inputB,isCaseSensative):
    # case sensativity is enabled by default but can be
```

```
    # modified if desired.
    # if case sensative, then set inputA/B to lowercase
    if isCaseSensative:
        inputA = inputA.lower()
        inputB = inputB.lower()
    # store the identified topens
    identifiedTokens = []
    # for 0 to the length of inputA
    for x in range(len(inputA)):
        # for x+1 to the lenght of inputA plus one
        for y in range(x+1,len(inputA)+1):
            # get substring between x and y in inputA
            subString = inputA[x:y]
            # if the substring is found in inputB then it is a
            # raw token and then that token is added to identifiedTokens
            if subString in inputB:
                identifiedTokens.append(subString)
    # return the list of raw tokens
    return list(identifiedTokens)

#**************************************************************************
# findAllTokenIndexes() METHOD:
#**************************************************************************
# return all index pairs of the given token within a string to search.
def findAllTokenIndexes(string,token,isCaseSensative=False):
    # create a list to store the result
    result = []
    # by default case sensativity is respected by default

    #result.append(token)
    startPosition = 0
    # while the start position is less than the lenth of the given string
    while startPosition < len(string):
        # if case sensative perform the find operation
        if(isCaseSensative):
            currentPosition = string.find(token, startPosition)
        # if not case sensative then perform the given find operation
        else:
            currentPosition = string.lower().find(token.lower(), startPosition)
        # if the start position is -1 then stop of the operation
        if currentPosition == -1:
            break
        # -1 added to correct offset error
        # update the result
        result.append([currentPosition,currentPosition+len(token)])
```

```
        # update the startPosition variable
        startPosition = currentPosition + 1
    # return the final result
    return result

#############################################################################
# PROGRAM CLASSES
#############################################################################
# Token Class:
#*****************************************************************************
# This class stores token strings with their inputAIndexes and
# inputBIndexes
class Token:
    #default constructor
    def __init__(self, tokenString, inputATokenIndexes, inputBTokenIndexes):
        self.inputATokenIndexes = inputATokenIndexes
        self.inputBTokenIndexes = inputBTokenIndexes
        self.tokenString = tokenString
    # getter method for tokenString
    def getTokenString(self):
        return self.tokenString
    # getter method for inputATokenIndexes
    def getInputATokenIndexes(self):
        return self.inputATokenIndexes
    # getter method for inputBTokenIndexes
    def getInputBTokenIndexes(self):
        return self.inputBTokenIndexes
    # toString method for token class
    def __str__(self):
        return "\""+self.tokenString+"\" | A = "+str(self.inputATokenIndexes)+" | B =
"+str(self.inputBTokenIndexes)

#*****************************************************************************
# TokenCollection class
#*****************************************************************************
# This class manages all of the token objects
class TokenCollection:
    # default constructor
    def __init__(self):
        # changed from a tuple to a list
        self.allTokenData = []
    # add token to Collection class object
    def update(self, tokenString, inputATokenIndexes, inputBTokenIndexes):
        # if the tokenString object already exists then update the existing
        # object witht he inputA and inputB TokenIndexess
```

```
        if tokenString in [token.tokenString for token in self.allTokenData]:
            existingObject = next(token for token in self.allTokenData if token.tokenString ==
tokenString)
            existingObject.inputATokenIndexes.extend(inputATokenIndexes)
            existingObject.inputBTokenIndexes.extend(inputBTokenIndexes)
        #if the object does not exist, then create it
        else:
            self.allTokenData.append(Token(tokenString, inputATokenIndexes,
inputBTokenIndexes))
    # remove redundant index pairs from token objects in TokenCollection
    def removeRedundantPairs(self):
        for tokens in self.allTokenData:
            # create list to store result
            uniqueInputAPairs = []
            # for every pair in inputATokenIndexes for the given object
            # if the pair does not already exist then update the list
            for pair in tokens.inputATokenIndexes:
                if pair not in uniqueInputAPairs:
                    uniqueInputAPairs.append(pair)
            # update the inputATokenIndexes to the filtered result
            tokens.inputATokenIndexes = uniqueInputAPairs
            uniqueInputBPairs = []
            # for every pair in inputATokenIndexes for the given object
            # if the pair does not already exist then update the list
            for pair in tokens.inputBTokenIndexes:
                if pair not in uniqueInputBPairs:
                    uniqueInputBPairs.append(pair)
            # update the inputATokenIndexes to the filtered result
            tokens.inputBTokenIndexes = uniqueInputBPairs
        #print("redundant pairs removed")
    # remove underlapping index pairs from token objects in TokenCollection
    def removeUnderlaps(self):
        # underlap cleanup for inputATokenIndexes
        for token in range(0,len(self.allTokenData)):
            #print("*"+str(self.allTokenData[token]))
            pairsToRemove = []  # to store the index pairs to be removed from
inputATokenIndexes
            for indexPair in self.allTokenData[token].inputATokenIndexes:
                #print("\t"+str(indexPair))
                for compareToken in range(0,len(self.allTokenData)):
                    # for every comparedIndexPair in allTokenData
                    for comparedIndexPair in
self.allTokenData[compareToken].inputATokenIndexes:
                        if token == compareToken and indexPair == comparedIndexPair:
                            #print("\t\tomit")
```

```
                    pass
                else:
                    if indexPair[0] >= comparedIndexPair[0] and indexPair[1] <=
comparedIndexPair[1]:
                        #print("\t\t PAIR UNDERLAPPED UNDER: "+str(comparedIndexPair))
                        pairsToRemove.append(indexPair)  # add the underlapping pair to the
removal list
                    else:
                        #print("\t\t"+str(comparedIndexPair))
                        pass
        self.allTokenData[token].inputATokenIndexes = [pair for pair in
self.allTokenData[token].inputATokenIndexes if pair not in pairsToRemove]
    # same process as above but for inputBTokenIndexes
    for token in range(0,len(self.allTokenData)):
        #print("*"+str(self.allTokenData[token]))
        pairsToRemove = []  # to store the index pairs to be removed from
inputATokenIndexes
        for indexPair in self.allTokenData[token].inputBTokenIndexes:
            #print("\t"+str(indexPair))
            for compareToken in range(0,len(self.allTokenData)):
                for comparedIndexPair in
self.allTokenData[compareToken].inputBTokenIndexes:
                    if token == compareToken and indexPair == comparedIndexPair:
                        #print("\t\tomit")
                        pass
                    else:
                        if indexPair[0] >= comparedIndexPair[0] and indexPair[1] <=
comparedIndexPair[1]:
                            #print("\t\t PAIR UNDERLAPPED UNDER: "+str(comparedIndexPair))
                            pairsToRemove.append(indexPair)  # add the underlapping pair to the
removal list
                        else:
                            #print("\t\t"+str(comparedIndexPair))
                            pass
        self.allTokenData[token].inputBTokenIndexes = [pair for pair in
self.allTokenData[token].inputBTokenIndexes if pair not in pairsToRemove]
    #print("removing underlaping tokens")
  # remove empty token pairs from token objects in TokenCollection
  def cleanEmpty(self):
    #self.allTokenData = [token for token in self.allTokenData if token.inputATokenIndexes
and token.inputBTokenIndexes]
    result = []
    for token in self.allTokenData:
        # if the indexes for inputA and inputB are both NOT empty then keep them
        # and add them to the list to return
```

```
            if token.inputATokenIndexes and token.inputBTokenIndexes:
                result.append(token)
        # return the result
        self.allTokenData = result

    # remove single character tokens from the token Collection class
    def filter(self):
        # create list to store result
        filteredList = []
        # for every token in allTokenData
        for token in self.allTokenData:
            # if the length is over 1 then keep the entry
            if len(token.tokenString) > 1:
                filteredList.append(token)
        self.allTokenData = filteredList
        #print("filtering irrelevant tokens")
    # getter method to compile and return the datastream for inputA index stream
    def getADataStream(self):
        stream = []
        # for every object in allTokenData
        for token in self.allTokenData:
            # update the inputAPairs from the current object values
            # for all of the pairs, append and create the data stream
            inputAPairs = token.getInputATokenIndexes()
            for pairs in inputAPairs:
                stream.append([pairs,str(token.getTokenString())])
            #print("\t"+str(token.getTokenString()))
        stream = sorted(stream, key=lambda x: x[0])
        return stream
    # getter method to compile and return the datastream for inputB index stream
    def getBDataStream(self):
        stream = []
        for token in self.allTokenData:
            # update the inputBPairs from the current object values
            inputBPairs = token.getInputBTokenIndexes()
            # for all of the pairs, append and create the data stream
            for pairs in inputBPairs:
                stream.append([pairs,str(token.getTokenString())])
            #print("\t"+str(token.getTokenString()))
        stream = sorted(stream, key=lambda x: x[0])
        return stream
    #def export(self):
    #    return self.allTokenData
    # toString method
    def __str__(self):
```

