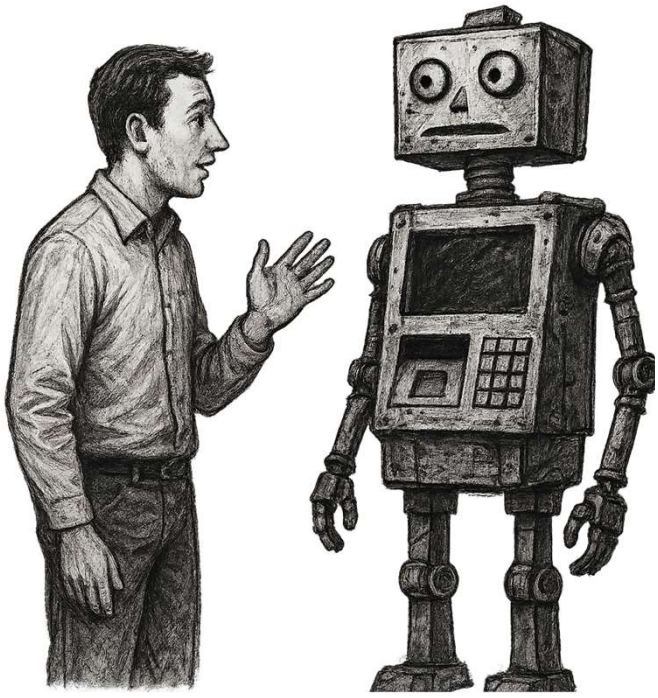
generateText.py

Self-Exploration (≈ 5 min)

1. Skim the code and add a one-sentence docstring to each function describing its purpose, parameters, and return value.
2. Run the code.
3. Try out different prompts.

Deepening Tasks

1. Modify the parameter temperature and set it to 0.0 and after that to 2.0.
2. Explain what the input_len-Variable is doing and what output[0][input_len:] does.



bechdelPipeline.py

Self-Exploration (≈ 5 min)

1. Skim the code and add a one-sentence docstring to each function describing its purpose, parameters, and return value.
2. Locate where the pipeline steps are defined. Add an inline comment to each step (e.g., “load file,” “prompt call,” “extract metadata”).
3. Run the code and check the result.

Deepening Tasks

1. Explain in which ways the concept ‚pipeline‘ is applied in this code.
2. What is the advandtage of importing the method generateText from a different module? Wouldn‘t it be easier, to just copy the method into SimpleBechdel?
3. Modify the prompt so it also asks for a third label, “Ambiguous.” Run the script and verify that a new column appears in ratings.csv.
4. Explain in two sentences why extracting script name and style via regex is preferable to hard-coding them.
5. Do you have any idea how to improve the quality of the result?

bechdelChoices.py

Self-Exploration (≈ 5 min)

1. Find the StatementSet class and write a docstring that explains how the three statements (a/b/c) are constructed.
2. Identify the code mapping responses a→–1, b→0, c→+1. Add a comment explaining how this mapping works.
3. Start the script. Read the console output. If the script runs to long and you think you understand, what it does, stop it after some minutes.

Deepening Tasks

1. Either extend the mapping to support two new responses (e.g., d→+2, e→–2) **OR** add a new style of your choice to filmSzene.py (see below). Test your changes with a mock prompt or by running the modified script.
2. Compare numeric scoring versus label-only output. In a short paragraph, assess which seems more informative and why.

BechdelReasoning.py

Self-Exploration (≈ 5 min)

1. Add a one-sentence docstring to each of the three reasoning steps (Info-Extraction, Statement-Generation, Final-Label) explaining expected input and output.
2. Open up generateText_with_messages.py. What is the difference to generateText?
3. Highlight each generate_text_with_messages call and note the message sequence (system, user, assistant).
4. Run the script and interpret, what you read in the console.

Deepening Tasks

1. What is reasoning? How many steps does this reasoning pipeline have? How is it applied here?
2. After Step 2, insert a simple consistency check: compare the generated statements for contradictions, and write an inconsistent flag to the CSV.
3. Run the script on one sample dialogue (change the folder to look for text and put some text there), then run SimpleBechdel3 and SimpleBechdel4 on the same dialogue. In a few sentences, compare the reliability and expressiveness of the three approaches.

filmSzene.py

Self-Exploration (≈ 5 min)

1. Write a docstring for load_scripts and another for save_script summarizing each function's input, output, and purpose.
2. Comment the loop that iterates over styles and descriptions; explain how filenames are constructed.
3. Run with a selection of the styles. (You can simply ‚comment out' other styles in the list styles). Use only ONE of the scripts in the folder data/scripts (remove all the others for the experiment).

Deepening Tasks

1. Either add a new style of your choice **OR** write a new short dialogue script and put it in folder data/scrips.
2. Run to produce new dialogues.
3. Run your own generated scene through SimpleBechdel3 or SimpleBechdel4 to produce a ratings CSV.
4. Execute your chosen option, then discuss in pairs: Which pipeline produced the clearest results? Were any outputs surprising or ambiguous?

agentDialog.py

Self-Exploration (≈ 5 min)

1. Add a docstring at the start of the agent loop describing which topics the agent tracks and how it selects the next utterance.
2. Mark the code where the agent's internal state is updated and add a brief comment on what happens there.
3. Run and look into the console to understand what is happening. Stop when you do.

Deepening Tasks

1. What is an agent? How does this concept differ from a pipeline?
2. Implement a "mood" attribute (e.g., positive/neutral/negative) that the agent updates each turn and influences its next choice.
3. In a short paragraph, contrast the agent-based design (stateful, autonomous) with the pipeline design (linear, stepwise).