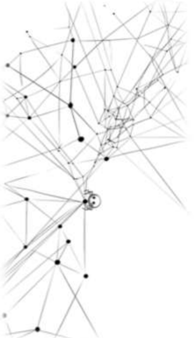
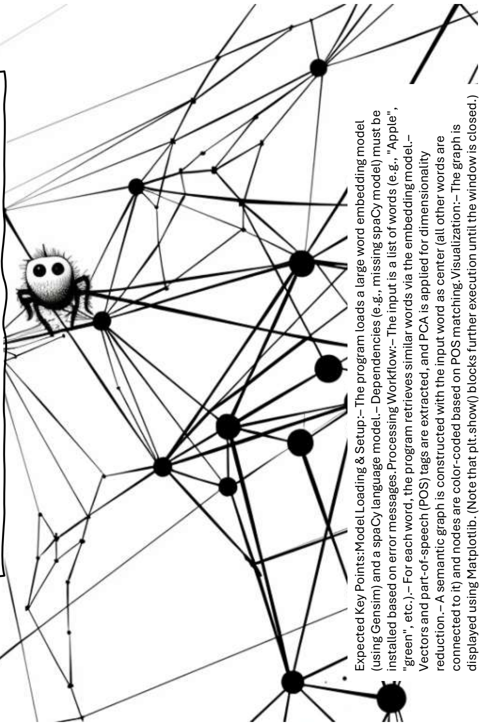


Program code

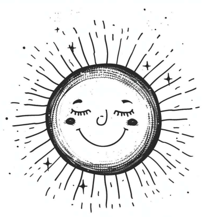
- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behavior. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behavior. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__’:` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It’s okay if you don’t understand everything immediately.
Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle.
Most importantly, have fun investigating the code!

- **Save Output as .png:**
  - Adjust the code so that instead of displaying the plot with plt.show(), the graph is saved as a PNG file (e.g., using plt.savefig("output.png")).
- **Test More Words:**
  - Modify the code to include additional words in the configuration (append extra terms to the input_words_orig list).
- **Analysis of Misspellings:**
  - Explain why, for a word such as **"beautiful"**, misspelled forms (e.g., "beatiful") might appear as similar words.
- **Linguistic Relationships:**
  - Define the terms *Hyperonym*, *Synonym*, *Hyponym*, and *Antonym*.
  - Reflect on how these concepts might relate to the results of the program?
- **Multi-Language Support:**
  - Propose what changes would be needed to run the program for German or Italian.



Expected Key Points:Model Loading & Setup:– The program loads a large word embedding model (using Gensim) and a spaCy language model.– Dependencies (e.g., missing spaCy model) must be installed based on error messages.Processing Workflow:– The input is a list of words (e.g., "Apple", "green", etc.).– For each word, the program retrieves similar words via the embedding model.– Vectors and part-of-speech (POS) tags are extracted, and PCA is applied for dimensionality reduction.– A semantic graph is constructed with the input word as center (all other words are connected to it) and nodes are color-coded based on POS matching.Visualization:– The graph is displayed using Matplotlib. (Note that plt.show() blocks further execution until the window is closed.)

