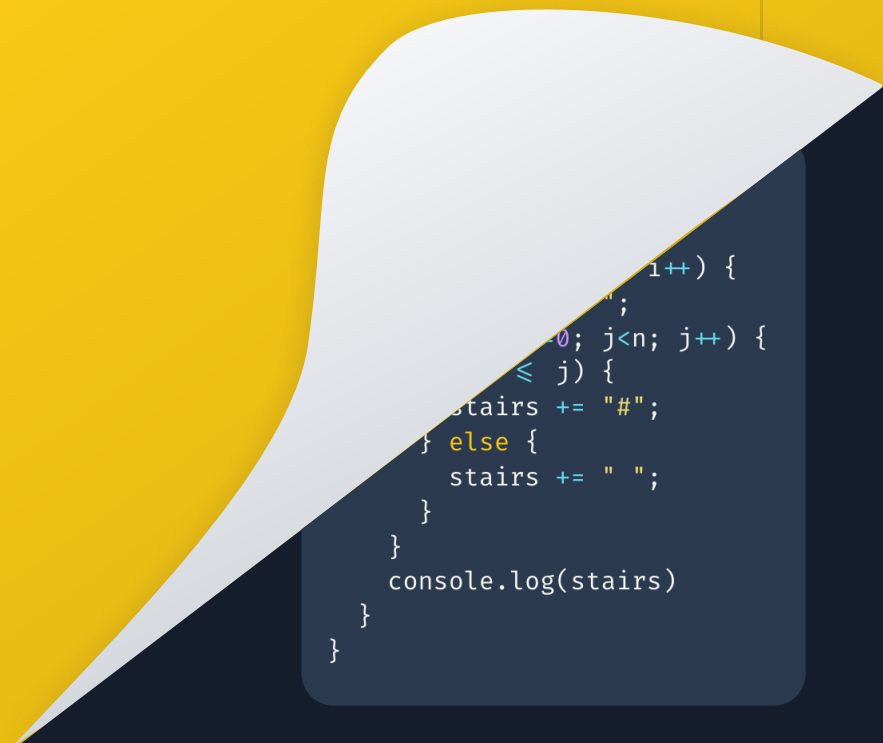
JavaScript: Interview Questions & Challenges

30 questions and challenges you will find in **most** interviews.



Welcome to "**JavaScript: Interview Questions & Challenges**"!

This ebook is designed to help you prepare for JavaScript job interviews, you will find a collection of 30 commonly asked interview questions and challenges that will help you practice and master your JavaScript skills.

These challenges and questions are designed to not only test your knowledge but also to help you improve your problem-solving skills.

Whether you are a beginner or an experienced developer, this ebook is a valuable resource for anyone who wants to excel in JavaScript interviews.

I'm confident that by working through the challenges and questions in this book, you will gain confidence in your skills and be better prepared to impress potential employers.

So, if you are ready to take your JavaScript skills to the next level and land your dream job, **let's get started!**

01 OBJECT KEYS VALUES

Problem:
Display all the keys and values of a nested object

```
function keyValuePrinter(obj) {
  for (let key in obj) {
    if (typeof obj[key] !== "object") {
      console.log("[" + key + " : " + obj[key] + "]");
    } else {
      keyValuePrinter(obj[key]);
    }
  }
}
```

Solution:

1. Base case: If the type of a value (under a key) is different from an object, then we print the key-value pair.
2. If the value is an object, call the function again with that object (obj[key])

02 VARIABLES SWAP

Problem:
Swap 2 integers present in variables num1 and num2 without using temporary variable.

```
let num1 = 10, num2 = 20;
[num1, num2] = [num2, num1];
```

Solution:

It is possible with simple destructuring assignment using array.

1. Create an array with [num2, num1]
2. Assign num1 to first position of newly created array, num2 to second position, using array destructuring.

