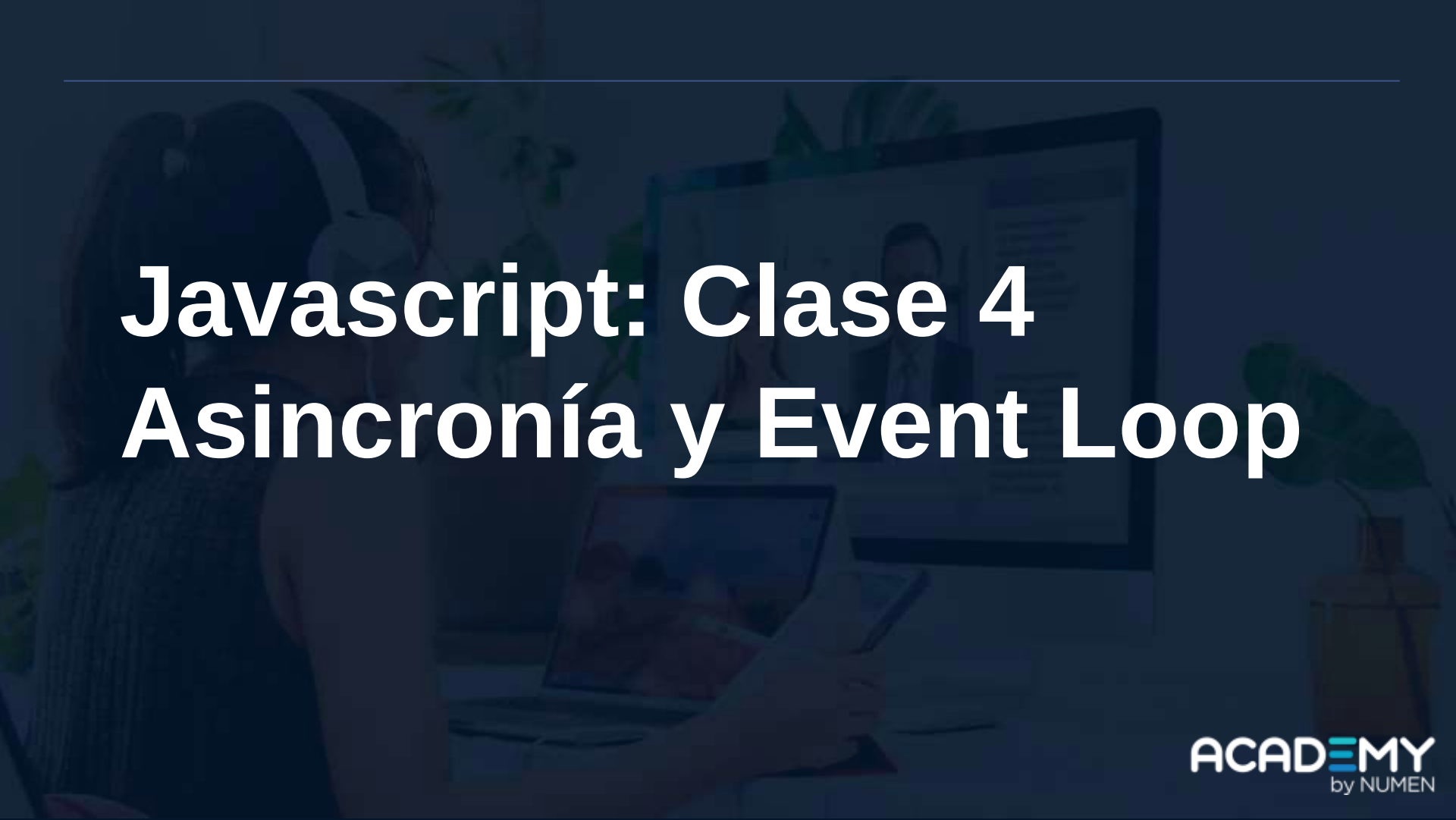
ACADEMY by NUMEN

FULL STACK

Comenzamos en unos minutos


ACADEMY
by NUMEN

Javascript: Clase 4
Asincronía y Event Loop



ACADEMY by NUMEN

Asincronía en JavaScript

setTimeOut()

El setTimeOut() es un método que recibe una función y nos permite establecer el momento de ejecución.

