

Spread Operator

Rest Parameter

Índice

1. [Spread Operator](#)
2. [Rest Parameter](#)

1 | Spread Operator

“

Este operador permite **expandir** cada uno de los datos de un **elemento iterable** dentro de otro elemento.



”

Uso y sintaxis

El operador de **propagación** se puede usar sobre cualquier elemento iterable. Nos sirve para copiar y mover datos de un lugar a otro de una forma eficaz.



Spread en arrays

Implementando este operador, podemos **copiar** todos los datos de un array en un **array nuevo**.

```
{ } let clubesUno = ['Boca', 'River', 'Racing'];  
let clubesDos = ['San Lorenzo', 'Lanús', 'Gimnasia'];  
let todosLosClubes = [...clubesUno, ...clubesDos];
```

También podemos **agregar** todos los datos de un array **dentro** de un **array existente**.

```
{ } let parte = ['los', 'cumplás'];  
let oracion = ['Que', ...parte, 'feliz'];
```

Spread en objetos

Implementando este operador, podemos **copiar** todas las propiedades de un objeto **dentro** de otro **objeto existente**.

```
{ } let auto = {marca:'Ferrari', kms:0, anio:2019};  
let corredorUno = {nombre:'Vettel', edad:32, ...auto};  
let corredorDos = {nombre:'Leclerc', edad:21, ...auto};
```

Tanto `corredorUno` como `corredorDos` ahora tienen todas las propiedades que definimos en el objeto `auto` sin tener que definirlas a mano en cada uno de ellos.

Spread y funciones

Implementando este operador, podemos pasarle a una función un array como argumento. El operador `...` se encargará de expandir los datos para que la función los tome como argumentos separados.

Para ejemplificar usaremos el método de JavaScript `Math.min()`, que recibe N cantidad de argumentos y devuelve el menor.

```
{
```

```
let notas = [9.3, 8.5, 3.2, 7, 10];  
Math.min(...notas); // Devuelve 3.2
```


2 | Rest Parameter

“

Utilizado como **último parámetro** de una función nos permite **capturar** cada uno de los **argumentos** adicionales pasados a esa **función**.



”

El parámetro rest

El **parámetro rest** se escribe de la misma manera que el **operador spread** `...`. La diferencia es que se utiliza durante la definición de la función, y no durante su ejecución.

El parámetro rest **generará un array** con todos los argumentos adicionales que se le pasen a la función.

```
{ }  
function miFuncion(param1, param2, ...otros) {  
    return otros;  
}  
miFuncion('a', 'b', 'c', 'd', 'e');  
// retornará ['c', 'd', 'e']
```

El parámetro rest

Implementando el parámetro rest, podemos definir una función que acepte cualquier número de argumentos.

```
{  
  function sumar(...numeros) {  
    // Sabiendo que números es ahora un array utilizamos  
    // el método reduce para obtener la sumatoria  
    return numeros.reduce((acum, num) => acum += num);  
  }  
}
```

```
sumar(1, 4); // devuelve 5
```

```
sumar(13, 6, 8, 12, 23, 37); // devuelve 99
```

El parámetro rest

Como el **parámetro rest** captura todos los argumentos restantes, **siempre debe ser el último parámetro de la función**, de lo contrario, recibiremos un error.

```
{}  
function sumar(...numeros, otroParámetro) {  
    // Utilizamos el método reduce para obtener la suma  
    return numeros.reduce((acum, num) => acum += num);  
}
```

SyntaxError: parameter after rest parameter

DigitalHouse >
Coding School