gitextract.py and diffCompare.py

Code is just Text

- **Set up the environment**
    - pip install pandas gitpython python-Levenshtein (difflib and itertools are in the standard library).
- **Generate version snapshots & metadata**
    - Run gitextract.py (adjust repo_path, file_path, output_folder in the header if required).
    - Confirm that:
        - multiple *.py_<commit>.py snapshots now live in ./data/, and
        - ./data/commit_data.csv holds the true commit dates/times.
- **Infer version order with the heuristic script**
    - Execute diffCompare.py.
    - Note the “probable order (old → new)” it prints and the average similarity values for each file.
- **Annotate the code**
    - Open **both** scripts and insert concise English comments explaining every non-trivial block:
        - file loading, similarity calculation, greedy TSP-style ordering, etc. in diffCompare.py;
        - repository traversal, blob extraction, DataFrame creation in gitextract.py.

- **Research the diff algorithm**
    - Look up how difflib.SequenceMatcher computes its .ratio() (hint: an O(ND) “gestalt pattern matching” algorithm by Ratcliff-Obershelp).
    - Summarise the core idea and its time/space complexity directly in the script as comments.
- **Validate the guessed order**
    - Compare the sequence produced in step 3 with the chronological order in commit_data.csv.
    - Record mismatches and speculate why they occur (e.g., two edits with very similar text lengths but reversed timestamps).

- **Critique & extend the approach**
    - Briefly discuss—either in a separate report.md or as end-of-file comments—
        - **Collaboration issues**: branching, multiple authors, interleaved edits, non-textual changes, large refactors.
        - **Alternative metric**: swap the similarity measure for *Levenshtein distance* (python-Levenshtein provides it), rerun the ordering, and note whether the sequence improves or degrades.