


A mostly reasonable collection of technical software development interview questions solved in Javascript

[#interview-questions](#) [#javascript](#) [#stack](#) [#recursion](#) [#strings](#) [#array](#) [#interview-practice](#) [#interviews](#)

 90 commits 1 branch 0 releases 12 contributors MIT

Branch: master ▾

New pull request

Create new file

Upload files

Find File

Clone or download ▾

 kennymkchan Merge pull request #33 from Fundebug/master ...

Latest commit 79348b2 on Oct 26, 2018

 [LICENSE.txt](#) Adding MIT License. 3 years ago [README.md](#) Code bug fix: 2.3 Check if a given string is a isomorphic 6 months ago [README.md](#)

Interview Algorithm Questions in Javascript() {...}

A mostly reasonable collection of technical software development interview questions solved in Javascript in ES5 and ES6

Table of Contents

1. [Array](#)
2. [Strings](#)
3. [Stacks and Queues](#)
4. [Recursion](#)
5. [Numbers](#)
6. [Javascript Specific](#)
7. To Be Continued

Array

- 1.1 Given an array of integers, find the largest product yielded from three of the integers

```
var unsortedArray = [-10, 7, 29, 30, 5, -10, -70];

computeProduct(unsortedArray); // 21000

function sortIntegers(a, b) {
    return a - b;
}

// Greatest product is either (min1 * min2 * max1 || max1 * max2 * max3)
function computeProduct(unsorted) {
    var sortedArray = unsorted.sort(sortIntegers),
        product1 = 1,
        product2 = 1,
        array_n_element = sortedArray.length - 1;

    // Get the product of three largest integers in sorted array
    for (var x = array_n_element; x > array_n_element - 3; x--) {
        product1 = product1 * sortedArray[x];
    }

    product2 = sortedArray[0] * sortedArray[1] * sortedArray[array_n_element];
```

```

    if (product1 > product2) return product1;

    return product2;
}

```

View on Codepen: <https://codepen.io/kennymkchan/pen/LxoMvm?editors=0012>

- **1.2** Being told that an unsorted array contains (n - 1) of n consecutive numbers (where the bounds are defined), find the missing number in $O(n)$ time

```

// The output of the function should be 8
var arrayOfIntegers = [2, 5, 1, 4, 9, 6, 3, 7];
var upperBound = 9;
var lowerBound = 1;

findMissingNumber(arrayOfIntegers, upperBound, lowerBound); // 8

function findMissingNumber(arrayOfIntegers, upperBound, lowerBound) {
    // Iterate through array to find the sum of the numbers
    var sumOfIntegers = 0;
    for (var i = 0; i < arrayOfIntegers.length; i++) {
        sumOfIntegers += arrayOfIntegers[i];
    }

    // Find theoretical sum of the consecutive numbers using a variation of Gauss Sum.
    // Formula: [(N * (N + 1)) / 2] - [(M * (M - 1)) / 2];
    // N is the upper bound and M is the lower bound

    upperLimitSum = (upperBound * (upperBound + 1)) / 2;
    lowerLimitSum = (lowerBound * (lowerBound - 1)) / 2;

    theoreticalSum = upperLimitSum - lowerLimitSum;

    return theoreticalSum - sumOfIntegers;
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/rjgoXw?editors=0012>

- **1.3** Removing duplicates of an array and returning an array of only unique elements

```

// ES6 Implementation
var array = [1, 2, 3, 5, 1, 5, 9, 1, 2, 8];

Array.from(new Set(array)); // [1, 2, 3, 5, 9, 8]

// ES5 Implementation
var array = [1, 2, 3, 5, 1, 5, 9, 1, 2, 8];

uniqueArray(array); // [1, 2, 3, 5, 9, 8]

function uniqueArray(array) {
    var hashmap = {};
    var unique = [];

    for (var i = 0; i < array.length; i++) {
        // If key returns undefined (unique), it is evaluated as false.
        if (!hashmap.hasOwnProperty(array[i])) {
            hashmap[array[i]] = 1;
            unique.push(array[i]);
        }
    }

    return unique;
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/ZLNwze?editors=0012>

