LAB-01
EC-9640
ARTIFICIAL INTELLIGENCE

LAKSHAN W.G.

2020/E/079

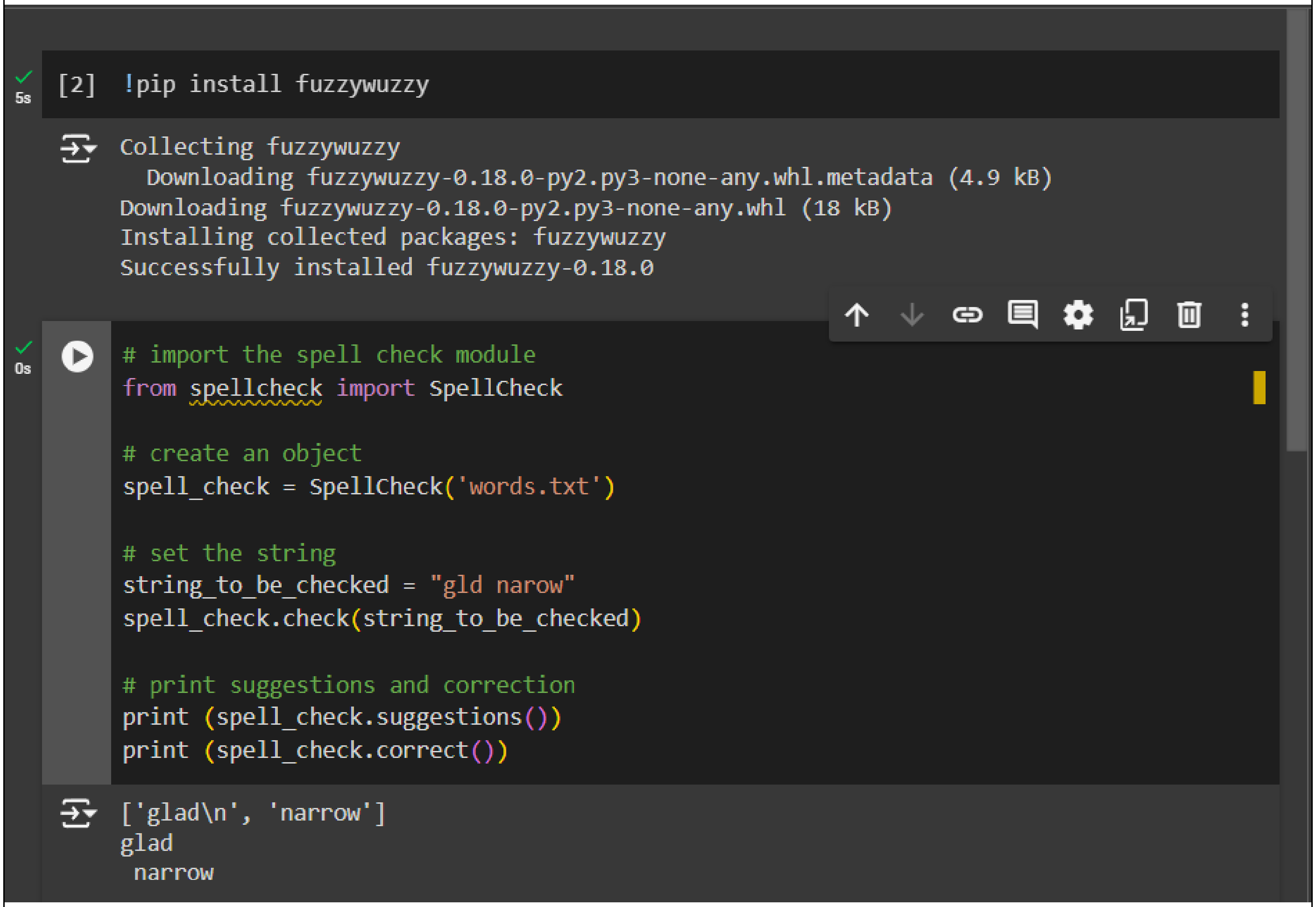
GROUP EG10

SEMESTER 7

16 OCT 2024

ILO: Apply the basic principles, models, and algorithms of Artificial Intelligence to solve problems;

