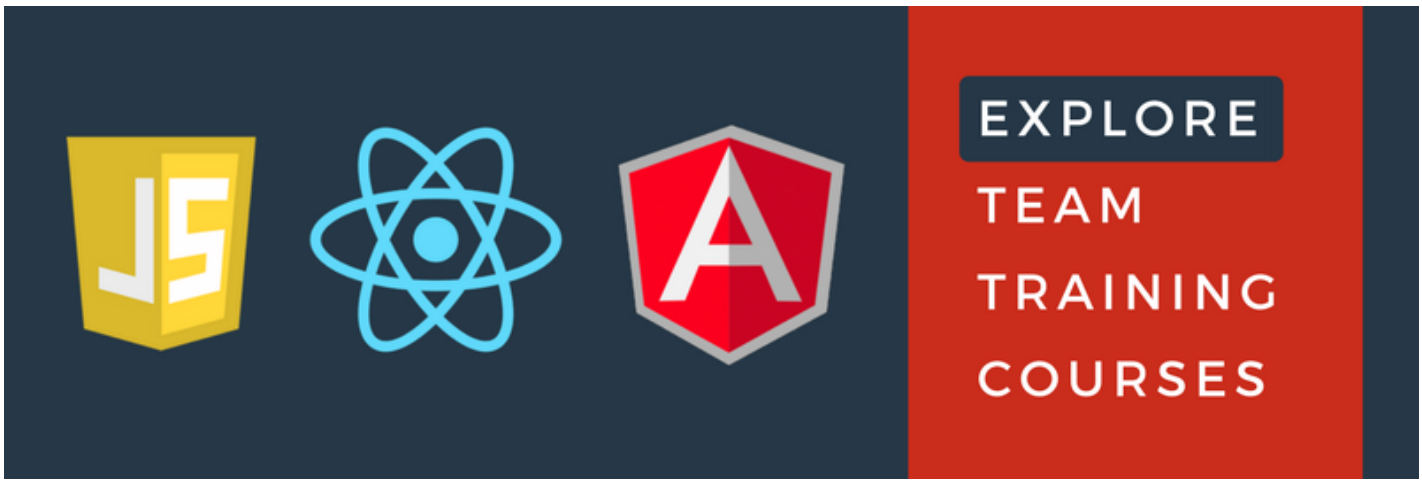
Finding the Most Frequent String in



In the following tutorial, I'll show you how to write a function that takes an array of strings and returns the most frequent string(s) in that array. First I'll go through the basic concepts and setup, we'll test it to make sure it works, then I'll explore a few ways in which you can handle bad data and errors. We'll also consider some ways to make the outputted data more user friendly. Let's get started!

We'll start with a pure function, one that takes an array of strings as input and returns another array as output (the outputted array will ultimately contain the string(s) that occur most frequently in the input). The setup looks like this:

```
1    function mostFreqStr (str) {
2       var which = [];
3       return which;
4     }
```

Next we'll define an object that we'll fill with data, using the strings in the array as keys and the number of times those strings occur as values. We'll also add in a

`mostFreq` variable that will be used to keep track of how many times the most frequent string(s) occur(s). The code now looks like this:

```
function (arr) {
  var obj = {}, mostFreq = 0, which = [];
  return which;
}
```

Now we'll get in to the meat of the function, the processing logic. I've chosen to use JavaScript's native `forEach` method that is an inherent to all arrays. If for some reason you wanted to use a for or while loop, you could do that as well. Here's what the `forEach` approach looks like using ECMAScript 6 syntax (don't worry, we'll discuss the logic involved right after looking at the code):

```
arr.forEach(ea => {
  if (!obj[ea]) {
    obj[ea] = 1;
  } else {
    obj[ea]++;
  }

  if (obj[ea] > mostFreq) {
    mostFreq = obj[ea];
    which = [ea];
  } else if (obj[ea] === mostFreq) {
    which.push(ea);
  }
});
```

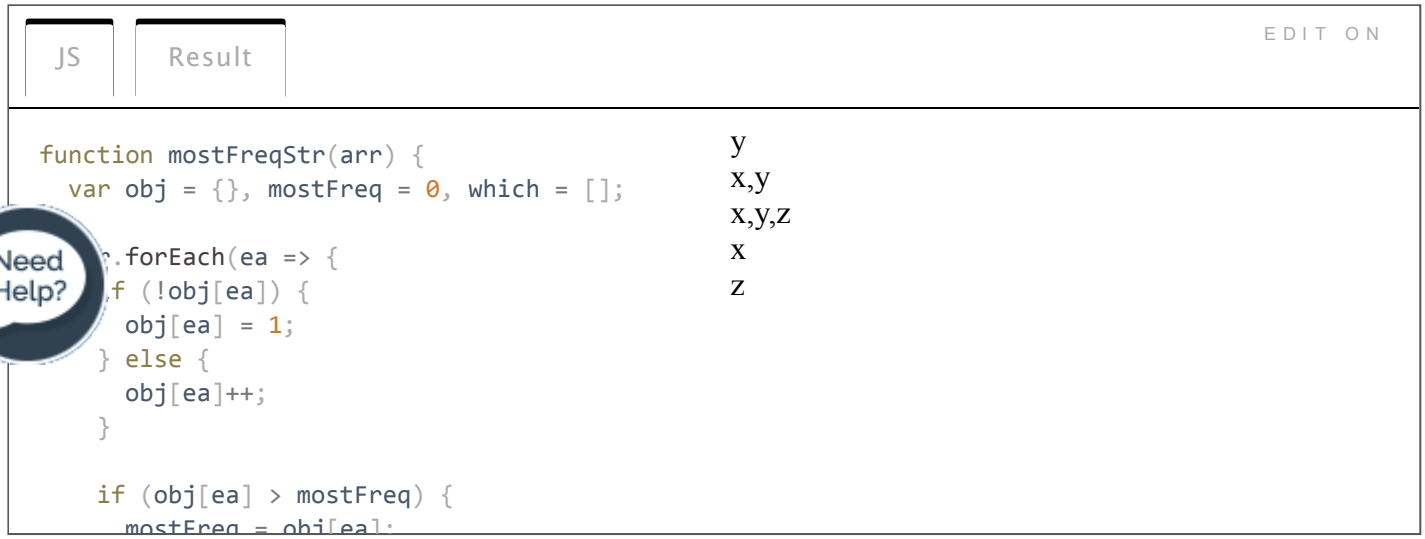
The logic is this: the `ea` variable represents the current array element on each iteration of `forEach`. The first if statement asks *Does the string represented by ea exist as a key in obj?* If the string is not the name of a key in `obj` (i.e., if `obj[ea] === undefined` evaluates to `true`), then we declare and initialize in `obj` the string represented by `ea`. We initialize it to a value of `1`. The value 1 here represents the number of times we've encountered the string in `arr`. On the other hand, If the string is already the name of a key in `obj`, then we increment that key's value by 1 (i.e., we add one more to the count of how many times we've seen the string in question in `arr`).

