**Rest Operator**

Syntactically, it looks exactly the same as spread operator. But it’s function is the exact opposite of the **spread** Operator. If Spread operator expands individual items, then **rest operator collects a bunch of items and puts them into arrays and objects.**Let’s see an example —

const numbers = [1, 2, 3];const [ firstNumber, **...restOfTheNumbers** ] = numbers;console.log(firstNumber, restOfTheNumbers);// 1 [ 2, 3 ]

Using destructuring, we extract the firstNumber. We don’t want individual numbers anymore.**So we *collect*them in the**restOfTheNumbers**array.**

Notice that restOfTheNumbers is an array. Since rest operator collects *items,*it needs to place them in a *container*. **That container is an array when we destructure from an array or strings.**

const **[ firstLetter, ...restOfTheLetters ] = 'Codeburst';**  
console.log(firstLetter, restOfTheLetters);// C [ 'o', 'd', 'e', 'b', 'u', 'r', 's', 't' ]

A few other things to note are —

* All concepts of destructuring apply with rest operator since rest operator is nothing more than a specialised form of destructuring.
* It is always to the left side of the = whereas the spread operator will be on the right side.
* In arrays and objects, the rest operator can only come at the end. That is, this syntax won’t work —

const [ firstLetter, ...restOfTheLetters, **lastLetter** ] = 'Codeburst';

Let’s see how the rest operator behaves with objects with an example —

const details = {  
 firstName: 'Code',  
 lastName: 'Burst',  
 age: 22  
};const **{ age, ...restOfTheDetails }** = details;console.log(age, restOfTheDetails);// 22 { firstName: 'Code', lastName: 'Burst' }