Some Problem-Solving Tips

CS 211 is a part of the total curriculum offered by the College of Engineering. If there is one thing that engineers are supposed to do, that's solve problems. If it were easy, they probably wouldn't require college educations. Below is a list of some tips with explanations that will help you solve problems effectively.

- Always have a plan
  - *"In preparing for battle I have always found that plans are useless, but planning is indispensable."* —Dwight D. Eisenhower
  - The plan will likely change as you solve the problem, but having a plan is still key
  - Without a plan, you are hoping to get lucky
  - Planning allows you to set immediate goals and achieve them; otherwise your goal is solve the whole problem at once.
- Restate the problem
  - Restating the problem can lead to new and valuable insights
  - Restatement can also be used to demonstrate your understanding of the problem
  - This should be the first step in any plan
- Divide the problem
  - Dividing the problem into smaller pieces can significantly reduce the amount of work
  - Sometimes the division can hide itself
- Start with what you know
  - A working partial solution can spark ideas about the remainder
  - Build momentum and confidence while making meaningful progress
- Reduce the problem
  - For example, remove some constraints, or use a smaller set of data
  - Solving a simpler version of the problem makes meaningful progress towards the final solution
  - Allows meaningful discussion when trying to solve a problem (Instead of "It doesn't work")
- Look for analogies
  - If you've solved a problem before and encounter a similar problem, most of the work is done

  - Analogies may not always be complete, but partial analogies still solve part of your problem
  - In this course, this particular strategy may not work often because you are still building your own store of solved problems to draw from
    * Copy/pasting or modifying existing code at this stage of your learning is dangerous for this very reason.
    * You need to be able to write AND internalize your own code; this is how you build a store of solutions to make analogies from
    * Relying on the code of others now means you will rely on the code of others in the future

- Experiment
  - **NOT** the same as guessing
  - Experimenting is controlled, and allows you to learn
    * Testing a library
    * Making small changes and observing if output changes
    * Scratch coding that deepens your understanding

- **Don't get frustrated**
  - Frustration is a vicious cycle
    * You won't think as clearly
    * You won't work as effectively
    * Everything will take longer and seem harder
    * Frustration feeds frustration
  - ALLOWING yourself to get frustrated is giving yourself an excuse to continue to fail
  - Have a plan, and tweak/change it when necessary
  - If choosing between getting frustrated or taking a break, take the break
    * Put the problem completley outside of your mind while you do something to get your blood flowing