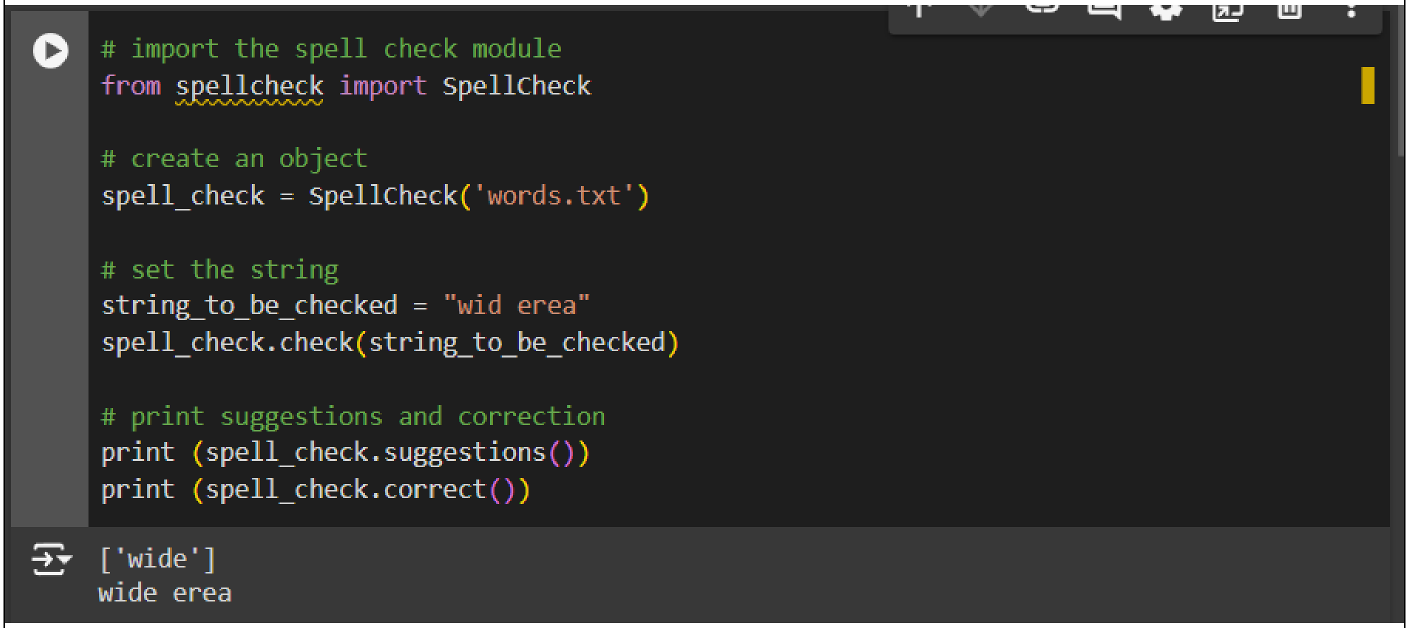
1. Execute the given application and find the spelling suggestions for the given example. Place the screenshot in the output file.



```
[2] !pip install fuzzywuzzy

Collecting fuzzywuzzy
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl.metadata (4.9 kB)
Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0
```

```
# import the spell check module
from spellcheck import SpellCheck

# create an object
spell_check = SpellCheck('words.txt')

# set the string
string_to_be_checked = "gld narow"
spell_check.check(string_to_be_checked)

# print suggestions and correction
print (spell_check.suggestions())
print (spell_check.correct())
```

```
['glad\n', 'narrow']
glad
 narrow
```



```
# import the spell check module
from spellcheck import SpellCheck

# create an object
spell_check = SpellCheck('words.txt')

# set the string
string_to_be_checked = "wid erea"
spell_check.check(string_to_be_checked)

# print suggestions and correction
print (spell_check.suggestions())
print (spell_check.correct())
```

