Steps of the software development process:

1. Based on a given starting point (feature, task, code block, etc.), what is the expected end result?
2. What are the written-out steps to go from point A to point B? You need to solve the problem before you begin coding it.
3. Implementation (coding it out, researching)
4. Test and debug code (run code with breakpoint, unit test, etc.)
5. Refactor if necessary. Test again. This continues until functionality is solidified.

The above steps should be rinse and repeated for every single problem you encounter. Ignoring these steps or straying from them will result in the long way around to solving a problem or even possibly never solving the problem.

To be a good problem solver, it is important to be able to break problems down. One way to go about this is to write out the steps it will take to solve the problem. These steps are written down in English in a manner that are easily explainable to someone who may not be technical. The idea is that in order to code something out, you first need to have a good understanding of what it is you are attempting to solve. For each of the problems below, write out the steps it will take to go about solving the problem. Then code it out and test!

You may jump around in these problems. If you get stuck on one problem, begin working on another. If you get stuck on that new problem, go back to working on the previous one.

The use cases below are just examples to give you a better idea of what might be passed into the method or what might be outputted from the method. You shouldn’t be coding exactly to these examples, but rather, be flexible to handle any data of that data type.

Whiteboard Challenges

1. Given an array of integers, return **indices** of the two numbers such that they add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice.
   1. Use Case:
      1. Given numbers in an array: [5, 17, 77, 50]
      2. Target: 55
2. Palindrome is a word, phrase, or sequence that reads the same backward as forward i.e. madam. Write code that takes a user input and checks to see if it is a Palindrome and reports the result. You must handle spaces.
3. Given a number, return the reciprocal of the reverse of the original number, as a double.
   1. Use case: If given 17, return 0.01408 (1/71)
4. A briefcase has a four-digit rolling-lock. Each digit is a number from 0-9 that can be rolled either forwards or backwards. Write a method that returns the smallest number of turns it takes to transform the lock from current combination to the target combination. One turn is equivalent to rolling a number forwards or backwards by one.
   1. Use case:
      1. Current lock: 3893
      2. Target lock: 5296
5. Given a list of integers, return a bool that represents whether or not all integers in the list can form a sequence of incrementing integers
   1. Use case:
      1. {5, 7, 3, 8, 6} 🡪 false (no 4 to complete the sequence)
      2. {17, 15, 20, 19, 21, 16, 18} 🡪 true
6. Create a method that takes an array of positive and negative numbers. Return an array where the first element is the count of the positive numbers and the second element is the sum of negative numbers.
   1. Use case: [7, 9, -3, -32, 107, -1, 36, 95, -14, -99, 21]
7. Create a method that accepts a string of space separated numbers and returns the highest and lowest number as a string
   1. Use case: “3 9 0 1 4 8 10 2” 🡪 “0 10”
8. Happy Numbers
   1. A happy number is a number defined by the following process: starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1. An example of a happy number is 19
9. Create a method that accepts a string, check if it’s a valid email address and returns either true or false depending on the valuation. Think about what is necessary to have a valid email address.
   1. Use case:
      1. “mike1@gmail.com” 🡪 true
      2. “gmail.com” 🡪 false
10. Create a method that takes in a string and replaces each letter with its appropriate position in the alphabet and returns the string
    1. Use case:
       1. “abc” 🡪 “1 2 3”
       2. “coding is fun” 🡪 “3 15 4 9 14 7 9 19 6 21 14”