파이썬프로그래밍및실습[3]

Week 8. Project

Date : 2023. 10. 27.

Name : 손준흠

ID : 234099

1. Introduction

1) Background

If you don't remember the sentence you want to find correctly, you often can't find it in the record. However, if you remember something even if it is not accurate, it will be much easier to access the results that users want among them if you search and find similar sentences based on the text. The result of this project is to develop a program that meets these needs.

2) Project Goal

A query is received from a user, and the top 10 sentences having the highest similarity to the input value are outputted. However, the input value and target sentence are not case-sensitive.

3) Differences from Existing Programs

In most cases, existing programs do not perform special actions unless they find a "matching value" with the value entered by the user or have a "matching value". However, this program differs from existing programs in that it outputs not only "matched values" but also the ranking and score of sentences with the highest similarity to the entered value, the location of the sentence, and the corresponding sentence.

2. Functional Requirement

1) Function 1

- a function that indexes each sentence in a file; preprocess(sentence)

(1) Detailed function 1

- After eliminating spaces before and after the parameters, the strings are separated based on spaces included in the string and stored in a list format.

2) Function 2

- A function that preprocesss sentences within a search target and stores them in the form of a list in file_tokens_pairs; indexing(file_name)

(1) Detailed function 2

- Open the file in a format that reads each sentence of file_name to be searched for, and save each sentence to lines. Each sentence stored in lines is processed using Function1, and the result is added to file_tokens_pairs and returned.

3) Function 3

- A function that scores and returns similarities through two values, parameter 1 and parameter 2; cals_similarity(parameter1, parameter2)

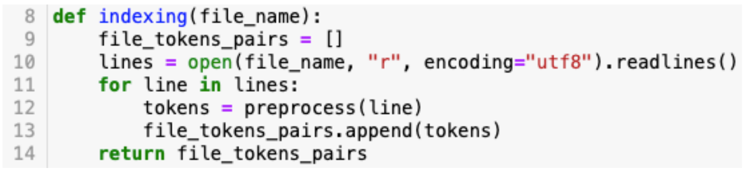
(1) Detailed function 3

- This function uses two values in the parameters of Function1 that are returned the sentence of the search target and the user input query. It is a code block to eliminate the case distinction between the sentence to be searched and the user input query in the internal repeating sentence. The code for external iterations is all_tokens, a combination of preprocessed_query and file_token_set, a similarity score obtained by dividing the intersection same_tokens and all_tokens by same_tokens, and a code block that controls the length of the similarity score.

3. Design and Implementation of Requirements

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

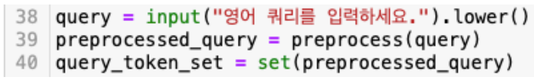
- It was implemented through the 2-2) indexing(file_name) function for the above function requirements.



```python
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
```

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

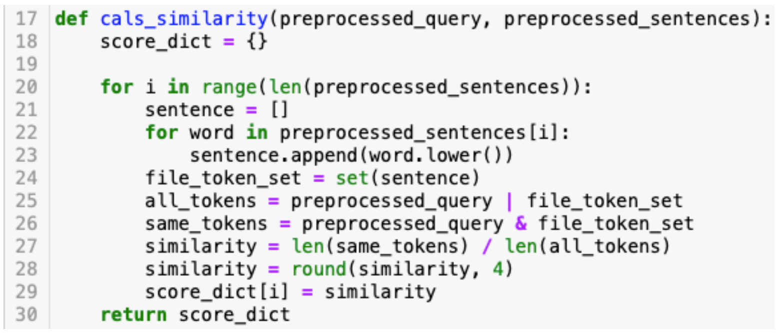
- In order to receive a string (query) from the user through line no.39: query = input("영어 쿼리를 입력하세요.").lower() in the source code, the function was implemented to meet the requirements using the string processing function lower() to receive the sentence and case-insensitive to the search target.



```python
38 query = input("영어 쿼리를 입력하세요.").lower()
39 preprocessed_query = preprocess(query)
40 query_token_set = set(preprocessed_query)
```

