

# UD 06. PROGRAMACIÓN ASÍNCRONA

Desarrollo Web en entorno cliente CFGS DAW

# **Ejercicios**

Álvaro Maceda Arranz

alvaro.maceda@ceedcv.es

2021/2022

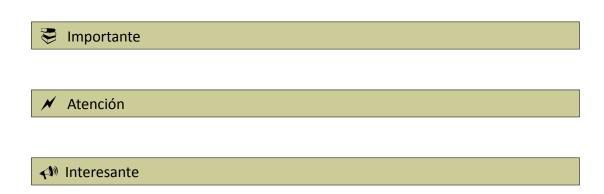
Versión:221127.1848

#### Licencia

Reconocimiento - NoComercial - Compartirlgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

#### Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



### **ÍNDICE DE CONTENIDO**

1. Ejercicio 1: Callbacks secuenciales	3
2. Ejercicio 2: Callbacks paralelos	
3. Ejercicio 3: cutriFetch	
4. Ejercicio 4: Fizz asíncrono	
5. Código de asyncRequest	

### UD06. Programación Asíncrona



El código de asyncRequest está al final del enunciado de los ejercicios

#### 1. EJERCICIO 1: CALLBACKS SECUENCIALES

Utilizando la función asyncRequest() realiza de forma secuencial peticiones para obtener recurso1, recurso2 y recurso3, en ese orden. Cuando hayas obtenido los tres recursos debes imprimir "¡Completado!" en la consola.

#### 2. EJERCICIO 2: CALLBACKS PARALELOS

Utilizando la función asyncRequest() realiza de forma simultánea peticiones para obtener recurso1, recurso2 y recurso3.

Debes imprimir el contenido de cada recurso en orden y lo antes posible. Cuando se hayan obtenido los tres recursos debes imprimir "¡Completado!" en la consola.

Esto es un ejemplo de cómo se podrían recibir los datos y qué tendría que imprimirse por pantalla:

Datos recibidos	Impreso en consola
resource2	(No se imprime nada ya que aún no tenemos el resultado de resource1)
resource1	The first resource The second resource
resource3	The third resource iCompletado!

#### 3. EJERCICIO 3: CUTRIFETCH

Crea un módulo con una función llamada cutriFetch() que reciba como parámetro un nombre de recurso, llame a asyncRequest() y devuelva una promesa que se resuelva cuando se han obtenido los datos del recurso.

Realiza el ejercicio 1 utilizando esa nueva función.

#### 4. EIERCICIO 4: FIZZ ASÍNCRONO

Crea una función llamada fizz() que devuelva si un número es divisible por 3 o contiene un 3. Sin embargo, debe devolver ese resultado al cabo de un tiempo aleatorio entre 100 y 10.000 ms (puedes reducir este número mientras estás probando el ejercicio)

Imprime los números del 1 al 300. Si fizz() devuelve false debes imprimir el número. Si fizz() devuelve true, debes imprimir "fizz" en lugar del número.

Debes imprimir los números en orden pero tan rápido como puedas.

#### 5. CÓDIGO DE ASYNCREQUEST

```
const DEFAULT_RESOURCES = {
    "resource1": "The first resource",
    "resource2": "The second resource",
    "resource3": "The third resource",
    "whatINeed": "resource2"
}

function asyncRequest(resource, callback, resources = DEFAULT_RESOURCES) {
    const MIN_DELAY = 1_000;
    const RANDOM_DELAY = 2_000;

    var randomDelay = Math.round(Math.random() * RANDOM_DELAY) + MIN_DELAY;

    console.log("**Requesting: " + resource + "**");

    setTimeout(function() {
        console.log("**Returning: " + resources[resource] + "**");
        callback(resources[resource]);
        }, randomDelay);
}

module.exports = asyncRequest;
```

A continuación ofrecemos en orden alfabético el listado de autores que han hecho aportaciones a este documento:

Álvaro Maceda Arranz