03 STAIRS PROBLEM

Problem:

Create a function that given an integer it console logs a step shaped stair with `n` levels using the # character.

e.g. steps(2) =>
```
#
##
```

e.g. steps(3) =>
```
#
##
###
```

```
const steps = (n) ⇒
  Array.from({ length: n }, (_, i) ⇒ i + 1)
    .map((x) ⇒ '#'.repeat(x))
    .map((x) ⇒ console.log(x));
```

Solution:

1. Create an array with n length using Array.from
2. Fill it with numbers from 0 to n with callback fn
3. Map over the array and create another array with # char repeated from 0 to n.
4. Iterate over the array and console log the results

04 ARRAY CHECK

Problem:
How do you check if a variable holds an array?

```
if (Array.isArray(arrayList)) {
  // arrayList is an array!
}
```

Solution:

You can use Array.isArray() method

05 AGE DIFFERENCE

Problem:
Given the following array of objects, find the difference in age between the oldest and youngest family members, and return their respective ages and the age difference.

```
const input = [
  {
    name: 'John',
    age: 13,
  },
  {
    name: 'Mark',
    age: 56,
  },
  {
    name: 'Rachel',
    age: 45,
  },
  {
    name: 'Nate',
    age: 67,
  },
]

// Output: [13, 67, 54]
```

05 AGE DIFFERENCE

```
const ages = input.map((person) ⇒ person.age)

[Math.min( ... ages), Math.max( ... ages), Math.max( ... ages) - Math.min( ... ages)]
```

Solution:

1. Save only the ages into an array. [13, 56, 45, 67]
2. Math.min() and Math.max() functions accepts an array of numbers, so create an array where the first position is the minimum value of the array of ages and the second position the max age.
3. The last position is the difference between the Math.min(...) and Math.max(...)

06 FLAT ARRAY

Problem:
Given an array of arrays, flatten them into a single array.

```
function flatten(arr) {
  return arr.reduce(function (flat, toFlatten) {
    return flat.concat(Array.isArray(toFlatten) ? flatten(toFlatten) : toFlatten);
  }, []);
}
```

Solution:

1. call reduce array helper method and have the accumulator (flat) be an empty array that will always be flat. toFlatten is the current item of the array.
2. On each iteration we concatenate the flat array with the result of calling flatten again if current item is an array, otherwise concat that value.
3. The base case for the recursive call will be when the toFlatten value is not an array, in which case we just add it to the array.

07 PALINDROME

Problem:
A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward.

Return true if the given string is a palindrome. Otherwise, return false.

```
function palindrome(str) {
  var re = /[\W_]/g;

  var lowRegStr = str.toLowerCase().replace(re, '');

  var reverseStr = lowRegStr.split('').reverse().join('');

  return reverseStr === lowRegStr;
}
```

Solution:

1. Lowercase the string and use the RegExp to remove unwanted characters from it.
2. Use the same chaining methods with built-in functions from the previous article 'Three Ways to Reverse a String in JavaScript'
3. Check if reverseStr is strictly equals to lowRegStr and return a Boolean

08 VOWELS

Problem:
Given a string of text containing 0 or more vowels, count the number of vowels that can be found within the text.

```
function countVowel(str) {
    const count = str.match(/[aeiou]/gi).length;
    return count;
}
```

Solution:

1. The match() method retrieves the result of matching a string against a regular expression.
2. We are looking for the vowels [aeiou].
3. `i` in a regular expression is meant to ignore the case, so it will match both a and A for example.
4. g stands for global, and it tells the match fn to perform a global search.

09 NULL VS UNDEFINED

Problem:
What is the difference between null and undefined in JavaScript?

Solution:

1. Undefined: means a variable has been declared but has not yet been assigned a value.
2. Null: Is an assignment value. It can be assigned to a variable as a representation of no value.

```
var testVar;
alert(testVar); //shows undefined
alert(typeof testVar); //shows undefined
```

```
var testVar = null;
alert(testVar); //shows null
alert(typeof testVar); //shows object
```

10 REVERSE A STRING

Problem:
Given the string ‘XWZ’ return ‘ZWX’

```
function reverseStr(str) {
  return str.split('').reverse().join('');
}
```

Solution:

1. split(‘’) when applied to a string returns a new array with each char of the string on a position of the array like [‘X’, ‘W’, ‘Z’].
2. reverse() function when applied to an array it reverses the string values like [‘Z’, ‘W’, ‘X’]
3. join(‘’) returns a new string by concatenating all of the elements in an array, separated by the specified separator string like ZWX

II FIZZ BUZZ

Problem:
Write a function that logs the numbers from 1 to n, but for multiples of 3 prints “fizz” and for multiples of 5 prints “buzz”. For numbers that are multiples of both (3 & 5) prints “fizzbuzz”.

```
function fizzBuzz(n) {
  for (let i = 1; i <= n; i++)
{   if (i % 15 === 0) {
      console.log('fizzbuzz')
    } else if (i % 3 === 0) {
      console.log('fizz')
    } else if (i % 5 === 0) {
      console.log('buzz')
    } else {
      console.log(i)
    }
  }
}
```

II FIZZ BUZZ

Problem:
Write a function that logs the numbers from 1 to n, but for multiples of 3 prints “fizz” and for multiples of 5 prints “buzz”. For numbers that are multiples of both (3 & 5) prints “fizzbuzz”.