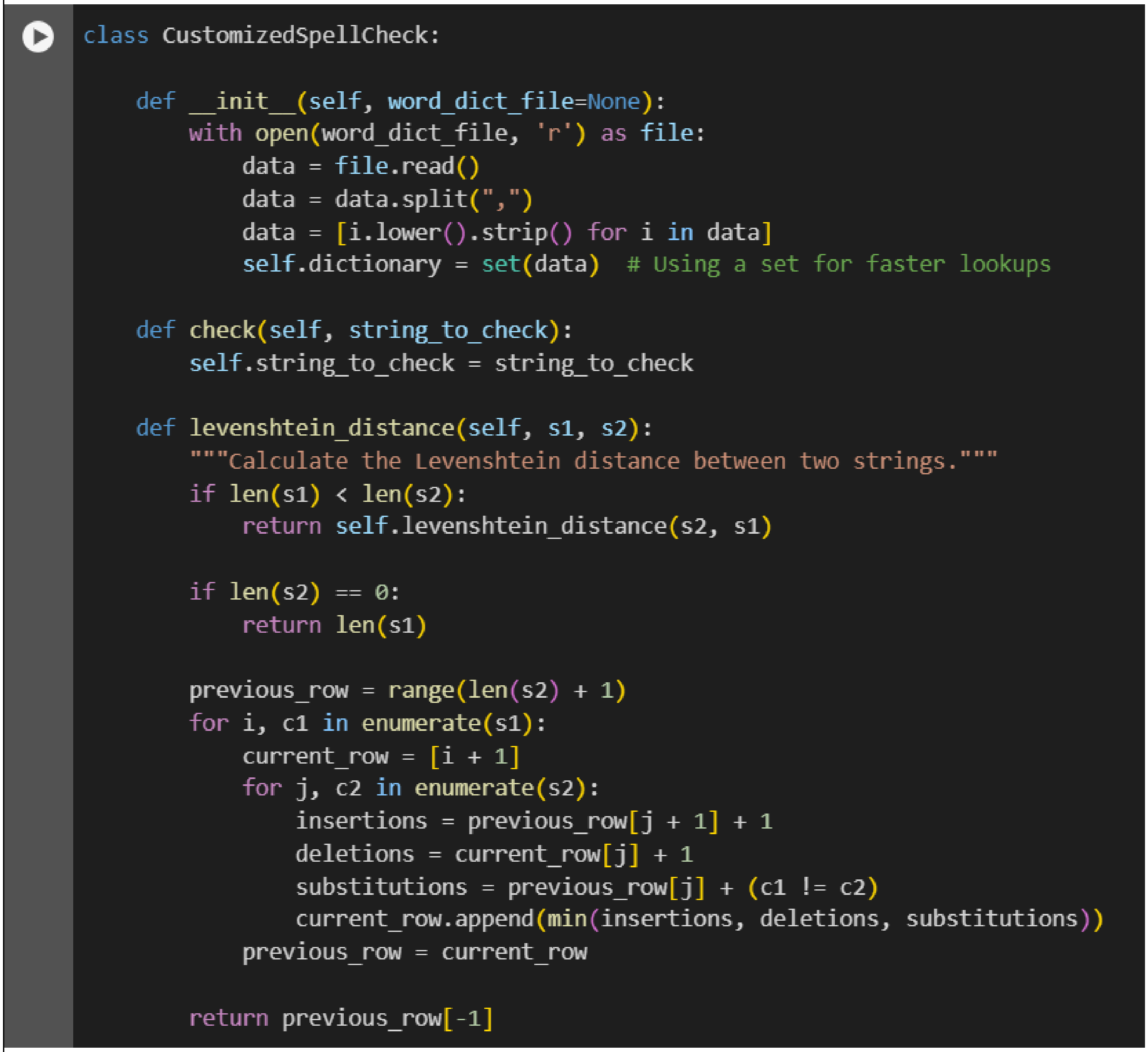
```
['wide']
wide erea
```

2. Design your own heuristic to list the word suggestions for the spelling mistake. Write your heuristic in the output file.

