# Project Part 1 Report

In the class InvertedIndex, three functions were implemented:

*index_document(documents),*
*split_query(Q, DoE),*
*max_score_query(query_splits, doc_id).*

Spacy, math.log, itertools were imported to help implement this InvertedIndex class.

Implementation:

1. *index_document(documents)*
   For given documents, iterate through each document, use '*nlp*' from *spacy* to analysis each document and obtain lists of tokens and entities. Remove single-word entity from token list. Count the occurrence (*term frequency*) of *tokens/ entities* and store it in the form {*docID: term frequency*} in *tf_tokens/tf_entities*. Then calculate Inverse Document Frequency(*idf*) of *tokens/entities* based on *tf_tokens/tf_entities*.

   Algorithm:
   1) Iterate through each document, get *token_list* and *entity_list* by *spacy*. Tokens only added to *token_list* when it's not *punctuation* or *stopword* or *space*. Remove the single-word entity in *token_list*.
   2) Iterate through *token_list* and *entity_list*, construct *self.tf_tokens* and *self.tf_entities* by recording the term frequency in every document.
   3) Calculate *self.idf_tokens* and *self.idf_entities* based on *self.tf_tokens*/*self.tf_entities* and total number of documents.
2. *split_query(Q, DoE)*
   Remove the entities in DoE which never appears in Q. Enumerate all possible subset of DoE and append this subset to the *returnValue* if all entities in this subset could be found in the Query in increasing order.

   Algorithm:
   1) split the *query* into list of tokens (*LoT*)
   2) Iterate through each entity in *DoE*, check if each word in entity appears in *LoT*, remove it from the *DoE* if it doesn't.
   3) Enumerate all possible subset of *DoE*, call it list of combination (*LoC*). Iterate through *LoC*, check if all entities appear in increasing order, if yes, append it to the *returnValue*.
   4) The way to check if all entities appears in increasing order: iterate through *entities.split()*, get index of each word if it's in the *query*, mark *current index*. Repeatedly check if word is in *query[current index:]* and get it's index. If *list of index = sorted(list of index)* -> indexes in this list is in ascending order.
3. *max_score_query(query_splits, doc_id)*
   Iterate through each *query_splits*, calculate *tf_idf scores* for each *query_splits* and return maximum score and corresponding *query_splits*.

   Algorithm:
   1) Iterate through *query_splits*.

Accumulate token scores through each token in *query_splits*: *normalize tf_tokens * idf_tokens*
Accumulate entity scores through each entity in *query_splits*: *normalize tf_entities * idf_entities*
Combines score = 0.4* *token_score* + *entity_score*

2) Return *max score* and corresponding *query_splits*.