Solution:

1. If the current number is a multiple of both 3 and 5, it means that it has to be multiple of 3*5 = **15** (i % 15 === 0), we print "FizzBuzz".
2. If the current number is only a multiple of 3 (i % 3 === 0), we print "Fizz".
3. If the current number is only a multiple of 5 (i % 5 === 0), we print

12 VAR, LET & CONST

Problem:
Please state the differences between **var**, **let** and **const** in JavaScript.

	is it block scoped?	creates a global variable?	is it reassignable?	is it redeclarable?
var	✗	✓	✓	✓
let	✓	✗	✓	✗
const	✓	✗	✗	✗

Subtle use cases to have in mind:

1. **var** is function scoped when is declared inside a function
2. **const** is not reassignable but will accept changes in values (for arrays and objects)
3. **let** can be redeclared outside of the scope it was declared.

I3 CHUNK PROBLEM

Problem:
Given an array and a chunk size, divide the array into many subarrays with each subarray being of the chunk size.

e.g. chunk([1,2,3,4,5], 2) -> [[1,2], [3,4], [5]]

length 2

```
function chunk(arr, size) {
  const chunked = [];
  let index = 0;

  while (index < arr.length) {
    let chunk = arr.slice(index, index + size);
    chunked.push(chunk);
    index += size;
  }

  return chunked;

}
```

I3 CHUNK PROBLEM

Problem:
Given an array and a chunk size, divide the array into many subarrays with each subarray being of the chunk size.

e.g. chunk([1,2,3,4,5], 2) -> [[1,2], [3,4], [5]]

length 2

Solution:

1. create an auxiliary array.
2. create an auxiliary index.
3. iterate the array from the 0 position.
4. create the first chunk by slicing the array from the index until the size parameter. (e.g. chunk = [1,2])
5. Increase the index by the size param.
6. e.g. index = 0 + 2 = 2
7. repeat the process with new index. This time the new chunk will be arr.slice(2, 4) = [3,4]

14 CAPITALIZE WORDS

Problem:
Create a function that given a string, it capitalizes the first letter of the words in a sentence.

e.g. capitalize(“hello world”) -> Hello World

```
function capitalize(str) {
  const words = []

  for (let word of str.split(' ')) {
    words.push(`${word[0].toUpperCase()}${word.slice(1)}`)
  }

  return words.join(' ')
}
```

Solution:

1. create an auxiliary words array and iterate through each word in the string.
2. capitalize each word by accessing its first letter and calling the toUpperCase function.
3. join the words on the auxiliary array with a space between them.

I5 REVERSE INTEGERS

Problem:
Create a function that given an integer, it returns an integer that is the reverse ordering of numbers.

e.g. reverseIntegers(7364) -> 4637
reverseIntegers(-15) -> -51
reverseIntegers(-90) -> -9

```
function reverseInteger(num) {
  const reversed = num.toString().split('').reverse().join('');

  return parseInt(reversed) * Math.sign(num);
}
```

Solution:

1. Convert the integer into a string and split the string so that every number is in a position of the array.
2. Reverse the array calling the reverser() helper method.
3. Convert the array back to string and convert it into an integer and multiply it by it's original sign.

I6 FIRST NON REPEATING CHARACTER.

Problem:
How could you find the first non repeating char in a string?

e.g. firstNonRepeatingChar("This is an example sentence") -> h

```
function firstNonRepeatingChar(str) {
  let length = str.length;
  let char = '';
  let charCount = {};

  for (let i = 0; i < length; i ++) {
    char = str[i].toLowerCase( );

    if (charCount[char]) {
      charCount[char] ++;
    } else {
      charCount[char] = 1;
    }
  }

  for (let j in charCount) {
    if (charCount[j] === 1) {
      return j
    }
  }
}
```

I6 FIRST NON REPEATING CHARACTER.