```
        # if the emptu is empty then return "this object is empty"
        if not len(self.allTokenData):
            return "This object is empty"
        result = ""
        # for every toke in the allTokenData add it to result
        for token in self.allTokenData:
            result += str(token) + "\n"
        # return result
        return result

#**************************************************************************
# TokenAnalytics Class
# This class manages the token collection to clean up and process
# the data found from the token collection class
#**************************************************************************
class TokenAnalytics:
    # default constructor
    def __init__(self,inputA,inputB,isCaseSensative):
        self.inputA = inputA
        self.inputB = inputB
        self.isCaseSensative = isCaseSensative
        # create tokenCollection object
        self.tokenData = TokenCollection()
        self.update(inputA,inputB,isCaseSensative)

    # update the token data for tokenData
    def update(self,inputA,inputB,isCaseSensative):
        tokens = findTokens(inputA, inputB,isCaseSensative)
        # for every token in tokens update tokenData object
        for items in tokens:
            self.tokenData.update(str(items),findAllTokenIndexes(inputA,items),findAllTokenInde
xes(inputB,items))
        # perform data cleaning functions for tokenData
        self.tokenData.removeRedundantPairs()
        self.tokenData.removeUnderlaps()
        self.tokenData.filter()
        self.tokenData.cleanEmpty()

    # getter method for inputA
    def getInputA(self):
        return self.getInputA

    # setter method for inputA
    def setInputA(self, inputA):
        self.inputA = inputA
```

```
        self.update(self.inputA,self.inputB,self.isCaseSensative)

    # getter method for inputB
    def getInputB(self):
        return self.getInputB

    # setter method for inputB
    def setInputB(self, inputB):
        self.inputB = inputB
        self.update(self.inputA,self.inputB,self.isCaseSensative)

    # getter method for isCaseSensative
    def getIsCaseSensative(self):
        return self.isCaseSensative

    # setter method for isCaseSensative
    def setIsCaseSensative(self,isCaseSensative):
        self.isCaseSensative = isCaseSensative
        self.update(self.inputA,self.inputB,self.isCaseSensative)

    # getter method for the tokenData
    def getTokenData(self):
        # this returns a "database" of all of the token
        # data shared between inputA and inputB
        return self.tokenData

    # getter method for the inputADataStream
    def getADataStream(self):
        # this is the data that is used for raw similarity analytics
        return self.tokenData.getADataStream()

    # getter method for the inputBDataStream
    def getBDataStream(self):
        # this is the data that is used for raw similarity analytics
        return self.tokenData.getBDataStream()

    # this method return the unique partially filtered tokens from inputA and inputB
    def getUniqueTokensStrings(self):
        # set allTokenStrings to every string from allTokenData
        allTokenStrings = [str(x.getTokenString()) for x in self.tokenData.allTokenData]
        # return the result
        return allTokenStrings

    # this method calculate the average lenght of tokens
    def getAverageTokenLength(self):
```

```
        # add all of the strings from the tokenData
        allTokenStrings = [str(x.getTokenString()) for x in self.tokenData.allTokenData]
        # take the leight of the allTokenStrings and divide that by the lenghth of allTokenStrings
        averageTokenLength = sum(len(token) for token in
allTokenStrings)/len(allTokenStrings)
        # return the average
        return averageTokenLength

    # this method delete all information in this object
    def clear(self):
        self.inputA = None
        self.inputB = None
        self.isCaseSensative = None
        self.tokenData = TokenCollection()

##############################################################################
# CODECHECK ENGINES
##############################################################################
# These are methods that actually perform analytical process
# of textual analysis
##############################################################################

# this method returns just the shared tokens between inputA and inputB
# this method does not remove token fragments that are cross-shared
# between actually tokens.
def getRawTokens(inputA,inputB,isCaseSensative):
    return findTokens(inputA, inputB,isCaseSensative)

# this method returns the shared tokens between inputA and inputB
# however it does remove the token fragments that are redundant.
# Some irrelevant tokens may be present since CodeCheck 2.0's
# pattern recognition algorithm was not implemented in this method.
def getRefinedTokens(inputA,inputB):
    allTokenData = TokenAnalytics(inputA,inputB,True)
    # return the all of the uniqueTokenStrings
    return allTokenData.getUniqueTokensStrings()

# This is the optimized legacy algorithm for CodeCheck 1.0
# This algorithm does not analyze the usage of tokens
# between inputs but instead it just checks raw matches
# of tokens between two inputs and not how they are use.
def CodeCheckLegacy(inputA,inputB):
    allTokenData = TokenAnalytics(inputA,inputB,True)
    #x = a.getTokenData()
    rawAStream = allTokenData.getADataStream()
```

```
    rawBStream = allTokenData.getBDataStream()

    # get similarity formatted html for both of the inputs relative to eachother
    inputAFormattedHTML = getSimilarityHTML(rawAStream,inputA)
    inputBFormattedHTML = getSimilarityHTML(rawBStream,inputB)

    # get similarity percentage score for both of the input relative to eachother
    inputASimilarityPercentage = calculatePercentage(rawAStream,inputA)
    inputBSimilarityPercentage = calculatePercentage(rawBStream,inputB)
    # put all of the variables/objects to return into a single list for
    # easse of consolidation.
    result = [rawAStream,
        inputASimilarityPercentage,
        inputAFormattedHTML,rawBStream,
        inputBSimilarityPercentage,
        inputBFormattedHTML]
    # return the result
    return result


def CodeCheck2(inputA,inputB):
    # get all token data from inputs
    allTokenData = TokenAnalytics(inputA,inputB,True)
    # save the raw data stream data
    rawAStream = allTokenData.getADataStream()
    rawBStream = allTokenData.getBDataStream()

    ###################################################
    # PATTERN CLASS
    # Internal Class
    ###################################################
    # this is a the class that take in and processes the
    # list and generated by the findSharedPatterns()
    # method, each of the repetitions of the list
    # have a unique pattern saved to the class and
    # for every reidentifid pattern the data
    # for the given object is updated, then the
    # data is cleaned and reformed with only
    # relevant information.
    class Pattern:
        # default constructor
        def __init__(self, pattern):
            self.pattern = pattern
            self.inputAPairs = []
            self.inputBPairs = []
```

```
    # add inputAPair and inputBPair data into existing data lists
    def addData(self, inputAPair, inputBPair):
        self.inputAPairs.append(inputAPair)
        self.inputAPairs = list(set(map(tuple, self.inputAPairs)))
        self.inputAPairs = [list(x) for x in self.inputAPairs]
        self.inputBPairs.append(inputBPair)
        self.inputBPairs = list(set(map(tuple, self.inputBPairs)))
        self.inputBPairs = [list(x) for x in self.inputBPairs]

#*********************************************
# This methods are for managing the pattern class
# this find or creates objects as needed be
def managePatternObject(patternList, patternObjects):
    # for every patternObject in patternObjects
    for patternObject in patternObjects:
        # if the patrtern is equal to patternlist then return patternObject
        if patternObject.pattern == patternList:
            return patternObject
    # otherwise make a new pattern object and return that
    newPatternObject = Pattern(patternList)
    patternObjects.append(newPatternObject)
    return newPatternObject
# this returns all of the inputAPairs from all of the object in the class
def getAPairs(patternObjects):
    result = []
    # for every patternObject in patternObjects
    for patternObject in patternObjects:
        for i in range(len(patternObject.inputAPairs)):
            # update the result with inputAPairs from patternObject
            result.append(patternObject.inputAPairs[i])
    return result
# this returns all of the inputBPairs from all of the object in the class
def getBPairs(patternObjects):
    result = []
    for patternObject in patternObjects:
        for i in range(len(patternObject.inputBPairs)):
            result.append(patternObject.inputBPairs[i])
    return result
# remove the underlapping pairs for inputAPairs
def removeAPairUnderlaps(patternObjects):
    for patternObject in patternObjects:
        i = 0
        # while the value of i is less than the patternObject inputAPairs
        while i < len(patternObject.inputAPairs):
            hasUnderLap = False
```

```
            for comparison in getAPairs(patternObjects):
                # if the current pair is inside of other pair then discard as an underlap
                if (patternObject.inputAPairs[i][0] >= comparison[0] and
                    patternObject.inputAPairs[i][1] <= comparison[1] and
                    (comparison != patternObject.inputAPairs[i])):
                    hasUnderLap = True
                    break
            # if an underlap is present then discard
            if hasUnderLap:
                patternObject.inputAPairs.pop(i)
            # updat the value of i
            else:
                i += 1
    # remove the underlapping pairs for inputBPairs
    def removeBPairUnderlaps(patternObjects):
        # for every patternObject in patternObject
        for patternObject in patternObjects:
            i = 0
            # while i is less than the lenght of patternObject inputBPairs
            while i < len(patternObject.inputBPairs):
                hasUnderLap = False
                for comparison in getBPairs(patternObjects):
                    # a -1 was added after patternObject.inputBPairs[i][1] to fix pattern offset errors
                    # if the current pair is inside of other pair then discard as an underlap
                    if (patternObject.inputBPairs[i][0] >= comparison[0] and
                        patternObject.inputBPairs[i][1] <= comparison[1] and
                            (comparison != patternObject.inputBPairs[i])):
                        hasUnderLap = True
                        break
                # if there is an underlap then discard
                if hasUnderLap:
                    patternObject.inputBPairs.pop(i)
                # update the value of i
                else:
                    i += 1
    # for every sublist (object update)
    # inside of the the parsed input
    # process that and update the process
    # class object
    def processList(input):
        # create a list to store the pattern objects
        patternObjects = []
        # for every item in the list
        for item in input:
            patternList = item[0]
```

