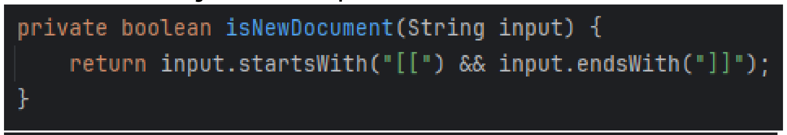
We have implemented 2 solutions, one in Java using the Lucene IR system and one in Python using the Whoosh IR system. This document will contain for both solutions the following:

- the description of the code
- results
- answers for all questions

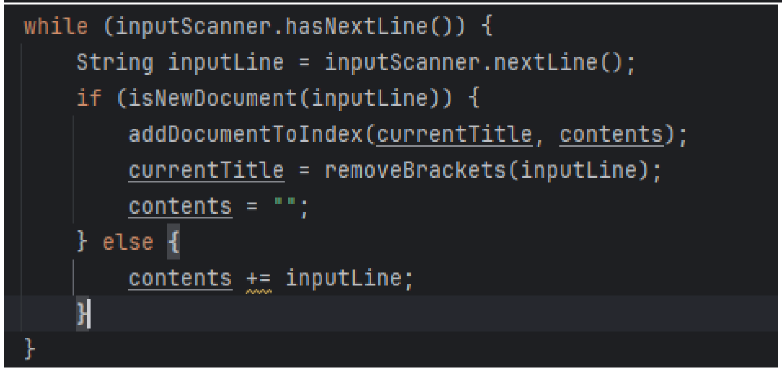
Lucene

Description of the code

In the **indexer.java** we separate the documents in the index for ach Wikipedia page:



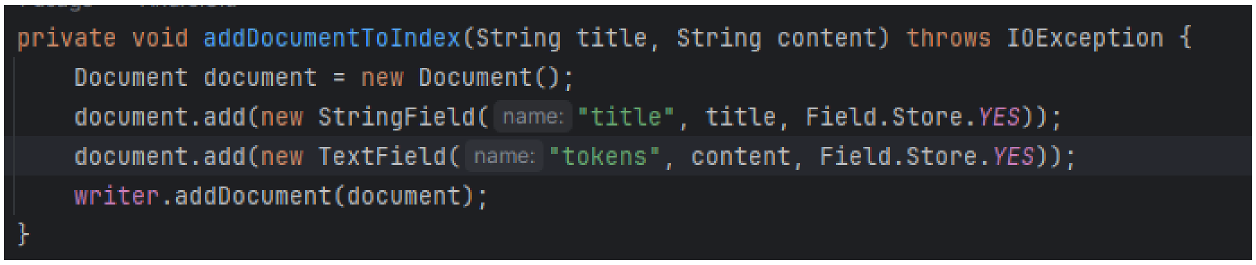
```java
private boolean isNewDocument(String input) {
    return input.startsWith("[[") && input.endsWith("]]");
}
```



```java
while (inputScanner.hasNextLine()) {
    String inputLine = inputScanner.nextLine();
    if (isNewDocument(inputLine)) {
        addDocumentToIndex(currentTitle, contents);
        currentTitle = removeBrackets(inputLine);
        contents = "";
    } else {
        contents += inputLine;
    }
}
```