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

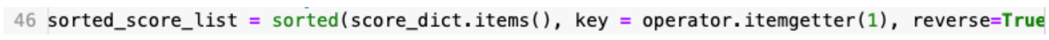
- As a detailed condition of Functional Requirement 3, the similarity was determined by the number of the same words, so that the data type was converted into a set so that duplicate words did not occur. As a result of 2-1) Function1 and 2-2) Function2, the code was written to be a parameter of 2-3) Function3. In addition, when looking at the results, it felt that the number of decimal places in the similarity score was long, so the code was added so that only 4 digits below the decimal point.



```
17 def cals_similarity(preprocessed_query, preprocessed_sentences):
18     score_dict = {}
19 
20     for i in range(len(preprocessed_sentences)):
21         sentence = []
22         for word in preprocessed_sentences[i]:
23             sentence.append(word.lower())
24         file_token_set = set(sentence)
25         all_tokens = preprocessed_query | file_token_set
26         same_tokens = preprocessed_query & file_token_set
27         similarity = len(same_tokens) / len(all_tokens)
28         similarity = round(similarity, 4)
29         score_dict[i] = similarity
30     return score_dict
```

4) Requirement 4. Rank the sentences based on similarity

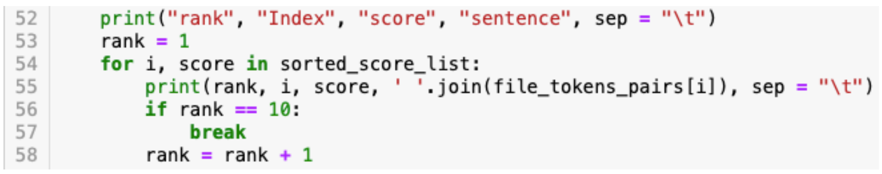
- Similarity scores are stored in score_dict in dictionary form, and since the default values are arranged in ascending order through general alignment, sort(), the function is implemented to check the similarity score from high to low by allowing reverse=True to reverse in ascending order of the default value.



```
46 sorted_score_list = sorted(score_dict.items(), key = operator.itemgetter(1), reverse=True
```

5) Requirement 5. Output the op 10 ranked sentences to the user from the ranked sentences

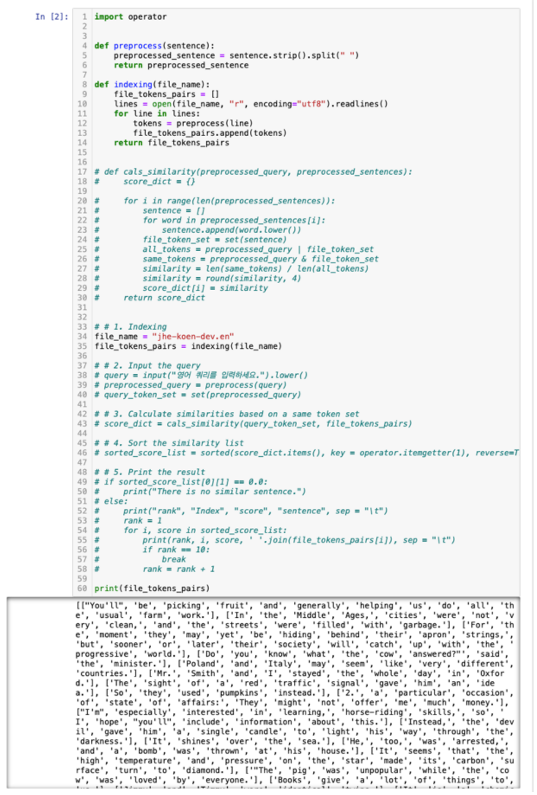
- In the sorted_score_list, where the similarity scores are arranged in descending order, if the index1 of index0 is not 0.0, the line no.53~63 of the source code was implemented so that the corresponding index, score (similarity score), and the corresponding sentence were output in the range of rank=1~10 through for repeated statements.



```
52     print("rank", "Index", "score", "sentence", sep = "\t")
53     rank = 1
54     for i, score in sorted_score_list:
55         print(rank, i, score, ' '.join(file_tokens_pairs[i]), sep = "\t")
56         if rank == 10:
57             break
58         rank = rank + 1
```

