Parameters

In ES5, we had varying ways to handle functions which needed **default values**, **indefinite arguments**, and **named parameters**. With ES6, we can accomplish all of this and more using more concise syntax.

Default Parameters

```
function addTwoNumbers(x, y) {
    x = x || 0;
    y = y || 0;
    return x + y;
}
```

In ES6, we can simply supply default values for parameters in a function:

```
function addTwoNumbers(x=0, y=0) {
    return x + y;
}
```

```
addTwoNumbers(2, 4); // 6
addTwoNumbers(2); // 2
addTwoNumbers(); // 0
```

Rest Parameters

In ES5, we handled an indefinite number of arguments like so:

```
function logArguments() {
    for (var i=0; i < arguments.length; i++) {
        console.log(arguments[i]);
    }
}
```

Using the **rest** operator, we can pass in an indefinite amount of arguments:

```
function logArguments(...args) {
    for (let arg of args) {
        console.log(arg);
    }
}
```

Named Parameters

One of the patterns in ES5 to handle named parameters was to use the **options object** pattern, adopted from jQuery.

```
function initializeCanvas(options) {
    var height = options.height || 600;
    var width  = options.width  || 400;
    var lineStroke = options.lineStroke || 'black';
}
```

We can achieve the same functionality using destructuring as a formal parameter to a function:

```
function initializeCanvas(
    { height=600, width=400, lineStroke='black'}) {
        // Use variables height, width, lineStroke here
    }
```

If we want to make the entire value optional, we can do so by destructuring an empty object:

```
function initializeCanvas(
    { height=600, width=400, lineStroke='black'} = {}) {
        // ...
    }
```

Spread Operator

In ES5, we could find the max of values in an array by using the `apply` method on `Math.max` like this:

```
Math.max.apply(null, [-1, 100, 9001, -32]); // 9001
```

In ES6, we can now use the spread operator to pass an array of values to be used as parameters to a function:

```
Math.max(...[-1, 100, 9001, -32]); // 9001
```

We can concat array literals easily with this intuitive syntax:

```
let cities = ['San Francisco', 'Los Angeles'];
let places = ['Miami', ...cities, 'Chicago']; // ['Miami', 'San Francisco', 'Los Angeles', 'Chicago']
```