```
            inputAPair = item[1]
            inputBPair = item[2]
            patternObject = managePatternObject(patternList, patternObjects)
            patternObject.addData(inputAPair, inputBPair)
        # return the patternObject object
        return patternObjects
    # remove an empty pattern object
    # from the pattern class if
    # either the inputAPairs or inputBPairs
    # are empty
    def removeEmptyPatterns(patternObjects):
        i = 0
        # while is less than the length of the patternObjects
        while i < len(patternObjects):
            # if there is a pattern that is empty then remove it
            if not patternObjects[i].inputAPairs or not patternObjects[i].inputBPairs:
                patternObjects.pop(i)
            # update the index counter
            else:
                i += 1
    # this method is used to drive the empty
    # object removal method above.
    def removeIrrelevantPatterns(patternObjects):
        removeEmptyPatterns(patternObjects)
    #**********************************************
    # CROSS REFERENCING PATTERN ALGORITHM:
    #**********************************************
    # this method take in two list of strings and goes through
    # and identified all patterns of string of two
    # or more strings in a sequential order.
    # these patterns are returned as a series of list in
    # the format below: [pattern][inputAPair][inputBPair]
    # each of the patterns are identified and merged together
    # in a set of list inside of list, so patterns
    # will be repeated but with different data points

    # This is an example output of the findSharedPatterns() method
    #    [['this ', ' is here'], [0, 1], [0, 1]],
    #    [['this ', ' is here', 'this ', ' a demo'], [0, 3], [5, 8]],
    #    [[' is here', 'this ', ' a demo'], [1, 3], [6, 8]],
    def findSharedPatterns(a, b):
        result = []
        for i in range(len(a)):
            for j in range(len(b)):
                # Check if the current elements match
```

```
            if a[i] == b[j]:
                # Initialize the shared pattern and indices
                sharedPattern = [a[i]]
                aIndices = [i, i]
                bIndices = [j, j]
                # Check for matching subsequent elements
                k = 1
                # using the value of i and k with length of a and b with intersections
                # cross compare and identify shared patterns
                while i + k < len(a) and j + k < len(b) and a[i + k] == b[j + k]:
                    sharedPattern.append(a[i + k])
                    aIndices[1] = i + k
                    bIndices[1] = j + k
                    k += 1
                # FIX: the second condition was added to solve the issue of dropping
                #   clusters with multiple words in it
                # IF the pattern is longer than 1 element
                # OR the element has two or more words in it,
                # THEN add it to the result
                if len(sharedPattern) > 1 or bool(len(sharedPattern[0].split()) > 2):
                    result.append([sharedPattern, aIndices, bIndices])
        # return the given list entry
        return result
    ###############################################################
    unProcessedPatterns = findSharedPatterns([x[1] for x in rawAStream],[x[1] for x in
rawBStream])
    processedPatterns = processList(unProcessedPatterns)
    # clean the data
    removeAPairUnderlaps(processedPatterns)
    removeBPairUnderlaps(processedPatterns)
    removeEmptyPatterns(processedPatterns)
    # create the variables that store the final processed pattern data
    filteredAStream = []
    filteredBStream = []
    # for every pattern object inside of the processedPatterns object
    for patternObject in processedPatterns:
        #print(f"Pattern: {patternObject.pattern}")
        #print(f"\tInput A Pairs: {patternObject.inputAPairs}")
        # for every index pair inside of inputAPairs for the given pattern
        for pairs in patternObject.inputAPairs:
            filteredAStream += rawAStream[pairs[0]:pairs[1]+1]
        #print(f"\tInput B Pairs: {patternObject.inputBPairs}")
        # for every index pair inside of inputBPairs for the given pattern
        for pairs in patternObject.inputBPairs:
            filteredBStream += rawBStream[pairs[0]:pairs[1]+1]
```

```
    # get similarity formatted html for both of the inputs relative to eachother
    inputAFormattedHTML = getSimilarityHTML(filteredAStream,inputA)
    inputBFormattedHTML = getSimilarityHTML(filteredBStream,inputB)
    # get similarity percentage score for both of the input relative to eachother
    inputASimilarityPercentage = calculatePercentage(filteredAStream,inputA)
    inputBSimilarityPercentage = calculatePercentage(filteredBStream,inputB)
    # compile the results into a single list to return
    result = [filteredAStream,
        inputASimilarityPercentage,
        inputAFormattedHTML,
        filteredBStream,
        inputBSimilarityPercentage,
        inputBFormattedHTML]
    # return the result
    return result


"""
Program Section:    Graphical User Interface
Description:        These are a series of classes that create
                    and handle the operation of graphical
                    user interface. This section also has a
                    default method that starts the application.
"""
################################################################################
#   MainInterfaceGUI Class
################################################################################
class MainInterfaceGUI(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.setObjectName("MainWindow")
        self.resize(1009, 412)
        self.setMinimumSize(QtCore.QSize(1009, 412))
        self.setMaximumSize(QtCore.QSize(1009, 412))
        self.CentralWidget = QtWidgets.QWidget(self)
        self.CentralWidget.setObjectName("CentralWidget")
        self.ProgramLabel = QtWidgets.QLabel(self.CentralWidget)
        self.ProgramLabel.setGeometry(QtCore.QRect(20, 10, 251, 41))
        font = QtGui.QFont()
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.ProgramLabel.setFont(font)
        self.ProgramLabel.setObjectName("ProgramLabel")
        self.InputATextEdit = QtWidgets.QPlainTextEdit(self.CentralWidget)
```

```
        self.InputATextEdit.setGeometry(QtCore.QRect(20, 120, 321, 221))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.InputATextEdit.setFont(font)
        self.InputATextEdit.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.InputATextEdit.setObjectName("InputATextEdit")
        self.Line = QtWidgets.QFrame(self.CentralWidget)
        self.Line.setGeometry(QtCore.QRect(20, 50, 971, 20))
        self.Line.setFrameShadow(QtWidgets.QFrame.Raised)
        self.Line.setFrameShape(QtWidgets.QFrame.HLine)
        self.Line.setObjectName("Line")
        self.InputALabel = QtWidgets.QLabel(self.CentralWidget)
        self.InputALabel.setGeometry(QtCore.QRect(20, 80, 91, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.InputALabel.setFont(font)
        self.InputALabel.setObjectName("InputALabel")
        self.InputBTextEdit = QtWidgets.QPlainTextEdit(self.CentralWidget)
        self.InputBTextEdit.setGeometry(QtCore.QRect(370, 120, 321, 221))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.InputBTextEdit.setFont(font)
        self.InputBTextEdit.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.InputBTextEdit.setObjectName("InputBTextEdit")
        self.InputBLabel = QtWidgets.QLabel(self.CentralWidget)
        self.InputBLabel.setGeometry(QtCore.QRect(370, 80, 91, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.InputBLabel.setFont(font)
        self.InputBLabel.setObjectName("InputBLabel")
        self.AlgorithmGroupBox = QtWidgets.QGroupBox(self.CentralWidget)
        self.AlgorithmGroupBox.setGeometry(QtCore.QRect(710, 110, 281, 231))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.AlgorithmGroupBox.setFont(font)
        self.AlgorithmGroupBox.setObjectName("AlgorithmGroupBox")
        self.legacyAlgoRadioButton = QtWidgets.QRadioButton(self.AlgorithmGroupBox)
        self.legacyAlgoRadioButton.setGeometry(QtCore.QRect(10, 110, 251, 41))
        self.legacyAlgoRadioButton.setObjectName("legacyAlgoRadioButton")
        self.StadardAlgoRadioButton = QtWidgets.QRadioButton(self.AlgorithmGroupBox)
        self.StadardAlgoRadioButton.setGeometry(QtCore.QRect(10, 60, 171, 31))
        self.StadardAlgoRadioButton.setChecked(True)
        self.StadardAlgoRadioButton.setObjectName("StadardAlgoRadioButton")
        self.AnalyzeButton = QtWidgets.QPushButton(self.AlgorithmGroupBox)
        self.AnalyzeButton.setEnabled(False)
```