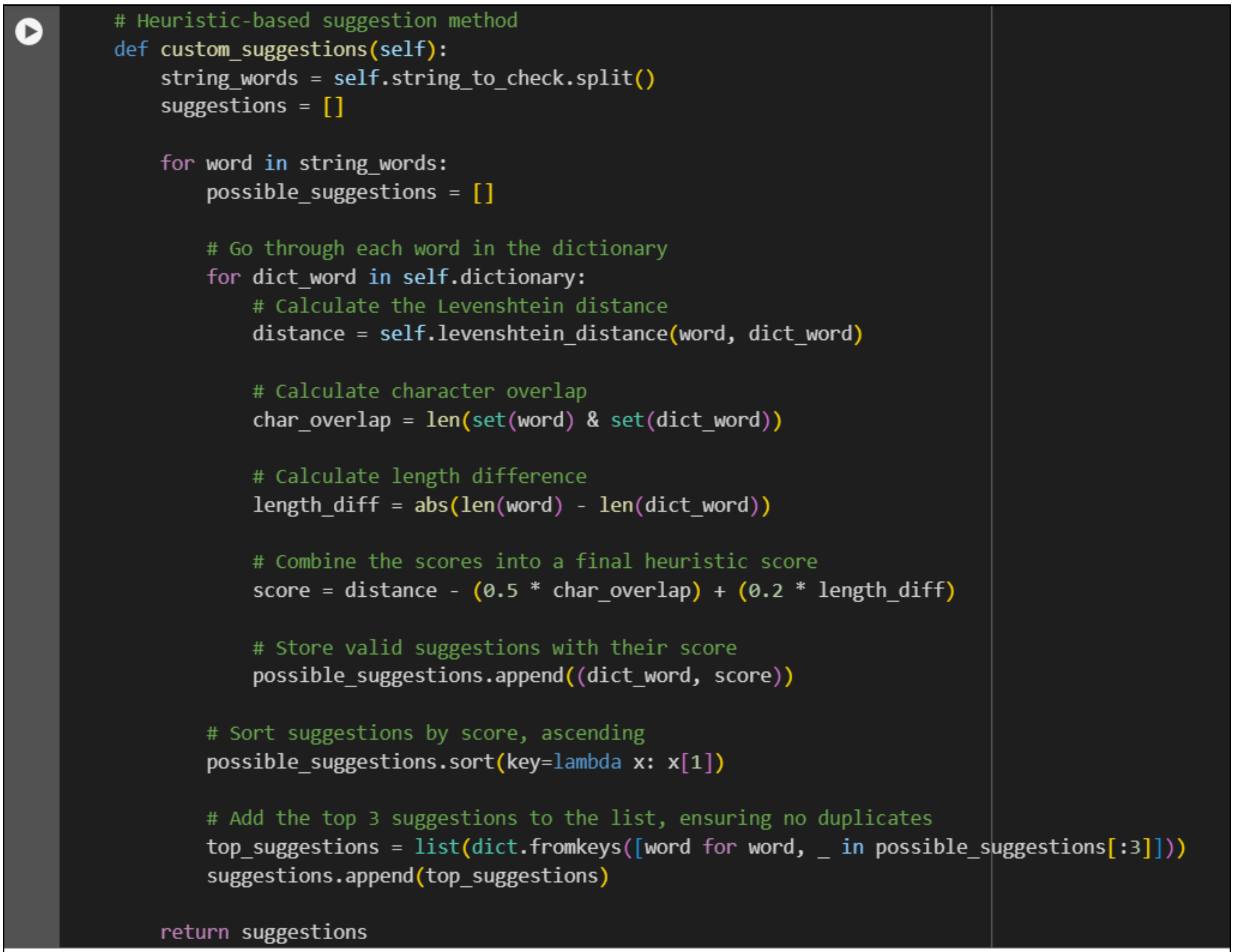
Heuristic for Customized Spell Checker

1. Initialize an empty set for the dictionary of words.
2. Open the file with the word list:
    - Read and split the contents by commas.
    - Clean each word (convert to lowercase, trim whitespace).
    - Add words to the dictionary set.
3. Store the input string to be checked.
4. Calculate the edit distance between two words:
    - If the first word is shorter, swap them.
    - If the second word is empty, return the length of the first word.
    - Create a range for the second word's length + 1.
    - For each character in the first word:
        - Initialize a list for the current row.
        - For each character in the second word:
            - Compute insertions, deletions, substitutions.
            - Store the minimum value in the current row.
        - Update the previous row.
5. Split the input string into individual words.
6. Initialize a list for suggestions.
7. For each word in the input:
    - Initialize a list for possible suggestions.
    - For each word in the dictionary:
        - Calculate the edit distance.
        - Count common characters.
        - Compute the length difference.
        - Calculate a score: edits - (0.5 * common) + (0.2 * length difference).
        - Add the word and score to possible suggestions.
8. Sort possible suggestions by score.
9. Select the top three suggestions, ensuring no duplicates.
10. Add selected suggestions to the main list.
11. Return the list of suggestions.

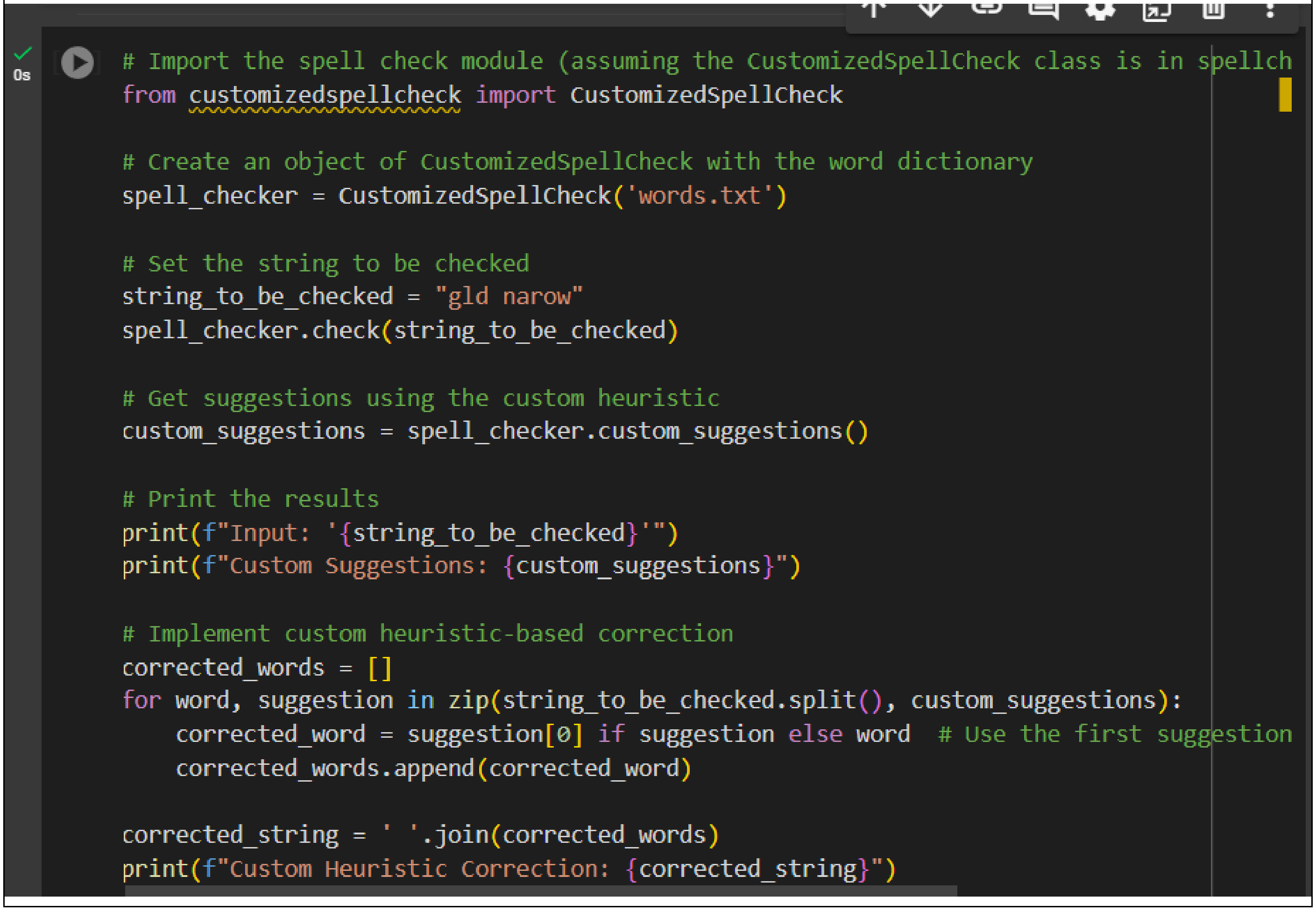
3. Implement and replace the suggestions function of the given application with your own heuristic in question 02.



```
class CustomizedSpellCheck:

    def __init__(self, word_dict_file=None):
        with open(word_dict_file, 'r') as file:
            data = file.read()
            data = data.split(",")
            data = [i.lower().strip() for i in data]
            self.dictionary = set(data)  # Using a set for faster lookups

    def check(self, string_to_check):
        self.string_to_check = string_to_check

    def levenshtein_distance(self, s1, s2):
        """Calculate the Levenshtein distance between two strings."""
        if len(s1) < len(s2):
            return self.levenshtein_distance(s2, s1)

        if len(s2) == 0:
            return len(s1)

        previous_row = range(len(s2) + 1)
        for i, c1 in enumerate(s1):
            current_row = [i + 1]
            for j, c2 in enumerate(s2):
                insertions = previous_row[j + 1] + 1
                deletions = current_row[j] + 1
                substitutions = previous_row[j] + (c1 != c2)
                current_row.append(min(insertions, deletions, substitutions))
            previous_row = current_row

        return previous_row[-1]
```

