Spread Operator and Rest Parameters

Learn how to to easily combine Array with the spread operator and much more.

WE'LL COVER THE FOLLOWING

The Spread operator

According to MDN:

> **Spread** syntax allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

Combine arrays

```
const veggie = ["tomato","cucumber","beans"];
const meat = ["pork","beef","chicken"];

const menu = [...veggie, "pasta", ...meat];
console.log(menu);
// Array [ "tomato", "cucumber", "beans", "pasta", "pork", "beef", "chicken" ]
```

The `...` is the spread syntax, and it allowed us to grab all the individual values of the arrays veggie and meat and put them inside the array menu. At the same time, we can add a new item in between them.

Copy arrays

The spread syntax is very helpful if we want to create a copy of an array.

```
const veggie = ["tomato","cucumber","beans"];
const newVeggie = veggie;

// this may seem like we created a copy of the veggie array but look now

veggie.push("peas");
console.log(veggie);
// Array [ "tomato", "cucumber", "beans", "peas" ]

console.log(newVeggie);
// Array [ "tomato", "cucumber", "beans", "peas" ]
```

Our new array also changed, but why? Because we did not actually create a copy but we just referenced our old array in the new one. The following is how we would usually make a **copy** of an array in ES5 and earlier

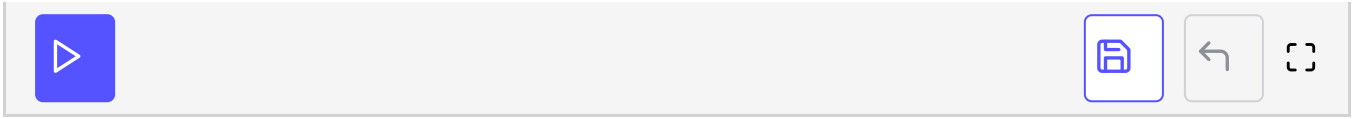
```
const veggie = ["tomato","cucumber","beans"];
// we created an empty array and put the values of the old array inside of it
const newVeggie = [].concat(veggie);
veggie.push("peas");
console.log(veggie);
// Array [ "tomato", "cucumber", "beans", "peas" ]
console.log(newVeggie);
// Array [ "tomato", "cucumber", "beans" ]
```

And this is how we would do the same using the spread syntax:

```
const veggie = ["tomato","cucumber","beans"];
const newVeggie = [...veggie];
```

```
veggie.push("peas");
console.log(veggie);
// Array [ "tomato", "cucumber", "beans", "peas" ]

console.log(newVeggie);
// Array [ "tomato", "cucumber", "beans" ]
```



The syntax for the **spread** operator is the following `...YourArray`. Since we wanted the variable `newVeggie` to be a copy of the array `veggie`, we assigned it to an Array. Inside of it we spread all the values from the variable `veggie` like so: `[...veggie]`.

Spread into a function

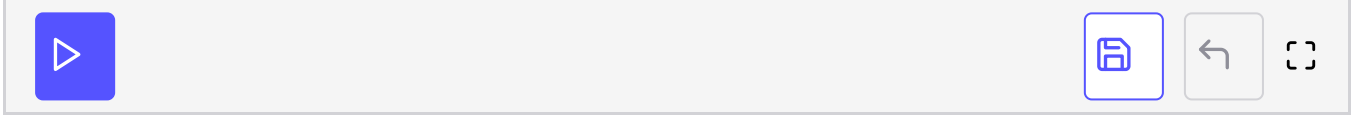
Thanks to the spread operator, we have an easier way of calling a function with an array of arguments.

```
// OLD WAY
function doStuff (x, y, z) {
  console.log(x + y + z);
 }
var args = [0, 1, 2];

// Call the function, passing args
doStuff.apply(null, args);

// USE THE SPREAD SYNTAX

doStuff(...args);
// 3 (0+1+2);
console.log(args);
// Array [ 0, 1, 2 ]
```



As you can see in the example, our `doStuff` function accepts 3 parameters. We are passing them by spreading the array `args` into the function like so: `...args` replacing the need of resorting to use `.apply()`.