```
        self.AnalyzeButton.setGeometry(QtCore.QRect(10, 167, 261, 51))
        self.AnalyzeButton.setObjectName("AnalyzeButton")
        self.InputBTextEdit.raise_()
        self.ProgramLabel.raise_()
        self.InputATextEdit.raise_()
        self.Line.raise_()
        self.InputALabel.raise_()
        self.InputBLabel.raise_()
        self.AlgorithmGroupBox.raise_()
        self.setCentralWidget(self.CentralWidget)
        self.StatusBar = QtWidgets.QStatusBar(self)
        self.StatusBar.setObjectName("StatusBar")
        self.setStatusBar(self.StatusBar)
        self.MenuBar = QtWidgets.QMenuBar(self)
        self.MenuBar.setGeometry(QtCore.QRect(0, 0, 1009, 29))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.MenuBar.setFont(font)
        self.MenuBar.setObjectName("MenuBar")
        self.InfoRibbonButton = QtWidgets.QMenu(self.MenuBar)
        self.InfoRibbonButton.setObjectName("InfoRibbonButton")
        self.menuHelp = QtWidgets.QMenu(self.MenuBar)
        self.menuHelp.setObjectName("menuHelp")
        self.setMenuBar(self.MenuBar)
        self.actionHelp = QtWidgets.QAction(self)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.actionHelp.setFont(font)
        self.actionHelp.setObjectName("actionHelp")
        self.actionGitHub = QtWidgets.QAction(self)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.actionGitHub.setFont(font)
        self.actionGitHub.setObjectName("actionGitHub")
        self.actionDocumentation = QtWidgets.QAction(self)
        self.actionDocumentation.setObjectName("actionDocumentation")
        self.actionHelp_2 = QtWidgets.QAction(self)
        self.actionHelp_2.setObjectName("actionHelp_2")
        self.actionGithubPage = QtWidgets.QAction(self)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.actionGithubPage.setFont(font)
        self.actionGithubPage.setObjectName("actionGithubPage")
        self.actionProgramVersion = QtWidgets.QAction(self)
        font = QtGui.QFont()
```

```
        font.setPointSize(10)
        self.actionProgramVersion.setFont(font)
        self.actionProgramVersion.setObjectName("actionProgramVersion")
        self.actionDocumentation = QtWidgets.QAction(self)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.actionDocumentation.setFont(font)
        self.actionDocumentation.setObjectName("actionDocumentation")
        self.InfoRibbonButton.addAction(self.actionGithubPage)
        self.InfoRibbonButton.addAction(self.actionProgramVersion)
        self.menuHelp.addAction(self.actionDocumentation)
        self.MenuBar.addAction(self.InfoRibbonButton.menuAction())
        self.MenuBar.addAction(self.menuHelp.menuAction())
        # ACTION LISTENERS
            # ribbon
        self.actionProgramVersion.triggered.connect(self.showProgramVersion)
        self.actionGithubPage.triggered.connect(self.showGithubPage)
        self.actionDocumentation.triggered.connect(self.showGithubDocumentationPage)

            # main page
        self.InputATextEdit.textChanged.connect(self.updateAnalyzeButton)
        self.InputBTextEdit.textChanged.connect(self.updateAnalyzeButton)
        self.legacyAlgoRadioButton.toggled.connect(self.updateAnalyzeButton)
        self.StadardAlgoRadioButton.toggled.connect(self.updateAnalyzeButton)
        self.AnalyzeButton.clicked.connect(self.analyzeInputs)
        # END OF ACTION LISTENERS
        self.retranslateUi(self)
        QtCore.QMetaObject.connectSlotsByName(self)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "CodeCheck 2.0"))
        self.ProgramLabel.setText(_translate("MainWindow", "CodeCheck 2.0"))
        self.InputALabel.setText(_translate("MainWindow", "Input A:"))
        self.InputBLabel.setText(_translate("MainWindow", "Input B:"))
        self.AlgorithmGroupBox.setTitle(_translate("MainWindow", "Algorithm Options:"))
        self.legacyAlgoRadioButton.setText(_translate("MainWindow", "CodeCheck 1.0
(Legacy)"))
        self.StadardAlgoRadioButton.setText(_translate("MainWindow", "CodeCheck 2.0"))
        self.AnalyzeButton.setText(_translate("MainWindow", "Analyze"))
        self.InfoRibbonButton.setTitle(_translate("MainWindow", "Info"))
        self.menuHelp.setTitle(_translate("MainWindow", "Help"))
        self.actionHelp.setText(_translate("MainWindow", "Version"))
        self.actionGitHub.setText(_translate("MainWindow", "GitHub"))
        self.actionDocumentation.setText(_translate("MainWindow", "Documentation"))
```

```
        self.actionHelp_2.setText(_translate("MainWindow", "Help"))
        self.actionGithubPage.setText(_translate("MainWindow", "Github Page"))
        self.actionProgramVersion.setText(_translate("MainWindow", "Version"))
        self.actionDocumentation.setText(_translate("MainWindow", "Documentation"))

    # this method is activated by "analyze" button
    # and all of the inputs are pulled and fed into the
    # correct CodeCheck algoritm
    def analyzeInputs(self):
        # this is a internal method that create the html for the general data browser in
        # the results GUI
        def makeGeneralDataHTML(algorithmType,inputAPercentage,inputBPercentage):
            html = "<h2>Similarity Percentages Relative to Comparison:</h2>"
            html += f"<div style='font-size: 10pt;'><strong>Algorithm:</strong>
{algorithmType}</div>"
            html += f"<div style='font-size: 10pt;'><strong>Input A:</strong>
{inputAPercentage}%</div>"
            html += f"<div style='font-size: 10pt;'><strong>Input B:</strong>
{inputBPercentage}%</div>"
            return html

        # if the inputs are 100% identical
        if((self.InputATextEdit.toPlainText().lower())==(self.InputBTextEdit.toPlainText().lower
())):
            # this if statement is intended to save computational power and to reduce any
            # chance of error as identical inputs are marked as such and no analytical algorithms
            # are required.
            # create custom mark tags since standard <mark></mark> tags don't work in PyQT5
            mark = "<span style=\"background-color: yellow;\">"
            endMark = "</span>"
            # create custom div tag with custom size to simplify code
            customDiv = "<div style='font-size: 10pt;'>"
            customDivEnd = "</div>"
            # create the results gui with the percentages as 100% as input with full highlight match
            self.openAnalyticalResultsGUI(
                makeGeneralDataHTML("None. Both inputs are identical","100","100"),
                f"{customDiv}{mark}{self.InputATextEdit.toPlainText()}{endMark}{customDivE
nd}",
                f"{customDiv}{mark}{self.InputBTextEdit.toPlainText()}{endMark}{customDivE
nd}")
        # if the inputs are not 100% identical
        else:
            # create custom mark tags since standard <mark></mark> tags don't work in PyQT5
            customDiv = "<div style='font-size: 10pt;'>"
            customDivEnd = "</div>"
```

```
        # if CodeCheck 1.0 (legacy) was selected
        if(self.legacyAlgoRadioButton.isChecked()):
            # store all of the data returned from codeCheckLegacy into a tuple
            data =
CodeCheckLegacy(self.InputATextEdit.toPlainText(),self.InputBTextEdit.toPlainText())
            # create the analysis results GUI from the data gathered from CodeCheckLegacy
            self.openAnalyticalResultsGUI(
                makeGeneralDataHTML("CodeCheck 1.0",data[1],data[4]),
                customDiv+data[2]+customDivEnd,
                customDiv+data[5]+customDivEnd)
        # if CodeCheck 2.0 was selected
        if(self.StadardAlgoRadioButton.isChecked()):
            # store all of the data returned from codeCheckLegacy into a tuple
            data =
CodeCheck2(self.InputATextEdit.toPlainText(),self.InputBTextEdit.toPlainText())
            # create the analysis results GUI from the data gathered from CodeCheck 2.0
            self.openAnalyticalResultsGUI(
                makeGeneralDataHTML("CodeCheck 2.0",data[1],data[4]),
                customDiv+data[2]+customDivEnd,
                customDiv+data[5]+customDivEnd)

    # action listener endpoint to process UX "analyze" button
    def updateAnalyzeButton(self):
        # if an algorithm option is checked and both text inputs are filled then enable the
        # analyze button
        if(self.InputATextEdit.toPlainText() and self.InputBTextEdit.toPlainText() and
           (self.legacyAlgoRadioButton.isChecked() or
           self.StadardAlgoRadioButton.isChecked())):
            self.AnalyzeButton.setEnabled(True)
        # otherwise, disable it.
        else:
            self.AnalyzeButton.setEnabled(False)

    # this prints out the program and codecheck version data
    def showProgramVersion(self):
        tk.messagebox.showinfo(title='Program Info', message='CodeCheck Version:
2.0v\nProgram Version: 1.0v')

    # this is standin for poping up a webpage
    def showGithubPage(self):
        url = 'https://github.com/Austin-Daigle/CodeCheck-2.0-Python-Research/tree/main'
        webbrowser.open(url)

    # this is standin for poping up a webpage
    def showGithubDocumentationPage(self):
```

```
        url = 'https://github.com/Austin-Daigle/CodeCheck-2.0-Python-
Research/blob/main/Documentation.md'
        webbrowser.open(url)