- 1.4 Given an array of integers, find the largest difference between two elements such that the element of lesser value must come before the greater element

```
var array = [7, 8, 4, 9, 9, 15, 3, 1, 10];
// [7, 8, 4, 9, 9, 15, 3, 1, 10] would return `11` based on the difference between `4` and `15`
// Notice: It is not `14` from the difference between `15` and `1` because 15 comes before 1.

findLargestDifference(array);

function findLargestDifference(array) {
    // If there is only one element, there is no difference
    if (array.length <= 1) return -1;

    // currentMin will keep track of the current lowest
    var currentMin = array[0];
    var currentMaxDifference = 0;

    // We will iterate through the array and keep track of the current max difference
    // If we find a greater max difference, we will set the current max difference to that variable
    // Keep track of the current min as we iterate through the array, since we know the greatest
    // difference is yield from `largest value in future` - `smallest value before it`

    for (var i = 1; i < array.length; i++) {
        if (array[i] > currentMin && (array[i] - currentMin > currentMaxDifference)) {
            currentMaxDifference = array[i] - currentMin;
        } else if (array[i] <= currentMin) {
            currentMin = array[i];
        }
    }

    // If negative or 0, there is no largest difference
    if (currentMaxDifference <= 0) return -1;

    return currentMaxDifference;
}
```

View on Codepen: <http://codepen.io/kennymkchan/pen/MJdLWJ?editors=0012>

- 1.5 Given an array of integers, return an output array such that output[i] is equal to the product of all the elements in the array other than itself. (Solve this in O(n) without division)

```
var firstArray = [2, 2, 4, 1];
var secondArray = [0, 0, 0, 2];
var thirdArray = [-2, -2, -3, 2];

productExceptSelf(firstArray); // [8, 8, 4, 16]
productExceptSelf(secondArray); // [0, 0, 0, 0]
productExceptSelf(thirdArray); // [12, 12, 8, -12]

function productExceptSelf(numArray) {
    var product = 1;
    var size = numArray.length;
    var output = [];

    // From first array: [1, 2, 4, 16]
    // The last number in this case is already in the right spot (allows for us)
    // to just multiply by 1 in the next step.
    // This step essentially gets the product to the left of the index at index + 1
    for (var x = 0; x < size; x++) {
        output.push(product);
        product = product * numArray[x];
    }

    // From the back, we multiply the current output element (which represents the product
    // on the left of the index, and multiplies it by the product on the right of the element)
    var product = 1;
    for (var i = size - 1; i > -1; i--) {
        output[i] = output[i] * product;
        product = product * numArray[i];
    }
}
```

```

    }

    return output;
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/OWYdJK?editors=0012>

- **1.6 Find the intersection of two arrays.** An intersection would be the common elements that exists within both arrays. In this case, these elements should be unique!

```

var firstArray = [2, 2, 4, 1];
var secondArray = [1, 2, 0, 2];

intersection(firstArray, secondArray); // [2, 1]

function intersection(firstArray, secondArray) {
    // The logic here is to create a hashmap with the elements of the firstArray as the keys.
    // After that, you can use the hashmap's O(1) look up time to check if the element exists in the hash
    // If it does exist, add that element to the new array.

    var hashmap = {};
    var intersectionArray = [];

    firstArray.forEach(function(element) {
        hashmap[element] = 1;
    });

    // Since we only want to push unique elements in our case... we can implement a counter to keep track of what
    secondArray.forEach(function(element) {
        if (hashmap[element] === 1) {
            intersectionArray.push(element);
            hashmap[element]++;
        }
    });

    return intersectionArray;

    // Time complexity O(n), Space complexity O(n)
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/vgwbEb?editors=0012>

[↑ back to top](#)

Strings

- **2.1 Given a string, reverse each word in the sentence** "Welcome to this Javascript Guide!" should become "emocleW ot siht tpircsavaJ !ediuG"

```

var string = "Welcome to this Javascript Guide!";

// Output becomes !ediuG tpircsavaJ siht ot emocleW
var reverseEntireSentence = reverseBySeparator(string, "");

// Output becomes emocleW ot siht tpircsavaJ !ediuG
var reverseEachWord = reverseBySeparator(reverseEntireSentence, " ");

function reverseBySeparator(string, separator) {
    return string.split(separator).reverse().join(separator);
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/VPOONZ?editors=0012>

- **2.2 Given two strings, return true if they are anagrams of one another** "Mary" is an anagram of "Army"