4. Test

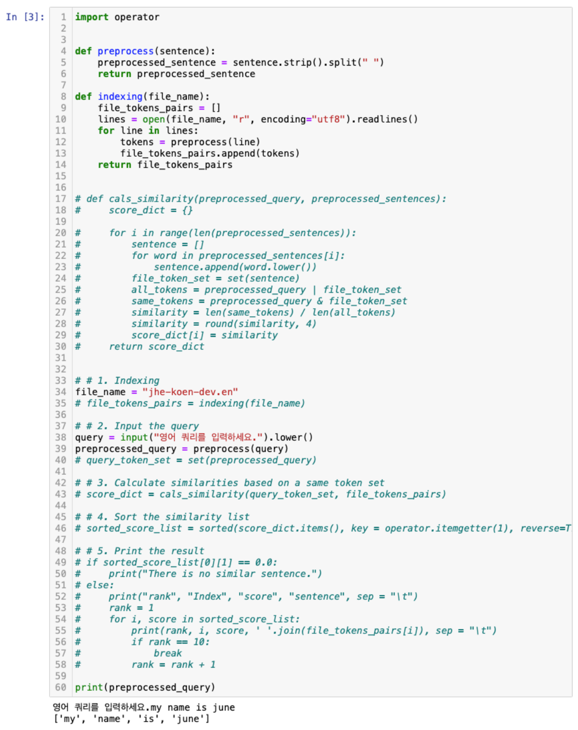
1) Requirement 1.



```
In [2]:
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


# def cals_similarity(preprocessed_query, preprocessed_sentences):
#     score_dict = {}

#     for i in range(len(preprocessed_sentences)):
#         sentence = []
#         for word in preprocessed_sentences[i]:
#             sentence.append(word.lower())
#         file_token_set = set(sentence)
#         all_tokens = preprocessed_query | file_token_set
#         same_tokens = preprocessed_query & file_token_set
#         similarity = len(same_tokens) / len(all_tokens)
#         similarity = round(similarity, 4)
#         score_dict[i] = similarity
#     return score_dict


# # 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

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

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

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

# # 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

print(file_tokens_pairs)
```

```
[["You'll", 'be', 'picking', 'fruit', 'and', 'generally', 'helping', 'us', 'do', 'all', 'th
e', 'usual', 'farm', 'work.'], ['In', 'the', 'Middle', 'Ages,', 'cities', 'were', 'not', 'v
ery', 'clean,', 'and', 'the', 'streets', 'were', 'filled', 'with', 'garbage.'], ['For', 'th
e', 'moment', 'they', 'may', 'yet', 'be', 'hiding', 'behind', 'their', 'apron', 'strings,',
'but', 'sooner', 'or', 'later', 'their', 'society', 'will', 'catch', 'up', 'with', 'the', '
progressive', 'world.'], ['Do', 'you', 'know', 'what', 'the', 'cow', 'answered?"', 'said',
'the', 'minister.'], ['Poland', 'and', 'Italy', 'may', 'seem', 'like', 'very', 'different',
'countries.'], ['Mr.', 'Smith', 'and', 'I', 'stayed', 'the', 'whole', 'day', 'in', 'Oxfor
d.'], ['The', 'sight', 'of', 'a', 'red', 'traffic', 'signal', 'gave', 'him', 'an', 'ide
a.'], ['So', 'they', 'used', 'pumpkins', 'instead.'], ['2.', 'a', 'particular', 'occasion',
'of', 'state', 'of', 'affairs:', 'They', 'might', 'not', 'offer', 'me', 'much', 'money.'],
["I'm", 'especially', 'interested', 'in', 'learning,', 'horse-riding', 'skills,', 'so', '
I', 'hope', "you'll", 'include', 'information', 'about', 'this.'], ['Instead,', 'the', 'dev
il', 'gave', 'him', 'a', 'single', 'candle', 'to', 'light', 'his', 'way', 'through', 'the',
'darkness.'], ['It', 'shines', 'over', 'the', 'sea.'], ['He,', 'too,', 'was', 'arrested,',
'and', 'a', 'bomb', 'was', 'thrown', 'at', 'his', 'house.'], ['It', 'seems', 'that', 'the',
'high', 'temperature', 'and', 'pressure', 'on', 'the', 'star', 'made', 'its', 'carbon', 'su
rface', 'turn', 'to', 'diamond.'], ['"The', 'pig', 'was', 'unpopular', 'while', 'the', 'co
w', 'was', 'loved', 'by', 'everyone.'], ['Books', 'give', 'a', 'lot', 'of', 'things', 'to',
```