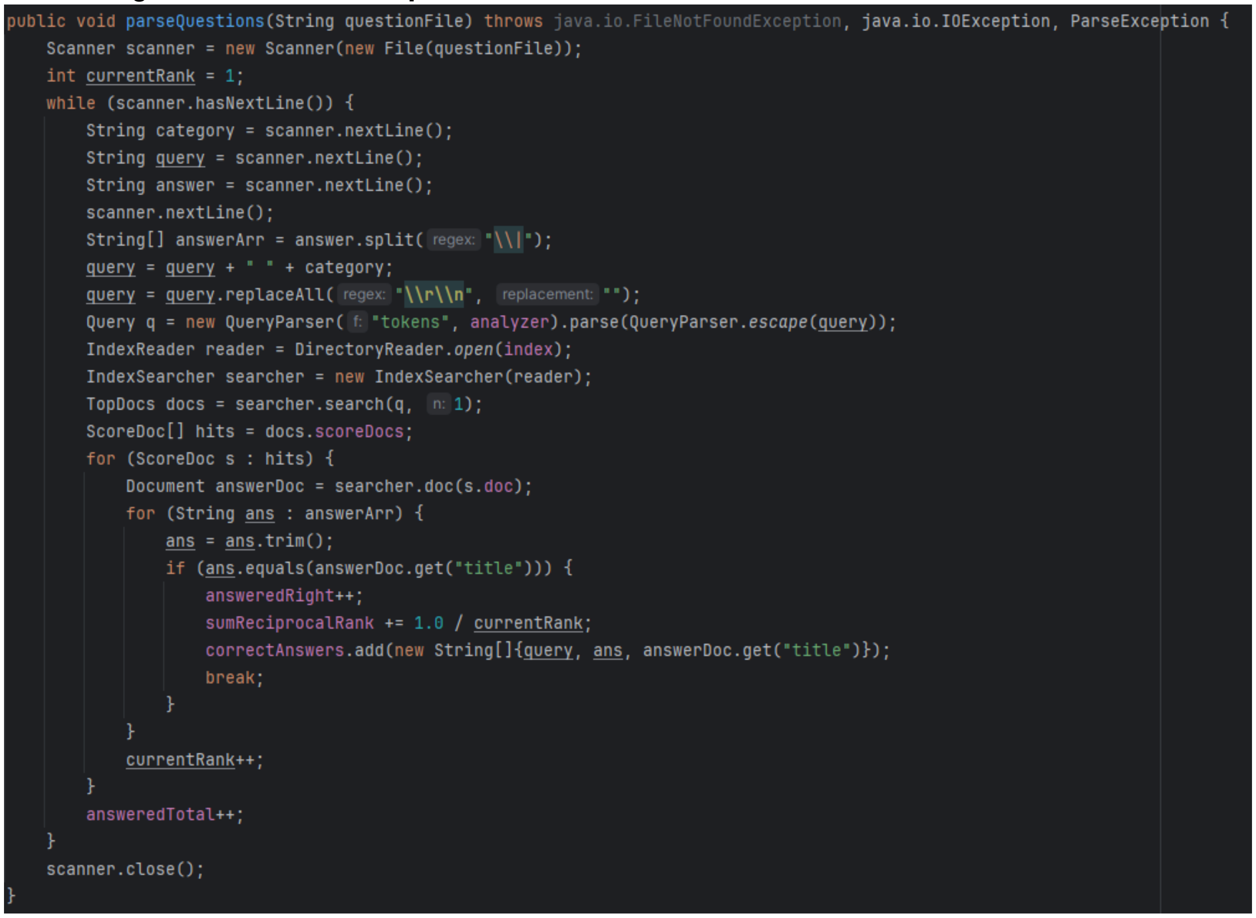
In this class we have a method called **buildIndex** that indexes pages and processes terms. We have tried ‘stop word’ and ‘stemming’ provided by Lucene`s **EnglishAnalyzer** method but we got better results using Lucene`s **StandardAnalyzer** method.