```

var firstWord = "Mary";
var secondWord = "Army";

isAnagram(firstWord, secondWord); // true

function isAnagram(first, second) {
  // For case insensitivity, change both words to lowercase.
  var a = first.toLowerCase();
  var b = second.toLowerCase();

  // Sort the strings, and join the resulting array to a string. Compare the results
  a = a.split("").sort().join("");
  b = b.split("").sort().join("");

  return a === b;
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/NdVVVj?editors=0012>

- **2.3 Check if a given string is a palindrome** "racecar" is a palindrome. "race car" should also be considered a palindrome. Case sensitivity should be taken into account

```

isPalindrome("racecar"); // true
isPalindrome("race Car"); // true

function isPalindrome(word) {
  // Replace all non-letter chars with "" and change to lowercase
  var lettersOnly = word.toLowerCase().replace(/\/s/g, "");

  // Compare the string with the reversed version of the string
  return lettersOnly === lettersOnly.split("").reverse().join("");
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/xgNNNB?editors=0012>

- **2.3 Check if a given string is a isomorphic**

For two strings to be isomorphic, all occurrences of a character in string A can be replaced with another character to get string B. The order of the characters must be preserved. There must be one-to-one mapping for every char of string A to every char of string B.

`paper` and `title` would return true.
 `egg` and `sad` would return false.
 `dgg` and `add` would return true.

```

isIsomorphic("egg", 'add'); // true
isIsomorphic("paper", 'title'); // true
isIsomorphic("kick", 'side'); // false

function isIsomorphic(firstString, secondString) {

  // Check if the same length. If not, they cannot be isomorphic
  if (firstString.length !== secondString.length) return false

  var letterMap = {};

  for (var i = 0; i < firstString.length; i++) {
    var letterA = firstString[i],
        letterB = secondString[i];

    // If the letter does not exist, create a map and map it to the value
    // of the second letter
    if (letterMap[letterA] === undefined) {

```

```

    // If letterB has already been added to letterMap, then we can say: they are not isomorphic.
    if(secondString.indexOf(letterB) < i){
        return false;
    } else {
        letterMap[letterA] = letterB;
    }
} else if (letterMap[letterA] !== letterB) {
    // Else if letterA already exists in the map, but it does not map to
    // letterB, that means that A is mapping to more than one letter.
    return false;
}
}
// If after iterating through and conditions are satisfied, return true.
// They are isomorphic
return true;
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/mRZgaj?editors=0012>

[↑ back to top](#)

Stacks and Queues

- **3.1 Implement enqueue and dequeue using only two stacks**

```

var inputStack = []; // First stack
var outputStack = []; // Second stack

// For enqueue, just push the item into the first stack
function enqueue(stackInput, item) {
    return stackInput.push(item);
}

function dequeue(stackInput, stackOutput) {
    // Reverse the stack such that the first element of the output stack is the
    // last element of the input stack. After that, pop the top of the output to
    // get the first element that was ever pushed into the input stack
    if (stackOutput.length <= 0) {
        while(stackInput.length > 0) {
            var elementToOutput = stackInput.pop();
            stackOutput.push(elementToOutput);
        }
    }

    return stackOutput.pop();
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/mRYYZV?editors=0012>

- **3.2 Create a function that will evaluate if a given expression has balanced parentheses -- Using stacks** In this example, we will only consider "{}" as valid parentheses. {}{} would be considered balancing. {{{}} is not balanced

```

var expression = "{{{}}{}}"
var expressionFalse = "{{}}{}";

isBalanced(expression); // true
isBalanced(expressionFalse); // false
isBalanced(""); // true

function isBalanced(expression) {
    var checkString = expression;
    var stack = [];

    // If empty, parentheses are technically balanced
    if (checkString.length <= 0) return true;

    for (var i = 0; i < checkString.length; i++) {

```

```

    if(checkString[i] === '{') {
        stack.push(checkString[i]);
    } else if (checkString[i] === '}') {
        // Pop on an empty array is undefined
        if (stack.length > 0) {
            stack.pop();
        } else {
            return false;
        }
    }
}

// If the array is not empty, it is not balanced
if (stack.pop()) return false;
return true;
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/egaawj?editors=0012>

[↑ back to top](#)