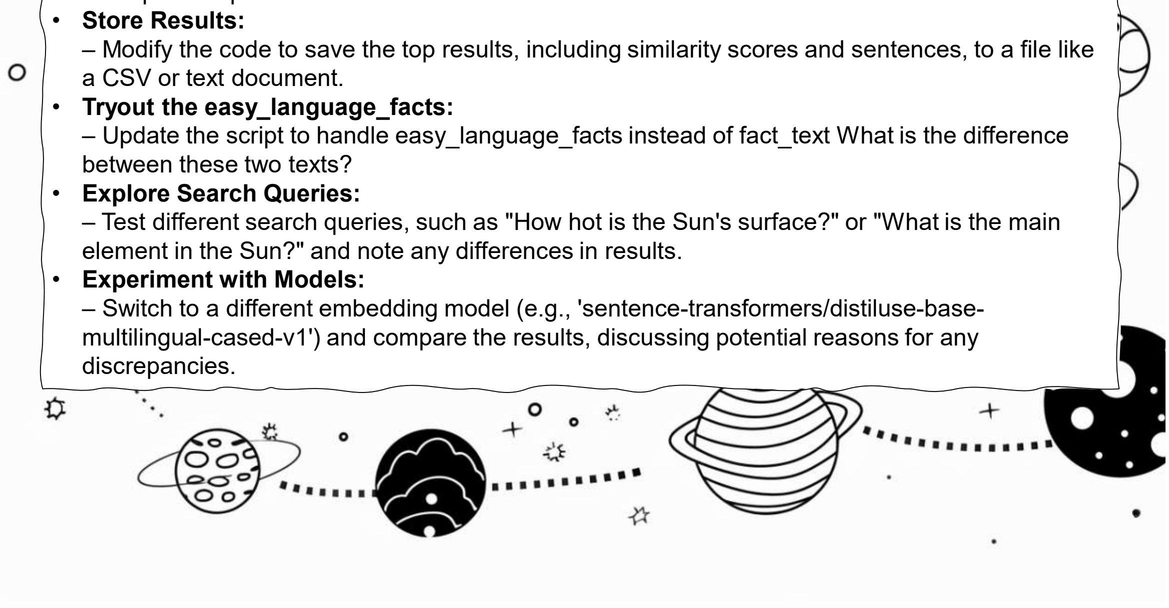


Program code

- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behavior. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behavior. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__':` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It's okay if you don't understand everything immediately.
Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle.
Most importantly, have fun investigating the code!

- **Enhance Functionality:**
  – Adjust the code to return both sentence texts and similarity scores. Test by displaying either the top 1 or top 5 results.
- **Store Results:**
  – Modify the code to save the top results, including similarity scores and sentences, to a file like a CSV or text document.
- **Tryout the easy_language_facts:**
  – Update the script to handle easy_language_facts instead of fact_text What is the difference between these two texts?
- **Explore Search Queries:**
  – Test different search queries, such as "How hot is the Sun's surface?" or "What is the main element in the Sun?" and note any differences in results.
- **Experiment with Models:**
  – Switch to a different embedding model (e.g., 'sentence-transformers/distiluse-base-multilingual-cased-v1') and compare the results, discussing potential reasons for any discrepancies.

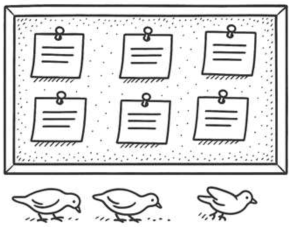


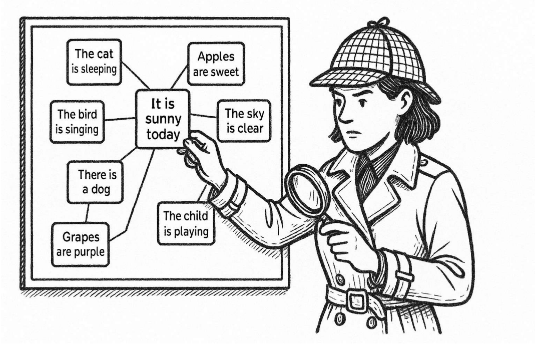
Model Loading: Use SentenceTransformer(model_name) and spacy.load(model_name) to load the embedding and spaCy models.Text Splitting: Call split_text(text, spacy_nlp) to split text into sentences.Generic Retrieval: Call closest_results(query, text, n, embedding_model, spacy_nlp) to get top n similar sentences.Similarity Computation: Cosine similarity is used: cosine_similarity(query_embedding, sentence_embeddings).

- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behaviour. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behaviour. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__':` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It's okay if you don't understand everything immediately. Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle. Most importantly, have fun investigating the code!

- **Vary Model Hyperparameters:**
  – Adjust the Doc2Vec model parameters (e.g., vector_size, epochs, window). Observe how changes in these parameters affect the clustering of documents in the plot.
- **Expand the Corpus:**
  – Add more documents to the corpus in get_corpus(). Analyze whether a larger or more thematic corpus improves clustering quality.
- **Annotation Enhancement:**
  – Instead of showing only document IDs on the plot, modify the code to display a short snippet (for instance, the first 20 characters) of each document along with its ID.
- **Automated Cluster Detection:**
  – Implement a simple clustering algorithm (like K-Means) on the document vectors. Update the plot to color-code documents based on their cluster membership.

