Developer Skills & Problem Solving

> 🎓 **CS Perspective:** This section covers **algorithmic thinking** and **problem decomposition**—fundamental skills from computational thinking. The 4-step process mirrors the classic software engineering approach: requirements analysis, modular design, research (leveraging existing solutions/libraries), and implementation. Breaking problems into sub-problems is essentially **top-down design** or **divide-and-conquer**, a paradigm used in algorithm design. Debugging techniques here relate to the broader discipline of **software verification and testing**.

📌 Problem Solving Framework

A structured approach to tackle coding challenges:

4-Step Process

```
1. UNDERSTAND the problem
   ↓
2. BREAK it into sub-problems
   ↓
3. RESEARCH if needed
   ↓
4. WRITE pseudocode and code
```

📌 Step 1: Understanding the Problem

Ask yourself:

- ✅ What is the problem asking?
- ✅ What are the inputs?
- ✅ What is the expected output?
- ✅ What are the edge cases?

Example

Problem: Calculate temperature amplitude from an array

Questions:

- What is "amplitude"? → Difference between max and min
- What if there are errors in data? → Skip them
- What type of data is expected? → Numbers

📌 Step 2: Breaking Into Sub-Problems

Divide complex problems into smaller, manageable tasks.

Example

Problem: Calculate temperature amplitude

Sub-problems:

1. How to find the maximum value?
2. How to find the minimum value?
3. How to handle errors in the array?
4. How to calculate the difference?

📌 Step 3: Research

Use these resources:

- **MDN Web Docs** - Official JavaScript documentation
- **Stack Overflow** - Community Q&A
- **Google** - Search for specific solutions

Search Tips

- Be specific: "javascript find max value in array"
- Include error messages in searches
- Look for recent answers (JS evolves)

📌 Step 4: Write Code

Example Solution

```
const temperatures = [3, -2, -6, -1, 'error', 9, 13, 17, 15, 14, 9, 5];

const calcTempAmplitude = function(temps) {
  let max = temps[0];
  let min = temps[0];

  for (let i = 0; i < temps.length; i++) {
    const curTemp = temps[i];

    // Skip non-numbers (errors)
    if (typeof curTemp !== 'number') continue;

    if (curTemp > max) max = curTemp;
    if (curTemp < min) min = curTemp;
  }

  return max - min;
};
```

```
const amplitude = calcTempAmplitude(temperatures);
console.log(amplitude); // 23 (17 - (-6))
```

📌 Merging Arrays

Use `concat()` to combine arrays:

```
const calcTempAmplitude = function(t1, t2) {
  const temps = t1.concat(t2);  // Merge arrays

  let max = temps[0];
  let min = temps[0];

  for (let i = 0; i < temps.length; i++) {
    const curTemp = temps[i];
    if (typeof curTemp !== 'number') continue;

    if (curTemp > max) max = curTemp;
    if (curTemp < min) min = curTemp;
  }

  return max - min;
};

calcTempAmplitude([3, 5, 1], [9, 0, 5]); // 9
```

📌 Debugging

What is a Bug?

A defect or problem in code that causes unexpected behavior.

Debugging Process

```
A) IDENTIFY → Know there's a bug
B) FIND → Locate the bug
C) FIX → Correct the code
D) PREVENT → Avoid similar bugs
```

📌 Console Methods for Debugging

```
console.log(value);    // Standard output
console.warn(value);   // Warning (yellow)
console.error(value);  // Error (red)
console.table(obj);    // Display object as table
```

console.table() Example

```
const measurement = {
  type: 'temp',
  unit: 'celsius',
  value: 10
};

console.table(measurement);
// ┌─────────┬───────────┐
// │ (index) │  Values   │
// ├─────────┼───────────┤
// │  type   │ 'temp'    │
// │  unit   │ 'celsius' │
// │  value  │    10     │
// └─────────┴───────────┘
```

📌 Common Debugging Techniques

1. Console Logging

```
function calcAge(birthYear) {
  console.log(birthYear); // Check input
  const age = 2037 - birthYear;
  console.log(age); // Check output
  return age;
}
```

