4. Algorithmic thinking

- **Algorithm in programming**:
    - In our previous class on functions and conditions, we have discussed that mastering to write a JavaScript instruction/function involves chopping down an instruction to very small components and describing it.
    - We call the step-by-step instructions that tell a computer to solve a problem algorithm. This where algorithms come. There are many common examples of algorithms, from sorting sets of numbers to finding routes through maps to displaying information on a screen. A recipe is a good example of an algorithm because it says what must be done, step by step. It takes inputs (ingredients) and produces an output (the completed dish).
- **Steps to follow when solving algorithmic problem**
    - **1. Understand the problem statement**
        - You can't solve a problem you don't understand.
        - Read the problem as many times as possible until you understand every word used within the problem
        - Use your own words to describe the problem.
        - Try explaining the problem to a friend and see if their understanding of your explanation matches the problem
    - **2. Hand-write the solution or solve it on paper first**
        - Take out a piece of paper and work through the problem manually. If needed, make a drawing depicting the problem
        - Use all the important information available in the problem
        - Use example cases/sample data to make sure that you can solve the problem on paper
        - NOTE: The most difficult part for newcomers is not actually solving the problem. The difficulty is in understanding and identifying the smallest steps the brain takes to solve the problem. Our brain is sometimes too fast for us to comprehend
    - **3. Craft a general step that can be followed to solve any kind of similar problem**
        - This is where you would consider scenarios that might not have been covered in the examples you solved above and tweak the solution accordingly
        - Have you solved any similar problem? If so, take advantage of that experience and its information
        - See if you can reduce any steps or if you are repeating any steps

- Eliminate redundant or unnecessary data

- **4. Write the pseudo code line by line**
  - Writing out pseudocode that you can translate into code will help with defining the structure of your code and make coding a lot easier
  - Pseudocode is a simpler version of a programming code in plain English which uses short phrases to write code for a program before it is implemented in a specific programming language.
  - Simplify the problem by dividing it into simpler cases and pick what JavaScript functionality you need to use to cover that case
  - Pseudocode generally does not actually have specific rules but sometimes, you might end up including some syntax from JavaScript. Don't get caught up with the syntax. Focus on the logic and steps.
  - **Look at these two pseudocode examples for a function that returns only even numbers from an array**:
    - Example 1, pseudocode that has more words

      function selectEvenNumbers

      create a variable with an empty array called evenNumbers to store the selected even numbers from the original array

      for each element in that array

      see if that element is even

      if element is even (if the remainder is 0 when divided by 2), add/push that to the array evenNumbers

      return evenNumbers

    - Example 2, pseudocode that less more words

```
function selectEvenNumbers
evenNumbers = []
for i = 0 to i = length of the original array
if (element % 2 === 0)
Add/push that to the array called evenNumbers
return evenNumbers
```

- **5. Translate pseudocode to JavaScript**
    - When you have your pseudocode ready, translate each line into real code in JavaScript
    - Follow the pseudocode and write the solution
        - Use all JavaScript tools you know
        - Variable
        - Operators
        - function
        - Conditional statements
        - Loops
        - Arrays
        - All the JS methods you know
- **6. Test/debug**
    - Use sample test cases to see if your code returns the results, you want
    - It is highly recommended to use console.log () after each variable or code line This helps you check if the values and code are behaving as expected before you move on. By doing this, you catch any issues before you get too far.
- **7. Write useful comments in your code**
    - You may not always remember what every single line meant a month later. And someone else working on your code may not know either. That's why it's important to write useful comments to avoid problems and save time later if you need to come back to it.
    - Write brief, high-level comments
- **8. Practice, practice, practice**
    - programming, like with anything, comes easier and more naturally with time
    - GOAL: Solve at least 30 problems within the next 10 days. With each problem you solve, the better a developer you become.
- Watch this YouTube video to understand how to solve algorithmic problems using pseudocode: https://youtu.be/6hfOvs8pY1k
- Please use https://edabit.com/ as your main resource to get sample problems and practice solving them on your own computer