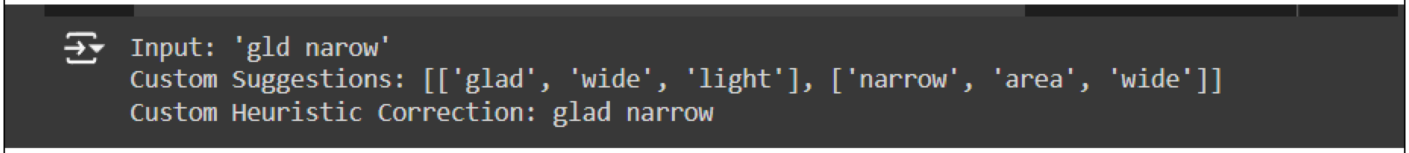


```python
    # Heuristic-based suggestion method
    def custom_suggestions(self):
        string_words = self.string_to_check.split()
        suggestions = []

        for word in string_words:
            possible_suggestions = []

            # Go through each word in the dictionary
            for dict_word in self.dictionary:
                # Calculate the Levenshtein distance
                distance = self.levenshtein_distance(word, dict_word)

                # Calculate character overlap
                char_overlap = len(set(word) & set(dict_word))

                # Calculate length difference
                length_diff = abs(len(word) - len(dict_word))

                # Combine the scores into a final heuristic score
                score = distance - (0.5 * char_overlap) + (0.2 * length_diff)

                # Store valid suggestions with their score
                possible_suggestions.append((dict_word, score))

            # Sort suggestions by score, ascending
            possible_suggestions.sort(key=lambda x: x[1])

            # Add the top 3 suggestions to the list, ensuring no duplicates
            top_suggestions = list(dict.fromkeys([word for word, _ in possible_suggestions[:3]]))
            suggestions.append(top_suggestions)

        return suggestions
```