Another method worth mentioning in this class is the **addDocumentToIndex** method that complements the one mentioned above that adds the pages` title and content to the index



```java
private void addDocumentToIndex(String title, String content) throws IOException {
    Document document = new Document();
    document.add(new StringField( name: "title", title, Field.Store.YES));
    document.add(new TextField( name: "tokens", content, Field.Store.YES));
    writer.addDocument(document);
}
```

Next is the class called **Watson.java** that searches the indexed files for the answers. Worth mentioning here is the method **parseQuestions**

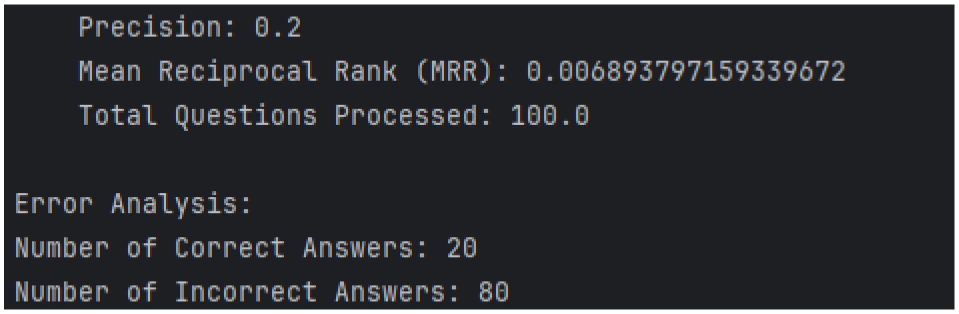


```java
public void parseQuestions(String questionFile) throws java.io.FileNotFoundException, java.io.IOException, ParseException {
    Scanner scanner = new Scanner(new File(questionFile));
    int currentRank = 1;
    while (scanner.hasNextLine()) {
        String category = scanner.nextLine();
        String query = scanner.nextLine();
        String answer = scanner.nextLine();
        scanner.nextLine();
        String[] answerArr = answer.split( regex: "\\|");
        query = query + " " + category;
        query = query.replaceAll( regex: "\\r\\n",  replacement: "");
        Query q = new QueryParser( f: "tokens", analyzer).parse(QueryParser.escape(query));
        IndexReader reader = DirectoryReader.open(index);
        IndexSearcher searcher = new IndexSearcher(reader);
        TopDocs docs = searcher.search(q,  n: 1);
        ScoreDoc[] hits = docs.scoreDocs;
        for (ScoreDoc s : hits) {
            Document answerDoc = searcher.doc(s.doc);
            for (String ans : answerArr) {
                ans = ans.trim();
                if (ans.equals(answerDoc.get("title"))) {
                    answeredRight++;
                    sumReciprocalRank += 1.0 / currentRank;
                    correctAnswers.add(new String[]{query, ans, answerDoc.get("title")});
                    break;
                }
            }
            currentRank++;
        }
        answeredTotal++;
    }
    scanner.close();
}
```

Here we only yield the first provided answer by the system as to provide the p@1 measurement.

In this class we also measure and return the result of MRR.

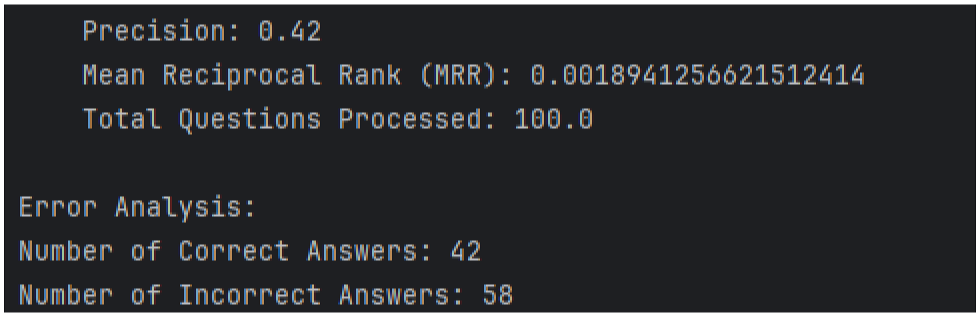
Results



```
    Precision: 0.2
    Mean Reciprocal Rank (MRR): 0.006893797159339672
    Total Questions Processed: 100.0

Error Analysis:
Number of Correct Answers: 20
Number of Incorrect Answers: 80
```

When considering only the questions to which the system got the answer right on the first try we get the above results.