The second `if` block there is used to compare values of the keys in obj (i.e., the strings in `arr`) to the current most frequent count (i.e., the value of `mostFreq`). `mostFreq` was initialized equal to `0`, so if the value of `ea` in obj is greater than 0 (as it will be as for any string encountered in `arr`), then `mostFreq` is updated to the value of `obj[ea]`. Additionally, the `which` array gets added to it the current value of `ea` (i.e., the current string in our iteration through `arr`). The `else if` portion of the second `if` block accounts for the condition where two or more strings in `arr` share the current value of `mostFreq`. If two or more strings in `arr` share the condition of being most frequent in `arr`, then the latter string(s) encountered is/are pushed to the `which` array so that they can equitably share the title.

The final step in our function is to return the newly populated `which` array. The entirety of the code looks like this:

```
function mostFreqStr (arr) {
  var obj = {}, mostFreq = 0, which = [];

  arr.forEach(ea => {
    if (!obj[ea]) {
      obj[ea] = 1;
    } else {
      obj[ea]++;
    }

    if (obj[ea] > mostFreq) {
      mostFreq = obj[ea];
      which = [ea];
    } else if (obj[ea] === mostFreq) {
      which.push(ea);
    }
  });

  return which;
}
```

Here it is live with a few tests so that we know it's working:



But let's say we want to make it a little more user friendly with respect to the output. We can convert the `which` array to a string by using Array's `toString` method. We can then append a sentence (using ES6 template literals) in order to clarify the output. Like this:

```
1     which = `"${which.toString()}" is the most frequent string
  in the array.`
```

Examples of input and output:

```
1     console.log(mostFreqStr(["x", "x", "y", "y", "y", "z"]));
2     //"y" is the most frequent string in the array.
3
4     console.log(mostFreqStr(["x", "x", "x", "y", "y", "y",
  "z"]));
5     //"x,y" is the most frequent string in the array.
```

Hmm... that second one doesn't look quite right. We can write in some logic to handle our output sentence's grammar when there's more than one most frequent string. I'm going to use Array's `join` method in order to help us out here. Here's the live version:

JS | Result

EDIT ON

```
function mostFreqStr(arr) {
  var obj = {}, mostFreq = 0, which = [];

  arr.forEach(ea => {
    if (!obj[ea]) {
      obj[ea] = 1;
    } else {
      obj[ea]++;
    }

    if (obj[ea] > mostFreq) {
      mostFreq = obj[ea];
```


```
"y" is the most frequent string in the array.
"x" and "y" are the most frequent strings in the array.
```

There we go!

Now… for one final consideration… error handling. What if someone passes in some bad data? Or let's say someone tries to pass in two arrays instead of just one; or perhaps an array that contains an element that isn't a string. What someone tries to pass in data that isn't an array at all? Let's write some logic to handle these erroneous conditions. Live example:

JS | Result

EDIT ON

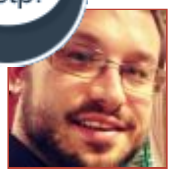
```
console.clear();

function mostFreqStr(arr) {
  //ERROR HANDLING
  //more than one argument passed
  if (arguments.length > 1) {
    return "Sorry, you may only pass one
array of strings to mostFreqStr."
  }
  //the argument is not an array OR if it's
empty
  if (!Array.isArray(arr) || arr.length < 1)
```

```
Sorry, you may only pass one array of strings to mostFreqStr.
Sorry, you may only pass an array of strings to mostFreqStr.
Sorry, you may only pass an array of strings to mostFreqStr.
Sorry, you may only pass an array of strings to mostFreqStr.
Sorry, element at index 2 is not a string.
"y" is the most frequent string in the array.
"x" and "y" are the most frequent strings in the
```

And there we have it! A function that takes an array of strings as input, checks for and handles potential errors, and ultimately returns a user friendly response indicating the most common string(s) in the array. Please feel free to comment and share. Thank you for reading!

Share this:

se Allen



Chase is a self-taught HTML, CSS, and JavaScript enthusiast. He enjoys creating web pages and web applications. His interests include art, music, movies, food, tech, and just general musings.

October 21, 2016 by Chase Allen in

core language, ES5, es6, JavaScript,
JavaScript

Comments

Jake October 23, 2016 Reply

Nice article, that was my first guess on how to do it (based on the title)! The error handling part also drives home how useful Typescript is moving forward, as you get both ES6 features and type checking be default 🙂

Comment

Name

Email

Add Comment

Write for