Search Engine Report

학과(부)	인공지능학부 소프트웨어전공
학번	213954
이름	이영채

1. Introduction

1-1. Project Purpose and Background

This project was undertaken to apply the knowledge acquired in the Python Programming and Practical Course, including the functions of variables, data types, syntax, files, and functions. Through this project, we aim to assess our overall understanding of the learning materials and individually complete code to review and reinforce our knowledge.

1-2. Goals

Based on what we have learned so far, we aim to create a simple search engine with Python to meet the requirements presented in the project task and operate it correctly.

2. Requirements

2-1. User Requirements

When a user enters an English query, we implement a system that searches 10 sentences similar to the input English query and outputs the sentence body, the index of the sentence, and the similarity ranking of the sentence.

2-2. Functional Requirements

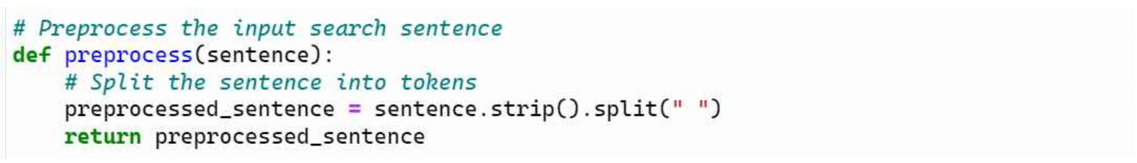
1. Preprocess the sentences in the search target and store them in the list.
2. It receives English queries from users and preprocesses them.
3. Calculate the similarity between the input English query and the sentences in the search target. Here, the similarity is calculated by the number of the same words.
4. Rank the sentences based on the calculated similarity.
5. Among the ranked sentences, the top 10 sentences are output to the user along with the index and similarity score.

3. Design and Implementation

3-1. Implementation by Function

1. preprocess(sentence)
- input: a query or each sentence of search targets
- return: preprocessed query or sentence (=a set of tokens)
- Explanation

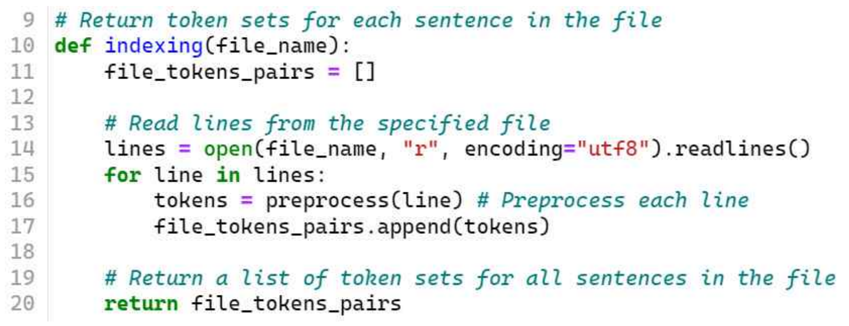
The input sentences to be preprocessed provided in the form of strings are divided based on spaces.



```
# Preprocess the input search sentence
def preprocess(sentence):
    # Split the sentence into tokens
    preprocessed_sentence = sentence.strip().split(" ")
    return preprocessed_sentence
```

2. indexing(file_name)
- input: a file name with its path for search targets
- return: a set of tokens for each sentence in the file
- Explanation

