Day 3: Changelog and Versioning

CHANGELOG.md

Definition: A CHANGELOG.md file **tracks all changes made to a project**, including new features, fixes, and updates. It's essential for documenting project history and helps collaborators or users understand what's new.

Steps to Create a CHANGELOG.md File:

!! SKIP STEP 1 to STEP 3 if you have already run these and have created a new branch for day 3 !!

1. Navigate to Your Project Repository:

```
cd <path to your repository>
```

2. Ensure you're up to date with the main branch:

```
git pull origin main
```

3. Create a New Branch for Your Work. Naming your branch with the day or feature being worked on is a common practice.

```
git checkout -b day-3-content
```

!! SKIP STEP 1 to STEP 3 if you have already run these and have created a new branch for day 3 !!

---

4. In your project directory, create a markdown file named CHANGELOG.md:

```
touch CHANGELOG.md
```

5. Add Initial Text to CHANGELOG.md. Use markdown syntax *(see below)* to add the changelog format. Start with the title and an initial version entry.

```
# Changelog
## [0.1.0] - YYYY-MM-DD
### Initial Version
- Added core functionalities covered in Day 3, including
lists, tuples, dictionaries, and sets.
- Implemented changelog and version control instructions.
```

6. Stage, Commit, and Push the Changes:
   a. Stage the new changelog file:

```
git add CHANGELOG.md
```

   b. Commit your changes with a message:

```
git commit -m "Add CHANGELOG.md with initial version
entry"
```

   c. Push your branch to the remote repository:

```
git push origin day-3-content
```

Markdown Basics for Changelog Formatting:

Markdown is a lightweight language for formatting text that **renders nicely in GitHub and many other platforms**:

- **Headers** are created with **#** symbols. More **#** symbols mean smaller headers (e.g., **##** for version numbers).
- **Bulleted Lists** use **-** or *****.
- **Example Entry**:

```
# Changelog
```

```
## [1.0.1] - 2024-10-26
### Fixed
- Corrected typos in the set operations documentation.
```

SEMANTIC VERSIONING

Semantic versioning is a system for labelling versions using the format:
MAJOR.MINOR.PATCH

- Major
  - Increase when making significant changes that break backward compatibility
  - e.g. `v1.0.0` to `v2.0.0`

- Minor
  - Increase when adding new features that are backward compatible
  - e.g. `v1.0.0` to `v1.1.0`

- Patch/Bug/Bump
  - Increase when making small fixes or patches that don't change existing functionality
  - e.g. `v1.0.0` to `v1.0.1`