S4.Problem Solving Approach

1. It comes up on interviews, a LOT!
2. Develop problem solving skills
3. Keep practicing

Problem Solving Approach

Undersdand the problem

1. Can I restate the problem in my own words?
2. What are the input that go into the problem?
3. What are the outputs / return that comes from the solution?
4. Can I get the outputs from the inputs? In other words, do I have all the info I need to solve the problem?
5. How should I call / label the important pieces in my problem

Explore Concrete Examples

1. Write 2-3 simple examples
2. Write 2-3 more comple examples
3. Explore examples with empty inputs
4. Explore example with invalid inputs

Break It Down

1. Explicitly write out the steps / skeleton you need to take
2. Helps build strategy
3. Help expose foggy points

4. It shows interviewer that we know how to solve it, even if time runs out (some are purposefully made so you don't finish)

Solve/Simplify

1. Solve problem, if you still can't solve a smaller problem
2. Once broken down, you'll have the easy chunks, and the hard chunks
3. Don't get stuck on it, and spend all your time doing that hard chunck
4. Do some of the other parts, demonstrating your knowledge, and it will also help giving way/ideas for the harder chunck
5. Once you get to the hard part, you get some hints from the interviewer, you can ask leading questions, explore potential techniques and ask for suggestions

Look Back and Refactor

1. Look back and see what you don't like, what could be done so it is more understandable?
2. How can it be more efficient?
3. Are there another solutions
4. Does it make sence, can other people read it?
5. How have other people solved this problem? Google
6.

```
//_____________________________________________
// Refactoring a solution

// given a string, return the count of each of its characters
// e.g. "hello" => {h:1, e:1, l:2, o:2}

function charCount(str) {
  // create an empty object
  var obj = {};
  // iterate thru string characters
  for (var i = 0; i < str.length; i++) {
    // char is string character lowercased
    var char = str[i].toLowerCase();
    // if alphanumeric test on char is true
    if (/[a-z0-9]/.test(char)) {
```

```
      // if object's key[of selected char] is >0 (meaning, it exist and it already has a
 1)
      if (obj[char] > 0) {
        // then, +1 its count
        obj[char]++;
        // else (if object's key[of selected char] is 0, meaning it doesn't exist)
      } else {
        // initialize the key at 1
        obj[char] = 1;
      }
    }
  }
  // return object
  return obj;
}

//_____________________________________________
// Refactor #1

function charCount(str) {
  var obj = {};
  // iterate thru string characters [with a "for of" loop]
  for (var char of str) {
    // char [is lowercased]
    char = char.toLowerCase();
    if (/[a-z0-9]/.test(char)) {
      if (obj[char] > 0) {
        obj[char]++;
      } else {
        obj[char] = 1;
      }
    }
  }
  return obj;
}

//_____________________________________________
// Refactor #2

function charCount(str) {
  var obj = {};
  for (var char of str) {
    char = char.toLowerCase();
    if (/[a-z0-9]/.test(char)) {
      // accesses object's key[of iterated character] (e.g. obj[h])
      // if it true, meaning there is such key => we are +1 the key, if false, obj[key] =
 1
      obj[char] = ++obj[char] || 1;
    }
  }
  return obj;
}

//_____________________________________________
```

```
// Refactor #3

function charCount(str) {
  var obj = {};
  for (var char of str) {
    char = char.toLowerCase();

    // if char is alphanumeric
    if (isAlphaNumeric(char)) {
      obj[char] = ++obj[char] || 1;
    }
  }
  return obj;
}

// Regex has had compatibility issues in the past with browsers, quoting Chrome
// Regex performance is higher vs simpler operations

// Instead, we can get a character's code (e.g. i.charCodeAt(0) = 105), and check if its c
ode fall within the alphanumeric code range

// Sidenote: isAlphaNumeric function could be within charCount, but it reads to much bette
r
function isAlphaNumeric(char) {
  var code = char.charCodeAt(0);
  if (
    !(code > 47 && code < 58) && // numeric (0-9)
    !(code > 64 && code < 91) && // upper alpha (A-Z)
    !(code > 96 && code < 123) // lower alpha (a-z)
  ) {
    // return false if code is out of alphanumeric code range
    return false;
  }
  // return true if code is within range
  return true;
}

//_____________________________________________
// Refactor #3

function charCount(str) {
  var obj = {};
  for (var char of str) {
    if (isAlphaNumeric(char)) {
      // after verification (for performance), lower case it
      char = char.toLowerCase();
      obj[char] = ++obj[char] || 1;
    }
  }
  return obj;
}

function isAlphaNumeric(char) {
  var code = char.charCodeAt(0);
```

```
  if (
    !(code > 47 && code < 58) && // numeric (0-9)
    !(code > 64 && code < 91) && // upper alpha (A-Z)
    !(code > 96 && code < 123) // lower alpha (a-z)
  ) {
    return false;
  }
  return true;
}

console.log(charCount("he8373)+***llo"));

//_____________________________________________________
// reference => S5, Frequency counter, refactoring from double loop (O(n^2)) -> 2 object l
oops (O(n))

function same(arr1, arr2) {
  // short circuit length comparison test
  if (arr1.length !== arr2.length) {
    return false;
  }

  // initiate 2 object where we will store the count of array1 and array2 elements
  let frequencyCounter1 = {};
  let frequencyCounter2 = {};

  // for each val of arr1
  for (let val of arr1) {
    // let frequencyCounter1[key] be value +1, or initialized value at 1
    frequencyCounter1[val] = (frequencyCounter1[val] || 0) + 1;
  }
  // for each val of arr2
  for (let val of arr2) {
    // let frequencyCounter2[key] be value +1, or initialized value at 1
    frequencyCounter2[val] = (frequencyCounter2[val] || 0) + 1;
  }
  console.log(frequencyCounter1); // {1:1, 2:2, 3:1, 5:1}
  console.log(frequencyCounter2); // {1:1, 4:2, 9:1, 11:1}

  // for each key of frequencyCounter1
  for (let key in frequencyCounter1) {
    // if !(it is not that) key squared is in frequencyCounter2 => false
    if (!(key ** 2 in frequencyCounter2)) {
      return false;
    }

    // if frequencyCounter2[frequencyCounter1[key] squared] is not equal to frequencyCount
er1 key => false
    if (frequencyCounter2[key ** 2] !== frequencyCounter1[key]) {
      return false;
    }
  }
  return true;
}
```

```
same([1, 2, 3, 2, 5], [9, 1, 4, 4, 11]);
```