1. **Understand the Problem:**
   * Before you jump into coding, thoroughly understand the problem. Read the requirements multiple times. If possible, discuss it with others to get different perspectives.
2. **Break It Down:**
   * Decompose the problem into smaller, more manageable components or steps. Solve each component individually.
3. **Pseudocode First:**
   * Before diving into actual code, write pseudocode. This allows you to focus on the logic without getting bogged down by syntax.
4. **Choose the Right Data Structures and Algorithms:**
   * Think about the best data structures or algorithms that can help solve the problem efficiently.
5. **Start Simple:**
   * Write a basic version of the solution first, don't worry about optimization. This is sometimes referred to as a 'brute force' solution.
6. **Iterative Development:**
   * Code a bit, test a bit. Don't try to write the entire solution in one go. Regular testing can catch bugs early.
7. **Edge Cases:**
   * Always consider edge cases. Think about potential outliers or unique scenarios that might break your code.
8. **Debugging:**
   * When faced with bugs, stay calm. Use systematic debugging techniques. Print variables, use debuggers, or manually trace through your code.
9. **Reflect and Refactor:**
   * Once your code works, reflect on parts that can be improved or optimized. Clean, readable code is as important as a working solution.
10. **Practice Regularly:**

* Just as with any other skill, coding gets better with practice. Platforms like LeetCode, HackerRank, and Codewars offer coding challenges that can help sharpen your skills.

1. **Stay Updated and Learn:**

* The tech field evolves rapidly. Regularly invest time in learning new algorithms, data structures, and techniques.

1. **Rubber Duck Debugging:**

* Explaining your problem or code to someone else (or even an inanimate object like a rubber duck) can often help you see the issue in a new light and solve it.

1. **Seek Feedback:**

* Share your solutions with peers or mentors. They might offer insights or alternative approaches.

Remember, everyone encounters problems they can't solve initially. It's consistent practice, reflection, and the willingness to learn from mistakes that make you a better coder over time.

Top of Form