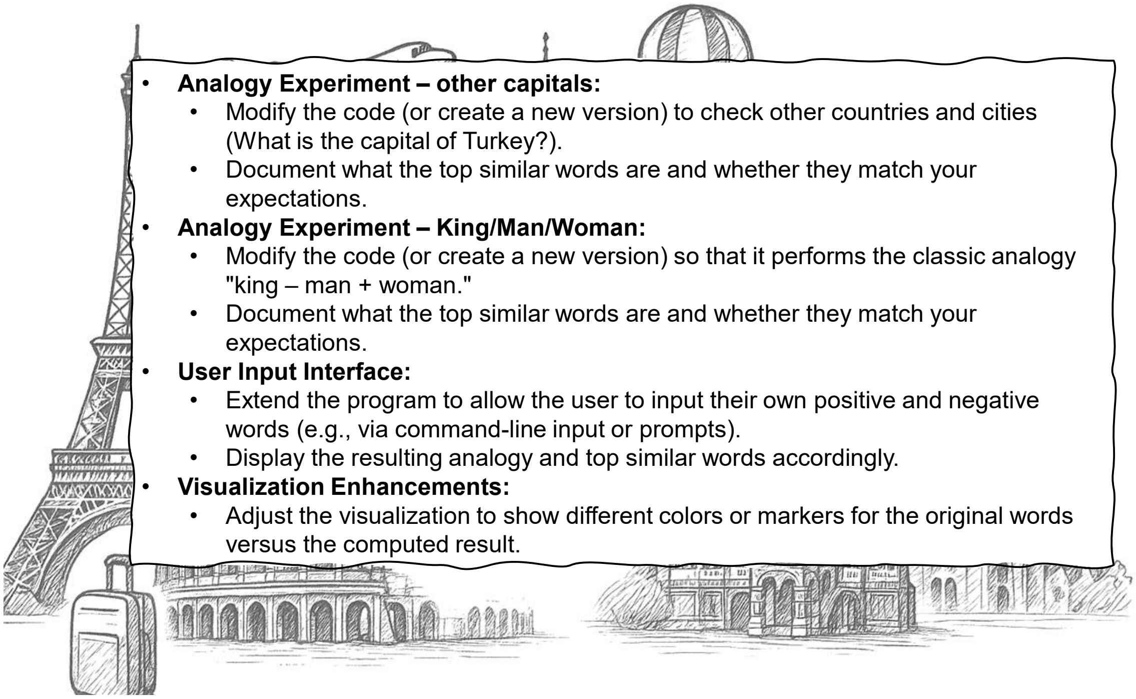




Corpus Preparation: get_corpus() returns document strings; preprocess_corpus() turns them into TaggedDocument objects.Model Training: train_doc2vec() builds and trains a Doc2Vec model on your corpus. Plotting: plot_documents() displays the 2D points with annotations.Legend: print_legend() maps document IDs to full text.

- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behavior. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behavior. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__’:` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It’s okay if you don’t understand everything immediately.
Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle.
Most importantly, have fun investigating the code!