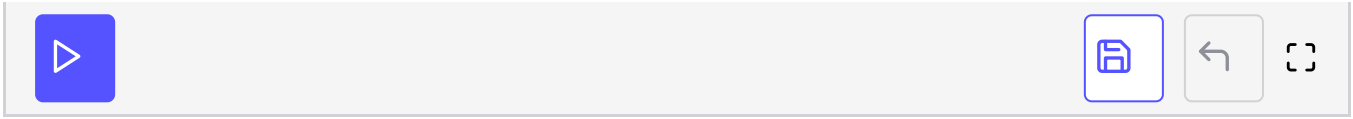
Let's look at another example:

```
const name = ["Alberto", "Montalesi"];
```

```
function greet(first, last) {
  console.log(`Hello ${first} ${last}`);

}

greet(...name);
// Hello Alberto Montalesi
```

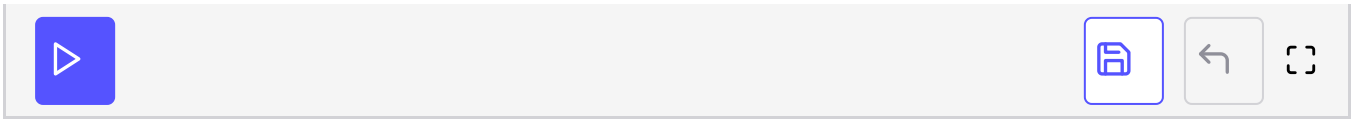


The two values of the array are automatically assigned to the two arguments of our function.

What if we provide more values than arguments?

```
const name = ["Jon", "Paul", "Jones"];

function greet(first, last) {
  console.log(`Hello ${first} ${last}`);
}
greet(...name);
// Hello Jon Paul
```



In the example given above, we provided three values inside the array but we only have two arguments in our function. Therefore, the last one is left out.

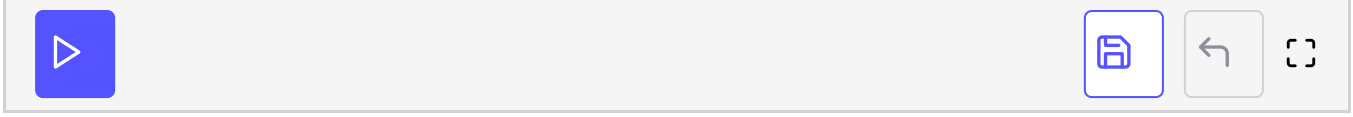
Spread in Object Literals (ES 2018 / ES9)

This feature is not part of ES6, but as we are already discussing this topic, it's worth mentioning that ES2018 introduced the spread operator for objects.

Let's look at an example:

```
let person = {
  name : "Alberto",
  surname: "Montalesi",
  age: 25,
}

let clone = {...person};
console.log(clone);
// Object { name: "Alberto", surname: "Montalesi", age: 25 }
```



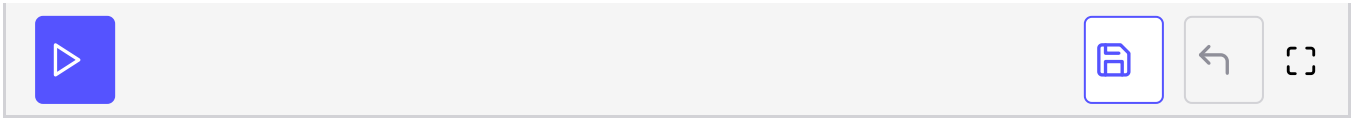
You can read more about ES2018 in the lesson: “ES2018: Async Iteration and more".

The Rest parameter

The **rest** syntax looks exactly the same as the **spread**- 3 dots `...`. But it is quite the opposite of it. **Spread** expands an array, while **rest** condenses multiple elements into a single one.

```
const runners = ["Tom", "Paul", "Mark", "Luke"];
const [first,second,...losers] = runners;

console.log(...losers);
// Mark Luke
```



We stored the first two values inside the `const` first and second and whatever was left we put inside losers using the rest parameter.

Another quiz and coding challenge is next on the docket.