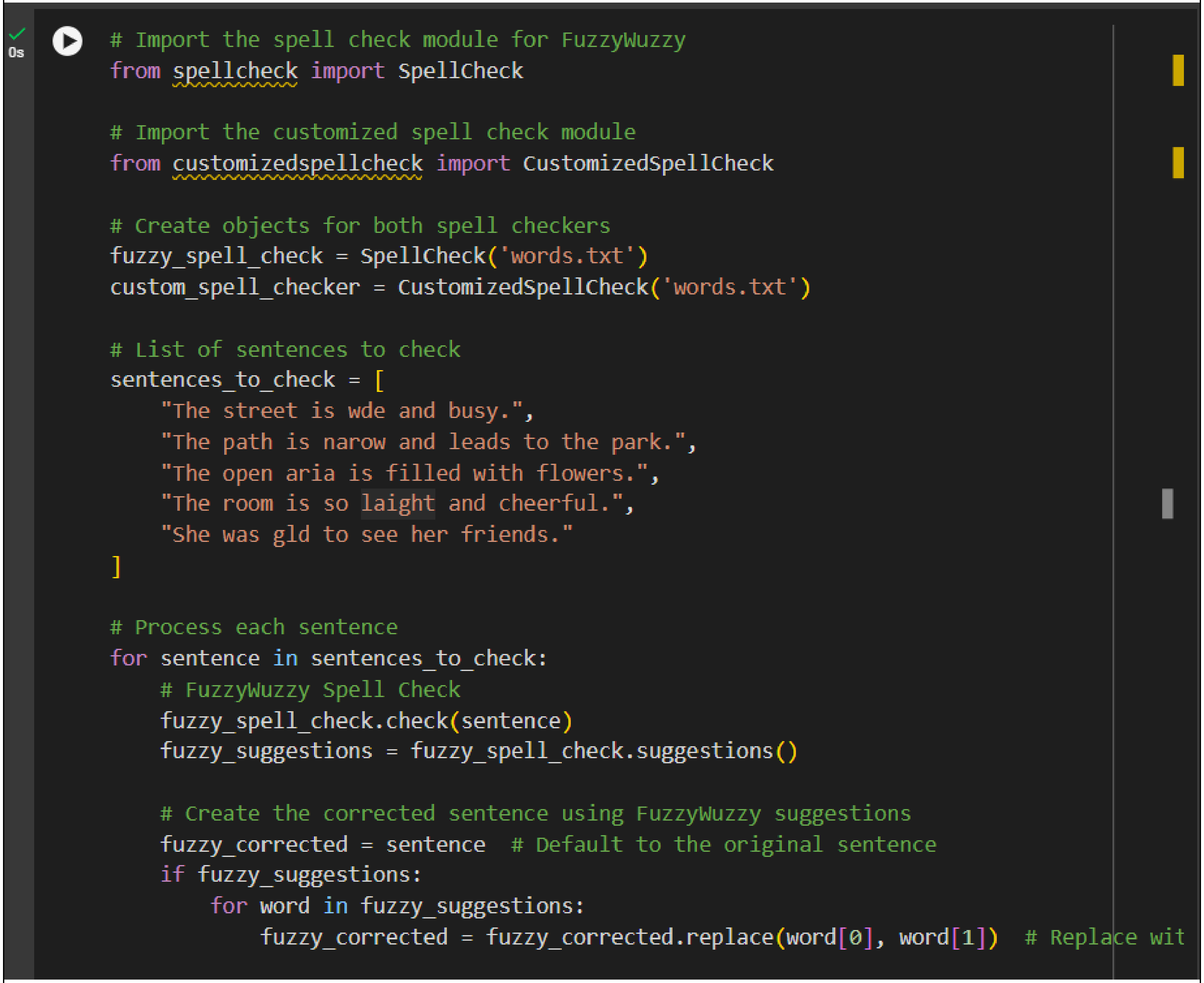
```python
# Import the spell check module (assuming the CustomizedSpellCheck class is in spellch
from customizedspellcheck import CustomizedSpellCheck

# Create an object of CustomizedSpellCheck with the word dictionary
spell_checker = CustomizedSpellCheck('words.txt')

# Set the string to be checked
string_to_be_checked = "gld narow"
spell_checker.check(string_to_be_checked)

# Get suggestions using the custom heuristic
custom_suggestions = spell_checker.custom_suggestions()

# Print the results
print(f"Input: '{string_to_be_checked}'")
print(f"Custom Suggestions: {custom_suggestions}")

# Implement custom heuristic-based correction
corrected_words = []
for word, suggestion in zip(string_to_be_checked.split(), custom_suggestions):
    corrected_word = suggestion[0] if suggestion else word  # Use the first suggestion

    corrected_words.append(corrected_word)

corrected_string = ' '.join(corrected_words)
print(f"Custom Heuristic Correction: {corrected_string}")
```

Output :