Este método recibe 2 parámetros: Por un lado la función a ejecutar y por otro lado el tiempo que queremos que tarde en ejecutarse.

```
setTimeout(function() {
    console.log('Ejecutando un setTimeOut')
}, 1000);
```

setInterval()

El setInterval() es un metodo similar al setTimeOut() con la diferencia que en lugar de ejecutarse una única vez, se repite una y otra vez cada vez que transcurre el tiempo que se le indico como parametro.

```
setInterval(() => {
    console.log('Ejecutando un setInterval')
}, 1000);
```

```
setInterval(() => {
    console.log(new Date().toLocaleTimeString())
}, 1000);
```

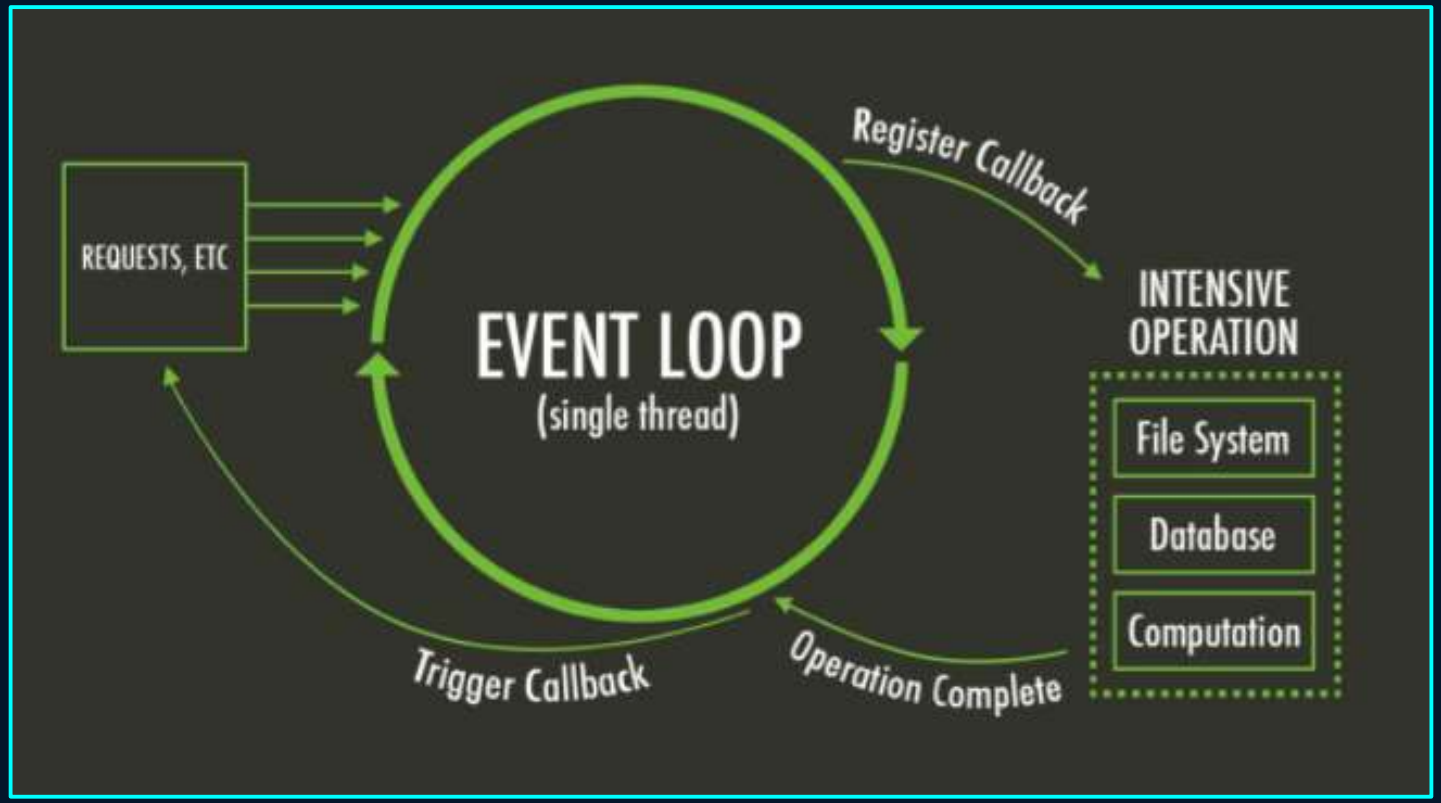
Caracteristicas de Javascript

1- Es un lenguaje de programación de un solo subproceso o hilo (single thread), lo que significa que sólo puede ejecutar una cosa a la vez.