Problem:
How could you find the first non repeating char in a string?

e.g. firstNonRepeatingChar("This is an example sentence") -> h

Solution:

1. create a char auxiliary variable to store each charater on the sentence.
2. create a charCount object to store a key value map of each character and the amount of times it exists on the sentence. Something like { 's': 2, 'j': 1 ...}
3. iterate through each letter on the sentence.
4. store each character on a temp auxiliary variable.
5. If that character doesn't exist on the charCount object we add it with an occurrence of 1, otherwise we just increase its occurrence.
6. Finally we iterate over the charCount object and return the first key (character) which occurrence count is 1.

I7 HIGHEST OCCURRENCE

Problem:
Given an array, get the element with the highest occurrence in an array.

e.g. highestOccurrence([1,2,'a',4,3,'a','b','a',6,1]) -> 'a'

```
function highestOccurrence(array) {
    if (array.length == 0) {
        return null;
    }
    var occurrenceMap = {};
    var maxEl = array[0], maxCount = 1;
    for(var i = 0; i < array.length; i++)
    {
        var el = array[i];
        if (occurrenceMap[el] == null) {
            occurrenceMap[el] = 1;
        } else {
            occurrenceMap[el]++;
        }

        if (occurrenceMap[el] > maxCount) {
            maxEl = el;
            maxCount = occurrenceMap[el];
        }
    }
    return maxEl;
}
```

17 HIGHEST OCCURRENCE

Problem:
Given an array, get the element with the highest occurrence in an array.

e.g. highestOccurrence([1,2,'a',4,3,'a','b','a',6,1]) -> 'a'

Solution:

1. if array has no items we return null
2. create auxiliary variables for the amount of element occurrences, the current max element, and the max count an element has been repeated.
3. iterate through the array and store each element on the el variable.
4. if our occurrence map object has already a count for that element, we increase its value, otherwise we initialize the element's occurrence count with 1.
5. if the current element's occurrence count is higher than the max count, we set the current element as the element with maximum count, and we update our maxCount variable with the element's count.
6. we return the max element after finishing the iteration of the array.

I8 INTERSECTION

Problem:
Given two arrays of primitives, return the elements that are included in both arrays (intersection).

e.g. intersection([1,2,3], [2,3,4,5]) -> [2, 3]

```
function intersection(arr1, arr2) {
    return arr1.filter(value ⇒ arr2.includes(value));
}
```

Solution:

1. filter all the elements from the first array that are **not** included in the second array.
2. for this we use the filter array helper, and the condition is that the current value of the array has to be included in the second array.

19 MAKE PAIRS

Problem:
Write a method that given an object with key value pairs, returns a deep array like [[key, value]]

e.g. makePairs({ a: 1, b: 2 }) -> [['a', 1], ['b', 2]]

```
function makePairs(obj) {
    return Object.keys(obj).map((key) ⇒ [key, obj[key]]);
}
```

Solution:

1. Object.keys returns an array with all the keys of the object.
2. we iterate over those, and for each key, we return an array that has the key in the first position and the value of that key (obj[key]) in the second position.

20 TOTAL PRICE

Problem:
Create a function that takes an array of objects (groceries) which calculates the total price and returns it as a number. A grocery object has a product, a quantity and a price, for example:

e.g. getTotalPrice([{ product: "Milk", quantity: 3, price: 1.50 },
{ product: "Cereals", quantity: 2, price: 2.50 }]) → 9.5

```
function getTotalPrice(groceries) {
  return groceries.reduce((acc, curr) => {
    acc += curr.price * curr.quantity;
    return acc;
  }, 0)
}
```

20 TOTAL PRICE

Problem:
Create a function that takes an array of objects (groceries) which calculates the total price and returns it as a number. A grocery object has a product, a quantity and a price, for example:

e.g. getTotalPrice([{ product: "Milk", quantity: 3, price: 1.50 },
{ product: "Cereals", quantity: 2, price: 2.50 }]) → 9.5

Solution:

1. we use reduce array helper and create an accumulator value that starts from 0.
2. On each iteration we increment the accumulator by the current element price * current element quantity.
3. we return the accumulator on each iteration

