Exercise 1

Code is in `exercise1.py`.

The code for `df`.

```
def df(letters, docs):
    _df = defaultdict(int)

    # populate dft
    for letter_target in letters:
        for doc_values in docs.values():
            if letter_target in doc_values:
                _df[letter_target] += 1

    return dict(_df)
```

The code for `idf`.

```
def idf(letters, docs, smoothing=0):
    N = len(docs)

    _df = df(letters, docs)

    # calculate idf
    _idf = defaultdict(int)
    for letter_target in letters:
        _idf[letter_target] = math.log10((N + smoothing) /
(_df[letter_target] + smoothing))

    return dict(_idf)
```

The code for `c` score.

```
def c(letters, docs, smoothing=0):
    N = len(docs)

    _df = df(letters, docs)

    _c = defaultdict(int)
    for letter_target in letters:
        _c[letter_target] = math.log10( (N - _df[letter_target] +
smoothing) / (_df[letter_target] + smoothing) )
```

```
    return dict(_c)
```

The code for `rsv` score.

```
def rsv(c, query, docs):
    _rsv = defaultdict(int)

    for letter in query:
        for doc, doc_values in docs.items():
            if letter in doc_values:
                _rsv[doc] += c.get(letter, 0)

    return dict(_rsv)
```

Sorting code

```
query = "mvykw"
_rsv = rsv(_c, query, docs)
print(f"RSV with given c = {_rsv}")

print(f"Sorting Based on RSV (ascending), {sorted(_rsv, key = _rsv.get)}")
```

Here is the result for all.

```
IDF score, With smoothing = 0
 idf={'x': 0.2218487496163564, 'y': 0.0, 'z': 0.3979400086720376, 'w':
0.2218487496163564, 'v': 0.2218487496163564, 'k': 0.6989700043360189, 'm':
0.3979400086720376}
IDF score, With smoothing = 0.5
 idf={'x': 0.1962946451439682, 'y': 0.0, 'z': 0.3424226808222063, 'w':
0.1962946451439682, 'v': 0.1962946451439682, 'k': 0.5642714304385625, 'm':
0.3424226808222063}
C Score, With smoothing = 0.5
 C={'x': -0.146128035678238, 'y': -1.041392685158225, 'z':
0.146128035678238, 'w': -0.146128035678238, 'v': -0.146128035678238, 'k':
0.47712125471966244, 'm': 0.146128035678238}
RSV with given c = {'D3': -1.041392685158225, 'D5': -1.041392685158225,
'D2': -0.7103994661168005, 'D4': -1.3336487565147008, 'D1':
-1.1875207208364629}
Sorting Based on RSV (ascending), ['D4', 'D1', 'D3', 'D5', 'D2']
```

Extra notes, here keep in mind that `D3` and `D5` has the same RSV, so it equal in relevance. However, the python function output would still compare it and put it in place.

Exercise 2

Code is in `exercise2.py`.

To find `pt`, this is the function

```
def pt(letter_target, relevant_docs):
    # <number-of-term-exist-in-relevant-docs> / <total-relevant-docs>
    count_exist = 0 # s
    relevant_doc_size = 0 # S

    for letters in relevant_docs.values():
        if letter_target in letters:
            count_exist += 1
        relevant_doc_size += 1

    return count_exist / relevant_doc_size

for letter_target in ['x', 'y', 'z', 'w', 'v', 'k', 'm']:
    print(f"pt({letter_target}, ...) = {pt(letter_target,
relevant_docs)}")
```

Here's the result

```
pt(x, ...) = 1.0
pt(y, ...) = 1.0
pt(z, ...) = 0.5
pt(w, ...) = 0.5
pt(v, ...) = 0.5
pt(k, ...) = 0.5
pt(m, ...) = 0.0
```

TP find `ut`, this is the function

```
def ut(letter_target, N, relevant_docs, word_frequency):
    # <number-of-term-exist-in-non-relevant-docs> / (N - <total-relevant-
docs>)
    count_exist = 0
    relevant_doc_size = 0

    for letters in relevant_docs.values():
        if letter_target in letters:
            count_exist += 1
        relevant_doc_size += 1

    return (word_frequency[letter_target] - count_exist) / (N -
relevant_doc_size)
```

```
for letter_target in ['x', 'y', 'z', 'w', 'v', 'k', 'm']:
    print(f"ut({letter_target}, ...) = {ut(letter_target, N,
relevant_docs, word_frequency)}")
```