2- Su modelo está basado en un loop (ciclo) de eventos.

3- Al no poder procesar instrucciones simultáneas necesita de un modelo asíncrono para realizar largas solicitudes de red sin bloquear el hilo principal, ahorrando cualquier clase de problema de concurrencia.

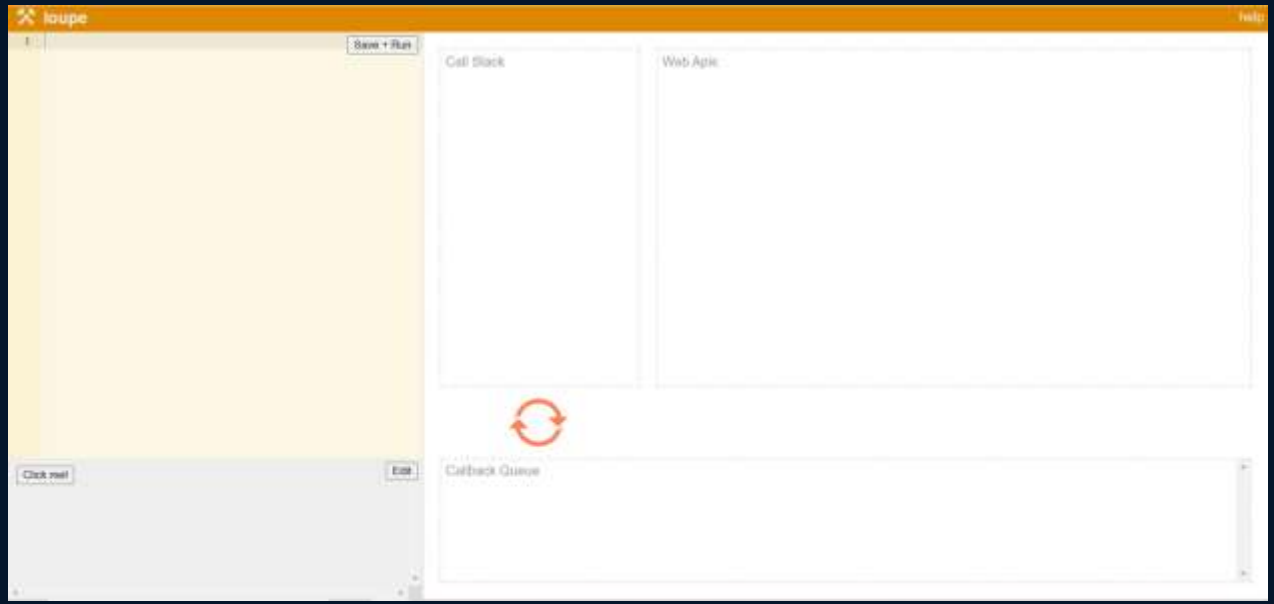
Ciclo de eventos / Event loop



Modelo de Event Loop

En este programa podremos ver de manera dinámica cómo funciona el event loop.

Vamos a ello!!
http://latentflip.com/loupe/?code=!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D



ACADEMY by NUMEN

LIFO: Last in - First out
Modelo Sincrono

Este es el formato en el que Javascript procesa las instrucciones. Cuando una instrucción se dispara, se une a la cola de espera, pero se pone en primer lugar. De este modo, la última instrucción en unirse es la primera en ejecutarse.

Esto permite a las instrucciones que estaban en espera ejecutarse apenas se resuelven evitando que se bloqueen otras instrucciones ya sea en la espera de resolución como en la espera de ejecución.
Veámoslo en código:

```
console.log('Codigo Sincrono');
    console.log('Inicio');

    function dos() {
        console.log('Dos');
    }

    function uno() {
        console.log('Uno');
        dos();
        console.log('Tres');
    }

    uno();
    console.log('Fin');
```

Modelo Asincrono

En la diapositiva anterior vimos cómo funcionaba el LIFO en un modelo síncrono.

