Array/Object spread operator

Assume you have the following object:

```
const adrian = {
  fullName: 'Adrian Oprea',
  occupation: 'Software developer',
  age: 31,
  website: 'https://oprea.rocks'
};
```

Let's assume you want to create a new object(person) with a different name and website, but the same occupation and age.

You could do this by specifying only the properties you want and use the spread operator for the rest, like below:

```
const bill = {
  ...adrian,
  fullName: 'Bill Gates',
  website: 'https://microsoft.com'
};
```

What the code above does, is to spread over the `adrian` object and get all its properties, then overwrite the existing properties with the ones we're passing. It copies the properties of the `adrian` object, over to the newly created object, and then explicitly overwrites `firstName` and `webSite`. Think of this spread thing as extracting all the individual properties one by one and transferring them to the new object.

In this case, since we specified the `fullName` and `website` properties after the spread operator kicked in, the JavaScript engine is smart enough to know that we want to overwrite the original values of those properties that are coming from the original object.

It's a similar situation with arrays. Except that instead of spreading over keys and values, the operator spreads indexes and values. Unlike object spread, where you won't have duplicate properties because that's just how JavaScript objects work

(you can't have an object with two `fullName` properties), with arrays you may end up with duplicate values, if you plan to implement something similar to our object example.

This means that the code below will result in you having an array with duplicate elements.

```
const numbers1 = [1, 2, 3, 4, 5];
const numbers2 = [ ...numbers1, 1, 2, 6,7,8]; // this will be [1, 2, 3, 4, 5, 1, 2, 6, 7, 8]
```

Think of it as a replacement for Array.prototype.concat.

Rest operator

When used within the signature of a function, where the function's arguments should be, either replacing the arguments completely or alongside the function's arguments, the three dots are also called the rest operator.

When it is used like that, the rest operator enables the developer to create functions that can take an indefinite number of arguments, also called functions of variable arity or variadic functions.

Here's the simplest example of such a function. Let's assume you want to create a function that calculates the sum of all its arguments. Note that it's not the sum of two, three or four numbers but the sum of all the numbers the function would receive as arguments.

Here is a naive implementation, using the rest operator

```
function sum(...numbers) {
      return numbers.reduce((accumulator, current) => {
            return accumulator += current;
      });
};

sum(1,2) // 3
sum(1,2,3,4,5) // 15
```

The simplest explanation would be that the rest operator takes the arguments that a function receives and dumps them into a real array that you could later use.

You might argue that you can do this by requesting the user to pass an array of numbers. That's technically doable but poor UX for your API, since users expect to call a sum function with plain numbers, not a list of numbers.

You might also argue that you could use the `arguments` array. That's also true but be careful, `arguments` is not a real array but an array-like object(an object with a length property). It actually looks like this, for the first call of our sum function, in the previous example:

```
{
  '0': 1,
  '1': 2,
  'length': 2
}
```

To manipulate this object and use array methods on it such as reduce, from my previous example, you'd have to do
the `Array.prototype.slice.call(arguments)` thing.

Here's the previous function written using `Array#slice`:

```
function sum() {
      const args = Array.prototype.slice.call(arguments);
      return args.reduce((accumulator, current) => {
            return accumulator += current;
      });
}


sum(1,2) // 3
sum(1,2,3,4,5) // 15
```

This code works just like the spread version but with some major differences. For starters, V8 compiler optimizations are not possible. passing

the `arguments` object to most functions will trigger a leak due to aliasing. Arguments is an object and will be passed by reference and so V8 is unable to pin down the type and shape of the elements inside the `arguments` object, since they can be overwritten by the functions they are passed to.

The biggest problem is that I used to do it myself. It's just so simple to write. The safer versions involved creating the array in-place, which was less elegant.

The bottom line is that the code above is just smartass code prone to confuse your junior colleagues. Don't do it. You need to look no further than the Mozilla Developer Network for options.

This should be everything you need to know to be productive with the rest/spread operator in JavaScript.