But when we tell the system that the answer should be in the first 10 yielded results we get the following results:



```
    Precision: 0.42
    Mean Reciprocal Rank (MRR): 0.0018941256621512414
    Total Questions Processed: 100.0

Error Analysis:
Number of Correct Answers: 42
Number of Incorrect Answers: 58
```

- **Why do you think the correct questions can be answered by such a simple system?:** Wikipedia pages often have structured and well-formatted information
  The clue captures only the essential information
  Clue and answer are usually straightforward

- **What problems do you observe for the questions answered incorrectly?**
  Questions can be ambiguous

Answers to the questions

- **Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.):** Used StandardAnalyzer for java. It lowercases each token, removes common words and punctuations and uses stop words for the English language.

- **What issues specific to Wikipedia content did you discover, and how did you address them?:** Ambiguity of inconsistent formats, because some pages have a lot of information whereas other don't and weird title.

- **Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset?:** Implemented using Lucene's IndexSearcher to query the indexed data, returning the most similar Wikipedia page title for a given Jeopardy clue.

- **If the latter, what is the best algorithm for selecting the subset of words from the clue?:**
  Queries are built using entire clues, optionally combined with category information for context. Lucene's query parser is used for this purpose.

- **Are you using the category of the question?:** No

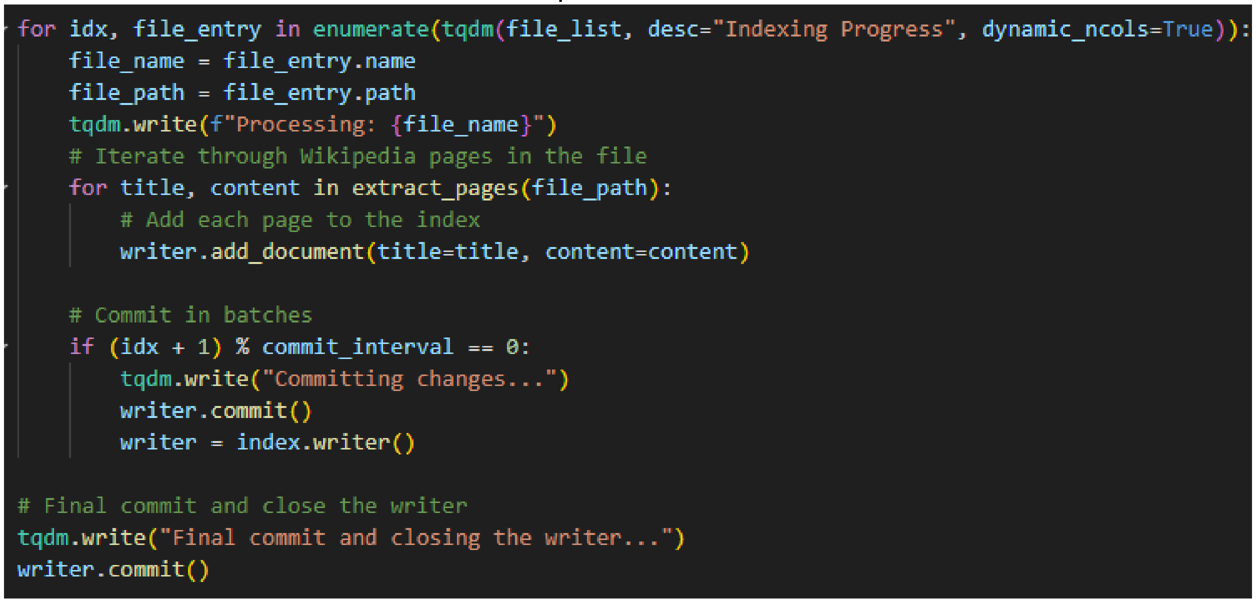
- **Note: not all the above metrics are relevant here! Justify your choice and then report performance using the metric(s) of your choice.:**
  Chose Precision at 1 (P@1) and Mean Reciprocal Rank (MRR) as metrics.

