Full Problem Solving Checklist

1. Understanding the problem	
a) Identify what you don't know	Read the description carefully and highlight terms you are not familiar with, keep a list of questions to ask the instructor or research later
b) Research	Look up terms, ask for clarification, answer any questions defined in part 1.a
c) Consolidation	Restate the problem in your own words
d) Deepening understanding	List potential user stories and possible inputs to the program, write down the expected output in general terms

2. Planning	
a) Task decomposition	Begin by making a list of all the components you will need to solve the problem. For each component, further break down the task into subtasks (tree structure) so that each subtask does only one simple thing (i.e. it could be contained in a single function)
b) Assess your knowledge	Write a list of problems you have encountered before that might help you with certain subtasks. Do you know the best tools (languages, libraries etc.) to use for each subtask?
c) Research	Research the best tools to use for each subtask based on any knowledge gaps that you identified in part 2.b. For each subtask, try to find example solutions or pseudocode.
d) Write pseudocode	Based on your research, write a plan of how each subtask will be implemented.
e) Make time estimates	Consider your prior experience with the subtask, whether new tools have to be installed, and the complexity of your pseudocode to assign time estimations to each of your subtasks.

3. Implementation	
a) Tools	Install any required tools (languages, software, libraries) and ensure versions are correct, run tests to validate correct installation
b) Inputs/Data	If the task requires input data, where will you gather it from? Was data provided to you? Does it have to be cleaned? Gather and clean your input data as required.
c) Environment	Build your base folder structure and skeleton code, get a simple program to compile and run using your skeleton environment and input data.
d) Code task-by-task	Now we are ready to begin writing code for each sub-task identified in part 2.a. Test each sub-task as you go (achieving expected outputs for a list of given inputs).

4. Evaluation	
a) Brainstorm test cases	Write down a list of potential test cases; think about possible inputs, especially edge cases. Create or reuse a testing library based on your list.
b) Test cases	Implement test cases and make notes of anything that needs to be fixed. Re-enter phase 3 and come back to this step as needed.
c) Verification	Look back at the problem description and the user stories you identified in part 1.a. Have you met all requirements and satisfied the assignment as a whole?
d) Reflection	Reflect on your learning. What was the biggest take-away? What are you proud of? What skills will you use again in the future?