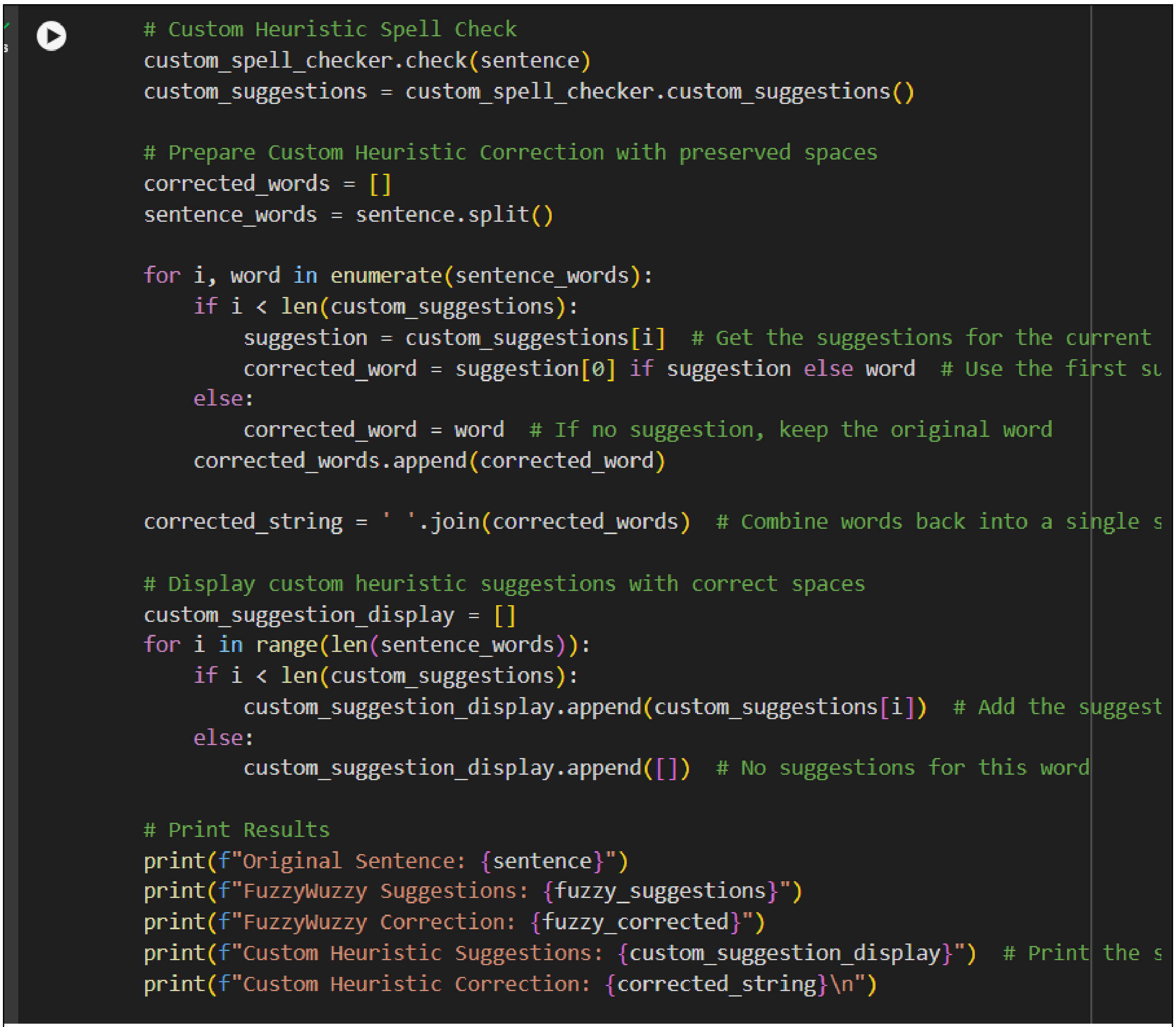


```
Input: 'gld narow'
Custom Suggestions: [['glad', 'wide', 'light'], ['narrow', 'area', 'wide']]
Custom Heuristic Correction: glad narrow
```

4. Compare five different example sentences for your own heuristic with the fuzzywuzzy matching of the given application. Place the screenshots in the output file.

