code
URJC

Tema 2

Spread, Rest y Destructuring

Micael Gallego
micael.gallego@gmail.com
@micael_gallego

EcmaScript 6+

- En 2015 se publicó el estándar EcmaScript 6 (ES6), también conocido como ES2015
- Trajo muchas novedades a **JavaScript**
  - **Lenguaje:** declaración de variables, *arrow functions*, módulos, clases, plantillas en strings, *destructuring*, operador *spread*, operador *rest*
  - **APIs:** Map, Set, Promises…
- El lenguaje ha seguido evolucionando cada año (ES7,ES8,ES9…)

EcmaScript 6+

- Estudiaremos nueva sintaxis de JavaScript que permite manipular los elementos de arrays o objetos de forma independiente para un código más conciso
  - Operador *spread*
  - Parámetro *rest*
  - *Destructuring assignment*

Spread operator

- Se definió en **ES6/ES2015** y se ha mejorado en **ES9/ES2018**
- Se puede traducir como **esparcir** o **extender**
- Se usa para **extraer** los elementos de un **array** (u otros objetos) y expandirlos en **contextos donde se esperan uno o más valores**

arrays strings objetos funciones

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax

Spread con arrays

code URJC

- Crear un nuevo array con elementos de otro array

```
var parts = ['shoulders', 'knees'];
var lyrics = ['head', ...parts, 'and', 'toes'];
// ["head", "shoulders", "knees", "and", "toes"]
```

- Copiar un array (sin copia profunda)

```
var arr = [1, 2, 3];
var arr2 = [...arr]; // like arr.slice()

arr2.push(4);
// arr2 becomes [1, 2, 3, 4]
// arr remains unaffected
```

Spread con arrays

- Concatenar arrays

```
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];

var arr3 = [...arr1, ...arr2];
// arr1 is now [0, 1, 2, 3, 4, 5]
```

- Insertar elementos de un array en otro

```
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];

arr1 = [...arr2, ...arr1];
// arr1 is now [3, 4, 5, 0, 1, 2]
```

Spread con strings

- Crear un array con los caracteres de un string

```
var name = "Pepe";
var letters = [...name];
// [ 'P', 'e', 'p', 'e' ]
```

Spread con objetos

- Crear un nuevo objeto con elementos de otro

```
var obj1 = { a:1, b:2 };
var obj2 = { c:3, d:4, ...obj1};
// {a:1, b:2, c:3, d:4}
```

- Los últimos elementos sobreescriben los anteriores

```
const obj = { a: 'a', b: 'b', c: 'c' };
{ a: 1, b: null, c: 3, ...obj }; // { a: 'a', b: 'b', c: 'c' }
{ a: 1, b: null, ...obj, c: 3 }; // { a: 'a', b: 'b', c: 3 }
{ a: 1, ...obj, b: null, c: 3 }; // { a: 'a', b: null, c: 3 }
{ ...obj, a: 1, b: null, c: 3 }; // { a: 1, b: null, c: 3 }
```

https://github.com/tc39/proposal-object-rest-spread

Spread con objetos

- Copiar un objeto (sin copia profunda)

```
const obj = { foo: 'bar' };
const clone = { ...obj }; // `{ foo: 'bar' }`
obj.foo = 'baz';

clone.foo; // 'bar'
```

- No copia propiedades de la clase/prototipo

```
class Point2D {
   constructor(x,y){
       this.x = x;
       this.y = y;
   }
   bar(){ return 3; }
}
```

```
var obj1 = new Point2D(0,0);
obj1.a = 0;

var obj2 = { ...obj1 };
//{ x: 0, y: 0, a: 0 }
```

Spread con objetos

code URJC

- Mezclar objetos

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var mergedObj = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }
```

- Insertar elementos de un objeto en otro

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var obj2 = { ...obj1, ...obj2 };
// Object { foo: "baz", x: 42, y: 13 }
```

Spread con funciones

code URJC

- Llamar a una función con los elementos de un array

```
function myFunction(x, y, z) { }

var args = [0, 1, 2];

myFunction(...args);
```

