JavaScript Algorithms and Data Structures Masterclass

# Section 4-5: Problem Solving Approach and Patterns

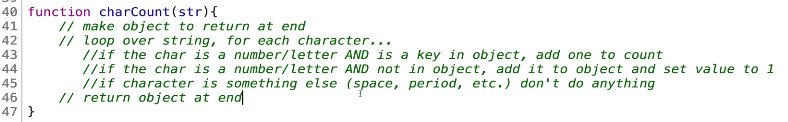
## Lessons

* Objectives
  + Define Algorithm
  + Devise Plan to Solve
  + Compare the problem solving patterns

### Intro

* Algorithm
  + Process/steps to accomplish a task

### Steps

1. Understand the Problem
   1. **Questions**:
      1. Restate/summarize the problem?
         1. ex. Adding 2 nums together
      2. What are the inputs and outputs to the problem?
         1. ex. input = integers, floats
         2. ex. outputs = integer
      3. Can outputs be produced by the inputs?
         1. ex. Ask
      4. How should I label the important pieces of data in the problem?
2. Concrete Examples
   1. Steps:
      1. Create simple examples (USE CASES) with inputs and outputs
      2. Explore Complex Cases
      3. Explore EDGE CASES
      4. Explore INVALID or Empty CASES
3. Breakdown the Problem into Steps
   1. How do you communicate?
   2. Write out the steps you will be taking
      1. Force you to catch conceptual issues or misunderstanding and focuses the problem
   3. Example: *Count the number of times characters repeat in a string*
      1. 
      2. Focusing on the PROCESS
4. Solve or Simplify
   1. Simplify
      1. Find the core difficulty in what you are trying to solve
      2. Temp ignore the difficulty
      3. Write a simplified solution
         1. Try to solve the parts you do understand before tackling parts you don’t
      4. Incorporate difficulty back in
5. Look back and Refactor
   1. Does your code return the right thing?
   2. Any different Approaches?
   3. Can you understand the code at a glance
   4. Can you use this method or result to solve other similar problems
   5. Can you improve the performance
   6. How have others solved this problem?

## Problem Solving Patterns

* **Patterns**:
  + Frequency Counters
    - This uses objects or sets to collect values/frequencies of values
    - Often uses nested loops O(n2) with arrays and strings
      * **ex**. Counting # of times a value appears in an array
  + Multiple Pointers
    - Create pointers/values that correspond to a spot/index and move towards beginning, end or middle based on condition
      * **ex**. find number of unique values in an array
  + Sliding Windows
    - Create a “window” (ex. Single value or Sub-array) from 1 position to another
      * Based on conditions, the windows inc or dec. Then a new window is created
        + **ex**. Find the maxSum of 2 adjacent values in an array
  + Divide and Conquer
    - Divide a data set into smaller chunks, then repeat the process on the individual subsets
      * **ex**. Binary Sort (cut array in half, then decide if it is above or below the middle value) then repeat
        + The array must already be sorted though