1. Introduction

1. Project Purpose and Background: In this project, we will create a simple search engine to review the concepts we have learned so far in Python programming and practical classes.

2. Goal: I need to create a search engine that outputs 10 sentences similar to the user's input string (query).

2. Requirements

1. User requirements: When a user enters a query, it should search for sentences similar to the query and then output 5 sentences that are similar to the query entered by the user.

2. Functional Requirements (The previously listed item)

1) Preprocess sentences within the search target and store them in a list.

2) Receive an input English string (query) from the user and preprocess it.

3) Calculate the similarity between the query and sentences within the search target (Similarity is based on the count of the same “word.")

4) Rank the sentences based on similarity.

5) Output the top 10 ranked sentences to the user from the ranked sentences.

3. Design and Implementation

```
# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence # preprocessed_sentence 반환 - split

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화
```

1) Preprocess sentences within the search target and store them in a list.

preprocess(sentence) function

Input: sentence: It takes a string parameter

Result: preprocessed_sentence: Return the list containing tokenized results, preprocessed_sentence
Description: The input string is split based on spaces to separate it into words (tokens), and the tokenized result is returned as a list.

indexing(file_name) function
Input: file_name: The string represents a file's path
Result: file_tokens_pairs:
It returns the list containing tokenized results, file_tokens_pairs.
Description: It reads the given file, tokenizes each line using the preprocess function, and appends the tokenized results to the file_tokens_pairs list. After processing all the lines in the file, it returns the list containing tokenized results, file_tokens_pairs.

```
# 2. Input the query
query = input("영어 쿼리를 입력하세요.")
preprocessed_query = preprocess(query) # query 전처리 - split
query_token_set = set(preprocessed_query)

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence # preprocessed_sentence 반환 - split
```

2) Receive an input English string (query) from the user and preprocess it.

Input: query: The string received from the user represents an English query.

Result: preprocessed_query: A list containing the results obtained by tokenizing the entered English query.
query_token_set: A set that stores tokenized results without duplicates. This represents the set of unique words (tokens) included in the English query.

Description: Users can input an English query into the program. The entered query string is divided into words (tokens) based on whitespace using the preprocess function. The tokenized results are stored in the "preprocessed_query" variable. To store the tokenized results without duplicates, they are converted into a set (set) and this resulting set is

stored in the "query_token_set" variable.

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

```
def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}
    for i in range(len(preprocessed_sentences)):
        # 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i]
        query_str = ' '.join(preprocessed_query).lower()
        sentence_str = ' '.join(sentence).lower()
        preprocessed_query = set(preprocess(query_str))
        preprocessed_sentence = preprocess(sentence_str)

        file_token_set = set(preprocessed_sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity

    return score_dict
```

3) Calculate the similarity between the query and sentences within the search target

Input: preprocessed_query: A list representing tokenized English queries. This is the result of tokenizing the query input by the user in the previous step.
preprocessed_sentences: A list containing tokenized sentences. This list is obtained using the indexing function and includes tokenized versions of each line in the file.

Result: score_dict: A dictionary with sentence indices as keys and the similarity between the respective sentence and query as values. This dictionary stores the similarity between each sentence and the query, with similarity represented as a value between 0 and 1. Higher similarity values indicate that two sentences are more similar.

Description: calc_similarity(preprocessed_query, preprocessed_sentences): It calls a function to calculate similarity using the provided tokenized query and a list of tokenized sentences. Through a loop, it calculates the similarity for each sentence and stores the results in the score_dict dictionary. score_dict is a dictionary with sentence indices as keys and the

similarity between each sentence and the query as values. When calculating similarity, it doesn't distinguish between uppercase and lowercase, and it converts the tokens of the query and sentence into sets. It then calculates the union and intersection of the two sets. Similarity is defined as the size of the intersection divided by the size of the union. The calculated similarity value is stored in score_dict with the sentence's index as the key.

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

4) Rank the sentences based on similarity.

