Module 11 Career Connection

Nice job this week. You explored JavaScript—a complex but valuable language to understand, particularly when working on client-side visualization. JavaScript is the go-to-standard for transferring information to consumers via the internet.

A strong foundation in JavaScript can set you apart from other data visualization candidates. It provides technologies that deliver data visualization across the internet (e.g., web hosting). Also, JavaScript opens up a wealth of fun, dynamic libraries to work with, such as **D3.js** (https://d3js.org/) and **Chart.js** (https://www.chartjs.org/). Of course, JavaScript has its challenges, but nothing this valuable was ever *too* easy.

In this section, you will learn the following:

1. How JavaScript can be used daily in your new position
2. How you can showcase this new skill to be **Employer Competitive**
3. How to solve common algorithm questions using JavaScript

Employer Competitive Advantage

To get noticed by recruiters, add JavaScript as a skill to your resume. By doing so, you're demonstrating knowledge that extends beyond Python and data analysis, which will set you apart from other candidates. For help developing your resume, see the resources listed at Milestone: Develop Your Resume.

See Milestone: Develop Your Resume by accessing your career services content using the link in the left-hand navigation menu.

Technical Interview

Whether you are pursuing a data role or considering a JavaScript development role, the industry expects candidates to demonstrate expertise through algorithms.

The algorithms explored in this section are not typically written on the job; however, they are often used to test coding ability during the interview process.

Preparation

When preparing for a technical interview, keep two things in mind: Just do your best, and demonstrate your thought process.

Just Do Your Best

Remember that potential employers are measuring your performance against other interviewees, and you have no way of knowing who is interviewing for a role. Maybe the other candidates aren't as strong as you. Maybe nobody gets all the answers right. It's hard to know how to perform better in relation to other candidates when you don't know who they are. That frees you up to focus on **yourself** and delivering your best performance when interviewing.

Demonstrate Your Thought Process

The algorithms we are about to explore are quite difficult. They are used in interviews because they are hard. Still, that's no reason to worry. During a technical interview, you're not expected to get every answer right. In fact, interviewers are far more interested in your process. In other words, they want to see how you tackle the problem, not just the solution. In this situation, as in many things in life, it's about the journey, not the destination.

In fact, during the technical interview, the difficulty of the questions continues to increase until you reach a question you're unable to answer. The interviewer wants to observe you under pressure and will focus on your process for finding an answer, not whether you get the right answer.

Technical Interview

Solving JavaScript algorithms during the technical interview is often done as a whiteboarding exercise. You might use a dry-erase marker and a whiteboard and, with a coding prompt, be expected to write out your solution on the board. Sometimes, whiteboarding takes place using a computer or a pen and paper. Still, the steps are the same:

1. Pseudocode
2. Code
3. Optimize

Always write pseudocode for your solution first—don't write any actual code. Use plain English along with certain keywords (e.g., "loop" or "if-statement") to map the logical steps to solve the problem. Pseudocoding tackles two major parts of the coding interview:

1. Syntax

   This brings us back to the whole process of solving a problem again. Interviewers know that developers can forget syntax. We can't remember everything all the time—especially if JavaScript isn't our first language. To be fair, there are a lot of curly braces, brackets, and other weird things in JavaScript. Pseudocoding our solutions allows the interviewer to see which steps we would take, in a logical and methodical approach, knowing that we could easily look up syntax on the internet if this were not an interview.

2. Time Management

   You might be given a huge problem with little time to solve it. Pseudocoding allows us to plug those gaps. Pseudocode first, then start coding your solution. If you don't have time to code out the entire thing, then at least you have the pseudocode already there to fill those spaces. Only if you manage to code out the entire solution do you then want to look at optimizing it, time-permitting.

So, remember: pseudocode, code, then optimize.

Now tackle the following issues using your favorite code editor. Don't use the internet, and give yourself a maximum of 30 minutes to work on the solution. You can complete the solution in either ES5 or ES6.

Challenges

Challenge 1: Array Left Rotation

A left rotation on an array shifts each element of the array one unit to the left. In this challenge, given an array `arr` and a number `n`, perform `n` rotations on the array and return the updated array.