P@1 is relevant for single-answer questions, and MRR accounts for the rank of correct answers. It is very important for a jeopardy show to yield the correct answer in the first try!

Whoosh

Description of the code

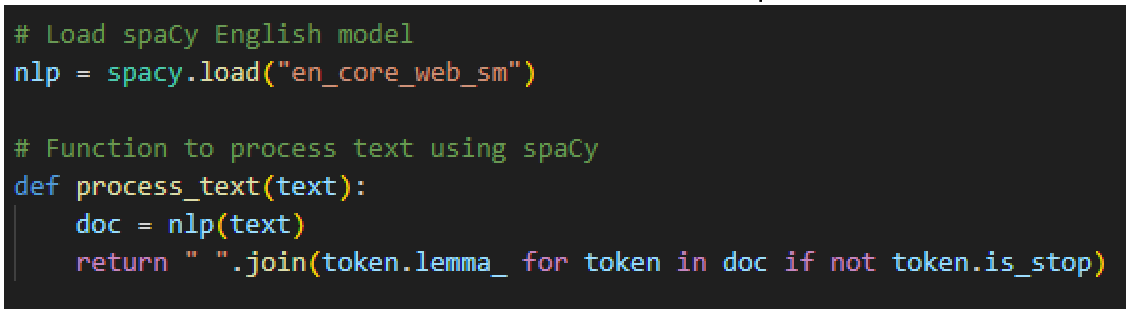
naive_indexer /indexer.py – This script uses Whoosh for indexing the Wikipedia pages. It is called naïve indexer because the whole unprocessed text is used as content in the document.



```python
for idx, file_entry in enumerate(tqdm(file_list, desc="Indexing Progress", dynamic_ncols=True)):
    file_name = file_entry.name
    file_path = file_entry.path
    tqdm.write(f"Processing: {file_name}")
    # Iterate through Wikipedia pages in the file
    for title, content in extract_pages(file_path):
        # Add each page to the index
        writer.add_document(title=title, content=content)

    # Commit in batches
    if (idx + 1) % commit_interval == 0:
        tqdm.write("Committing changes...")
        writer.commit()
        writer = index.writer()

# Final commit and close the writer
tqdm.write("Final commit and closing the writer...")
writer.commit()
```

Here it can be seen that each Wikipedia page appears as a separate document in the index. The extract_pages function (found in the same script) take

Improved_indexer/indexer.py - This script uses Whoosh for indexing the Wikipedia pages. It is called improved indexer because the content of the page is processed before being added as content in the document. Processed = lemmatization + stop words



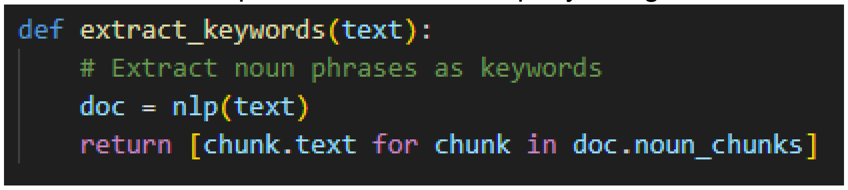
```python
# Load spaCy English model
nlp = spacy.load("en_core_web_sm")

# Function to process text using spaCy
def process_text(text):
    doc = nlp(text)
    return " ".join(token.lemma_ for token in doc if not token.is_stop)
```

Here it can be seen that Spacy was used in order to retrieve the lemma of a token. If the token is a stop word it will be skipped. The rest of the script is identic with the one from the naïve indexer

indexer.py – Run this script with the following arguments: “python ./indexer.py True True” in order to run both the naïve indexer and the improved indexer

measure_raw_clue.py – Script used for measuring the performance of the Jeopardy system. All 3 types of metrics are being used (P@1, NDCG, MRR)

measure_keywords_clue.py - Script used for measuring the performance of the Jeopardy system. All 3 types of metrics are being used (P@1, NDCG, MRR). Compared to the raw_clue measurement script, this one build the query string as a subset of keywords from the clue.



