Springboard

Rest / Spread Operator

Goals

- Understand what the rest operator does
- Understand what the spread operator does
- Use spread to copy arrays and objects
- Use rest to gather remaining arguments in an array

What we used to do

- In JS, every function created using the ***function*** keyword has access to a special keyword called ***arguments***

```
function displayArguments(){ console.log("The first argument is", arguments[0]) return
`You passed in ${arguments.length} arguments!` } displayArguments(1,2) // "The first
argument is 1" displayArguments() // "The first argument is undefined"
```

You *cannot* use ***arguments*** with arrow functions

The problem

- Unfortunately, ***arguments*** is not an *actual* array, it is an "array-like-object"
- It has a length property and can be accessed at a specific index, but does not have build in array methods like map, filter, etc.

```
function doubleArgs(){ return arguments.map(function(arg){ return arg * 2 }) }
doubleArgs(1,2) // Uncaught TypeError: arguments.map is not a function
```

So what did we do?

- We turned the array-like-object into an array, by making a copy of an array and setting the target of the copy to be the arguments array-like-object

```
function doubleArgs(){ let arrayFromArguments = [].slice.call(arguments) return
arrayFromArguments.map(function(arg){ return arg * 2 }) } doubleArgs(1,2) // [2, 4]
```

Kind of gross...

Introducing Rest and Spread!

Rest / Spread Operator

- They are both the "same" operator and use three dots ...
- When the three dots are part of a function definition, we call the operator the "rest" operator
- The rest operator is the last parameter defined in a function and will evaluate to an array of all additional arguments passed in
- Let's look at that first!

```
// we can call the value after the ... whatever we want function
displayArguments(...restOfArgs){ console.log("The first argument is", restOfArgs[0])
return `You passed in ${restOfArgs.length} arguments!` } displayArguments(1,2,3); // "You
passed in 3 arguments!"
```

This looks exactly like what we had before, but instead of an "array-like-object", we have an array.

We can now use methods like ***map***, ***filter***, etc without having to make a copy of the array-like-object first.

```
function sumMany(...nums) { return nums.reduce((a, b) => a + b, 0); } sumMany(5, 10); //
15 sumMany(10, 10, 10, 10, 10, 10, 10, 10, 10); // 90 sumMany(1); // 1
```

`...nums` collects additional arguments into single ***nums*** array.

Collecting Remaining Arguments

You can also specify several named parameters and collect the rest.

```
function oneOrMoreArguments(first, ...more) { console.log(first); more.forEach(arg => {
console.log(arg); }); }
```

Here, ***first*** will be first item and ***more*** will be array of everything else.

Spread Operator

The `...` syntax, in a different context, is called the *spread operator*.

For example: when calling a function, you can “spread out” array elements:

```
function takesFour(one, two, three, four) { console.log(one); console.log(two);
console.log(three); console.log(four); } const names = ['Mary', 'Colt', 'Angela', 'Abe'];
takesFour(...names); // Mary // Colt // Angela // Abe
```

Spread Operator for copies

You can use the same operator to copy over pieces of an array into a new array. This is a shorter version of using concat or writing your own loops

```
const nums = [1,2,3] const newNums = [...nums, 4, 5, 6] const newNumsFirst = [4, 5, 6,
...nums] const newNumsInMiddle = [0, ...nums, 4]
```

Spreading Objects

You can use the same operator to copy over pieces of an object into a new object.

```
const whiskeyTheDog = { name: 'Whiskey', species: 'canine', cool: true, }; // make a new
dog but override the 'name' key const gandalfTheDog = { ...whiskeyTheDog, name: 'Gandalf'
}; console.log(gandalfTheDog); /* { name: 'Gandalf', species: 'canine', cool: true, } */
```

Spreading Objects Warning

This is a newer feature to JavaScript so it may not work in older browsers - specifically Internet Explorer and Microsoft Edge.

Spreading and copies

- Using spread will only make a “shallow” copy.
- If there are nested objects, they will still contain the same reference
- For a deep copy, use libraries like lodash

```
let odds = [1,3,5] let evens = [2,4,6] let collection = [odds, evens] let copy =
[...collection] console.log(collection[0][0]) // 1 copy[0][0] = "REPLACED!"
console.log(collection[0][0]) // "REPLACED"
```

Where this is commonly used

For built in methods like Math.max that do not accept an array

```
Math.max([1,2,3]) // NaN Math.max(...[1,2,3]) // 3
```

Spreading objects and arrays is incredibly common in React and Redux and other modern front-end frameworks!