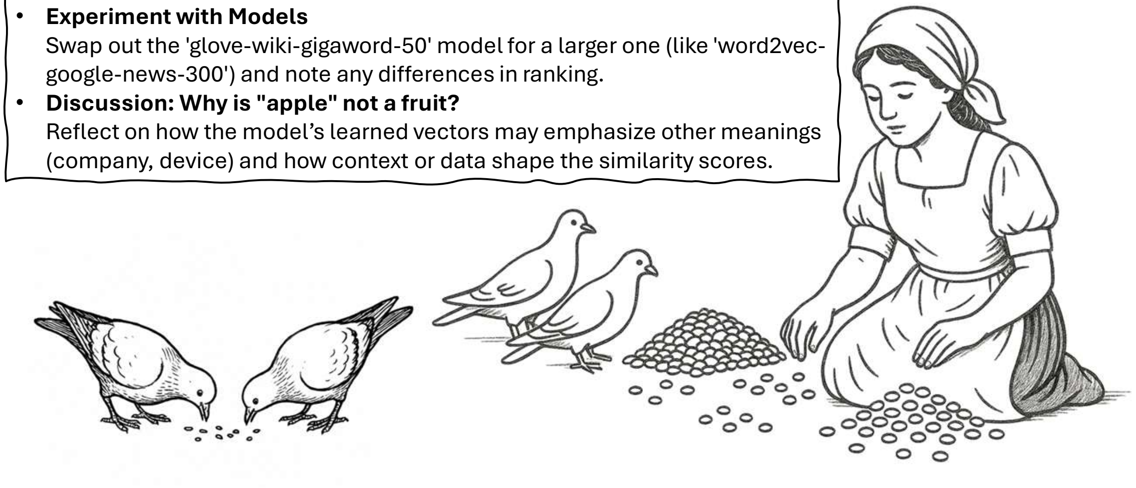
- **Analogy Experiment – other capitals:**
  - Modify the code (or create a new version) to check other countries and cities (What is the capital of Turkey?).
  - Document what the top similar words are and whether they match your expectations.
- **Analogy Experiment – King/Man/Woman:**
  - Modify the code (or create a new version) so that it performs the classic analogy "king – man + woman."
  - Document what the top similar words are and whether they match your expectations.
- **User Input Interface:**
  - Extend the program to allow the user to input their own positive and negative words (e.g., via command-line input or prompts).
  - Display the resulting analogy and top similar words accordingly.
- **Visualization Enhancements:**
  - Adjust the visualization to show different colors or markers for the original words versus the computed result.

This program demonstrates word vector arithmetic using pretrained embeddings. In the demonstration, it performs the analogy: Paris – France + Italy, which should yield a vector close to the embedding of “Rome.” The program then visualizes the embeddings of the selected words and the computed result in a 2D plot using PCA. Key Functions: load_model(model_name): Loads the embedding model.word_arithmetic(model, positive, negative): Computes the resultant vector using vector arithmetic.visualize_words(model, words, result_vector): Projects word embeddings into 2D and plots them.

- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behavior. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behavior. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__’:` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It’s okay if you don’t understand everything immediately.
Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle.
Most importantly, have fun investigating the code!

- **Enhance Functionality**
  Try adding extra target words (e.g., "carrot", "keyboard") with multiple candidate hypernyms, and compare which candidate comes out on top.
- **Store Results**
  Modify the script to save each best hypernym along with its similarity score in a CSV file.
- **Experiment with Models**
  Swap out the 'glove-wiki-gigaword-50' model for a larger one (like 'word2vec-google-news-300') and note any differences in ranking.
- **Discussion: Why is "apple" not a fruit?**
  Reflect on how the model’s learned vectors may emphasize other meanings (company, device) and how context or data shape the similarity scores.

