Coding

1. **Please encapsulate your coding logic in a Python class**

- Classes help in managing your coding logic
    - Writing a bunch of python functions in a file leads to unreadable code
    - Classes also helps you test your code easily.
    - But don't forget to check if the object is None or not before you are accessing the object's attribute. i.e. Try to avoid `AttributeError: 'NoneType' object has no attribute 'something'`

2. **Test your code against a wide set of queries**

- Some of the code submissions fail to run because of programming error (will be included in your submission feedback)
- Make sure that you write tests (look at pytest if you can.). It seems like a waste of time at first. But makes you efficient (On an average every submission for the 2nd assignment has ~400 lines of code.)
- Some test queries were made available by other students on the forum

3. **Be careful passing dictionaries around in your code**

- Dictionaries are mutable in python. If you pass it to another function and change it, it remains changed even after the function finishes execution. Tracking what happens to the dictionary becomes really difficult.

4. **Keep it simple**

    - The simpler the code, the easier it is to modify and read.

5. **Use pathlib**
    - Take a look at path lib library to handle paths and operations on them for creating files, iterating through directories, obtaining specific parts of the filename etc.
    - Here is a tutorial https://realpython.com/python-pathlib/
    - Always check and fix any hard coding of paths before submission. One way to make sure is to upload the script on sunfire/tembusu and re-run it.
6. **Use error handling**
    - Please use error handling so that we are able to troubleshoot issues faster and get back to you if needed. Where possible please add informative print statements in the error handling.
    - For example, you might want to include `try… except…` statement and print out the query if there is an error.

Indexing

- Try using the defaultdict module available in python. You don't have to check whether key exists in the dictionary in this case

```
my_dict = defaultdict(list)
my_dict[key].append(1)
```

- Skip pointers have to be created at the indexing time and not during search. The search will get slowed down if you create indexing at search time.

  Here is how I would do it

    - Create a class for the skip list.. say `SkipList`
    - It implements something like a Linked List
    - The node in the Linked List has three values 1) The doc id where the term is found 2) the pointer to the next node 3) The skip field that points to another node in the list far away (This might be empty)

- Create an appropriate way to serialise the objects of this class and deserialise them by reading them from posting file.
- If you find a term twice in the document and want to find the unique document-ids, a simple way to do it would be

```
unique_doc_ids = list(set(doc_ids))
```

- Would you or would you not create a skip list for the intermediate result. Say you have a query with 50 ANDs, then would you save time during search ?? Think this in tandem with the first point we mentioned in this section
- You have to tokenise by sentences first and then words. Just splitting by words may not given an appropriate answer. For more sophisticated word tokenisation you can refer to

  https://spacy.io/usage/spacy-101. But nltk may not follow a similar algorithm. So you have to be careful.

Searching

- The parsing of queries should be efficient and follow the precedence order which is always not the case with students
- The best way is to use the Shunting Yard Algorithm
- Consider how you tokenise the query as well. What If the

query has "(word1 AND word2)" and "(word1 AND word2)". If the tokenisation of the query is not done properly, you would not be producing the correct answers.
- You are required to implement the AND operation as described in the lecture notes. However for other operations, you can implement using set operations.

General

- Do not copy your code from public repositories found on Github for this course. We have a separate set of test queries for your code. The GitHub code might be performing bad and the logic may be flawed.
- You Readme should explain your logic in the python files concisely. Please do not detail every line of code that you have written.
- Good practices for documentation in the python files
    - Write the purpose of the function you write
    - Document the type of data-structure that you return from the function
    - Describe the type for every function parameter.

  If you use PyCharm then type `"""` inside a function or class and hit enter. The documentation template is automatically generated.