Ahora nos toca ver como funciona en un modelo asíncrono.

Vamos a ello!!

```
console.log('Codigo Asincrono');
console.log('Inicio');

function dos() {
    setTimeout(function () {
        console.log('Dos');
    }, 1000)
}

function uno() {
    setTimeout(function () {
        console.log('Uno');
    }, 0)
    dos();
    console.log('Tres');
}

uno();
console.log('Fin')
```

Callbacks

Una función de tipo **Callback** nos permite pasar otra función como argumento e invocarla a través de esta cuantas veces se requiera.

En esta ocasión podemos observar como las funciones **saludarUsuario()** y **despedirUsuario()** se pasan como parámetro en la función **crearSaludo()** y luego se invocan al llamar a esta última.

```
function saludarUsuario(usuario) {
    return `Hola ${usuario}!`
}

function despedirUsuario(usuario) {
    return `Adiós ${usuario}!`
}

function crearSaludo(usuario, callback) {
    return callback(usuario)
}

crearSaludo('Numen', saludarUsuario);
crearSaludo('Numen', despedirUsuario);
```

.forEach()

forEach un método que nos permite iterar arreglos, elemento por elemento, a través de una ejecución repetitiva de funciones.

```
var profesores = ['Cinthia', 'Matias', 'Aaron',
'Santi']

// sin callback

for (var i = 0; i < profesores.length; i++) {
    console.log(profesores[i])
}
```

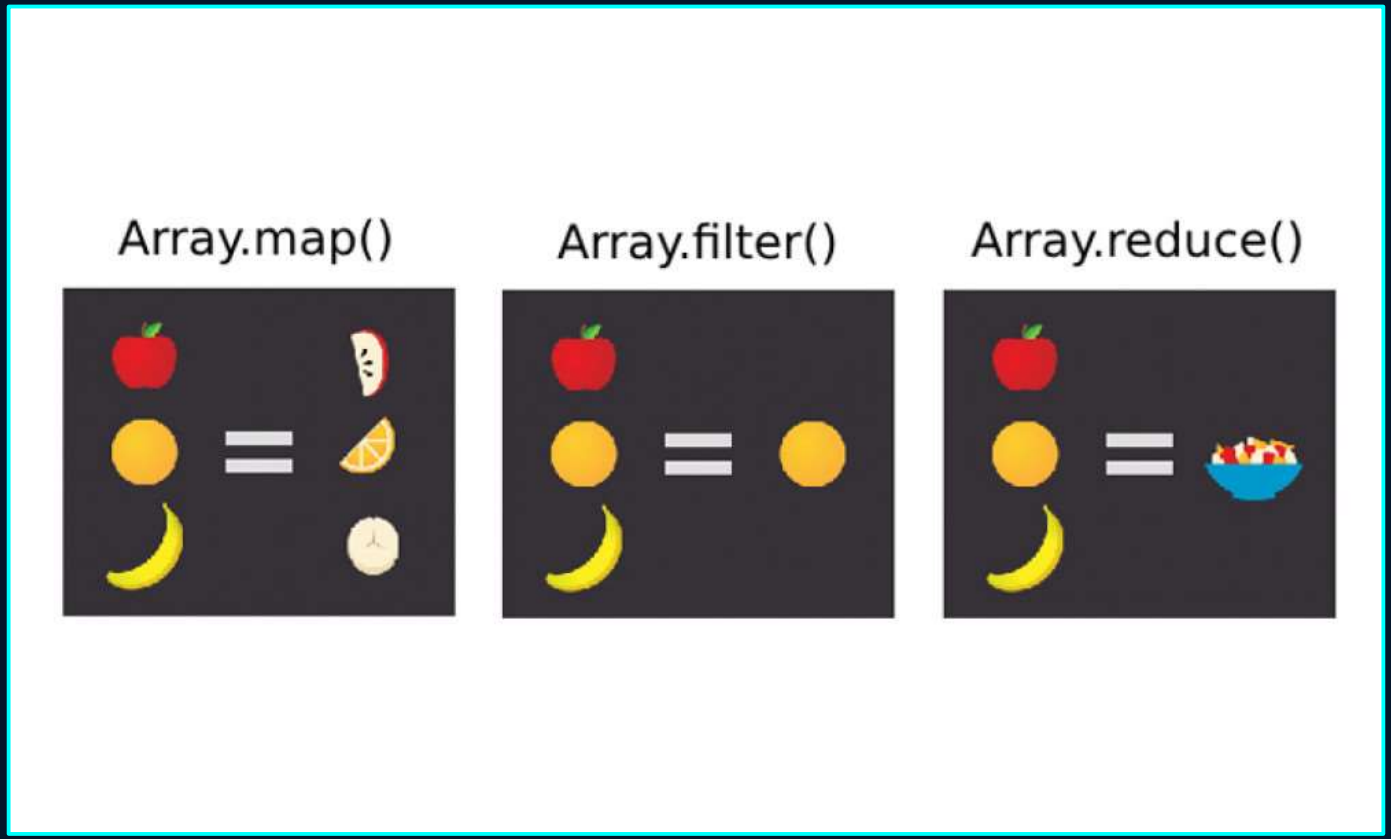
```
var profesores = ['Cinthia', 'Matias', 'Aaron',
'Santi']

// con callback

profesores.forEach(function(elemento, indice) {
    console.log(elemento)
})
```