Recursion

- **4.1** Write a recursive function that returns the binary string of a given decimal number Given 4 as the decimal input, the function should return 100

```

decimalToBinary(3); // 11
decimalToBinary(8); // 1000
decimalToBinary(1000); // 1111101000

function decimalToBinary(digit) {
    if(digit >= 1) {
        // If digit is not divisible by 2 then recursively return proceeding
        // binary of the digit minus 1, 1 is added for the leftover 1 digit
        if (digit % 2) {
            return decimalToBinary((digit - 1) / 2) + 1;
        } else {
            // Recursively return proceeding binary digits
            return decimalToBinary(digit / 2) + 0;
        }
    } else {
        // Exit condition
        return '';
    }
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/OWYYKb?editors=0012>

- **4.2** Write a recursive function that performs a binary search

```

function recursiveBinarySearch(array, value, leftPosition, rightPosition) {
    // Value DNE
    if (leftPosition > rightPosition) return -1;

    var middlePivot = Math.floor((leftPosition + rightPosition) / 2);
    if (array[middlePivot] === value) {
        return middlePivot;
    } else if (array[middlePivot] > value) {
        return recursiveBinarySearch(array, value, leftPosition, middlePivot - 1);
    } else {
        return recursiveBinarySearch(array, value, middlePivot + 1, rightPosition);
    }
}

```

View on Codepen: <http://codepen.io/kennymkchan/pen/ygWWmK?editors=0012>

[↑ back to top](#)

Numbers

- **5.1** Given an integer, determine if it is a power of 2. If so, return that number, else return -1. (0 is not a power of two)

```
isPowerOfTwo(4); // true
isPowerOfTwo(64); // true
isPowerOfTwo(1); // true
isPowerOfTwo(0); // false
isPowerOfTwo(-1); // false

// For the non-zero case:
function isPowerOfTwo(number) {
  // `&` uses the bitwise n.
  // In the case of number = 4; the expression would be identical to:
  // `return (4 & 3 === 0)`
  // In bitwise, 4 is 100, and 3 is 011. Using &, if two values at the same
  // spot is 1, then result is 1, else 0. In this case, it would return 000,
  // and thus, 4 satisfies are expression.
  // In turn, if the expression is `return (5 & 4 === 0)`, it would be false
  // since it returns 101 & 100 = 100 (NOT === 0)

  return number & (number - 1) === 0;
}

// For zero-case:
function isPowerOfTwoZeroCase(number) {
  return (number !== 0) && ((number & (number - 1)) === 0);
}
```

View on Codepen: <http://codepen.io/kennymkchan/pen/qRGGeG?editors=0012>

[↑ back to top](#)

Javascript

- **6.1** Explain what is hoisting in Javascript

Hoisting is the concept in which Javascript, by default, moves all declarations to the top of the current scope. As such, a variable can be used before it has been declared. Note that Javascript only hoists declarations and not initializations

- **6.2** Describe the functionality of the `use strict;` directive

the ``use strict`` directive defines that the Javascript should be executed in ``strict mode``. One major benefit that strict mode provides is that it prevents developers from using undeclared variables. Older versions of javascript would ignore this directive declaration

```
// Example of strict mode
"use strict";

catchThemAll();
function catchThemAll() {
  x = 3.14; // Error will be thrown
  return x * x;
}
```

- **6.3** Explain event bubbling and how one may prevent it

Event bubbling is the concept in which an event triggers at the deepest possible element, and triggers on parent elements in nesting order. As a result, when clicking on a child element one may exhibit the handler of the parent activating.

One way to prevent event bubbling is using ``event.stopPropagation()`` or ``event.cancelBubble`` on IE < 9

- **6.4 What is the difference between `==` and `===` in JS?**

``===`` is known as a strict operator. The key difference between ``==`` and ``===`` is that the strict operator matches for both value and type, as opposed to just the value.

```
// Example of comparators
0 == false; // true
0 === false; // false

2 == '2'; // true
2 === '2'; // false
```

- **6.5 What is the difference between `null` and `undefined`**

In Javascript, `null` is an assignment value, and can be assigned to a variable representing that it has no value. `Undefined`, on the other hand, represents that a variable has been declared but there is no value associated with it

- **6.6 How does `prototypal inheritance` differ from `classical inheritance`**

In classical inheritance, classes are immutable, may or may not support multiple inheritance, and may contain interfaces, final classes, and abstract classes. In contrast, prototypes are much more flexible in the sense that they may be mutable or immutable. The object may inherit from multiple prototypes, and only contains objects.

[↑ back to top](#)