General Process and Strategies for cs124/160 Assignments and Computational Problem Solving
(Following the Principles in How to Solve It, by G. Polya)

Understand the problem
- study and understand the given information: be able to explain the given info., examples …
- develop a clear big picture of the assignment and overall program structure (classes, methods, …)

Devise a plan -- develop a correct and efficient computational/algorithmic solution
- break a big problems into smaller, more manageable ones
  - at the assignment level: largely done already, reflected in given class APIs (study/reflect)
  - for individual methods: your job
- solution development: focus on individual public methods, one at a time, and design private helper methods if needed. For each method,
  - be very clear about input, output, required functionality and efficiency
  - develop algorithmic solutions at different levels of detail: be able to clearly articulate and justify big ideas, intermediate refinement, and eventually all details needed for (on-paper) analysis, testing and debugging, as well as coding
    - break bigger sub-problems into smaller problems
    - refine rough ideas and figure out all details
    - identify properties needed for computation
    - use small examples, related past problems, existing design patterns, your problem solving strategies to gain insights, get new (useful) ideas, make progresses
    - study textbook (again) for related concepts and examples
    - select/design suitable data structures to support your computation
    - …
- analyze, test and debug your algorithms for correctness and efficiency
  - be critical of your own solution: the goal is to identify errors and inefficiency of your own solution and fix them yourself
  - do thorough analysis, comprehensive testing using cases with different properties
  - debug your algorithms when there are errors
    - be very clear about what are expected right behaviors, variable values …
    - figure out what’s really happening within your own algorithms, e.g. by tracing (line by line execution of your pseudo-code for general cases, specific cases …)

Carry out the plan – implement your algorithms in code, along with more analysis, testing & debugging
- coding is generally straightforward after you have developed detailed algorithms
- coding should be done in a systematic and disciplined process ➔ clean, profession code
  - focus on one class, one method at a time
  - do thorough analysis, testing and debugging of one unit before starting the next unit
  - follow style guidelines along the way

Reflect, review and extend
- look back at what you have done, what worked and what didn't
- think about alternative approaches (their pros and cons), related problems and adapted algorithms
- summarize learned lessons and successful strategies -- useful for solving future problems

Computation problem solving is an iterative process. Errors identified in a later stage may bring us back to earlier stages and require us to revise and improve our understanding and solution, which is an important part of process and learning. With computation thinking and problem solving, you learn it by doing it.