    # this class open the results gui data
    def
openAnalyticalResultsGUI(self,generalDataAnalytics,inputADataAnalytics,inputBDataAnalyt
ics):
        # create instance of AnalyticalResultsGUI class
        self.second_window =
AnalyticalResultsGUI(generalDataAnalytics,inputADataAnalytics,inputBDataAnalytics)
        # close first window
        self.close()
        # show second window
        self.second_window.show()

##########################################################################
#   AnalyticalResultGUI Class
##########################################################################
class AnalyticalResultsGUI(QtWidgets.QMainWindow):
    def __init__(self,generalDataAnalytics,inputADataAnalytics,inputBDataAnalytics):
        super().__init__()
        # create the internal variables for storing display info for the GUI
        self.generalDataAnalytics = generalDataAnalytics
        self.inputADataAnalytics = inputADataAnalytics
        self.inputBDataAnalytics = inputBDataAnalytics
        self.setObjectName("MainWindow")
        self.resize(575, 717)
        self.setMinimumSize(QtCore.QSize(575, 717))
        self.setMaximumSize(QtCore.QSize(575, 717))
        self.centralwidget = QtWidgets.QWidget(self)
        self.centralwidget.setObjectName("centralwidget")
        self.GeneralBrowser = QtWidgets.QTextBrowser(self.centralwidget)
        self.GeneralBrowser.setGeometry(QtCore.QRect(20, 110, 531, 111))
        self.GeneralBrowser.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.GeneralBrowser.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        self.GeneralBrowser.setObjectName("GeneralBrowser")
        self.InputABrowser = QtWidgets.QTextBrowser(self.centralwidget)
        self.InputABrowser.setGeometry(QtCore.QRect(20, 300, 531, 131))
        self.InputABrowser.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.InputABrowser.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        self.InputABrowser.setObjectName("InputABrowser")
        self.InputBBrowser = QtWidgets.QTextBrowser(self.centralwidget)
        self.InputBBrowser.setGeometry(QtCore.QRect(20, 470, 531, 131))
        self.InputBBrowser.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
```

```
        self.InputBBrowser.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)
        self.InputBBrowser.setObjectName("InputBBrowser")
        self.GeneralLabel = QtWidgets.QLabel(self.centralwidget)
        self.GeneralLabel.setGeometry(QtCore.QRect(20, 70, 311, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.GeneralLabel.setFont(font)
        self.GeneralLabel.setObjectName("GeneralLabel")
        self.InputALabel = QtWidgets.QLabel(self.centralwidget)
        self.InputALabel.setGeometry(QtCore.QRect(20, 260, 311, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.InputALabel.setFont(font)
        self.InputALabel.setObjectName("InputALabel")
        self.InputBLabel = QtWidgets.QLabel(self.centralwidget)
        self.InputBLabel.setGeometry(QtCore.QRect(20, 430, 311, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.InputBLabel.setFont(font)
        self.InputBLabel.setObjectName("InputBLabel")
        self.line = QtWidgets.QFrame(self.centralwidget)
        self.line.setGeometry(QtCore.QRect(20, 50, 531, 21))
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.ProgramLabel = QtWidgets.QLabel(self.centralwidget)
        self.ProgramLabel.setGeometry(QtCore.QRect(20, 10, 351, 41))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.ProgramLabel.setFont(font)
        self.ProgramLabel.setObjectName("ProgramLabel")
        self.CloseProgramButton = QtWidgets.QPushButton(self.centralwidget)
        self.CloseProgramButton.setGeometry(QtCore.QRect(20, 630, 531, 28))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.CloseProgramButton.setFont(font)
        self.CloseProgramButton.setObjectName("CloseProgramButton")
        self.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(self)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 575, 29))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.menubar.setFont(font)
        self.menubar.setObjectName("menubar")
        self.menuFile = QtWidgets.QMenu(self.menubar)
```

```
        font = QtGui.QFont()
        font.setPointSize(10)
        self.menuFile.setFont(font)
        self.menuFile.setObjectName("menuFile")
        self.setMenuBar(self.menubar)
        # Due to deadline the save options from the results page have been left
        # out until further notice
        #self.statusbar = QtWidgets.QStatusBar(self)
        #self.statusbar.setObjectName("statusbar")
        #self.setStatusBar(self.statusbar)
        #self.actionSave_Results_as_HTML = QtWidgets.QAction(self)
        #self.actionSave_Results_as_HTML.setObjectName("actionSave_Results_as_HTML")
        #self.actionSave_as = QtWidgets.QAction(self)
        #self.actionSave_as.setObjectName("actionSave_as")
        #self.actionSave_as_2 = QtWidgets.QAction(self)
        #self.actionSave_as_2.setObjectName("actionSave_as_2")
        #self.menuFile.addAction(self.actionSave_Results_as_HTML)
        #self.menuFile.addAction(self.actionSave_as_2)
        #self.menubar.addAction(self.menuFile.menuAction())

        # ACTION LISTENERS
        self.CloseProgramButton.clicked.connect(self.close)
        # END OF ACTION LISTENERS
        self.retranslateUi(self)
        QtCore.QMetaObject.connectSlotsByName(self)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "CodeCheck 2.0 Result"))
        self.GeneralBrowser.setHtml(_translate("MainWindow", self.generalDataAnalytics))
        self.InputABrowser.setHtml(_translate("MainWindow", self.inputADataAnalytics))
        self.InputBBrowser.setHtml(_translate("MainWindow", self.inputBDataAnalytics))
        self.GeneralLabel.setText(_translate("MainWindow", "General Analysis:"))
        self.InputALabel.setText(_translate("MainWindow", "Input A:"))
        self.InputBLabel.setText(_translate("MainWindow", "Input B:"))
        self.ProgramLabel.setText(_translate("MainWindow", "CodeCheck Results:"))
        self.CloseProgramButton.setText(_translate("MainWindow", "Close Program"))
        self.menuFile.setTitle(_translate("MainWindow", "File"))
        #self.actionSave_Results_as_HTML.setText(_translate("MainWindow", "Save"))
        #self.actionSave_as.setText(_translate("MainWindow", "Save as..."))
        #self.actionSave_as_2.setText(_translate("MainWindow", "Save as"))