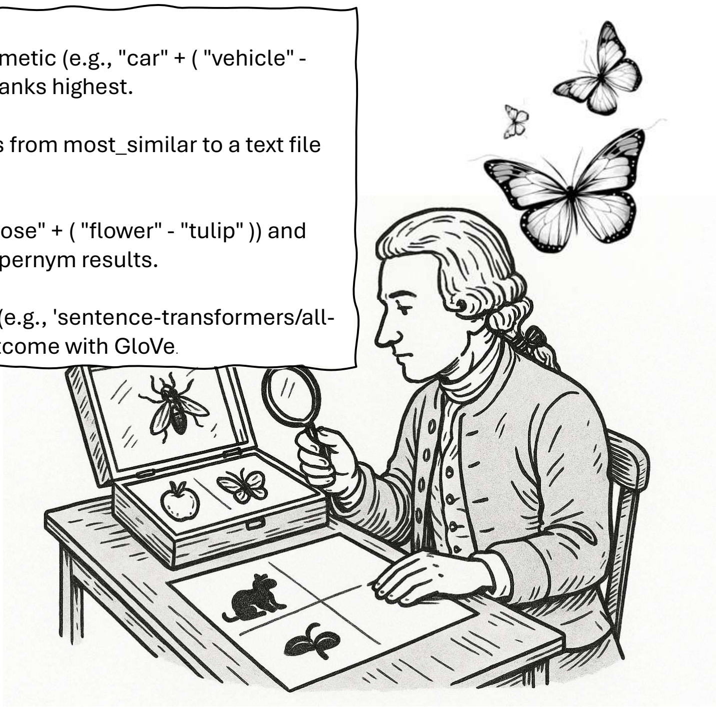


This program loads a pretrained word embedding model (GloVe) to compute cosine similarities between a target word and candidate hypernyms, then visualizes the vectors in 2D using PCA and matplotlib. It uses gensim to download embeddings, numpy for vector operations, scikit-learn’s PCA for dimensionality reduction, and matplotlib for plotting.

- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behavior. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behavior. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__':` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It's okay if you don't understand everything immediately.
Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle.
Most importantly, have fun investigating the code!

- **Enhance Functionality**
  Experiment with different word arithmetic (e.g., "car" + ("vehicle" - "bike")), then see which candidate ranks highest.
- **Store Results**
  Log the top three most similar words from most_similar to a text file alongside their scores.
- **Explore Search Queries**
  Change the query words (e.g., use "rose" + ("flower" - "tulip")) and compare the new best candidate hypernym results.
- **Experiment with Models**
  Test a sentence-transformer model (e.g., 'sentence-transformers/all-MiniLM-L6-v2') and compare the outcome with GloVe.



This program performs word arithmetic (computing tulip + (fruit – apple)) to derive a new vector and then selects the best hypernym candidate from a given list by evaluating cosine similarities. It relies on gensim for the embeddings, numpy for mathematical operations, scikit-learn's PCA for reducing dimensions, and matplotlib for visualization.

- **Download the Code:** Use the repo link (QR code) to save the Python-script to your computer. Ensure you have a working Python environment.
- **Run the Script:** Execute the script and observe its behavior. If errors occur, don't worry—move to the next step.
- **Troubleshoot Errors:** If you face errors like "Module not found," read the messages for clues on missing libraries. Collaborate with your partner to install necessary packages. After fixing issues, rerun the script until it works.
- **Observe the Output:** After successful execution, note the output or visual result. Discuss with your partner what you think the program's purpose is based on its behavior. **(Do not write this down yet—just discuss.)**
- **Read and Understand the Code:** Open the script in a code editor and review it section by section. Work with your partner to interpret each part. If needed, use the ChatGPT assistant (QR code) for explanations on specific lines.
- **Test out methods:** In the section with the heading: `if __name__=='__main__’:` replace the call to `main()` with calls to the methods to find out, what they do.
- **Annotate with Comments:** As you grasp each code portion, add helpful comments to explain major steps, key lines, and how different parts connect. Use your own words to make the code clear for others who haven't seen it before. ☛ Active Reading

Keep in mind: This is an open exploration. It's okay if you don't understand everything immediately.
Use your programming knowledge, teamwork, and the ChatGPT assistant to piece together the puzzle.
Most importantly, have fun investigating the code!

- **Enhance Functionality**
  Add or remove items in the candidate_list or target_words (for example, include "rose" or "bow") to see how it changes the final assignments.
- **Store Results**
  Write each word's assigned hypernym and similarity score to a CSV for record-keeping.
- **Experiment with the Relation Vector**
  Swap the reference pair (e.g., "cat" - "kitten" or "bird" - "sparrow") to see if it classifies words differently.
- **Discussion: Why do some words seem to be in the wrong category (especially flowers like "lily")?**
  Consider the limitations of the "relation vector" approach, how embeddings are learned, and the overlap between semantic groups (e.g., "lily" might share vector features with other categories).

This program assigns a hypernym to each target word by adding a relation vector (derived from dog – poodle) to its word vector and then choosing the candidate with the highest cosine similarity, finally visualizing the assignments as a directed graph using Graphviz. It uses gensim for loading word embeddings, numpy for computing vector similarities, and Graphviz to render the hypernym assignment graph.