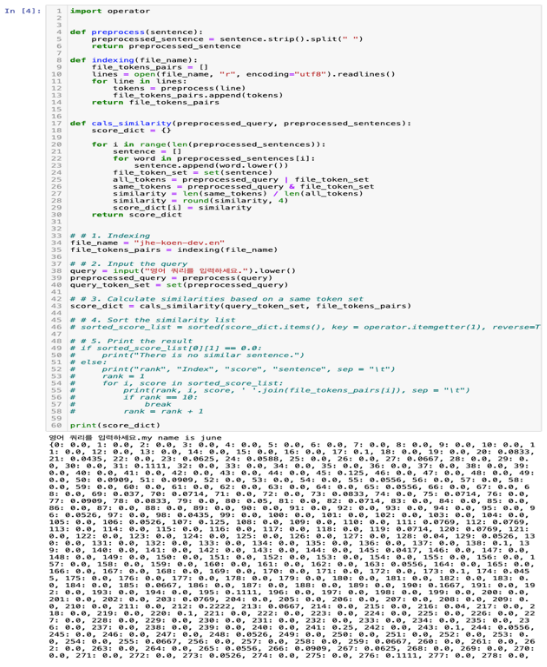
2) Requirement 2.



```
In [3]:
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


# def cals_similarity(preprocessed_query, preprocessed_sentences):
#     score_dict = {}

#     for i in range(len(preprocessed_sentences)):
#         sentence = []
#         for word in preprocessed_sentences[i]:
#             sentence.append(word.lower())
#         file_token_set = set(sentence)
#         all_tokens = preprocessed_query | file_token_set
#         same_tokens = preprocessed_query & file_token_set
#         similarity = len(same_tokens) / len(all_tokens)
#         similarity = round(similarity, 4)
#         score_dict[i] = similarity
#     return score_dict


# # 1. Indexing
file_name = "jhe-koen-dev.en"
# file_tokens_pairs = indexing(file_name)

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

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

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

# # 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

print(preprocessed_query)
```

```
영어 쿼리를 입력하세요.my name is june
['my', 'name', 'is', 'june']
```