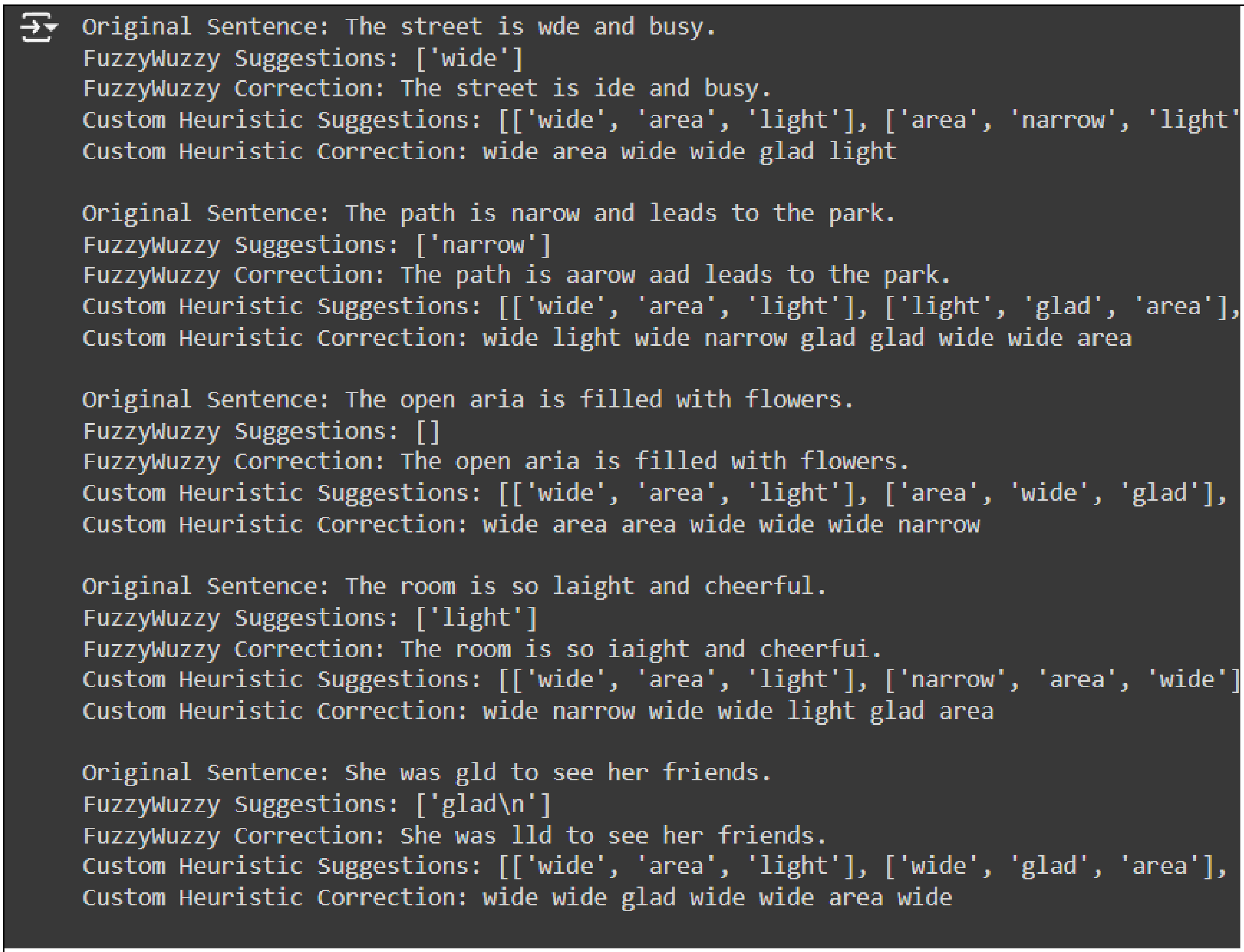


```
# Import the spell check module for FuzzyWuzzy
from spellcheck import SpellCheck

# Import the customized spell check module
from customizedspellcheck import CustomizedSpellCheck

# Create objects for both spell checkers
fuzzy_spell_check = SpellCheck('words.txt')
custom_spell_checker = CustomizedSpellCheck('words.txt')

# List of sentences to check
sentences_to_check = [
    "The street is wde and busy.",
    "The path is narow and leads to the park.",
    "The open aria is filled with flowers.",
    "The room is so laight and cheerful.",
    "She was gld to see her friends."
]

# Process each sentence
for sentence in sentences_to_check:
    # FuzzyWuzzy Spell Check
    fuzzy_spell_check.check(sentence)
    fuzzy_suggestions = fuzzy_spell_check.suggestions()

    # Create the corrected sentence using FuzzyWuzzy suggestions
    fuzzy_corrected = sentence  # Default to the original sentence
    if fuzzy_suggestions:
        for word in fuzzy_suggestions:
            fuzzy_corrected = fuzzy_corrected.replace(word[0], word[1])  # Replace wit
```



```python
# Custom Heuristic Spell Check
custom_spell_checker.check(sentence)
custom_suggestions = custom_spell_checker.custom_suggestions()

# Prepare Custom Heuristic Correction with preserved spaces
corrected_words = []
sentence_words = sentence.split()

for i, word in enumerate(sentence_words):
    if i < len(custom_suggestions):
        suggestion = custom_suggestions[i]  # Get the suggestions for the current
        corrected_word = suggestion[0] if suggestion else word  # Use the first su
    else:
        corrected_word = word  # If no suggestion, keep the original word
    corrected_words.append(corrected_word)

corrected_string = ' '.join(corrected_words)  # Combine words back into a single s

# Display custom heuristic suggestions with correct spaces
custom_suggestion_display = []
for i in range(len(sentence_words)):
    if i < len(custom_suggestions):
        custom_suggestion_display.append(custom_suggestions[i])  # Add the suggest
    else:
        custom_suggestion_display.append([])  # No suggestions for this word

# Print Results
print(f"Original Sentence: {sentence}")
print(f"FuzzyWuzzy Suggestions: {fuzzy_suggestions}")
print(f"FuzzyWuzzy Correction: {fuzzy_corrected}")
print(f"Custom Heuristic Suggestions: {custom_suggestion_display}")  # Print the s
print(f"Custom Heuristic Correction: {corrected_string}\n")
```

Output :



```
Original Sentence: The street is wde and busy.
FuzzyWuzzy Suggestions: ['wide']
FuzzyWuzzy Correction: The street is ide and busy.
Custom Heuristic Suggestions: [['wide', 'area', 'light'], ['area', 'narrow', 'light'
Custom Heuristic Correction: wide area wide wide glad light

Original Sentence: The path is narow and leads to the park.
FuzzyWuzzy Suggestions: ['narrow']
FuzzyWuzzy Correction: The path is aarow aad leads to the park.
Custom Heuristic Suggestions: [['wide', 'area', 'light'], ['light', 'glad', 'area'],
Custom Heuristic Correction: wide light wide narrow glad glad wide wide area

Original Sentence: The open aria is filled with flowers.
FuzzyWuzzy Suggestions: []
FuzzyWuzzy Correction: The open aria is filled with flowers.
Custom Heuristic Suggestions: [['wide', 'area', 'light'], ['area', 'wide', 'glad'],
Custom Heuristic Correction: wide area area wide wide wide narrow

Original Sentence: The room is so laight and cheerful.
FuzzyWuzzy Suggestions: ['light']
FuzzyWuzzy Correction: The room is so iaight and cheerfui.
Custom Heuristic Suggestions: [['wide', 'area', 'light'], ['narrow', 'area', 'wide']
Custom Heuristic Correction: wide narrow wide wide light glad area

Original Sentence: She was gld to see her friends.
FuzzyWuzzy Suggestions: ['glad\n']
FuzzyWuzzy Correction: She was lld to see her friends.
Custom Heuristic Suggestions: [['wide', 'area', 'light'], ['wide', 'glad', 'area'],
Custom Heuristic Correction: wide wide glad wide wide area wide
```