The Spread Operator and Rest Parameters

WE'LL COVER THE FOLLOWING

Two of my favorite new features in ES6 are the Spread Operator and Rest Parameters. These two features really allow us to create powerful operations that might have required more work, and added more confusion than needed. Let's first take a look at Rest Parameters.

Rest Parameters

From time to time you might want to write a function that would take an unknown number of arguments. JavaScript has the ability to do this through the `arguments` keyword. Take a `sum()` function, say it takes only two numbers

```
function sum(a,b) {
  return a + b;
}
```

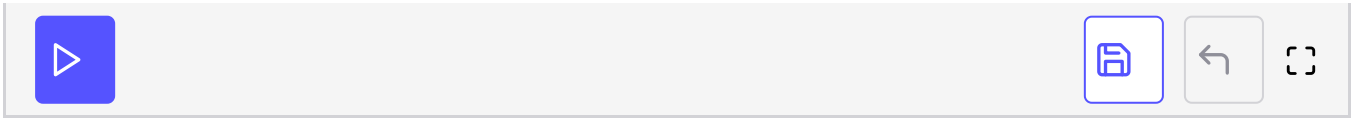
Maybe we want this `sum()` to take any number of arguments? We can tell how many arguments have been passed to a function by using the `arguments` keyword.

```
function sum() {
  console.log(arguments);
}

let total = sum(1,3,4,6); //[1,3,4,6]
```

This is great: it looks like it returns an array for us to work with. We can go ahead and use the `reduce` method to reduce the numbers.

Reduce is a method that accepts a function and runs it for each value in an array. It can be used for much more than just getting the sum of some values, but in our case we will use it just for that!

```
const sum = function() {
  return arguments.reduce( (pre,cur) => {
    return pre + cur;
  });
};

const total = sum(1,2,3,4); //"TypeError: arguments.reduce is not a function
```



There is a small problem, however, as you will see if you run this code. The problem is that `arguments` is not an array. It is an array-LIKE object, meaning that it does not have all the array methods on it. So, how can we solve this problem? We can use the `.call()` method.

call()

`.call()` is a method that allows us to call a function with a provided value for `this`. As a quick example, imagine a simple object that has a `fullName` property.

```
const person = {
  fullName: 'Ryan Christiani'
}
```

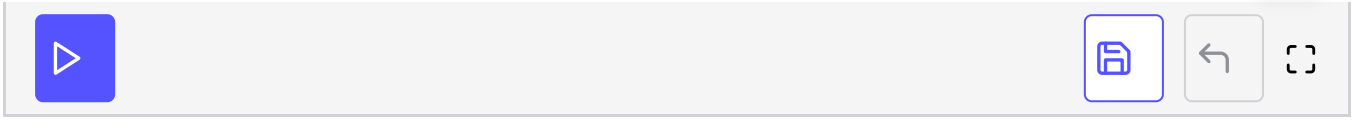
Let's also assume we have a function like this:

```
function sayName() {
  console.log(this.fullName);
}
```

In the function above, `this` will be bound to the window. Using the `.call()` method we can call the `sayName` function and set the `this` context to be our

`person` object!

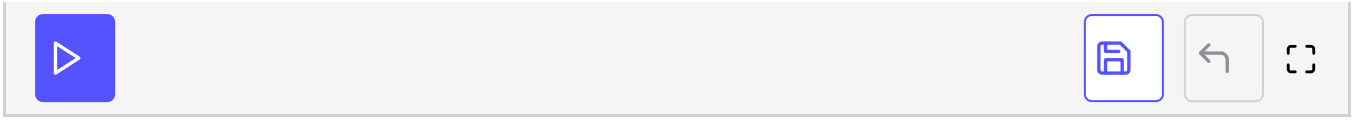
```
sayName.call(person); //Ryan Christiani
```



With this in mind we can use `Array.prototype.reduce`, the method that all Array's get, and use the `.call()` method on it passing `arguments` to get it to work.

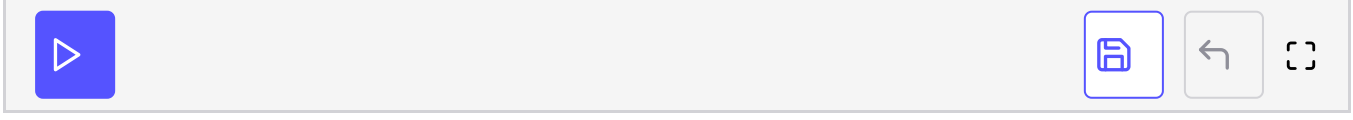
Although this works, and it can be very powerful, it is not great as a beginner, and can be confusing to understand. Rest Parameters allow us to define a function that can accept any number of arguments, with a more straight forward syntax.

```
const sum = function(...numbers) {
    console.log(numbers);
};

function sayName() {
  console.log(this.fullName);
}

sum(1,2,3,4,5,6); //[1,2,3,4,5,6]
```



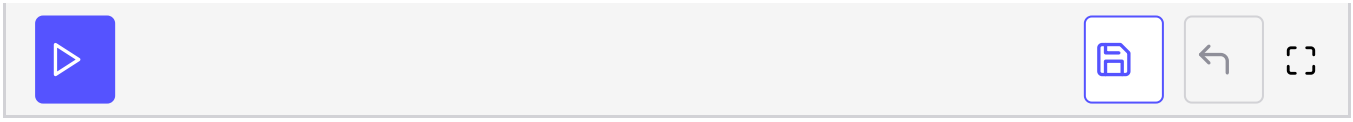
The `...` is the special prefix that both Rest Parameters and the Spread Operator use, as we will see later. The name following the `...` is the parameter name. The great thing about this is that the `numbers` parameter value is now an actual array, and we can call the reduce method on it!

```
const sum = function(...args) {
    return args.reduce(function(prev,curr) {
        return prev + curr
    });
};

console.log(sum(1, 2, 3, 4, 5));
```



There is something that I want to point out: you can use Rest Parameters to just gather the rest of the arguments from a function. Let's write a function that will take a value as the first argument and then multiply the rest of the values from there.