This is the result

```
ut(x, ...) = 0.4642857142857143
ut(y, ...) = 0.35714285714285715
ut(z, ...) = 0.4642857142857143
ut(w, ...) = 0.5357142857142857
ut(v, ...) = 0.6071428571428571
ut(k, ...) = 0.32142857142857145
ut(m, ...) = 0.2857142857142857
```

To find `ct` this is the function

```
def ct(letter_target, N, relevant_docs, word_frequency, smoothing=0.5):
    count_exist = 0 # s
    relevant_doc_size = 0 # S

    for letters in relevant_docs.values():
        if letter_target in letters:
            count_exist += 1
        relevant_doc_size += 1

    numerator = (count_exist+smoothing) / (relevant_doc_size-
count_exist+smoothing)
    denominator = (word_frequency[letter_target]-count_exist+smoothing) /
(N-word_frequency[letter_target]-relevant_doc_size+count_exist+smoothing)

    return numerator/denominator


ct_scores = defaultdict(int)
for letter_target in ['x', 'y', 'z', 'w', 'v', 'k', 'm']:
    print(f"ct({letter_target}, ...) = {ct(letter_target, N,
relevant_docs, word_frequency)}")
    ct_scores[letter_target] = ct(letter_target, N, relevant_docs,
word_frequency)
```

This is the result.

```
ct(x, ...) = 0.7589679340113041
ct(y, ...) = 0.9449524336690945
ct(z, ...) = 0.05999792967528537
ct(w, ...) = -0.05999792967528537
```

```
ct(v, ...) = -0.18234020833268275
ct(k, ...) = 0.3123110060736703
ct(m, ...) = -0.3166350689945572
```

For `rsv` for each `additional_docs`.

```
def rsv(query, doc_values, ct_scores):
    score = 0
    for letter in query:
        if letter in doc_values:
            score += ct_scores[letter]

    return score

query = "mvykw"
for doc, doc_values in additional_docs.items():
    print(f"rsv({query}, {doc}, ...) = {rsv(query, doc_values,
ct_scores)}")
```

This is the result.

```
rsv(mvykw, D6, ...) = 0.2523130763983849
rsv(mvykw, D7, ...) = 0.7626122253364118
rsv(mvykw, D8, ...) = 0.44597715634185464
```

Based on this RSV lowest to highest, `D6` → `D8` → `D7`. We don't need to use info about non-relevant docs, because they contain irrelevant terms, which might increase RSV value unecessarily, even though the document is irrelevant.

Exercise 3

File is in `exercise3.py`.

To calculate `tf_`,`df_`, `idf_`, `_score`.

```
...
def tf_(doc: List[str]):
    frequencies = defaultdict(int)
    for letter in doc:
        frequencies[letter] += 1

    return dict(frequencies)
...
def df_(docs: List[List[str]]):
    df = defaultdict(int)
```

```
    for doc in docs:
        for letter in doc:
            df[letter] += 1

    return dict(df)
...
def idf_(df, corpus_size):
    idf = {}
    for term, freq in df.items():
        idf[term] = round(math.log((corpus_size) / (freq)),2)
    return idf
...
def _score(query, doc, docs, k1=1.5, b=0.75):
    score = 0.0
    tf = tf_(doc)
    df = df_(docs)
    idf = idf_(df, len(docs))
    avg_doc_len = sum(len(doc) for doc in docs)/len(docs) # calculate
average document length
    for term in query:
        if term not in tf.keys():
            continue

        numerator = (k1+1) * tf[term]
        denominator = k1 * ( (1-b) + b*len(doc)/avg_doc_len ) + tf[term]

        score += idf[term] * numerator/denominator
    return score
...
```

With this results for each of those function and given params.

```
tf_(['a', 'b', 'b', 'c', 'd']) = {'a': 1, 'b': 2, 'c': 1, 'd': 1}
df_([['a', 'b', 'c'], ['b', 'c', 'd'], ['c', 'd', 'e']]) = {'a': 1, 'b':
2, 'c': 3, 'd': 2, 'e': 1}
idf_({'a': 1, 'b': 2, 'c': 3, 'd': 2, 'e': 1}) = {'a': 1.1, 'b': 0.41,
'c': 0.0, 'd': 0.41, 'e': 1.1}
_score(['b', 'c', 'e'], ['b', 'c', 'd'],
[['a', 'b', 'c'], ['b', 'c', 'd'], ['c', 'd', 'e']]) = 0.41
```