Input: score_dict: A dictionary containing the previously calculated similarity between sentences and queries. Each entry in this dictionary has the sentence index as the key and the corresponding similarity between the sentence and the query as the value.

Result: sorted_score_list: A list of (sentence index, similarity score) pairs sorted in descending order based on similarity scores. This list is sorted from sentences with the highest similarity to the lowest similarity.

Description: sorted_score_list=sorted(score_dict.items(), key=operator.itemgetter(1), reverse=True): This involves sorting the score_dict dictionary.
score_dict.items(): It converts each entry in the score_dict dictionary into a list of tuples (key, value).
key=operator.itemgetter(1): When sorting this list of tuples, specify sorting based on the second element of each entry (similarity score). Set reverse=True to perform sorting in descending order.

```
# 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(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "\t")
        if rank == 10:
            break
        rank = rank + 1
```

5) Output the top 10 ranked sentences to the user from the ranked sentences

Input: sorted_score_list: A list containing the sorted sentences with their similarity scores from the previous step. Each entry consists of a sentence index and a similarity score pair.

Result: This code outputs the top 10 sentences sorted by similarity.
"rank": A number indicating the rank of a sentence.
"Index": The index of a sentence.
"score": The similarity score between a sentence and a query.
"sentence": The original text of the respective sentence, not in tokenized form, but in its original form.

Description: if sorted_score_list[0][1] == 0.0::
If the sentence with the highest similarity score has a similarity score of 0.0, a message is displayed indicating that there are no similar sentences.
In that case, a list of similar sentences is displayed.
In the output, "rank," "Index," "score," and "sentence" represent header columns, separated by tab characters (\t). The rank variable is a variable indicating the position, initially set to 1 and incremented through a loop. "i" and "score" represent the current sentence's index and similarity score.
"file_tokens_pairs[i]" represents the tokenized version of the respective sentence.
"If rank == 10:" signifies that the top 10 similar sentences have been output, and the loop should be terminated.
"rank = rank + 1" is used to increase the rank variable to calculate the next position.

4. Testing

1. Test Results for Each Functionality: (Screenshots by Requirement)

1) Preprocess sentences within the search target and store them in a list.

In [4]:

```
import operator

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화

# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)
print(file_tokens_pairs)
```

```
[["You'll", 'be', 'picking', 'fruit', 'and', 'generally', 'helping', 'us', 'do', 'all', 'the', 'usual', 'fa
rm', 'work.'], ['In', 'the', 'Middle', 'Ages,', 'cities', 'were', 'not', 'very', 'clean,', 'and', 'the', 's
treets', 'were', 'filled', 'with', 'garbage.'], ['For', 'the', 'moment', 'they', 'may', 'yet', 'be', 'hidin
g', 'behind', 'their', 'apron', 'strings,', 'but', 'sooner', 'or', 'later', 'their', 'society', 'will', 'ca
tch', 'up', 'with', 'the', 'progressive', 'world.'], ['Do', 'you', 'know', 'what', 'the', 'cow', 'answere
d?"', 'said', 'the', 'minister.'], ['Poland', 'and', 'Italy', 'may', 'seem', 'like', 'very', 'different',
'countries.'], ['Mr.', 'Smith', 'and', 'I', 'stayed', 'the', 'whole', 'day', 'in', 'Oxford.'], ['The', 'sig
ht', 'of', 'a', 'red', 'traffic', 'signal', 'gave', 'him', 'an', 'idea.'], ['So', 'they', 'used', 'pumpkin
s', 'instead.'], ['2.', 'a', 'particular', 'occasion', 'of', 'state', 'of', 'affairs:', 'They', 'might', 'n
ot', 'offer', 'me', 'much', 'money.'], ["I'm", 'especially', 'interested', 'in', 'learning,', 'horse-ridin
g', 'skills,', 'so', 'I', 'hope', "you'll", 'include', 'information', 'about', 'this.'], ['Instead,', 'th
e', 'devil', 'gave', 'him', 'a', 'single', 'candle', 'to', 'light', 'his', 'way', 'through', 'the', 'darkne
ss.'], ['It', 'shines', 'over', 'the', 'sea.'], ['He,', 'too,', 'was', 'arrested,', 'and', 'a', 'bomb', 'wa
s', 'thrown', 'at', 'his', 'house.'], ['It', 'seems', 'that', 'the', 'high', 'temperature', 'and', 'pressur
e', 'on', 'the', 'star', 'made', 'its', 'carbon', 'surface', 'turn', 'to', 'diamond.'], ['"The', 'pig', 'wa
s', 'unpopular', 'while', 'the', 'cow', 'was', 'loved', 'by', 'everyone.'], ['Books', 'give', 'a', 'lot',
'of', 'things', 'to', 'us.'], ['Jimmy', 'and', 'Timmy', 'were', 'identical', 'twins.'], ['It', 'is', 'a',
'chemical', 'that', 'cause', 'cancer.'], ['Ziege', 'from', 'Germany', 'and', 'Brazilian', 'superstars', 'Ro
naldo', 'and', 'Roberto', 'Carlos', 'belonged', 'to', 'the', 'bald', 'club.'], ['Now', 'the', 'Taliban', 'a
```