21 POSITIVE DOMINANT

Problem:
An array is positive dominant if it contains strictly more unique positive values than unique negative values. Write a function that returns true if an array is positive dominant.

e.g. isPositiveDominant([1,1,-2,-3]) -> false
isPositiveDominant([1,2,3,-1,-2]) -> true

```
function isPositiveDominant(arr) {
    const positives = new Set(arr.filter(n => n > 0));
    const negatives = new Set(arr.filter(n => n < 0));

    return positives.size > negatives.size;
}
```

21 POSITIVE DOMINANT

Problem:
An array is positive dominant if it contains strictly more unique positive values than unique negative values. Write a function that returns true if an array is positive dominant.

e.g. isPositiveDominant([1,1,-2,-3]) -> false
isPositiveDominant([1,2,3,-1,-2]) -> true

Solution:

1. we create two Sets, one with an array of positive values from arr, and the other with negatives values from arr.
2. because a value in the Set may only occur once, we guarantee that all the values are unique.
3. .size attribute of a set returns the size of the set, how many values does it hold. Is positive dominant if the size of the positive set is larger than the size of the negative set.

22 TWO DISTINCT ELEMENTS

Problem:
In each input array, every number repeats at least once, except for two. Write a function that returns the two unique numbers.

e.g. returnUnique([1, 9, 8, 8, 7, 6, 1, 6]) → [9, 7]

```
function returnUnique(arr) {
    return arr.filter(el ⇒ arr.indexOf(el) === arr.lastIndexOf(el));
}
```

Solution:

1. We filter the array using the following condition; the first index position of the element in the array has to be the same as the last index position of the element. This guarantees that the element is unique, because it doesn't occur again in the array.

23 REVERSE THE ODD LENGTH WORDS

Problem:
Given a string, reverse all the words which have odd length. The even length words are not changed.

e.g. reverseOdd("Bananas") → "sananaB"

```
function reverseOdd(str) {
  return str.split(" ")
         .map(w => w.length%2 ? [...w].reverse().join("") : w)
         .join(" ");
}
```

Solution:

1. we convert the string into an array of words by using str.split(“ “).
2. we check if the word has odd characters by checking the remainder of w.length/2. If the remainder is 0 it means the word has even characters, otherwise it has odd characters.
3. if it has odd characters, we split the words characters into a new array and we reverse it to return the reversed word.
4. last, we join the words with a space between them

24 HOURS PASSED

Problem:
Write a function that takes time t1 and time t2 and returns the numbers of hours passed between the two times.

e.g. hoursPassed("3:00 AM", "9:00 AM") → "6 hours"

```
function hoursPassed(t1, t2) {
    t1 = eval(t1.replace(' AM','').replace(' PM','+12').replace(':00',''))
    t2 = eval(t2.replace(' AM','').replace(' PM','+12').replace(':00',''))
    if (t1==t2) {
      t1 = 0;
    }
    return t1≠t2 ? Math.abs(t1-t2) + ' hours' : 'No time has passed.'
}
```

Solution:

1. we replace AM or PM with an empty string or with +12 respectively.
2. we also replace :00 with an empty string as we only care about the hours.
3. if after doing so, t1 == t2 we set t1 to 0 and return the difference between t1 and t2. We return no time has passed if t1 == t2.

25 WHITE SPACES

Problem:
Write a function that inserts a white space between every instance of a lower character followed immediately by an upper character.

e.g. whiteSpaces("HelloMyNameIsGeorge") -> Hello My Name Is George

```
const insertWhitespace = str ⇒
  str.replace(/([a-z][A-Z])/g, ([lower, upper]) ⇒ `${lower} ${upper}`);
```

Solution:

1. we use the replace function against a regular expression to solve this.
2. /([a-z][A-Z])/g will match all the lower case letters followed immediately by an uppercase letter. e.g. oM, yN.
3. the second parameter of the replace function can take a callback function with the maches from the regular expression, in our case lower is the lowercase (o,y,e,s) and the upper is the uppercase letter (M,N,Y,G)
4. we replace those by adding a space between them.

