JavaScript

En aquest apartat es descriuran les novetats per a tots aquells que venguin de jQuery.

Variable

La definició de variables amb `var` deixa de ser recomanada en favor de `let` i `const`

```
let nom = "Juan"
const edat = 10; // No permet reasignació
```

Objectes

Definició d'objectes amb json i accés a propietats per notació per punt

```
const person = {
   name: 'Steve',
   age: '30',
   hobbies: ['waterpolo', 'reading']
}

// Acccés a les propietats
const name = person.name;

/* Propietats per shorthand */
const name = 'Steve';
const per = {
   name,
   age: '30',
}
// És equivalent a:
const per = {
   name: 'Steve',
   age: '30',
}
```

Deconstruct

Podem deconstruir un objecte en les seves propietats

```
// Objecte
const {name, age} = person;
// name == person.name

// Array
```

```
let arr = [1, 2, 3];
let [a, b, c] = arr;
// a == 1
```

> Exemple pràctic

```
// Suposem que volem customitzar slot body d'un datatable:
<template #body="slotProps">{{ slotProps.data.name }}</template>
```

Podríem descompondre aquesta variable en les seves propietats

```
<template #body="{data}">{{ data.name }}</template>
```

Export/Import

Podem exportar distints elements i importar-ho a un altre fitxer

```
// Export en linea
export const a = 1;
export function b() {...}

// Export al final
const a = 1;
function b() {...}

export {a,b}

// Import amb deconstruct
import {a,b} from '...'

// Import com ha objecte
import * as c from '...'
// c.a = 1
```

Export default: Si exportam un default el podem importar directament

```
export default function c() {...}

import c from '...'
```

Rest/spread operator

Rest: Podem generar un array d'un llistat de valors

```
doMath('add', 1, 2, 3);

function doMath(operator, ...numbers) {
   // operator == 'add'
   // numbers == [1,2,3] converteix els tres darrers paràmetres en un array
}
```

Spread: descomposa un array o objecte a "valors separats per coma"

```
let arr = [1,2,3,-1];
console.log(Math.min(...arr)); // --1

// També es pot usar en objectes
const user1 = {
   name: 'Jen',
   age: 22
};
const user2 = {
   ...user1 ,
   llinatge: 'Diaz'
}
/*
user2 = {
   name: 'Jen',
   age: 22,
   llinatge: 'Diaz'
}
*/
```

Default funcion parameter

```
function multiply(num = 2, multiplier = 2){
   return num * multiplier;
}
multiply(); // 4
```

Arrow function

Els arrow function són una forma de crear funcions de forma "incrustada"

```
(e) => {
   return e*2;
}
// equivalent
function(e) {
```

```
    return e*2;
}
```

El format dels arrow funcion és: `(params...) => { //code }`. Podem tenir dues variacions (per si trobau el cas):

1. Si el mètode reb únicament un paràmetre es pot prescindir del parèntesi `param => { //code }`
2. Si el codi retorna un valor directament es pot prescindir del claudàtor {} `(e) => e.nom` seria equivalent a `(e) => { return e.nom }`

> Exemple pràctic

Un exemple clar en el qual podem usar aquestes funcions és en el retorn d'un callback

```
service.getAll().then(
    // Arrou funcion
    (data) => { this.list = data.list }
);
```

Template literals

En lloc d'usar cometes s'han d'usar els accents `.

```
const e = 'Juan';
const  a = `Hola ${juan}
Salutacions`
// Permet incrustar variables i suporta multilinia
```

Classes i Herencia

Es poden definir classes i herència entre elles

```
class Person {
    constructor(name){
        this.name = name;
    }

    introduce(){
        return `Hola ${this.name}`;
    }
}

class SuperHero extends Person{
    constructor(name, power) {
        super(name);
        this.power = power;
    }
```

```
    introduce() {
        return `${super.introduce()}. Poder ${this.power}`
    }
}
```