2) Receive an input English string (query) from the user and preprocess it.

In [6]:

```
import operator

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화

# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.")
preprocessed_query = preprocess(query) # query 전처리 - split
query_token_set = set(preprocessed_query)
print(query_token_set)
```

```
영어 쿼리를 입력하세요.i love you
{'love', 'you', 'i'}
```

3) Calculate the similarity between the query and sentences within the search target

In [7]:

```python
import operator

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화

def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}
    for i in range(len(preprocessed_sentences)):
        # 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i]
        query_str = ' '.join(preprocessed_query).lower()
        sentence_str = ' '.join(sentence).lower()
        preprocessed_query = set(preprocess(query_str))
        preprocessed_sentence = preprocess(sentence_str)

        file_token_set = set(preprocessed_sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity

    return score_dict

# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.")
preprocessed_query = preprocess(query) # query 전처리 - split
query_token_set = set(preprocessed_query)

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

```
영어 쿼리를 입력하세요.i love you
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.09090909090909091, 4: 0.0, 5: 0.08333333333333333, 6: 0.0, 7: 0.0, 8: 0.0,
9: 0.058823529411764705, 10: 0.0, 11: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.0, 18: 0.0,
19: 0.0, 20: 0.0, 21: 0.0, 22: 0.0, 23: 0.06666666666666667, 24: 0.0, 25: 0.0, 26: 0.1111111111111111, 27:
0.0, 28: 0.0, 29: 0.0, 30: 0.0, 31: 0.0, 32: 0.10526315789473684, 33: 0.08333333333333333, 34: 0.0, 35: 0.
0, 36: 0.0, 37: 0.0, 38: 0.0, 39: 0.0, 40: 0.0, 41: 0.0, 42: 0.07142857142857142, 43: 0.09090909090909091,
44: 0.0, 45: 0.0, 46: 0.0, 47: 0.0, 48: 0.0, 49: 0.0, 50: 0.0, 51: 0.0, 52: 0.0, 53: 0.0, 54: 0.0, 55: 0.
0, 56: 0.0, 57: 0.0, 58: 0.0, 59: 0.0, 60: 0.0, 61: 0.0, 62: 0.1111111111111111, 63: 0.058823529411764705,
64: 0.0, 65: 0.0, 66: 0.0, 67: 0.0, 68: 0.0, 69: 0.0, 70: 0.0, 71: 0.0, 72: 0.0, 73: 0.0, 74: 0.0, 75: 0.
0, 76: 0.0, 77: 0.0, 78: 0.0, 79: 0.0, 80: 0.0, 81: 0.0, 82: 0.0, 83: 0.0, 84: 0.0, 85: 0.0, 86: 0.0, 87:
0.0, 88: 0.0, 89: 0.0, 90: 0.0, 91: 0.0, 92: 0.0, 93: 0.0, 94: 0.0, 95: 0.0, 96: 0.0, 97: 0.0, 98: 0.0, 9
9: 0.0, 100: 0.0, 101: 0.0, 102: 0.0, 103: 0.0, 104: 0.06666666666666667, 105: 0.0, 106: 0.0, 107: 0.0, 10
8: 0.0, 109: 0.0, 110: 0.0, 111: 0.0, 112: 0.08333333333333333, 113: 0.05263157894736842, 114: 0.0, 115:
0.0, 116: 0.0, 117: 0.0, 118: 0.0, 119: 0.07692307692307693, 120: 0.0, 121: 0.0, 122: 0.0, 123: 0.0, 124:
0.0, 125: 0.0, 126: 0.0, 127: 0.0, 128: 0.0, 129: 0.0, 130: 0.0, 131: 0.0, 132: 0.0, 133: 0.0, 134: 0.0, 1
35: 0.0, 136: 0.0, 137: 0.0, 138: 0.0, 139: 0.0, 140: 0.07692307692307693, 141: 0.0, 142: 0.0, 143: 0.0, 1
44: 0.07692307692307693, 145: 0.0, 146: 0.0, 147: 0.08333333333333333, 148: 0.0, 149: 0.0, 150: 0.0, 151:
0.0, 152: 0.0, 153: 0.0, 154: 0.0, 155: 0.0, 156: 0.0, 157: 0.0, 158: 0.0, 159: 0.0, 160: 0.166666666666666
666, 161: 0.05555555555555555, 162: 0.0, 163: 0.058823529411764705, 164: 0.125, 165: 0.0, 166: 0.058823529
```

4) Rank the sentences based on similarity

In [8]:

```python
import operator

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화

def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}
    for i in range(len(preprocessed_sentences)):
        # 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i]
        query_str = ' '.join(preprocessed_query).lower()
        sentence_str = ' '.join(sentence).lower()
        preprocessed_query = set(preprocess(query_str))
        preprocessed_sentence = preprocess(sentence_str)

        file_token_set = set(preprocessed_sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity

    return score_dict

# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.")
preprocessed_query = preprocess(query) # query 전처리 - split
query_token_set = set(preprocessed_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)
print(sorted_score_list)
```

```
영어 쿼리를 입력하세요.i love you
[(439, 0.2222222222222222), (467, 0.2), (160, 0.16666666666666666), (251, 0.14285714285714285), (351, 0.14285714285714285), (405, 0.14285714285714285), (516, 0.14285714285714285), (582, 0.14285714285714285), (654, 0.14285714285714285), (164, 0.125), (388, 0.125), (500, 0.125), (541, 0.125), (556, 0.125), (617, 0.125), (672, 0.125), (697, 0.125), (704, 0.125), (718, 0.125), (26, 0.1111111111111111), (62, 0.1111111111111111), (182, 0.1111111111111111), (383, 0.1111111111111111), (427, 0.1111111111111111), (472, 0.1111111111111111), (509, 0.1111111111111111), (534, 0.1111111111111111), (620, 0.1111111111111111), (625, 0.1111111111111111), (627, 0.1111111111111111), (652, 0.1111111111111111), (32, 0.10526315789473684), (183, 0.1), (266, 0.1), (323, 0.1), (359, 0.1), (436, 0.1), (491, 0.1), (537, 0.1), (581, 0.1), (605, 0.1), (3, 0.09090909090909091), (43, 0.09090909090909091), (218, 0.09090909090909091), (441, 0.09090909090909091), (649, 0.09090909090909091), (677, 0.09090909090909091), (5, 0.08333333333333333), (33, 0.08333333333333333), (112, 0.08333333333333333), (147, 0.08333333333333333), (229, 0.08333333333333333), (271, 0.08333333333333333), (466, 0.08333333333333333), (493, 0.08333333333333333), (512, 0.08333333333333333), (542, 0.08333333333333333), (594, 0.08333333333333333), (610, 0.08333333333333333), (621, 0.08333333333333333), (623, 0.08333333333333333), (633, 0.08333333333333333), (641, 0.08333333333333333), (119, 0.07692307692307693), (140, 0.07692307692307693), (144, 0.07692307692307693), (232, 0.07692307692307693), (242, 0.07692307692307693), (257, 0.07692307692307693), (334, 0.07692307692307693), (468, 0.07692307692307693), (586, 0.07692307692307693), (601, 0.07692307692307693), (696, 0.07692307692307693), (700, 0.07692307692307693), (705, 0.07692307692307693), (42, 0.07142857142857142), (168, 0.07142857142857142), (185, 0.07142857142857142), (258, 0.07142
```

5) Output the top 10 ranked sentences to the user from the ranked sentences