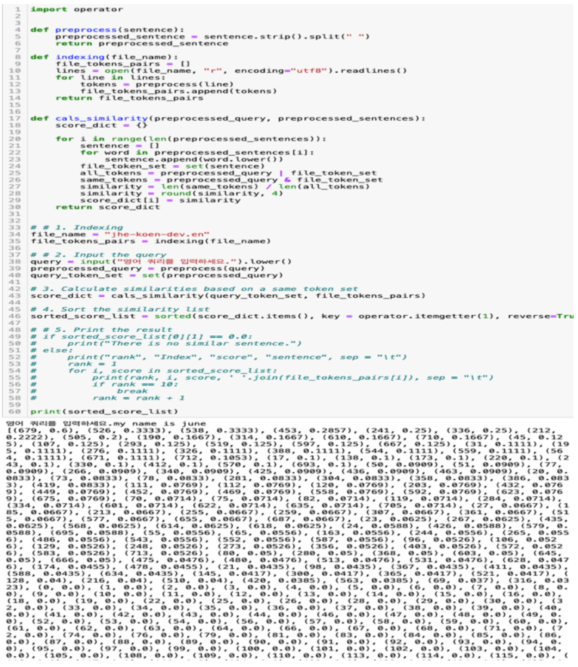
3) Requirement 3.



```
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


def cals_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}

    for i in range(len(preprocessed_sentences)):
        sentence = []
        for word in preprocessed_sentences[i]:
            sentence.append(word.lower())
        file_token_set = set(sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        similarity = round(similarity, 4)
        score_dict[i] = similarity
    return score_dict


# # 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

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

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

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

# # 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

print(score_dict)
```

```
영어 쿼리를 입력하세요.my name is june
{0: 0.0, 1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0, 5: 0.0, 6: 0.0, 7: 0.0, 8: 0.0, 9: 0.0, 10: 0.0, 1
1: 0.0, 12: 0.0, 13: 0.0, 14: 0.0, 15: 0.0, 16: 0.0, 17: 0.1, 18: 0.0, 19: 0.0, 20: 0.0833,
21: 0.0435, 22: 0.0, 23: 0.0625, 24: 0.0588, 25: 0.0, 26: 0.0, 27: 0.0667, 28: 0.0, 29: 0.
0, 30: 0.0, 31: 0.1111, 32: 0.0, 33: 0.0, 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: 0.0, 43: 0.0, 44: 0.0, 45: 0.125, 46: 0.0, 47: 0.0, 48: 0.0, 49:
0.0, 50: 0.0909, 51: 0.0909, 52: 0.0, 53: 0.0, 54: 0.0, 55: 0.0556, 56: 0.0, 57: 0.0, 58:
0.0, 59: 0.0, 60: 0.0, 61: 0.0, 62: 0.0, 63: 0.0, 64: 0.0, 65: 0.0556, 66: 0.0, 67: 0.0, 6
8: 0.0, 69: 0.037, 70: 0.0714, 71: 0.0, 72: 0.0, 73: 0.0833, 74: 0.0, 75: 0.0714, 76: 0.0,
77: 0.0909, 78: 0.0833, 79: 0.0, 80: 0.05, 81: 0.0, 82: 0.0714, 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, 9
6: 0.0526, 97: 0.0, 98: 0.0435, 99: 0.0, 100: 0.0, 101: 0.0, 102: 0.0, 103: 0.0, 104: 0.0,
105: 0.0, 106: 0.0526, 107: 0.125, 108: 0.0, 109: 0.0, 110: 0.0, 111: 0.0769, 112: 0.0769,
113: 0.0, 114: 0.0, 115: 0.0, 116: 0.0, 117: 0.0, 118: 0.0, 119: 0.0714, 120: 0.0769, 121:
0.0, 122: 0.0, 123: 0.0, 124: 0.0, 125: 0.0, 126: 0.0, 127: 0.0, 128: 0.04, 129: 0.0526, 13
0: 0.0, 131: 0.0, 132: 0.0, 133: 0.0, 134: 0.0, 135: 0.0, 136: 0.0, 137: 0.0, 138: 0.1, 13
9: 0.0, 140: 0.0, 141: 0.0, 142: 0.0, 143: 0.0, 144: 0.0, 145: 0.0417, 146: 0.0, 147: 0.0,
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, 1
57: 0.0, 158: 0.0, 159: 0.0, 160: 0.0, 161: 0.0, 162: 0.0, 163: 0.0556, 164: 0.0, 165: 0.0,
166: 0.0, 167: 0.0, 168: 0.0, 169: 0.0, 170: 0.0, 171: 0.0, 172: 0.0, 173: 0.1, 174: 0.045
5, 175: 0.0, 176: 0.0, 177: 0.0, 178: 0.0, 179: 0.0, 180: 0.0, 181: 0.0, 182: 0.0, 183: 0.
0, 184: 0.0, 185: 0.0667, 186: 0.0, 187: 0.0, 188: 0.0, 189: 0.0, 190: 0.1667, 191: 0.0, 19
2: 0.0, 193: 0.0, 194: 0.0, 195: 0.1111, 196: 0.0, 197: 0.0, 198: 0.0, 199: 0.0, 200: 0.0,
201: 0.0, 202: 0.0, 203: 0.0769, 204: 0.0, 205: 0.0, 206: 0.0, 207: 0.0, 208: 0.0, 209: 0.
0, 210: 0.0, 211: 0.0, 212: 0.2222, 213: 0.0667, 214: 0.0, 215: 0.0, 216: 0.04, 217: 0.0, 2
18: 0.0, 219: 0.0, 220: 0.1, 221: 0.0, 222: 0.0, 223: 0.0, 224: 0.0, 225: 0.0, 226: 0.0, 22
7: 0.0, 228: 0.0, 229: 0.0, 230: 0.0, 231: 0.0, 232: 0.0, 233: 0.0, 234: 0.0, 235: 0.0, 23
6: 0.0, 237: 0.0, 238: 0.0, 239: 0.0, 240: 0.0, 241: 0.25, 242: 0.0, 243: 0.1, 244: 0.0556,
245: 0.0, 246: 0.0, 247: 0.0, 248: 0.0526, 249: 0.0, 250: 0.0, 251: 0.0, 252: 0.0, 253: 0.
0, 254: 0.0, 255: 0.0667, 256: 0.0, 257: 0.0, 258: 0.0, 259: 0.0667, 260: 0.0, 261: 0.0, 26
2: 0.0, 263: 0.0, 264: 0.0, 265: 0.0556, 266: 0.0909, 267: 0.0625, 268: 0.0, 269: 0.0, 270:
0.0, 271: 0.0, 272: 0.0, 273: 0.0526, 274: 0.0, 275: 0.0, 276: 0.1111, 277: 0.0, 278: 0.0,
```