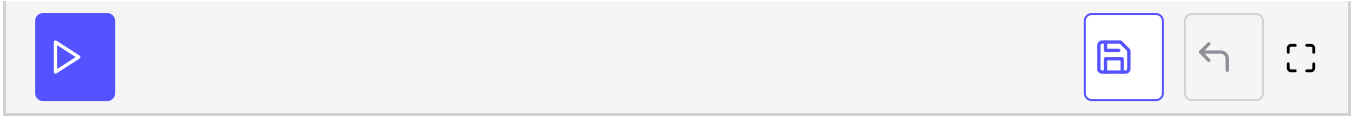
```
let multiply = function(mul,...args) {
  return args.map(function(num) {
    return mul * num;
  });
};

console.log(multiply(3,14,45,6,7,8) );//[42, 135, 18, 21, 24]
```



Spread Operator

As mentioned previously the syntax for the Spread Operator is similar to Rest Parameters: it uses `...` as well. It differs from Rest Parameters, however, in that you need to start with an array value and then you spread it out.

```
const numbers = [4,5,6,7];
console.log(...numbers); // 4 5 6 7
```

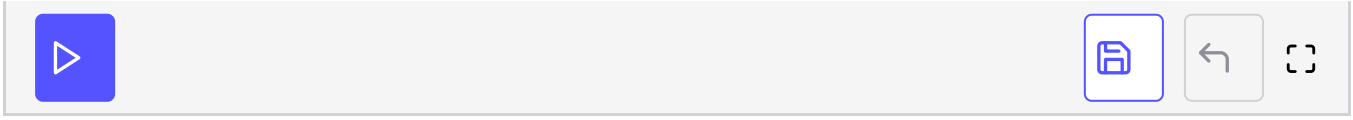


Let's look at a good example of this. Consider `Math.max()`, this method takes any number of arguments and it will return the maximum of them.

```
const max = Math.max(1,23,45,6,7,8);
console.log(max); // 45
```

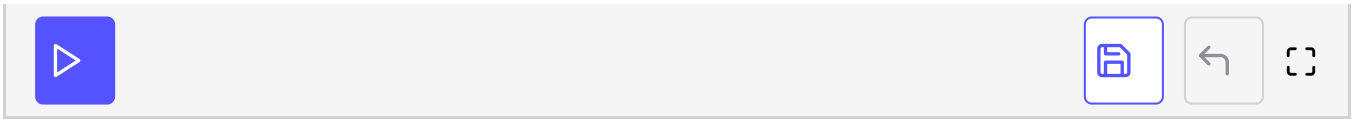
If you have ever tried passing an array to `Math.max` you will know that this does not work.

```
const numbers = [23,46,12,63,23];

console.log(Math.max(numbers)); // NaN
```



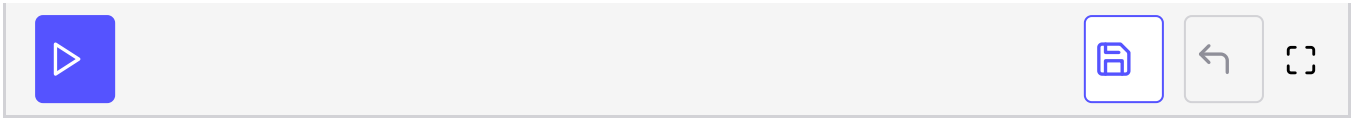
Similar to the `.call()` method, there is a method on functions called `.apply()`. This method calls a function and supplies a context for the `this` keyword, as well as an array that we want to use as the values for a function. With the same `numbers` array from before, we could do something like this.

```
const numbers = [23,46,12,63,23];
const total = Math.max.apply(null,numbers);
console.log(total); //63
```



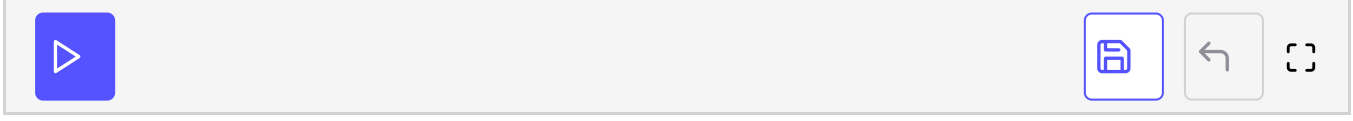
With the Spread Operator we can do what `.apply()` did, but much more cleanly. I showed a student `.apply` recently in a similar situation and it was a little confusing for them. However when I explained spread to them it made more sense! It is basically a better `.apply()` for this situation.

```
const numbers = [23,46,12,63,23];

const total = Math.max(...numbers);

console.log(total);// 63
```



Going further, we can use the Spread Operator as way to concatenate arrays together.

```
let numbers = [2,34,5];
let newNumbers = [74,52,...numbers];

console.log(newNumbers);
// [74, 52, 2, 34, 5]
```



With the ability to take an array of values and spread them out, it really opens up a lot of possibilities for us.