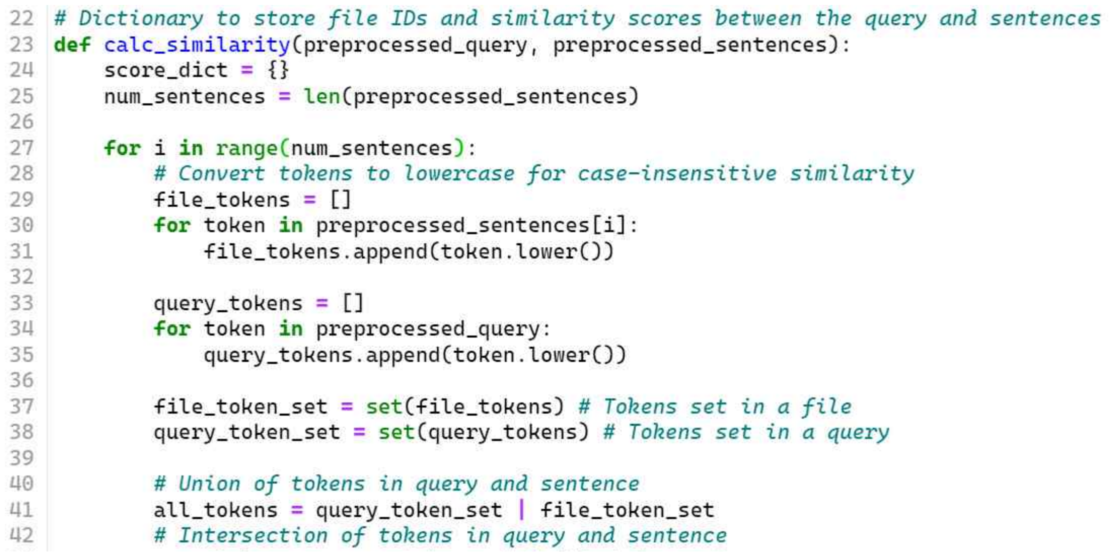
  Recall the file, read all the lines of the file, call the preprocessed function, perform the preprocessing, and add the return value to the file_tokens_pairs list.



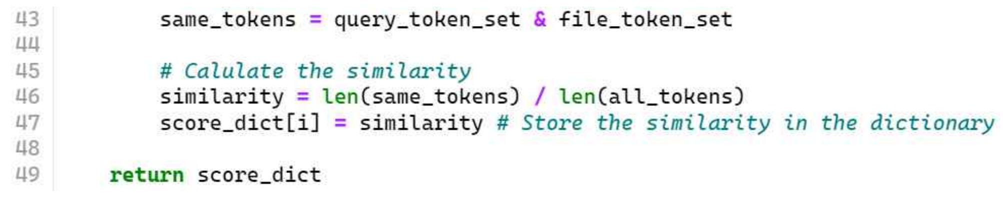
```python
# Return token sets for each sentence in the file
def indexing(file_name):
    file_tokens_pairs = []

    # Read lines from the specified file
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line) # Preprocess each line
        file_tokens_pairs.append(tokens)

    # Return a list of token sets for all sentences in the file
    return file_tokens_pairs
```

3. calc_similarity(preprocessed_query, preprocessed_sentences)
- input: preprocessed query, preprocessed sentences (=search target)
- return: a dictionary containing file_id and corresponding similarity score
- Explanation

  Calculate the similarity between the pre-processed query and the list of pre-processed sentences. For case-insensitive comparison, all file tokens and sentence tokens are converted to lowercase. Make a list of converted tokens into a set so that each token is counted only once. The similarity is calculated based on the set of tokens, and the similarity between the sentence and the query is stored in the dictionary and returned.



```python
# Dictionary to store file IDs and similarity scores between the query and sentences
def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}
    num_sentences = len(preprocessed_sentences)

    for i in range(num_sentences):
        # Convert tokens to lowercase for case-insensitive similarity
        file_tokens = []
        for token in preprocessed_sentences[i]:
            file_tokens.append(token.lower())

        query_tokens = []
        for token in preprocessed_query:
            query_tokens.append(token.lower())

        file_token_set = set(file_tokens) # Tokens set in a file
        query_token_set = set(query_tokens) # Tokens set in a query

        # Union of tokens in query and sentence
        all_tokens = query_token_set | file_token_set
        # Intersection of tokens in query and sentence
```