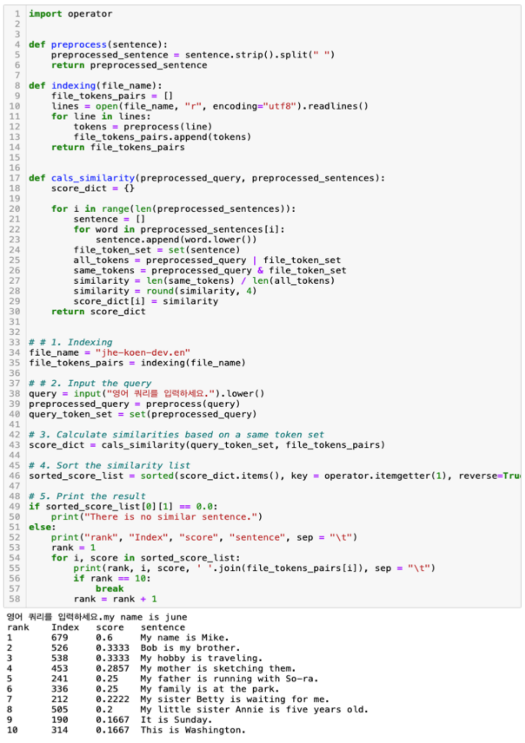
4) Requirement 4.



```
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


def cals_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}

    for i in range(len(preprocessed_sentences)):
        sentence = []
        for word in preprocessed_sentences[i]:
            sentence.append(word.lower())
        file_token_set = set(sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        similarity = round(similarity, 4)
        score_dict[i] = similarity
    return score_dict


# # 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

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

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

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

# # 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

print(sorted_score_list)
```

```
영어 쿼리를 입력하세요.my name is june
[(679, 0.6), (526, 0.3333), (538, 0.3333), (453, 0.2857), (241, 0.25), (336, 0.25), (212,
0.2222), (505, 0.2), (190, 0.1667), (314, 0.1667), (610, 0.1667), (710, 0.1667), (45, 0.12
5), (107, 0.125), (293, 0.125), (519, 0.125), (597, 0.125), (667, 0.125), (31, 0.1111), (19
5, 0.1111), (276, 0.1111), (326, 0.1111), (388, 0.1111), (544, 0.1111), (559, 0.1111), (56
4, 0.1111), (671, 0.1111), (712, 0.1053), (17, 0.1), (138, 0.1), (173, 0.1), (220, 0.1), (2
43, 0.1), (330, 0.1), (412, 0.1), (570, 0.1), (693, 0.1), (50, 0.0909), (51, 0.0909), (77,
0.0909), (266, 0.0909), (340, 0.0909), (425, 0.0909), (436, 0.0909), (463, 0.0909), (20, 0.
0833), (73, 0.0833), (78, 0.0833), (281, 0.0833), (304, 0.0833), (358, 0.0833), (386, 0.083
3), (419, 0.0833), (111, 0.0769), (112, 0.0769), (120, 0.0769), (203, 0.0769), (432, 0.076
9), (449, 0.0769), (452, 0.0769), (469, 0.0769), (558, 0.0769), (592, 0.0769), (623, 0.076
9), (675, 0.0769), (70, 0.0714), (75, 0.0714), (82, 0.0714), (119, 0.0714), (284, 0.0714),
(334, 0.0714), (601, 0.0714), (622, 0.0714), (635, 0.0714), (705, 0.0714), (27, 0.0667), (1
85, 0.0667), (213, 0.0667), (255, 0.0667), (259, 0.0667), (307, 0.0667), (361, 0.0667), (51
5, 0.0667), (577, 0.0667), (655, 0.0667), (687, 0.0667), (23, 0.0625), (267, 0.0625), (435,
0.0625), (568, 0.0625), (614, 0.0625), (618, 0.0625), (24, 0.0588), (426, 0.0588), (579, 0.
0588), (695, 0.0588), (55, 0.0556), (65, 0.0556), (163, 0.0556), (244, 0.0556), (265, 0.055
6), (486, 0.0556), (543, 0.0556), (552, 0.0556), (587, 0.0556), (96, 0.0526), (106, 0.052
6), (129, 0.0526), (248, 0.0526), (273, 0.0526), (356, 0.0526), (403, 0.0526), (572, 0.052
6), (583, 0.0526), (713, 0.0526), (80, 0.05), (280, 0.05), (368, 0.05), (603, 0.05), (645,
0.05), (666, 0.05), (475, 0.0476), (480, 0.0476), (513, 0.0476), (531, 0.0476), (628, 0.047
6), (174, 0.0455), (478, 0.0455), (21, 0.0435), (98, 0.0435), (367, 0.0435), (411, 0.0435),
(588, 0.0435), (634, 0.0435), (145, 0.0417), (300, 0.0417), (365, 0.0417), (521, 0.0417), (
128, 0.04), (216, 0.04), (510, 0.04), (420, 0.0385), (563, 0.0385), (69, 0.037), (316, 0.03
23), (0, 0.0), (1, 0.0), (2, 0.0), (3, 0.0), (4, 0.0), (5, 0.0), (6, 0.0), (7, 0.0), (8, 0.
0), (9, 0.0), (10, 0.0), (11, 0.0), (12, 0.0), (13, 0.0), (14, 0.0), (15, 0.0), (16, 0.0),
(18, 0.0), (19, 0.0), (22, 0.0), (25, 0.0), (26, 0.0), (28, 0.0), (29, 0.0), (30, 0.0), (3
2, 0.0), (33, 0.0), (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, 0.0), (43, 0.0), (44, 0.0), (46, 0.0), (47, 0.0), (48, 0.0), (49, 0.
0), (52, 0.0), (53, 0.0), (54, 0.0), (56, 0.0), (57, 0.0), (58, 0.0), (59, 0.0), (60, 0.0),
(61, 0.0), (62, 0.0), (63, 0.0), (64, 0.0), (66, 0.0), (67, 0.0), (68, 0.0), (71, 0.0), (7
2, 0.0), (74, 0.0), (76, 0.0), (79, 0.0), (81, 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), (97, 0.0), (99, 0.0), (100, 0.0), (101, 0.0), (102, 0.0), (103, 0.0), (104,
0.0), (105, 0.0), (108, 0.0), (109, 0.0), (110, 0.0), (113, 0.0), (114, 0.0), (115, 0.0), (
```