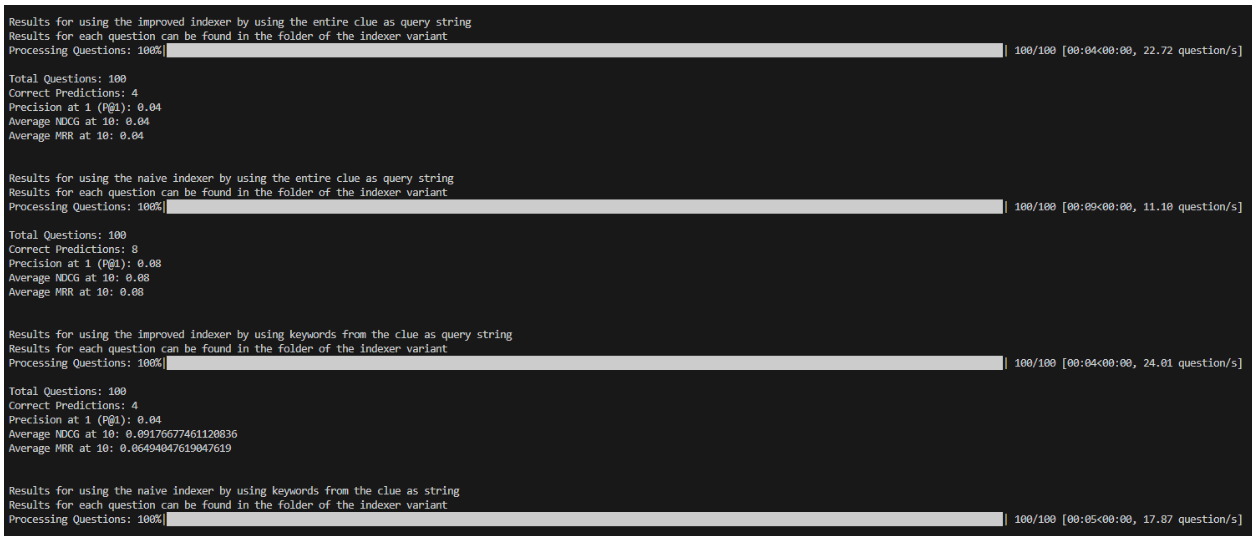
```python
def extract_keywords(text):
    # Extract noun phrases as keywords
    doc = nlp(text)
    return [chunk.text for chunk in doc.noun_chunks]
```

Spacy was also used for this task. It has been observed that using keywords for the query string yields better results

measure.py – Execute this script too run all possible combinations of measurements:

1. Improved indexer – entire clue query string
2. Naïve indexer – full clue query string
3. Improved indexer – keywords clue query string
4. Naïve indexer – keywords clue query string