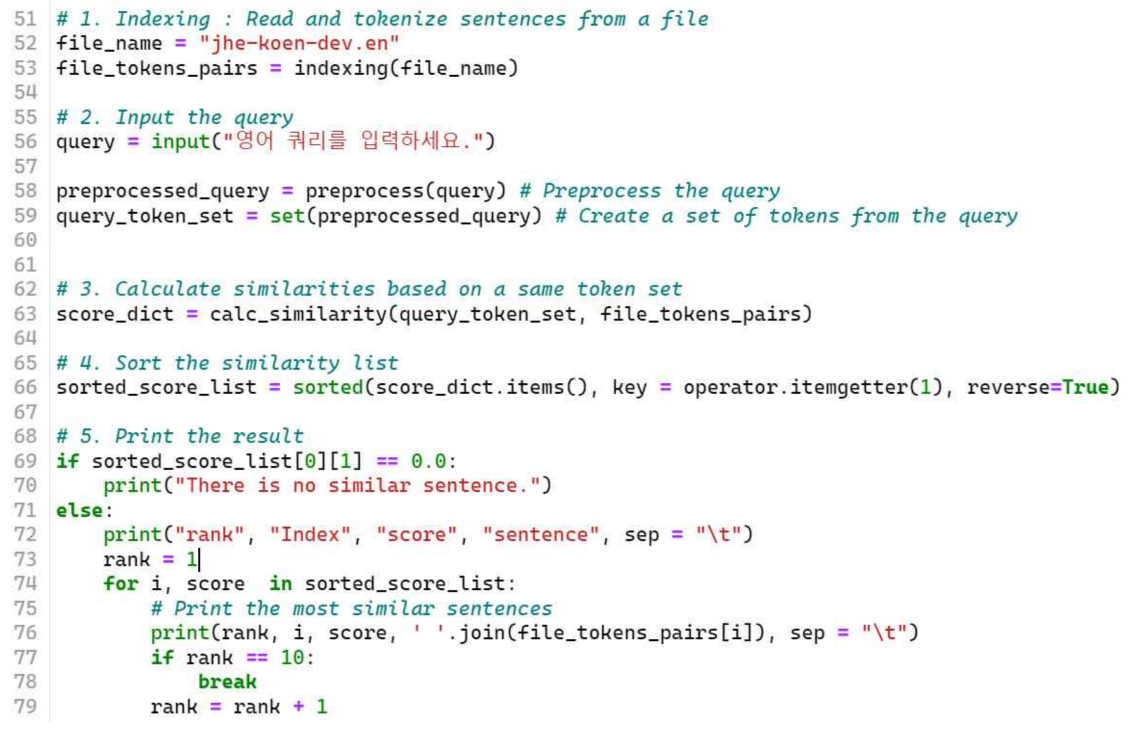


```python
        same_tokens = query_token_set & file_token_set

        # Calulate the similarity
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity # Store the similarity in the dictionary

    return score_dict
```

4. Main source code

- Explanation

  Read and tokenize sentences from a file. When a user enters an English query, the input query is preprocessed and stored in a set of tokens. The similarity between sentences is calculated based on the created set of tokens, the similarity list is sorted, and the result is output to the user.

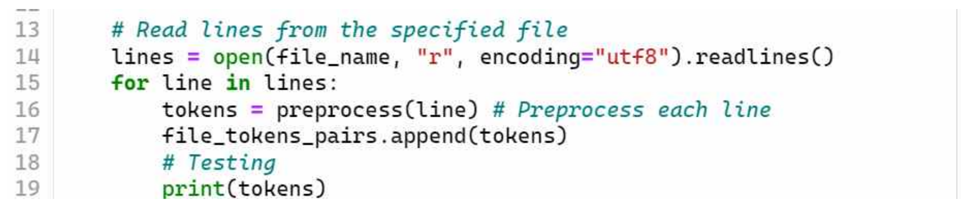


```python
# 1. Indexing : Read and tokenize sentences from a file
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.")

preprocessed_query = preprocess(query) # Preprocess the query
query_token_set = set(preprocessed_query) # Create a set of tokens from the query


# 3. Calculate similarities based on a same token set
score_dict = calc_similarity(query_token_set, file_tokens_pairs)

# 4. Sort the similarity list
sorted_score_list = sorted(score_dict.items(), key = operator.itemgetter(1), reverse=True)

# 5. Print the result
if sorted_score_list[0][1] == 0.0:
    print("There is no similar sentence.")
else:
    print("rank", "Index", "score", "sentence", sep = "\t")
    rank = 1
    for i, score  in sorted_score_list:
        # Print the most similar sentences
        print(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "\t")
        if rank == 10:
            break
        rank = rank + 1
```

4. Testing

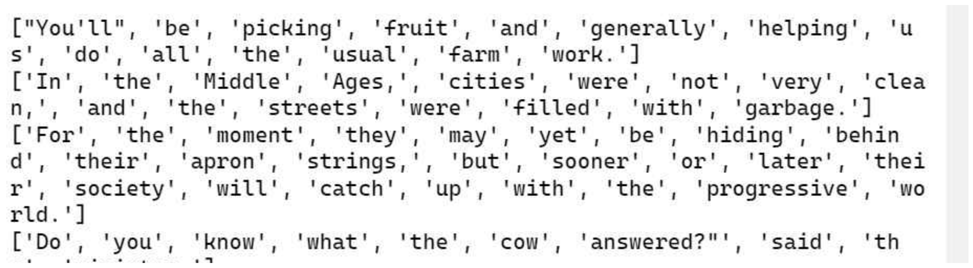
4-1. Test Results for Each Functionality