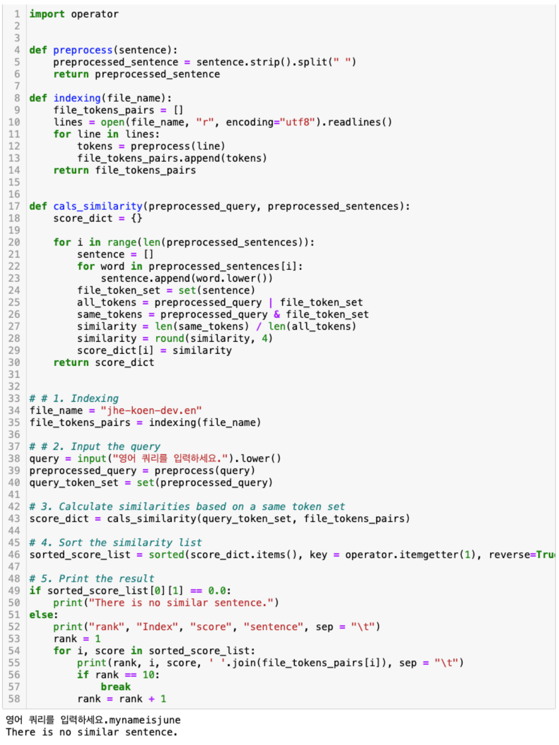
5) Requirement 5.



```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


def cals_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}

    for i in range(len(preprocessed_sentences)):
        sentence = []
        for word in preprocessed_sentences[i]:
            sentence.append(word.lower())
        file_token_set = set(sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        similarity = round(similarity, 4)
        score_dict[i] = similarity
    return score_dict


# # 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

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

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

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

# 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.my name is june
rank	Index	score	sentence
1	679	0.6	My name is Mike.
2	526	0.3333	Bob is my brother.
3	538	0.3333	My hobby is traveling.
4	453	0.2857	My mother is sketching them.
5	241	0.25	My father is running with So-ra.
6	336	0.25	My family is at the park.
7	212	0.2222	My sister Betty is waiting for me.
8	505	0.2	My little sister Annie is five years old.
9	190	0.1667	It is Sunday.
10	314	0.1667	This is Washington.
```