Results



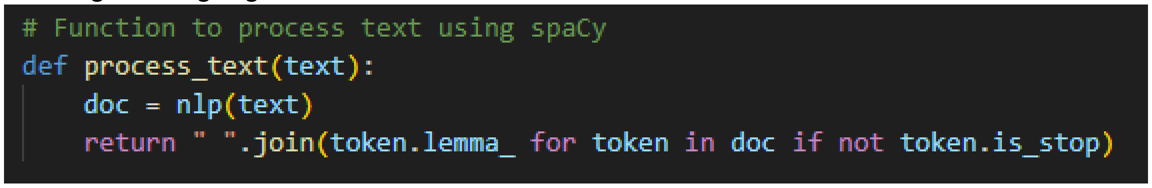
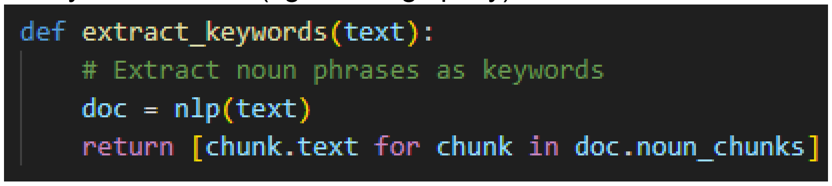
```
Results for using the improved indexer by using the entire clue as query string
Results for each question can be found in the folder of the indexer variant
Processing Questions: 100%|██████████████████████████████████████████████| 100/100 [00:04<00:00, 22.72 question/s]

Total Questions: 100
Correct Predictions: 4
Precision at 1 (P@1): 0.04
Average NDCG at 10: 0.04
Average MRR at 10: 0.04


Results for using the naive indexer by using the entire clue as query string
Results for each question can be found in the folder of the indexer variant
Processing Questions: 100%|██████████████████████████████████████████████| 100/100 [00:09<00:00, 11.10 question/s]

Total Questions: 100
Correct Predictions: 8
Precision at 1 (P@1): 0.08
Average NDCG at 10: 0.08
Average MRR at 10: 0.08


Results for using the improved indexer by using keywords from the clue as query string
Results for each question can be found in the folder of the indexer variant
Processing Questions: 100%|██████████████████████████████████████████████| 100/100 [00:04<00:00, 24.01 question/s]

Total Questions: 100
Correct Predictions: 4
Precision at 1 (P@1): 0.04
Average NDCG at 10: 0.09176677461120836
Average MRR at 10: 0.06494047619047619


Results for using the naive indexer by using keywords from the clue as string
Results for each question can be found in the folder of the indexer variant
Processing Questions: 100%|██████████████████████████████████████████████| 100/100 [00:05<00:00, 17.87 question/s]
```

- **Perform an error analysis of your best system. How many questions were answered correctly/incorrectly?:** Using the naïve indexer, 8-9 questions have been answered correctly. Using the improved indexer, only 4 questions have been answered correctly. It results that lemmatization and eliminating stop words has not helped in this case
- **Why do you think the correct questions can be answered by such a simple system?:** Wikipedia pages often have structured and well-formatted information

The clue captures only the essential information
Clue and answer are usually straightforward

- **What problems do you observe for the questions answered incorrectly?**
Questions can be ambiguous
Were context is relevant system can not perform

Answers to the questions

- **Describe how you prepared the terms for indexing (stemming, lemmatization, stop words, etc.):** Used spacy from Python, which is a Industrial-Strength Natural Language Processing library. It has implementation for lemmatization of a word and stop words for the English language

```python
# Function to process text using spaCy
def process_text(text):
    doc = nlp(text)
    return " ".join(token.lemma_ for token in doc if not token.is_stop)
```
Also tried without processing the text, by creating a document containing the entire content of the page. Surprisingly this yielded better results compared to the processed text.
- **Describe how you built the query from the clue. For example, are you using all the words in the clue or a subset?:** Tried both options, using the entire clue and just a set of keywords from it (again using spacy)

```python
def extract_keywords(text):
    # Extract noun phrases as keywords
    doc = nlp(text)
    return [chunk.text for chunk in doc.noun_chunks]
```
The searches yield better results when the query string is built out of keywords
- **If the latter, what is the best algorithm for selecting the subset of words from the clue?:** Only tried the above solution
- **Are you using the category of the question?:** No, using the category of the question yields worse results
- **Note: not all the above metrics are relevant here! Justify your choice and then report performance using the metric(s) of your choice.:** Tried all the 3 metrics discussed (P@1, NDCG, MRR), but I believe in our case Precision at 1 is the most suitable, because in the context of the Quiz Show Jeopardy, only the first answer matters 😆