If you can use the spread operator to *spread* an array into multiple elements, then certainly there should be a way to bundle multiple elements back into an array, right?

In fact, there is! It’s called the *rest parameter*, and it’s another new addition in ES6.

**Rest parameter**

The **rest parameter**, also written with three consecutive dots ( ... ), allows you to represent an indefinite number of elements as an array. This can be helpful in a couple of different situations.

One situation is when assigning the values of an array to variables. For example,

**const** order = [20.17, 18.67, 1.50, "cheese", "eggs", "milk", "bread"];

**const** [total, subtotal, tax, ...items] = order;

console.log(total, subtotal, tax, items);

***Prints:****20.17 18.67 1.5 ["cheese", "eggs", "milk", "bread"]*

This code takes the values of the order array and assigns them to individual variables using destructuring (as you saw in the Destructuring section earlier in this lesson). total, subtotal, and tax are assigned the first three values in the array, however, items is where you want to pay the most attention.

By using the rest parameter, items is assigned the *rest* of the values in the array (as an array).

Replay

Mute

Loaded: 100.00%

Remaining Time -0:00

1x

Playback Rate

Picture-in-Picture

Fullscreen

You can think of the rest parameter like the opposite of the spread operator; if the spread operator is like unboxing all of the contents of a package, then the rest parameter is like taking all the contents and putting them back into the package.

**Variadic functions**

Another use case for the rest parameter is when you’re working with variadic functions. **Variadic functions** are functions that take an indefinite number of arguments.

For example, let’s say we have a function called sum() which calculates the sum of an indefinite amount of numbers. How might the sum() function be called during execution?

sum(1, 2);

sum(10, 36, 7, 84, 90, 110);

sum(-23, 3000, 575000);

There’s literally an endless number of ways the sum() function could be called. Regardless of the amount of numbers passed to the function, it should always return the total sum of the numbers.

**Using the arguments object**

In previous versions of JavaScript, this type of function would be handled using the [arguments object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/arguments). The **arguments object** is an array-like object that is available as a local variable inside all functions. It contains a value for each argument being passed to the function starting at 0 for the first argument, 1 for the second argument, and so on.

If we look at the implementation of our sum() function, then you’ll see how the arguments object could be used to handle the variable amount of numbers being passed to it.

**function** **sum**() {

**let** total = 0;

**for**(**const** argument **of** arguments) {

total += argument;

}

**return** total;

}

Now this works fine, but it does have its issues:

1. If you look at the definition for the sum() function, it doesn’t have any parameters.
   * This is misleading because we know the sum() function can handle an indefinite amount of arguments.
2. It can be hard to understand.
   * If you’ve never used the arguments object before, then you would most likely look at this code and wonder where the arguments object is even coming from. Did it appear out of thin air? It certainly looks that way.

**Using the rest parameter**

Fortunately, with the addition of the rest parameter, you can rewrite the sum() function to read more clearly.

**function** **sum**(...nums) {

**let** total = 0;

**for**(**const** num **of** nums) {

total += num;

}

**return** total;

}

This version of the sum() function is both **more concise** and is **easier to read**. Remember, we use the [for...of loop](https://classroom.udacity.com/nanodegrees/nd016/parts/11a45d59-bb81-44a9-be76-042c99e5f051/modules/cbf6deb8-d2cc-4757-b3a9-a1c58a4acd82/lessons/42383e89-ac6a-491a-b7d0-198851287bbe/concepts/f1955923-744a-4906-8f64-1ddcb34c6da2) to loop over any type of data that is iterable. So we'll use for...of here rather than for...in.