Problem Solving Approach

1. Intro To Problem Solving

A. Objectives

- Define what an algorithm is
- Devise a plan to solve algorithms
- Compare and contrast problem solving patterns including frequency counters, two pointer problems and divide and conquer

B. What is an ALGORITHM?

> A **process** or **set of steps** to accomplish a certian task

C. How do you improve?

> 1. Devise a plan for solving problems
> 2. Master common problem solving patterns

Problem Solving Strategies

> 1. Understand the Problem
> 2. Explore Concrete Examples
> 3. Break It Down
> 4. Solve/Simplify
> 5. Look Back and Refactor

---

2. STEP 1 - Understand The Problem

Understand The Problem

Before you start solving the problems, ask yourself these questions:

> 1. Can I `RESTATE` the problem in my own words?
> 2. What are the `INPUTS` that go into the problem?
> 3. What are the `OUTPUTS` that should come from the solution to the problem?
> 4. Can the `outputs` be `determined` from the `inputs`? In other words, do I have enough information to solve the problem?
> 5. How should I `LABEL` the important pieces of `data` that are part of the problem?

Example

1. Write a function which takes two numbers and returns their sum.

- STEP 1: Understand The Problem
    1. Restate:
        - "implement addition"

2. Inputs:
    - integers?
    - floating point?
    - how large?
    - only 2 inputs?
    - what if there's only 1 input?
3. Outputs:
    - type of output?
4. Output determined from Input:
    - in most cases there's enough information
    - if there's only 1 input what do we return?
5. Labeling Important Data:
    - what matters?
    - num1, num2, sum (variable names)

---

3. STEP 2 - Concrete Examples

Explore Examples

Coming up with examples can help you understand the problem better and examples also provide sanity checks that your eventual solution works how it should.

> 1. Start with `Simple` Examples
> 2. Progress to More `Complex` Examples
> 3. Explore Examples with `Empty Inputs`
> 4. Explore Examples with `Invalid Inputs`

Example

1. Write a function which takes in a string and returns counts of each character in the string.

- STEP 1: Understand The Problem
- STEP 2: Explore Examples
    1. Simple Examples:
        - `charCount("aaaa") => {a: 4}`
        - `charCount("hello") => {h:1, e:1, l: 2, o:1}`
            - do we count for characters not in the string? set it to 0?
    2. Complex Examples:
        - `charCount("this is Complex input #$#$231111")`
            - spaces?
            - numbers
            - special characters?
            - ignore casing? uppercase? lowercase?
    3. Empty Inputs:
        - `charCount("")`
            - what do we return?
    4. Invalid Inputs:

        - `charCount(INVALID_INPUT)`
            - input?
            - object?
            - null?

---

4. STEP 3 - Break It Down

Write Down The Steps

> Explicitly write out the steps you need to take. This forces you to think about the code you'll write before you write it, and helps you catch any lingering coneptual issues or misunderstandings before you dive in and have to worry about details as well.

Example

1. Write a function which takes in a string and returns counts of each character in the string.

- STEP 1: Understand The Problem
- STEP 2: Explore Examples
- STEP 3: Break It Down
    - Type up the skeleton of our function

```
function charChount(str){
  // do something

  // return an object with keys that are lowercase alphanumeric
characters in string; values should be a number that represents the
total count of the characters in a string
}
```

    - Futher Expand

```
function charChount(str){
  // make object to return at end

  // loop over string for each character...
      // if the char is a number/letter AND key in object, add one to
count

      // if the char is not number/letter AND not in object, add it
and set value to 1

      // if the char is something else (space, period, etc) don't do
anything

  // return object at end
}
```

> If you don't manage to solve the problem, make sure to layout your thought process in layman terms or pseudo code, and make sure you verbally talk through your process of solving the problem.

---

5. STEP 4 - Solve Or Simplify

SIMPLIFY

> 1. `Find` the core `difficulty` in what you're trying to do
> 2. Temporarily `ignore` that difficulty
> 3. Write a `simplified` solution
> 4. Then `incorporate` that difficult back in

Example

1. Write a function which takes in a string and returns counts of each character in the string.

- STEP 1: Understand The Problem
- STEP 2: Explore Examples
- STEP 3: Break It Down
- STEP 4: Solve or Simplify
    1. Find the core difficulty:
        - difficulty looping over a string?
        - forgot which methods uppercase/lowercase characters
        - difficulty with alphanumeric values
    2. Ignore the difficulty (temporarily):
    3. Write a SIMPLIFIED solution:

```
  function charChount(str){

     // make object to return at end

     const result = {};

    // loop over string for each character

        IGNORE (Temporarily):
              // if the char is a number/letter AND key in
object, add one to count
              // if the char is not number/letter AND not in
object, add it and set value to 1
              // if the char is something else (space, period,
etc) don't do anything

      for(let i = 0; i < str.length; i++){
        const char = str[i];
        if(result[char] > 0){
          result[char] += 1;
        }
```

```
        else{
          result[char] = 1;
        }
      }

    // return object at end

    return result;
  }
```

4. Incorporate the difficulty back in:
    - Do some research if allowed
    - Ask the interviewer for hint/suggestions after demonstrating your though process and solving a simplified solution

---

6. STEP 5 - Look Back and Refactor

Refactoring Questions

> 1. Can you check the result?
> 2. Can you derive the result differently?
> 3. Can you understand it at a glance?
> 4. Can you use the result or method for some other problem?
> 5. Can you improve the performance of your solution?
> 6. Can you think of other ways to refactor?
> 7. How have other people solved this problem?

---

7. Recap and Interview Strategies

> Steps 1 to 5 help us in devising a plan for solving problems.