    # this closes the current window object and program
    def closeProgram(self):
        self.close()
```

```
##########################################################################
#   MAIN METHOD FOR APPLICATION
##########################################################################
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    window = MainInterfaceGUI()
    window.show()
    sys.exit(app.exec_())
```

Program Testing:

Test Cases:

OpenAI's GPT-3.5 (Deprecated) and 4.0 API Models were utilized to generate test cases and corresponding HTML similarity comparisons for evaluation purposes. Due to the inherent characteristics of these advanced large language models with natural language processing capabilities, there may be minor or significant discrepancies in the machine-generated HTML comparisons. It is important to note that these machine-generated comparisons should be regarded as an additional reference for the outputs of CodeCheck 2.0 (OpenAI's Terms of Service and user disclaimers even recommend that end user exercise discretion with model outputs as its outputs may be completely inaccurate or false and also somewhat unpredictable), not definitive results. OpenAI's products were employed in this instance to produce test cases and reference HTML comparisons, owing to a non-availability of available model data concerning plagiarized code, text samples, and biometric data. Both models have been cited in the works cited section. The prompts for generating the test cases and references have been included below.

The following are six test cases demonstrating the functionality of CodeCheck 2.0, including similarity analysis for plain text, biometric data (DNA), and code. Each test case is structured as follows: machine-generated test cases and their respective HTML comparisons are displayed on the left, accompanied by an image illustrating the HTML rendering. The output data generated by the CodeCheck 2.0 application based on the provided inputs are presented on the right.

These are the prompts used to generate test cases (GPT-3.5 Legacy):

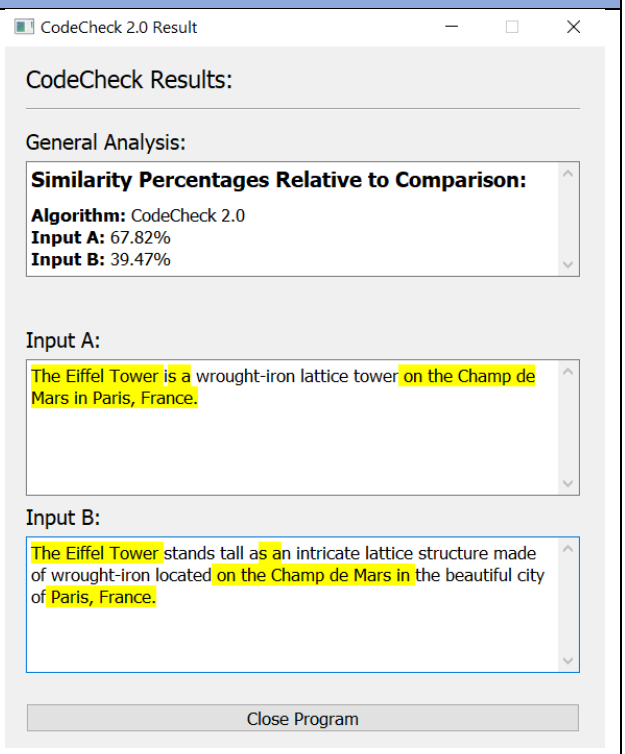
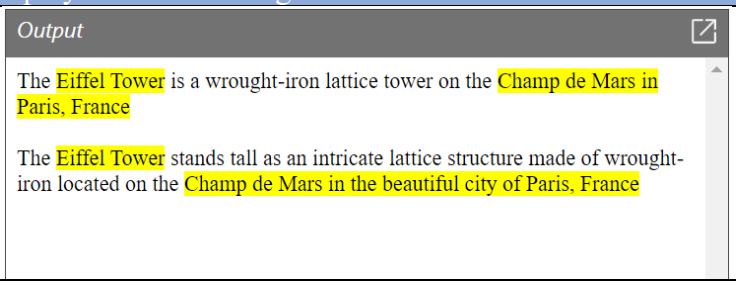
Get a short random sentence and print it out, then plagiarize that sentence (avoid changing the sentence structure, just change a few words) and print the results. Also, give the link to the page where the sentence was found.
Generate two stands of DNA of 36 characters with a space between every four characters and a shared gene of four characters between then
make a few lines of Python code and print it out, then plagiarize that code (avoid changing the structure, just change a few words) and print the results.

This prompt was used to create the html references:
This prompt was also modified to have the "sentences" portion of the prompt changed to code or DNA sequences for the other tests.

Take in these two sentences and for all of the text matched between them surround that text with a highlight tag in html and then print both sentences out sentence1: sentence2:

Textual Similarity Test Cases:

<table>
<tr><th colspan="2">Test Case #1</th></tr>
<tr><td>Machine-generated Test Case Sentences: (GPT-3)</td><td>CodeCheck 2.0 Algorithm Results:</td></tr>
<tr><td>Original sentence: "The Eiffel Tower is a wrought-iron lattice tower on the Champ de Mars in Paris, France."

Plagiarized sentence: "The Eiffel Tower stands tall as an intricate lattice structure made of wrought-iron located on the Champ de Mars in the beautiful city of Paris, France."</td><td rowspan="2"> 
</td></tr>
<tr><td>Machine-generated Test Case Sentences with HTML similarity highlights: (GPT-4)</td></tr>
<tr><td>Original sentence:
The <mark>Eiffel Tower</mark> is a wrought-iron lattice tower on the <mark>Champ de Mars in Paris, France</mark>
Plagiarized sentence:
The <mark>Eiffel Tower</mark> stands tall as an intricate lattice structure made of wrought-iron located on the <mark>Champ de Mars in the beautiful city of Paris, France</mark></td><td></td></tr>
<tr><td colspan="2">HTML similarity display from machine-generated test cases:</td></tr>
<tr><td colspan="2"> 
</td></tr>
</table>

Test Case #2

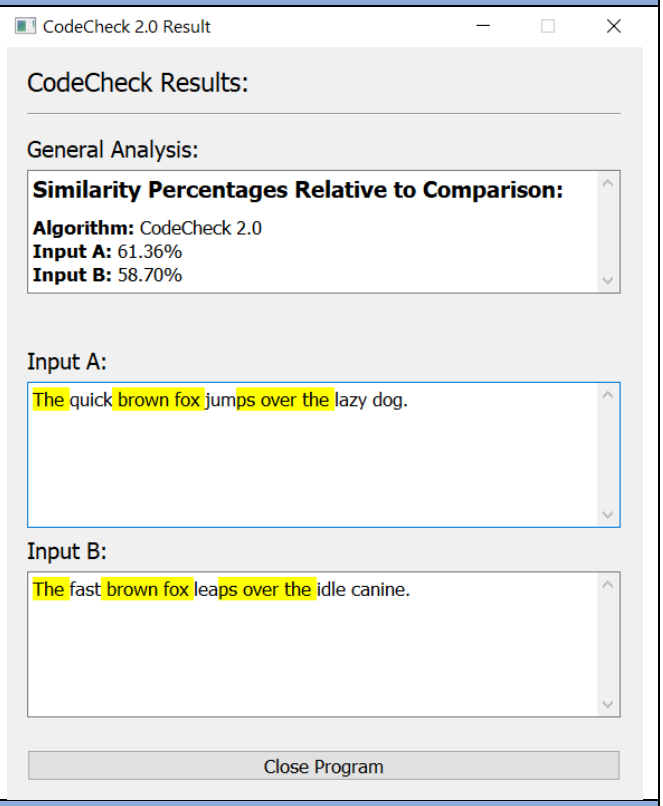
Machine-generated Test Case Sentences: (GPT-3)

Original sentence:
“The quick brown fox jumps over the lazy dog."
Plagiarized sentence:
"The fast brown fox leaps over the idle canine."

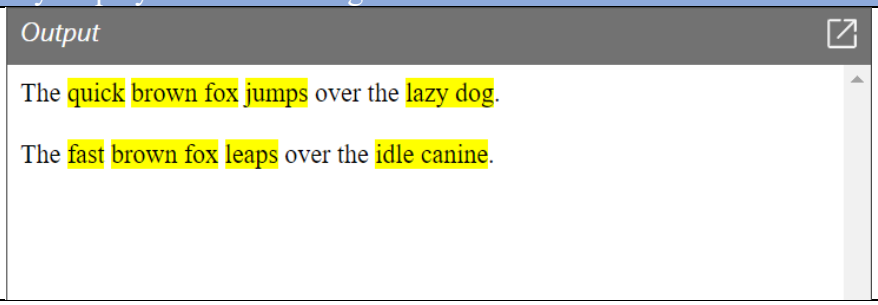
Machine-generated Test Case Sentences with HTML similarity highlights: (GPT-4)

Original sentence:
The <mark>quick</mark> <mark>brown fox</mark> <mark>jumps</mark> over the <mark>lazy dog</mark>.
Plagiarized sentence:
The <mark>fast</mark> <mark>brown fox</mark> <mark>leaps</mark> over the <mark>idle canine</mark>.

CodeCheck 2.0 Algorithm Results:



HTML similarity display from machine-generated test cases:



Biometric (DNA) Similarity Test Cases:

Test Case #3

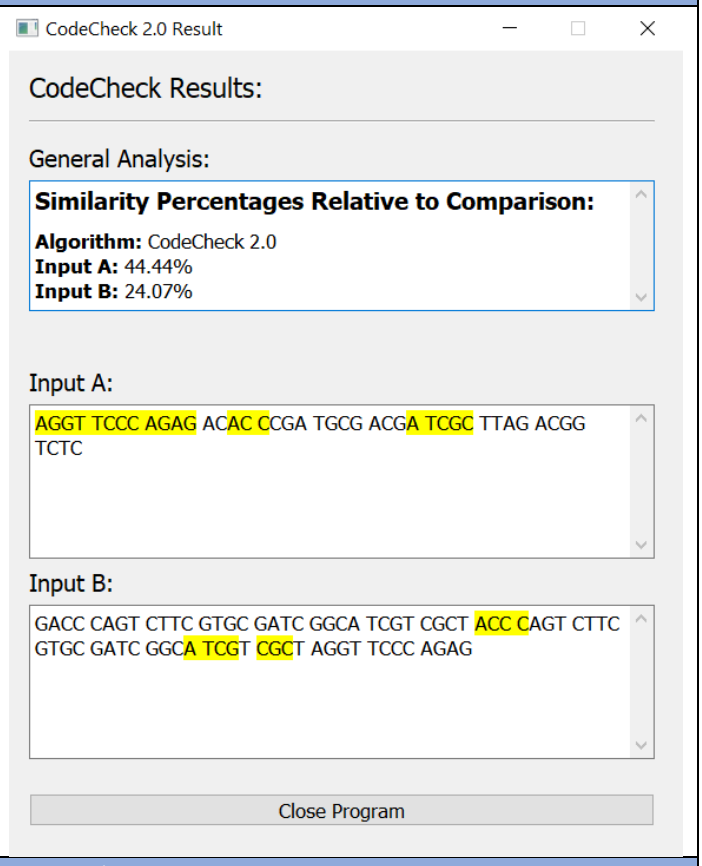
Machine-generated Test Case DNA Sequence: (GPT-3)

Sequence A:
AGGT TCCC AGAG ACAC CCGA TGCG ACGA TCGC TTAG ACGG TCTC
Sequence B:
GACC CAGT CTTC GTGC GATC GGCA TCGT CGCT AGGT TCCC AGAG

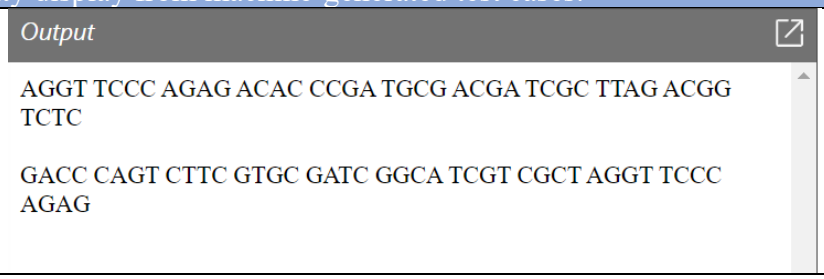
Machine-generated Test Case DNA Sequences with HTML similarity highlights: (GPT-4)

Sequence A:
AGGT TCCC AGAG ACAC CCGA TGCG ACGA TCGC TTAG ACGG TCTC
Sequence B:
GACC CAGT CTTC GTGC GATC GGCA TCGT CGCT AGGT TCCC AGAG

CodeCheck 2.0 Algorithm Results:



HTML similarity display from machine-generated test cases:



Test Case #4

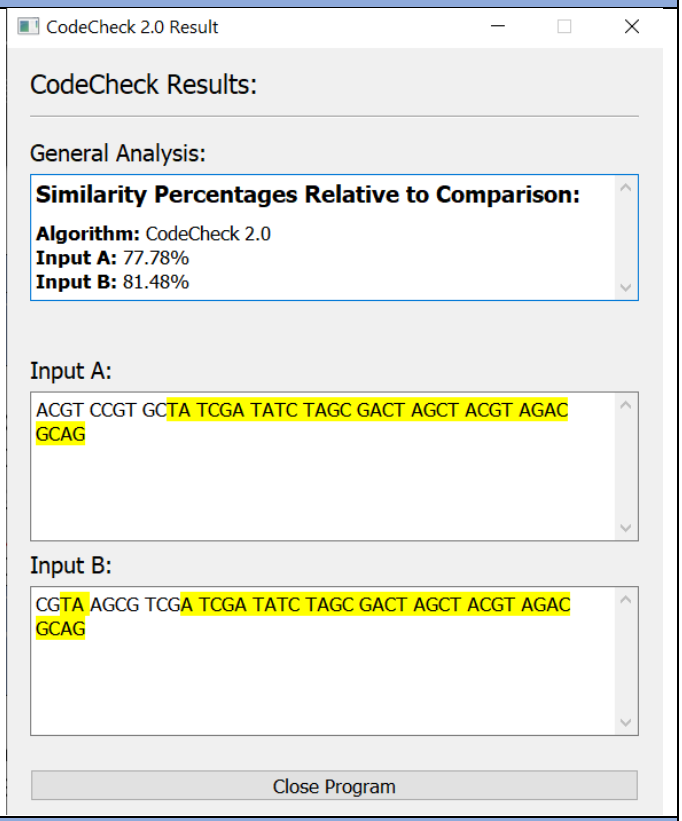
Machine-generated Test Case DNA Sequence: (GPT-3)

Sequence A:
ACGT CCGT GCTA TCGA TATC TAGC GACT AGCT ACGT AGAC GCAG
Sequence B:
CGTA AGCG TCGA TCGA TATC TAGC GACT AGCT ACGT AGAC GCAG

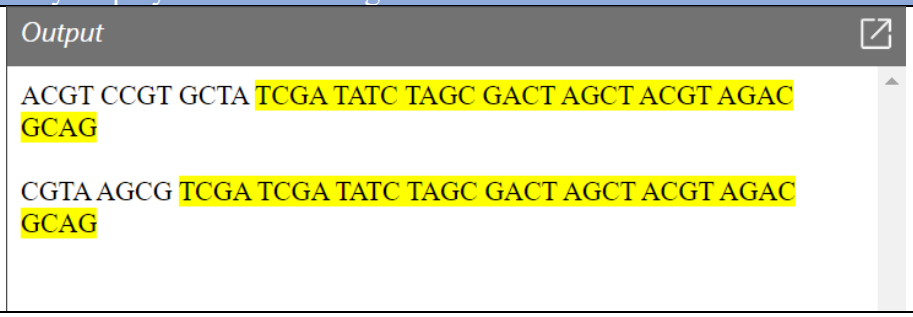
Machine-generated Test Case DNA Sequences with HTML similarity highlights: (GPT-4)

Sequence A:
ACGT CCGT GCTA <mark>TCGA TATC TAGC GACT AGCT ACGT AGAC GCAG</mark>
Sequence B:
CGTA AGCG <mark>TCGA TCGA TATC TAGC GACT AGCT ACGT AGAC GCAG</mark>

CodeCheck 2.0 Algorithm Results:



HTML similarity display from machine-generated test cases:



Python Code Similarity Test Cases:

Test Case #5

Machine-generated Test Case Code Sequence: (GPT-3)

Original Code:

```
length = 5
width = 3
area = length * width
print("The area of the rectangle is:", area)
```

Plagiarized Code:

```
breadth = 4
height = 6
total_area = breadth * height
print("The total area of the object is:", total_area)
```

Machine-generated Test Case Code Sequences with HTML similarity highlights: (GPT-4)

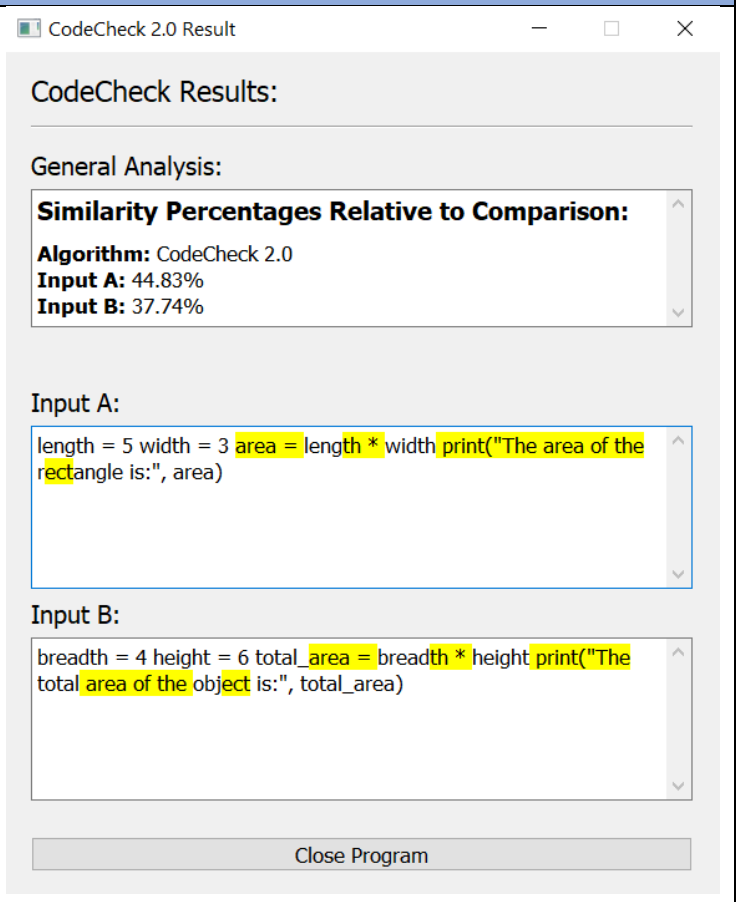
Original Code:

```
length = 5
width = 3
area = length * width
print("<mark>The area of the rectangle is:</mark>", area)
```

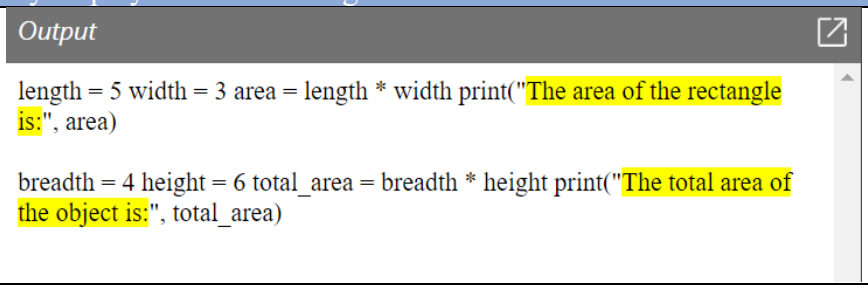
Plagiarized Code:

```
breadth = 4
height = 6
total_area = breadth * height
print("<mark>The total area of the object is:</mark>", total_area)
```

CodeCheck 2.0 Algorithm Results:



HTML similarity display from machine-generated test cases:



Test Case #6

Machine-generated Test Case Code Sequence: (GPT-3)

Original Code:

```
import random
num = random.randint(1, 10)
print("The random number is:", num)
```

Plagiarized Code:

```
import random
rand_num = random.randint(1, 10)
print("The random number is:",
rand_num)
```

Machine-generated Test Case Code Sequences with HTML similarity highlights: (GPT-4)

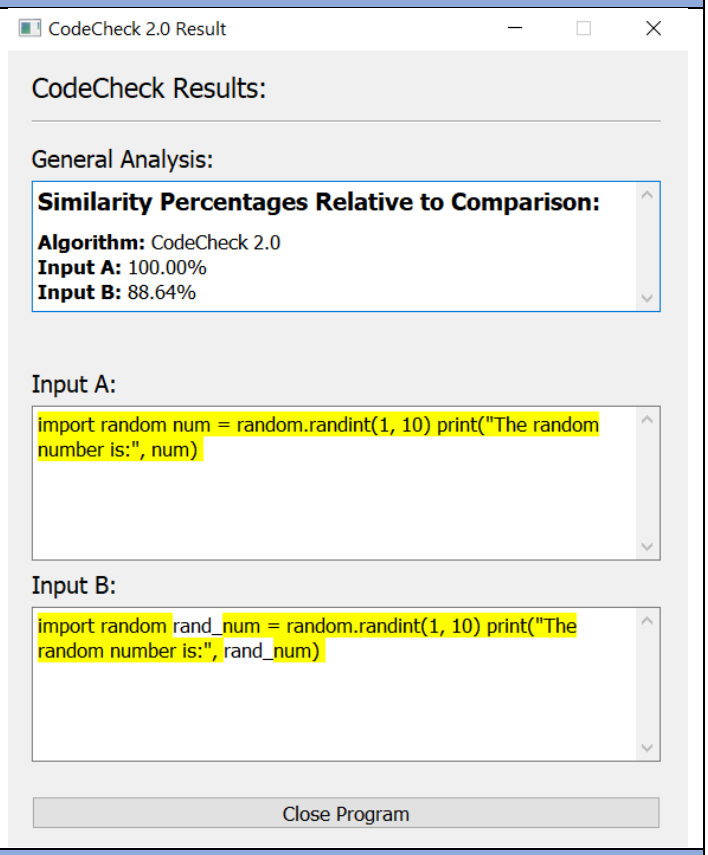
Original Code:

```
import random
<num = random.randint(1, 10)>
print("The random number is:", num)
```

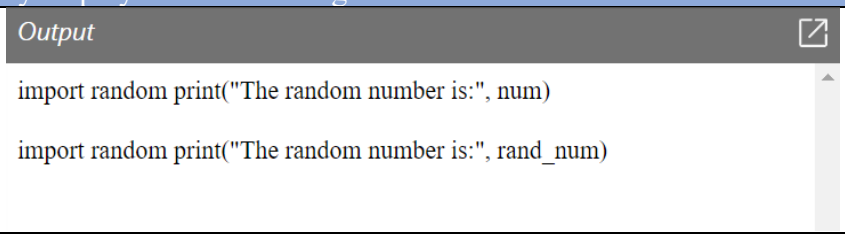
Plagiarized Code:

```
import random
<rand_num = random.randint(1, 10)>
print("The random number is:",
rand_num)
```

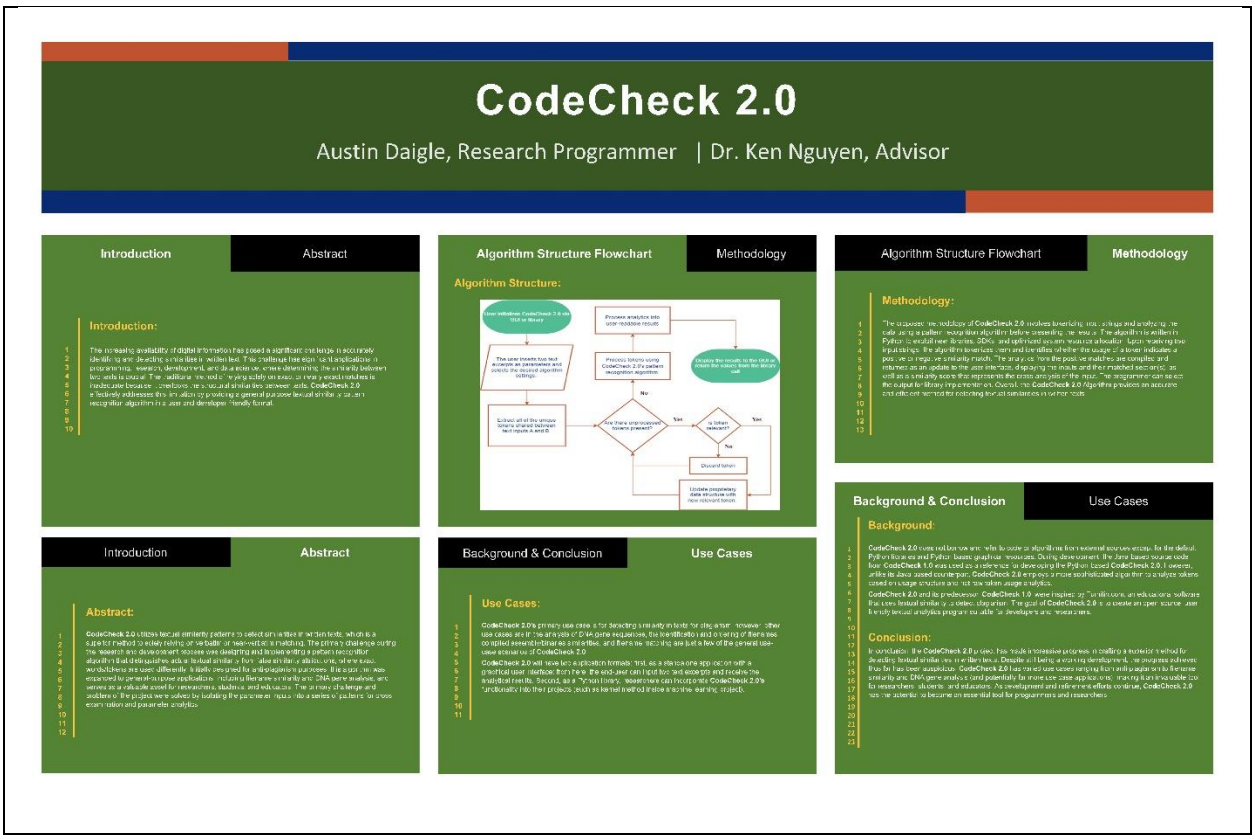
CodeCheck 2.0 Algorithm Results:



HTML similarity display from machine-generated test cases:



CIMS Symposium Poster:



Future Plans:

The subsequent phase of research will involve implementing large language models (LLMs) such as Generative Pre-trained Transformers (GPTs) with natural language processing (NLP) capabilities to enhance the code and textual similarity analysis process. This will identify similar usage patterns found in tokens and analyze semantic differences between inputs for paraphrasing detection.

Another crucial aspect to consider is integrating an input data storage system for retaining all prior inputs. This will facilitate the rapid creation of a database to identify similar text and code segments within the stored data. By incorporating these advanced, state-of-the-art features, a more robust and comprehensive similarity detection software package can be developed, capable of scaling and evolving to meet the needs of the computer science community.
Another crucial aspect to consider is integrating an input data storage system for retaining all prior inputs. This will facilitate the rapid creation of a database to identify similar text and code segments within the stored data. By incorporating these advanced, state-of-the-art features, a more robust and comprehensive similarity detection software package can be developed, capable of scaling and evolving to meet the needs of the computer science community.

Conclusion:

This research project has successfully developed an open-source, lightweight textual similarity algorithm that is versatile and applicable to multiple fields with various use cases without requiring external services, AI, machine learning, or proprietary software. The algorithm has successfully addressed the intended applications, such as anti-plagiarism (textual similarity analysis), repetitive file name detection, biometric data analysis, and textual similarity. The project has been optimized to cater to technical and non-technical researchers and users, making it easily integrated into existing Python projects.

References:

The core code for CodeCheck 2.0 and Legacy does not rely on external sources to design and construct the core algorithm. However, the CodeCheck 2.0 application utilizes the following external dependencies for the graphical user interface: PyQt5, QtCore, QtGui, QtWidgets, tkinter, messagebox from tkinter, and webbrowser. Each of these external Python packages is referenced on the works cited page.

Works Cited:

Developers, Staff. “Machine Generated Test Cases.” *ChatGPT3.5*, OpenAI, 2022, https://chat.openai.com/.

Developers, Staff. “Machine Generated HTML Test Case references.” *ChatGPT4*, OpenAI, 2023, https://chat.openai.com/.

Group, Qt. “PYQT5.” *PyPI*, Qt Group, 1995, https://pypi.org/project/PyQt5/.

Group, Qt. “PySide2.QtCore.” *PySide2.QtCore - Qt for Python*, Qt Group, 21 Apr. 2016, https://doc.qt.io/qtforpython-5/PySide2/QtCore/index.html.

Group, Qt. “PySide6.QtWidgets#.” *PySide6.QtWidgets - Qt for Python*, Qt Group, 1995, https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/index.html.

Group, Qt. “Qt Gui.” *Qt Documentation*, Qt Group, 2018, https://doc.qt.io/qt-5/qtgui-index.html.

Lumholt, Steen, and Guido van Rossum. “Tkinter - Python Interface to TCL/TK.” Edited by Fredrik Lundh, *Python Documentation*, Python Software Foundation, 1999, https://docs.python.org/3/library/tkinter.html.

Staff, Developer. “Webbrowser - Convenient Web-Browser Controller.” *Python Documentation*, Python Software Foundation, 2000, https://docs.python.org/3/library/webbrowser.html.