<u>1) An overview of the function of the code (i.e., what it does and what it can be used for).</u>
Often while browsing the web, we encounter an entity that we'd like to look up. This involves copy-pasting the word to a search engine and navigating to the article we want, which is often quite low on the search results because it is missing the specific context from the website we were browsing. We need to use the local context of the word/phrase as well as the context of the web page to perform entity linking.

Also, vital to learning about a new entity quickly is the ability to put it in context using relations extracted from public knowledge bases and semantically nearby entities.

My chrome extension performs the following functions

1. Entity Linking using local + webpage context using a combination of neural end2end entity linking and the DBPedia spotlight API.
2. Displaying the local knowledge graph of the entity using WikiData
3. Extracting similar entities using Wikipedia2Vec embeddings

<u>2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.</u>

The majority of the entity linking logic is handled by back_end/ELApp/__init__.py by the worker1() method. I perform entity linking in two ways - one way is by sending an HTTP request to end2end neural entity linker server running on the same machine. As a fallback option, I use the DBPedia spotlight API. After I have the Wikipedia id of an entity, I extract the local knowledge graph of the entity using the wikidata API.

I extract important phrases from the document using the textrank algorithm (also in worker1())

I perform similar entity search using Wikipedia2Vec embeddings in worker2()

Among the front-end files, the majority of the logic is in the file home_handler.js. I extract entities and nearest neighbors using an HTTP request and then populate the relevant HTML elements. I use the viz-graph library to display the local knowledge graph. The file app.js contains a listener crucial to extracting the entity sentence. The file home_handler.js has the logic required to add a context menu entry for our extension. It also has the listener.

<u>3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run the software, whichever is applicable.</u>

Step 0: Clone the directory

```
git clone https://github.com/karthik63/ChromeEntity
cd ChromeEntity
```

Step 1 :
Clone end2end entity linking and Create a virtual environment and install the requirements

```
pip install -r requirements_nel.txt
```

Run code/gerbil.py. Use a manual threshold of -0.4.

```
python -m gerbil.server --training_name=base_att_global
--experiment_name=paper_models  --persons_coreference_merge=True
--all_spans_training=True --entity_extension=extension_entities
--hardcoded_thr -0.4
```

This will start an HTTP server at http://127.0.0.1:5000/

Step 2 :
Create a virtual environment and install the requirements for the back end.

```
pip install -r requirements_back_end.txt
```

Run a second HTTP server using the code in the back_end directory.

```
cd back_end/ELApp
python __init__.py
```

This will start an HTTP server at http://127.0.0.1:5667/

Step 3:
Go to chrome://extensions/ and load `ChromEntity/front_end` as a new extension.
Once the extension has been loaded and the two servers are running, you are good to go! You can right-click and link any entity on any webpage.