2. Checking Types

```
console.log(typeof value);
```

3. Browser Debugger

- Open DevTools (F12 or Cmd+Option+I)
- Go to Sources tab
- Set breakpoints by clicking line numbers
- Step through code execution

4. debugger Statement

```
function buggyCode() {
  let x = 10;
  debugger;  // Execution pauses here
  x = x + 5;
  return x;
}
```

📌 Common Bug Patterns

1. Wrong Initial Values

```
// ❌ Bug: Starting with 0 misses negative numbers
let max = 0;
let min = 0;

// ✅ Fix: Start with first array element
let max = temps[0];
let min = temps[0];
```

2. Type Errors

```
// ❌ Bug: prompt returns string
const value = prompt('Enter number:');
const result = value + 10; // '510' (string concat)

// ✅ Fix: Convert to number
const value = Number(prompt('Enter number:'));
const result = value + 10; // 15
```

3. Off-by-One Errors

```
// ❌ Bug: < vs <=
for (let i = 0; i < arr.length; i++) // Correct
for (let i = 0; i <= arr.length; i++) // Off-by-one error
```

📌 Building Strings in Loops

```
const data = [17, 21, 23];
let str = '';
```

```
for (let i = 0; i < data.length; i++) {
  str += `${data[i]}ºC in ${i + 1} days ... `;
}

console.log('... ' + str);
// ... 17ºC in 1 days ... 21ºC in 2 days ... 23ºC in 3 days ...
```

📌 Code Quality Tips

1. Use Descriptive Variable Names

```
// ❌ Bad
const x = temps[0];
const y = temps[1];

// ✅ Good
const maxTemp = temps[0];
const minTemp = temps[1];
```

2. Write Clean Functions

```
// ✅ Single responsibility
function findMax(arr) {
  let max = arr[0];
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] > max) max = arr[i];
  }
  return max;
}
```

3. Handle Edge Cases

```
function calcAverage(arr) {
  if (arr.length === 0) return 0; // Edge case

  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    sum += arr[i];
  }
  return sum / arr.length;
}
```

📌 Using typeof for Validation

```
const temps = [3, -2, 'error', 9, 13];

for (let i = 0; i < temps.length; i++) {
  // Skip invalid data
  if (typeof temps[i] !== 'number') continue;

  console.log(temps[i]);
}
// Output: 3, -2, 9, 13
```

📌 Math Object Utilities

```
Math.random()            // Random 0-1
Math.trunc(4.7)          // 4 (remove decimals)
Math.round(4.5)          // 5 (round)
Math.floor(4.7)          // 4 (round down)
Math.ceil(4.2)           // 5 (round up)
Math.max(1, 5, 3)        // 5
Math.min(1, 5, 3)        // 1

// Random integer 1-6
Math.trunc(Math.random() * 6) + 1
```

🧪 Quick Reference Cheatsheet

```
// Problem Solving
// 1. Understand → 2. Break down → 3. Research → 4. Code

// Debugging
console.log(value);
console.table(object);
console.warn(message);
console.error(message);
debugger;

// Type checking
typeof value !== 'number'

// Array operations
arr.concat(arr2);          // Merge arrays
arr.length;                // Array length

// String building
```

```
str += 'new text';
str = str + 'new text';

// Skipping in loops
if (condition) continue;  // Skip iteration

// Math
Math.max(...arr);
Math.min(...arr);
Math.random();
Math.trunc(num);
```

✅ Exam Tips

1. **Always understand** the problem before coding
2. Break complex problems into **sub-problems**
3. Use `console.log()` liberally when debugging
4. `console.table()` is great for **objects and arrays**
5. Use `typeof` to validate data types
6. Initialize max/min with **first element**, not 0
7. Remember `prompt()` returns a **string**
8. Use `continue` to **skip** invalid data in loops
9. Use `concat()` to **merge** arrays
10. The `debugger` statement pauses execution in DevTools
11. Test with **edge cases** (empty arrays, invalid input)
12. Write **descriptive variable names** for readability