Using the Spread Operator and Rest Parameter Syntax

When writing functions in JavaScript you gain a certain flexibility that other programming languages don't allow. As we have previously covered, JavaScript functions will happily take fewer arguments than specified, or more arguments than specified. This flexibility can be taken advantage of by using the *spread operator*and *rest parameter*syntax.

When you finish this reading, you should be able to:

1. Use rest parameter syntax to accept an arbitrary number of arguments inside a function.
2. Use spread operator syntax with both Objects and Arrays.

Accepting arguments

Before we jump into talking about using new syntax let's quickly recap on what we already know about functions.

Functions with fewer arguments than specified

As we've previously covered, JavaScript functions can take fewer arguments than expected. If a parameter has been declared when the function itself was defined, then the default value of that parameter is `undefined`.

Below is an example of a function with a defined parameter both with and without an argument being passed in:

```
function tester(arg) {
  return arg;
}

console.log(tester(5)); // => prints: 5
console.log(tester()); // => prints: undefined
```

Always keep in mind that a function will still run even if it has been passed no arguments at all.

More arguments than specified

JavaScript functions will also accept more arguments than were previously defined by parameters.

Below is an example of a function with extra arguments being passed in:

```
function adder(num1, num2) {
  let sum = num1 + num2;
  return sum;
}

// adder will assign the first two parameters to the passed in arguments
// and ignore the rest
console.log(adder(2, 3, 4)); // => 5
console.log(adder(1, 5, 19, 100, 13)); // => 6
```

Utilizing Rest Parameters

We know that JavaScript functions can take in extra arguments - but how do we access those extra arguments? For the above example of the `adder`function:

how could we add all incoming arguments - even the ones we didn't define as parameters?

Rest parametersyntax allows us to capture all of a function's incoming arguments into an array. Let's take a look at the syntax:

```
// to use the rest parameter you use ... then the name of the array
// the arguments will be contained within
function tester(...restOfArgs) {
  // ...
}
```

In order to use *rest parameter*syntax a function's last parameter can be prefixed with `...`which will then cause all remaining arguments to be placed within an array. Only the **last parameter**can be a *rest parameter*.

Here is a simple example using *rest parameter*syntax to capture all incoming arguments into an array:

```
function logArguments(...allArguments) {
  console.log(allArguments);
}

logArguments("apple", 15, 3); // prints: ["apple", 15, 3]
```

For a more practical example let's expand on our `adder`function from before using *rest parameter syntax*:

```
function adder(num1, ...otherNums) {
  console.log("The first number is: " + num1);
  let sum = num1;

  // captures all other arguments into an array and adds them to our sum
  otherNums.forEach(function(num) {
    sum += num;
  });

  console.log("The sum is: " + sum);
}

adder(2, 3, 4);
// prints out:
// The first number is: 2
// The sum is: 9
```

To recap - we can use the *rest parameter*to capture a function's incoming arguments into an array.

Utilizing Spread Syntax

Let's now talk about a very interesting and useful operator in JavaScript. In essence, the *spread operator*allows you to break down a data type into the elements that make it up.

The *spread operator*has two basic behaviors:

1. Take a data type (i.e. an array, an object) and *spread*the values of that type where **elements**are expected
2. Take an iterable data type (an array or a string) and *spread*the elements of that type where **arguments**are expected.

Spreading elements

The spread operator is very useful for *spreading*the values of an array or object where comma-separated elements are expected.

Spread operatorsyntax is very similar to rest parameter syntax but they do very different things:

```
let numArray = [1, 2, 3];

// here we are taking `numArray` and *spreading* it into a new array where
// comma separated elements are expected to be
let moreNums = [...numArray, 4, 5, 6];

> moreNums
// => [1, 2, 3, 4, 5, 6]
```

In the above example you can see we used the spread operator to *spread* the values of `numArray` into a new array. Previously we used the `concat` method for this purpose, but we can now accomplish the same behavior using the *spread operator*.

We can also *spread* Objects! Using the spread operator you can *spread* the `key-value` pairs from one object and into another new object.

Here is an example:

```
let colors = { red: "scarlet", blue: "aquamarine" };
let newColors = { ...colors };

> newColors
// { red: "scarlet", blue: "aquamarine" };
```

Just like we previously showed with arrays, we can use this spread behavior to *merge* objects together:

```
let colors = { red: "scarlet", blue: "aquamarine" };
let colors2 = { green: "forest", yellow: "sunflower" };

let moreColors = { ...colors, ...colors2 };

> moreColors
// {red: "scarlet", blue: "aquamarine", green: "forest", yellow: "sunflower"}
```

Spreading arguments

The other scenario in which *spread* proves useful is *spreading* an iterable data type into the passed in arguments of a function. To clarify, when we say *iterable* data types we mean arrays and string, **not Objects**.

Here is a common example of spreading an array into a function's arguments:

```
function speak(verb, noun) {
  return "I like to go " + verb + " with " + noun + ".";
}

const words = ["running", "Jet"];

console.log(speak("running", "Jet")); // => I like to go running with Jet.
console.log(speak(...words)); // => I like to go running with Jet.
```

Using *spread* allowed us to pass in the `words` array, which was then broken down into the separate parameters of the `speak` function. The spread operator allows you to pass an array as an argument to a function and the values of that array be will be *spread* to fill in the separate parameters.

What you learned

Rest parameter syntax may look like *spread operator* syntax but they are pretty much opposites[1]:

1. Spread 'expands' a data type into its elements
2. Rest collects multiple elements and 'condenses' them into a single data type.

What this reading covered:

- JavaScript functions can accept any number of arguments

- Using rest parameter syntax we can capture the arguments of a JavaScript function in an array
- Using *spread operator*syntax to spread iterable data types where arguments or values are expected
    - Using the spread operator to spread an array and object into their separate elements