Aquí se puede observar como se ha reemplazado la iteración, a través de un ciclo for, de un arreglo por una función forEach.

Otras funciones callback...



.map()

El método map() nos permite iterar los elementos de un arreglo, transformarlos y devolver un nuevo arreglo con las transformaciones realizadas.

```
var profesores = [
    {
        nombre: 'Matias',
        edad: '33',
        profesion: 'Profesor',
    },
    {
        nombre: 'Cinthia',
        edad: '31',
        profesion: 'Coordinadora',
    },
    {
        nombre: 'Aaron',
        edad: '23',
        profesion: 'Profesor',
    },
    {
        nombre: 'Guillermo',
        edad: '25',
        profesion: 'Tutor',
    }
]
```

```
profesores.map(elemento => {
    console.log(`
      <h2>Bienvenidos a Academia
Numen</h2>

      <p>En esta ocasión quiero
presentarles
      a ${elemento.nombre} quien
será su ${elemento.profesion}
      a lo largo de este curso.</p>
    `)
})
```

.reduce()

El método reduce() nos permite recorrer un arreglo y nos devuelve un único elemento.

Ejemplo: Sumar todos los elemento (numericos) de un arreglo y devolvernos el resultado de esa suma.

```
var numeros = [1, 2, 3, 4, 5, 6, 7];

// sin callbacks

var suma = 0;
for (var i = 0; i < numeros.length; i++) {
    suma = suma + numeros[i];
}

// callbacks

var sumaReduce = numeros.reduce(function(acumulador,
elemento) {
    return acumulador + elemento;
}, 0);
```

.filter()

Como indica su nombre, el método **filter()** se utiliza para filtrar contenido de un arreglo, es decir, extraer solo ciertos elementos que coincidan con la condición especificada.

```
var palabras = ['chancleta', 'pato', 'bigote',
'ornitorrinco', 'termo', 'ajedrez'];

var resultado = palabras.filter(palabra => palabra.length
> 6);

console.log(resultado);
```

Closures

Las **closures** nos permiten acceder a variables que no estén exactamente definidas dentro del **scope** de una función.

Es decir, van a estar en otro contexto que ya no existirá más pero que Javascript nos reserva esa referencia para poder acceder a ella de todos modos.

```
function saludar( saludo ){
    return function( nombre ){
        console.log(`${saludo} ${nombre}`);
    }
}

var saludoHola = saludar('Hola'); // Esto devuelve una función

saludarHola('Numen'); // 'Hola Numen'
```

Closures

```
function hacerSaludo( lenguaje ) {
    if ( lenguaje === 'en' ) {
        return function () {
            console.log('Hi!')
        }
    }

    if (lenguaje === 'es') {
        return function () {
            console.log('Hola!')
        }
    }
}

let saludoIngles = hacerSaludo('en');
let saludoEspaniol = hacerSaludo('es');
```

ACADEMY
by NUMEN