- Se pueden usar parámetros y *spread*

```
function myFunction(v, w, x, y, z) {
   console.log(v, w, x, y, z);
}
var args = [0, 1];
myFunction(-1, ...args, 2, ...[3]);
//-1 0 1 2 3
```

Parámetro *rest*

- Tiene la misma sintaxis que el *spread* (...)
- Permite manipular los últimos parámetros de una función como un array

```
function myFun(a, b, ...manyMoreArgs) {
  console.log("a", a);
  console.log("b", b);
  console.log("manyMoreArgs", manyMoreArgs);
}

myFun("one", "two", "three", "four", "five", "six");

// a, one
// b, two
// manyMoreArgs, [three, four, five, six]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters

Parámetro *rest*

- Todos los parámetros pueden manipularse con un único array

```
function fun1(...args) {
   console.log(args.length);
}

fun1(); // 0
fun1(5); // 1
fun1(5, 6, 7); // 3
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters

Destructuring assignment

- Se puede traducir como “asignación mediante desestructuración”
- Permite asignar en una misma sentencia **elementos individuales** de un objeto o un array a variables o parámetros

objetos arrays

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Destructuring en objetos

- Simplifica el código para no tener que asignar cada elemento en su propia sentencia

```
const config = {lang: 'english', prod: true};

const lang = config.lang;
const prod = config.prod;
```

```
const config = {lang: 'english', prod: true};

const {lang, prod} = config;

console.log(lang); // english
console.log(prod); // true
```

Destructuring en objetos

- Se usa el la sintaxis del **parámetro rest** para gestionar el resto de propiedades no asignadas

```
const obj = {a:'a', b:'b', b:'b', c:'c'};

const {a, b, ...others} = obj;

console.log(a); // a
console.log(b); // b
console.log(others); // { b:'b', c:'c' }
```

Destructuring en objetos

- Se pueden desestructurar objetos anidados

```
const user = {
   id: 42,
   displayName: 'jdoe',
   fullName: {
       firstName: 'John',
       lastName: 'Doe'
   }
};

const { id, fullName: { firstName }} = user;

console.log(id);              // 42
console.log(firstName);       // John
```

Destructuring en objetos

- Se pueden crear variables con un nombre diferente a los atributos

```
const o = {p: 42, q: true};
const {p: foo, q: bar} = o;

console.log(foo); // 42
console.log(bar); // true
```

- Con objetos anidados

```
const user = {
  displayName: 'jdoe',
  fullName: { firstName: 'John', lastName: 'Doe' }
};

const {displayName, fullName: {firstName: name}} = user;

console.log(name); // John
```

Destructuring en objetos

- Valores por defecto

```
const {a = 10, b = 5} = {a: 3};

console.log(a); // 3
console.log(b); // 5
```

- Valores por defecto con objetos anidados

```
const chart = { size: 'small', radius: 10 }

const {size = 'big', coords = {x: 0, y: 0}, radius = 25} = chart;

console.log(coords); // {x:0, y:0}
```

Destructuring en objetos

- Cuidado usando : en vez de =

```
const chart = {};
const { coords = { x: 0, y: 0 } } = chart;
console.log(coords); // {x:0, y:0}
```

```
const chart = { coords: { x: 3 } };
const { coords = { x: 0, y: 0 } } = chart;
console.log(coords); // { x: 3 }
```

```
const chart = { coords: { x: 3 } };
const { coords: { x = 0, y = 0 } } = chart;
console.log(x,y); // 3 0
```

Destructuring en objetos

- Nombre diferente y valor por defecto

```
const {a: aa = 10, b: bb = 5} = {a: 3};

console.log(aa); // 3
console.log(bb); // 5
```

Destructuring en objetos

- Desestructuración en un bucle for

```
const people = [
  { name: 'Mike Smith'},
  { name: 'Tom Jones'}
];

for (const { name } of people) {
  console.log('Name: ' + name);
}

// Name: Mike Smith
// Name: Tom Jones
```