26 PRIME NUMBERS

Problem:
Create a function that returns true if there's at least one prime number in the given range (n1 to n2 (inclusive)), false otherwise.

e.g. primeInRange(10, 15) → true // Prime numbers in range: 11, 13

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

const primeInRange = (low, high) => {
    for (let num = low; num <= high; num += 1) {
        if (isPrime(num)) return true;
    }

    return false;
}
```

26 PRIME NUMBERS

Problem:
Create a function that returns true if there's at least one prime number in the given range (n1 to n2 (inclusive)), false otherwise.

e.g. primeInRange(10, 15) → true // Prime numbers in range: 11, 13

Solution:

1. we iterate from n1 (low) to n2 (high), and we call an aux function isPrime() on each number. It there is one number that is prime we return true.
2. to check if a number is prime, we check the remainder of dividing the number by num. If the remainder is 0 then it means the number is not prime, otherwise it's prime.

27 MAP THE LETTERS IN A STRING

Problem:
Given a word, create an object that stores the indexes of each letter in an array.

- Make sure the letters are the keys.
- Make sure the letters are symbols.
- Make sure the indexes are stored in an array and those arrays are values.

e.g. mapLetters("dodo") → { d: [0, 2], o: [1, 3] }

```
const mapLetters = (str) =>
  Array.from(str).reduce((accumulator, currValue, index) => {
    return { ...obj, [char]: [...(obj[char] || []), index] }
  }, {})
```

Solution:

1. we create an array from the string and we apply the reduce function with an object as the accumulator.
2. on each iteration we return whatever was in the obj before (...obj) and then in the char key ([char]) we spread the previous array if it exists, otherwise we just add the index to the array like [...(obj[char] || []), index]

28 MULTIPLICATION TABLE

Problem:
Create N x N multiplication table, of size n provided in parameter.

For example, when n is 5, the multiplication table is:

- *1, 2, 3, 4, 5*
- *2, 4, 6, 8, 10*
- *3, 6, 9, 12, 15*
- *4, 8, 12, 16, 20*
- *5, 10, 15, 20, 25*

```
const multiplicationTable = n ⇒
    Array.from({length: n}, (_, i) ⇒
        Array.from({length: n},(_, j) ⇒ (i + 1) * (j + 1)));
```

28 MULTIPLICATION TABLE

Problem:
Create N x N multiplication table, of size n provided in parameter.

For example, when n is 5, the multiplication table is:

- *1, 2, 3, 4, 5*
- *2, 4, 6, 8, 10*
- *3, 6, 9, 12, 15*
- *4, 8, 12, 16, 20*
- *5, 10, 15, 20, 25*

Solution:

1. we use Array.from function that takes a callback fn as second parameter to initialize each element.
2. we also use Array.from to create another array for the nested arrays.
3. On each iteration we use the index of the first array (i) as base number for multiplication.
4. On each iteration we use the index of the second array (j) as the multiplier which will always range between (0-n)

29 FINDING COMMON ELEMENTS

Problem:
Create a function that takes two "sorted" arrays of numbers and returns an array of numbers which are common to both the input arrays.

commonElements([-1, 3, 4, 6, 7, 9], [1, 3]) → *[3]*
commonElements([1, 3, 4, 6, 7, 9], [1, 2, 3, 4, 7, 10]) → *[1, 3, 4, 7]*

```
function commonElements(arr1,arr2) {
  return arr2.filter(el⇒arr1.includes(el));
}
```

Solution:

1. we filter the second array by the following condition: if the arr1 has the current element in the iteration, then we return that element, because it means it's in both arrays.

30 NTH FIBONACCI NUMBER

Problem:
Create a function that returns the Nth number in the Fibonacci sequence

fibonacci(10) → *"55"*
fibonacci(20) → *"6765"*

```
function fib(n) {
  if (n < 2){
    return n
  }
  return fib(n - 1) + fib (n - 2)
}
```

Solution:

1. the base case for fibonacci is when the number is less than 2, in that case we return the number itself.
2. for the remaining cases we return the result of running fib(n - 1) + fib(n - 2) because for any given number the result of a fibonacci sequence is the sum of the last two.

Thank you for taking the time to go through my Javascript interview questions and challenges.

I hope that my content has provided you with valuable insights and knowledge that can help you improve your skills as a Javascript developer.

If you found it helpful and want to take your skills to the next level, I highly recommend checking out my content on **HTML and CSS**, **UX/UI** and more here: https://georgemoller.gumroad.com/.

Thank you once again for your interest in my Javascript interview questions and challenges, and I wish you all the best in your learning journey.