① Save sentences in the search target to the list after preprocessing



```python
    # Read lines from the specified file
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line) # Preprocess each line
        file_tokens_pairs.append(tokens)
        # Testing
        print(tokens)
```

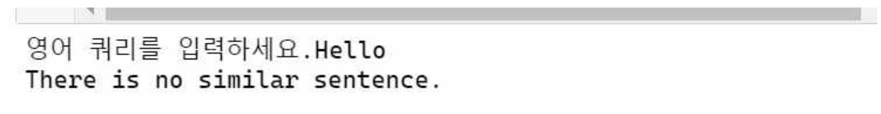
- Test result



```
["You'll", 'be', 'picking', 'fruit', 'and', 'generally', 'helping', 'u
s', 'do', 'all', 'the', 'usual', 'farm', 'work.']
['In', 'the', 'Middle', 'Ages,', 'cities', 'were', 'not', 'very', 'clea
n,', 'and', 'the', 'streets', 'were', 'filled', 'with', 'garbage.']
['For', 'the', 'moment', 'they', 'may', 'yet', 'be', 'hiding', 'behin
d', 'their', 'apron', 'strings,', 'but', 'sooner', 'or', 'later', 'thei
r', 'society', 'will', 'catch', 'up', 'with', 'the', 'progressive', 'wo
rld.']
['Do', 'you', 'know', 'what', 'the', 'cow', 'answered?"', 'said', 'th
```

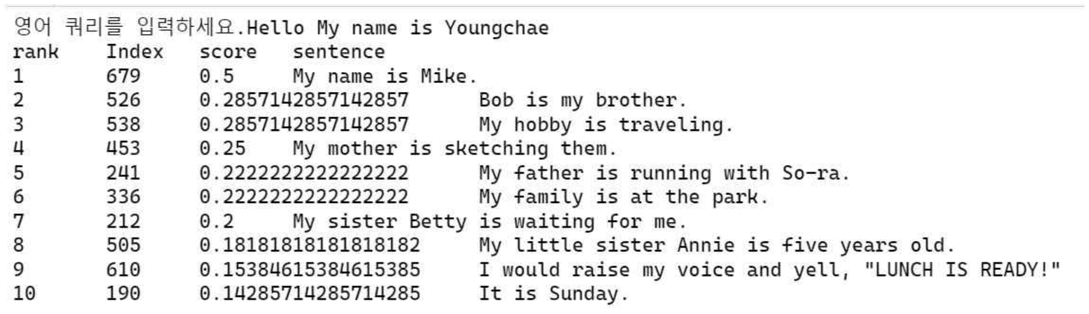
4-2. Final Test Screenshot

① If there is no similar sentence



```
영어 쿼리를 입력하세요.Hello
There is no similar sentence.
```

② If there is a similar sentence



```
영어 쿼리를 입력하세요.Hello My name is Youngchae
rank    Index   score   sentence
1       679     0.5     My name is Mike.
2       526     0.2857142857142857      Bob is my brother.
3       538     0.2857142857142857      My hobby is traveling.
4       453     0.25    My mother is sketching them.
5       241     0.2222222222222222      My father is running with So-ra.
6       336     0.2222222222222222      My family is at the park.
7       212     0.2     My sister Betty is waiting for me.
8       505     0.18181818181818182     My little sister Annie is five years old.
9       610     0.15384615384615385     I would raise my voice and yell, "LUNCH IS READY!"
10      190     0.14285714285714285     It is Sunday.
```

5. Results and Conclusion

5-1. Projects Results

As a result of the project, a simple search engine could be implemented to meet the requirements. When you create an English query, you calculate the similarity of the sentence, case-insensitive, and return the result to the user.

5-2. Conclusion

While conducting the project practice, I was able to find out how much I understood what I had learned so far. There were some difficulties in implementing the function of the function based on the skeleton code, but it was able to solve it well. It was useful to review and apply the contents.