Destructuring en objetos

- Desestructuración en los **parámetros** de una función

```
const user = { id: 42, displayName: 'jdoe' };

function userId({id}) {
  return id;
}

function whois({id, displayName}) {
  return `${displayName} is ${id}`;
}

console.log(userId(user)); // 42
console.log(whois(user));  // jdoe is 42
```

Destructuring en objetos

- Desestructuración en los **parámetros** con objetos anidados (y **cambio de nombre**)

```
const user = {
  id: 42,
  displayName: 'jdoe',
  fullName: {
    firstName: 'John',
    lastName: 'Doe'
  }
};

function whois({displayName, fullName: {firstName: name}}) {
  return `${displayName} is ${name}`;
}

console.log(whois(user)); // "jdoe is John"
```

Destructuring en objetos

- Desestructuración en los **parámetros** con objetos anidados y valores por defecto

```
function drawChart({size = 'big', coords = {x: 0, y: 0}} = {}) {

  console.log(size, coords);
}

drawChart({ coords: {x: 18, y: 30}}); // big {x:18, y:30}
drawChart();                          // big {x:0, y:0}
drawChart({});                        // big {x:0, y:0}
drawChart({size:'small'});            // small {x:0, y:0}
```

Destructuring en arrays

- Desestructuración de los elementos de un array en variables independientes

```
let [a, b] = [10, 20];
console.log(a); // 10
console.log(b); // 20
```

```
let [a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(a); // 10
console.log(b); // 20
console.log(rest); // [30, 40, 50]
```

Destructuring en arrays

- Declaración previa de variables

```
let a, b;
[a, b] = [1, 2];

console.log(a); // 1
console.log(b); // 2
```

- Valores por defecto

```
let a, b; [a=5, b=7] = [1];

console.log(a); // 1
console.log(b); // 7
```

Destructuring en arrays

code URJC

- Intercambio de valores de variables (*swapping*)

```
let a = 1;
let b = 3;
[a, b] = [b, a];
console.log(a); // 3
console.log(b); // 1
```

- Ignorar algunos valores

```
const [a,,b] = [1,2,3,4];

console.log(a); // 1
console.log(b); // 3
```

Destructuring en arrays

- Asignar el resto del array a una variable

```
const [a, ...b] = [1, 2, 3];

console.log(a); // 1
console.log(b); // [2, 3]
```

Destructuring en arrays

code URJC

- Combinando arrays y objetos

```
const props = [
  { id: 1, name: 'Fizz'},
  { id: 2, name: 'Buzz'},
  { id: 3, name: 'FizzBuzz'}
];

const [,, { name }] = props;

console.log(name); // "FizzBuzz"
```

Destructuring en arrays

- Combinando arrays y objetos

```
const metadata = {
  title: 'Scratchpad',
  translations: [
    {
      locale: 'de',
      localization_tags: [],
      last_edit: '2014-04-14',
      url: '/de/docs/Scratchpad',
      title: 'JS-Umgebung'
    }
  ],
  url: '/en-US/docs/Scratchpad'
};
```

```
let {
  title: englishTitle, // rename
  translations: [
    {
      title: localeTitle, // rename
    },
  ],
} = metadata;

console.log(englishTitle);
// "Scratchpad"

console.log(localeTitle);
// "JS-Umgebung"
```

Ejercicio 1

- Implementa varias funciones que usen la sintaxis vista en el tema
- Haz varias llamadas con diferentes parámetros y muestra el resultado por consola para verificar el funcionamiento.
  - **chars()**
  - **default()**
  - **params()**

Ejercicio 1

- **chars():** Recibe como parámetros varios Strings y devuelve un único array con todos los caracteres de todos los arrays.
- **default():** Recibe un objeto de configuración y devuelve ese mismo objeto pero con todos los valores no configurados con sus valores por defecto (usa propiedades y valores por defecto inventadas)

Ejercicio 1

- **params():**
  - Implementa una función que necesite tres valores
  - Cualquier valor es opcional porque tiene valor por defecto (interno)
  - En vez de pasar 3 parámetros, pasa un objeto con los valores en propiedades
  - Usa destructuring en la definición de parámetros