Step 1. Research and Problem Selection:

- **Identify a Problem:** Choose a real-world problem that interests you. This could be anything from environmental issues to everyday inconveniences.
    - Use the **PCC Library tools** to explore topic ideas and find credible academic resources. Feel free to follow up with your favorite search engine after discovering topics.
- **Explore Existing Solutions:** Investigate current solutions to your chosen problem. What software solutions exist, and how effective are they?
- **Justify Your Choice:** Explain why this problem interests you and why it's important to find a solution.

Step 2. Design a Solution:

- **Propose a Software Solution:** Create your own program idea! Outline a basic software solution that addresses the problem you are researching. Describe how your idea improves on or differs from existing solutions, feel free to build your ideas based on software that already exists.
- **Pseudocode:** Write pseudocode or code-like statements that outline the logic of the critical behavior of your software. Don't worry if you don't know exactly how it might be done in a programming language, stay general and think about the steps needed to complete the task. This helps visualize the programming logic without getting into syntax specifics.
    - **Note:** Actual coding is not required. Please focus on the conceptual design rather than specific syntax.
- **User Interaction:** Visualize how users would interact with your software. Describe user interactions or create mock-ups of the user interface. Your design should be informative and engaging.

Presentation:

Prepare a Presentation: Use PowerPoint, Google Slides, or another tool to create a presentation document that summarizes your research, proposed solution, and design documentation. Be creative and have fun with your project!

Structure Your Presentation:

- **Introduction to the Topic:** Explain your chosen topic and why it interests you.
- **Summary of Findings:** Discuss the history, current software solutions, and future potential of the problem you are researching. Teach us what you've discovered during your research in Step 1.
- **Real-World Applications:** Discuss how the concepts can be applied in real-world scenarios.
- **Describe Your Design Approach:** Even though no actual software is being developed, describe your planning process while thinking through the potential software design.
- **Solution Design Proposal:** Share the elements of your proposed software design in Step 2.
- **Open Questions:** Highlight at least two unanswered questions from your research. What surprised you? What do you want to know more about?
- **Citations:** List all resources you used, such as articles, journals, and websites.

Important Tips:

- **Start Early:** Give yourself plenty of time to research, design, and refine your project.
- **Stay Curious:** Always be open to discovering new tools, techniques, and perspectives. Use your imagination, don't limit yourself. Describe your process, your uncertainties, and your discoveries.

- **Exploration Encouraged:** While actual coding is not required, feel free to explore the possibilities of how your software might be developed in Python. Consider creating a short text based mock-up of your program design idea.

Learning Objectives:

- Understand the application of programming and problem solving to real-world issues.
- Develop skills in researching technology solutions and planning software projects.
- Articulate and visualize software solutions through descriptions, flowcharts, pseudocode, and user-facing design.

Submission Guidelines:

- **Final Presentation:** Upload your presentation document as a .PDF to D2L assignment folder
- **Design Documents:** Include all research notes, design documents (flowcharts, pseudocode, user-facing design, code if any), and anything else developed while creating this final project.
- **GitHub:** Include your presentation document and all related design documents as a project in your GitHub repository. Organize the files and name them clearly. Include a link to your GitHub project in the comments of your submission.

Problem: Spotify's current music shuffle program is actually so bad and specifically prioritizes songs you listen to the most, which then causes an endless cycle of that one handful of songs playing on repeat. The only existing solution I've found is to specifically go through your playlist and start playing songs you don't listen to as much to throw off the algorithm. I think this is arbitrary and could be simply solved with an improved shuffling algorithm. This problem interests me because I am always listening to music throughout the day and I have a massive playlist so a very big chunk of my music gets pushed aside by Spotify's defunct algorithm

https://lifehacker.com/the-reason-spotify-shuffles-aren-t-really-random-and-h-1849756947

https://community.spotify.com/t5/iOS-iPhone-iPad/Shuffle-play-is-not-random/td-p/750619

https://www.macobserver.com/news/spotify-shuffle-isnt-always-random-heres-how-to-truly-shuffle-your-albums-and-playlists/

https://www.makeuseof.com/how-spotify-shuffle-works/#:~:text=The%20major%20difference%20is%20that,same%20artist%20are%20spread%20out.

https://medium.com/immensity/how-spotifys-shuffle-algorithm-works-19e963e75171

Design a Solution: My solution will provide three options for a user's shuffle. One will be the way they have it now, prioritizing songs you listen to the most. The second will be the opposite, prioritizing songs you don't listen to much, giving more light to the songs the first option overshadows. Then the third option will be a fully randomized shuffle with no specific algorithm.