Sample Input

```
arr = [1,2,3,4,5];
```

```
n = 2;
```

Sample Output

```
[3,4,5,1,2]
```

The Code

```
function rotateLeft(arr, n) {
     // YOUR CODE HERE
}
```

The Pseudocode

```
    // create a copy of the array and save it in a temporary variable
    // loop through the array as many times as we need to do rotations
    // remove the first item of the array, store it in a temporary variable
    // add that temporary variable back at the end of the array
    // once completed, return the the newly rotated array
```

The Solution

```
function rotateLeft(arr, n) {
    // create a copy of the array and save it in a temporary variable
    var temp = arr.slice(0);
    // loop through the array as many times as we need to do rotations
    for (var i = 0; i < n; i++) {
        // remove the first item of the array, store it in a temporary variable
        var first = temp.shift();
        // add that temporary variable back at the end of the array
        temp.push(first);
    }
    // once completed, return the the newly rotated array
    return temp;
}
```

The Optimized ES6 Solution

```
rotateLeft = (arr, n) => {
    let temp = arr.slice(0);
    for (let i = 0; i < n; i++) {
        let first = temp.shift();
        temp.push(first);
    }
    return temp;
}
```

Challenge 2: CamelCase Converter

Write a function that takes in a string parameter, `str`, and converts any dashed or underscored words into CamelCase. If the first word of the input is capitalized, it should retain its capitalization in the output (i.e., upper

CamelCase)

Sample Input

```
toCamelCase("this-is-a-string")
```

```
toCamelCase("This_is_not_an_integer")
```

Sample Output

```
 "thisIsAString"
```

```
"ThisIsNotAnInteger"
```

The Code

```
function toCamelCase(str) {
     // YOUR CODE HERE
}
```

The Pseudocode

```
     // take the input string, and split it where there is an underscore or    a dash
    // use a loop to go through each item of the splitSentence
    // take the first item in that sentence, capitalize it, and then remove     the uncapitalized version
    // join the array back together

    // return the solved string
```

The Solution

```
function toCamelCase(str) {
     // take the input string, and split it where there is an underscore or    a dash
    var splitSentence = str.split(/[-_]/);
    // use a loop to go through each item of  the splitSentence
    for (var i = 1; i < splitSentence.length; i++) {
 // take the first item in that sentence, capitalize it, and then remove the uncapitalized version
        splitSentence[i] = splitSentence[i].charAt(0).toUpperCase() +    splitSentence[i].slice(1)

    }
    // join the  array back together
    var camelCaseStr = splitSentence.join('');

    // return the solved string
    return camelCaseStr;

}
```

The Optimized ES6 Solution

```
toCamelCase = str => {
    let splitSentence = str.split(/[-_]/);
    for (let i = 1; i < splitSentence.length; i++) {
        splitSentence[i] = splitSentence[i].charAt(0).toUpperCase() +    splitSentence[i].slice(1)
    }
    const camelCaseStr = splitSentence.join('');
    return camelCaseStr;
}
```

Challenge 3: isPrime

Write a function that takes in an integer argument and returns a Boolean value stating whether the integer is prime. For the purpose of this challenge, a prime number is considered a natural number greater than 1 that has no positive divisors.

Sample Input

```
console.log(isPrime(1));
console.log(isPrime(29));
console.log(isPrime(30));
```

Sample Output

```
false
true
false
```

The Code

```
function isPrime(integer){
// YOUR CODE HERE
}
```

The Pseudocode

```
        // Check if the number is equal to, or less than, 1. If so, return false because it isn't prime

        // Otherwise, start to loop through the numbers that precede the given integer, starting with 2
        // if the integer can be divisible by [i], return false because it isn't prime

    // Else, return true because the number IS prime
}
```

The Solution

```
function isPrime(integer) {
    // Check if the number is equal to, or less than, 1. If so, return false because it isn't prime
    if (integer <= 1) {
```

```
        return false;
    } else {

        // Otherwise, start to loop through the numbers that precede the given integer, starting with 2
        for (var i = 2; i < integer; i++) {
            // if the integer can be divisible by [i], return false because it isn't prime
            if (integer % i === 0) {
                return false;
            }
        }
    }

    // Else, return true because the number IS prime
    return true;
}
```

The Optimized ES6 Solution

```
isPrime = integer => {
    if (integer <= 1) {
        return false;
    } else {
        for (let i = 2; i < integer; i++) {
            if (integer % i === 0) {
                return false;
            }
        }
    }
    return true;
}
```

Continue to Hone Your Skills

If you're interested in learning more about the technical interviewing process and practicing algorithms in a mock interview setting, check out our **upcoming workshops.** **(https://careernetwork.2u.com/?utm_medium=Academics&utm_source=boot_camp)**