6) If there is no similar sentence



```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


def cals_similarity(preprocessed_query, preprocessed_sentences):
    score_dict = {}

    for i in range(len(preprocessed_sentences)):
        sentence = []
        for word in preprocessed_sentences[i]:
            sentence.append(word.lower())
        file_token_set = set(sentence)
        all_tokens = preprocessed_query | file_token_set
        same_tokens = preprocessed_query & file_token_set
        similarity = len(same_tokens) / len(all_tokens)
        similarity = round(similarity, 4)
        score_dict[i] = similarity
    return score_dict


# # 1. Indexing
file_name = "jhe-koen-dev.en"
file_tokens_pairs = indexing(file_name)

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

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

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

# 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.mynameisjune
There is no similar sentence.
```

5. Results and conclusions

1. Results of project

- We can now receive user input and return similar sentences from search targets. As a result, even if you remember inaccurate content, you can choose the correct sentence from among sentences with several similar words.

2. Conclusion

- I had never done like this before this project, and when I learned various functions one by one, including string processing functions used in the project, I thought, "What can I do with this?" and was quite surprised to be able to create a program that can be used with what I've learned so far. I was just an old man, evaluating myself beyond words, but I realized that I was able to mutter. However, he also realized that he still has a long way to go to become a developer. And... Professor, I'm sad that the progress is so fast...