```
# 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(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "\t")
        if rank == 10:
            break
        rank = rank + 1
```

```
영어 쿼리를 입력하세요.i love you
rank    Index   score   sentence
1       439     0.2222222222222222      I look forward to hearing from you soon.
2       467     0.2     You know you can't.
3       160     0.16666666666666666     I just wanted you to know how proud I am of Bobby.
4       251     0.14285714285714285     I am nine years old.
5       351     0.14285714285714285     "Do you have the time?"
6       405     0.14285714285714285     I felt sorry for it.
7       516     0.14285714285714285     I just got your flower.
8       582     0.14285714285714285     Thank you for helping me.
9       654     0.14285714285714285     You will have a toothache.
10      164     0.125   Let me give you another example.
```

2. Screenshots of the final test: (Screenshots of the complete program operation)

1) If there are no similar sentences

In [1]:

```python
import operator

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence # preprocessed_sentence 반환 - split

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화

def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}
    for i in range(len(preprocessed_sentences)):
        # 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i]
        query_str = ' '.join(preprocessed_query).lower()
        sentence_str = ' '.join(sentence).lower()
        preprocessed_query = set(preprocess(query_str))
        preprocessed_sentence = preprocess(sentence_str)

        file_token_set = set(preprocessed_sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity

    return score_dict

# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.")
preprocessed_query = preprocess(query) # query 전처리 - split
query_token_set = set(preprocessed_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(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "\t")
        if rank == 10:
            break
        rank = rank + 1
```

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

2) If there are similar sentences

In [1]:

```python
import operator

def preprocess(sentence):
    preprocessed_sentence = sentence.strip().split(" ")

    return preprocessed_sentence # preprocessed_sentence 반환 - split

def indexing(file_name):
    file_tokens_pairs = []
    lines = open(file_name, "r", encoding="utf8").readlines()
    for line in lines:
        tokens = preprocess(line)
        file_tokens_pairs.append(tokens)

    return file_tokens_pairs # file_tokens_pairs 반환 - 토큰화

def calc_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}
    for i in range(len(preprocessed_sentences)):
        # 대소문자 구분 없는 토큰 셋을 만들기 위한 코드
        sentence = preprocessed_sentences[i]
        query_str = ' '.join(preprocessed_query).lower()
        sentence_str = ' '.join(sentence).lower()
        preprocessed_query = set(preprocess(query_str))
        preprocessed_sentence = preprocess(sentence_str)

        file_token_set = set(preprocessed_sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        score_dict[i] = similarity

    return score_dict

# 1. Indexing
file_name = "C:\\Users\\82102\\Desktop\\PY202309\\project\\week8\\jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

# 2. Input the query
query = input("영어 쿼리를 입력하세요.")
preprocessed_query = preprocess(query) # query 전처리 - split
query_token_set = set(preprocessed_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(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "\t")
        if rank == 10:
            break
        rank = rank + 1
```

```
영어 쿼리를 입력하세요.Hello My name is Misoo
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

1. Result: I have created a search engine program where users input queries, and it searches for sentences similar to the query, then outputs five sentences that are similar to the user's input query.

2. My thoughts: I realized that handling, processing, and presenting data in a visually appealing manner is not an easy task. I thought about wanting to learn more about data preprocessing techniques.