

Unidad

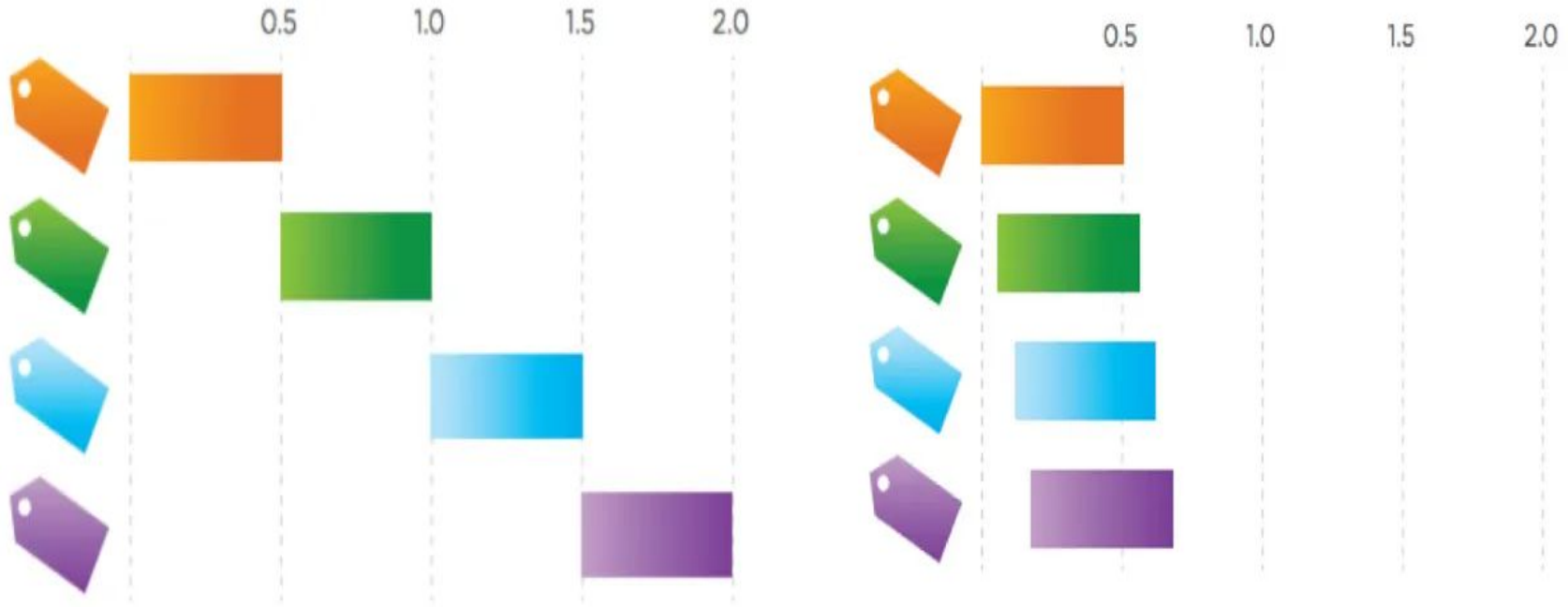
Javascript - clase 4



Universidad Tecnológica Nacional
Facultad Regional Córdoba
Cátedra: Desarrollo de Software



Programación síncrona vs asíncrona



JavaScript Síncrono

- Cada operación se hace de una vez, bloqueando el flujo de ejecución, el hilo es bloqueado mientras espera la respuesta, cuando esta se procesa pasa a la siguiente operación y así sucesivamente al terminar todas las operaciones.

```
console.log("Inicio proceso");
function procesoSecundario() {
  console.log("Etapa 2 del proceso");
}
function procesoPrincipal() {
  console.log("Etapa 1 del proceso");
  procesoSecundario(); //llamada síncrona
  console.log("Etapa 3 del proceso");
}
procesoPrincipal();
console.log("Fin");
```



JS IS SINGLE THREADED?



THEN HOW MULTITHREADING

JavaScript y su modelo de un solo hilo (Single Thread)

(Single Thread): en el contexto de JavaScript, significa que sólo un conjunto de instrucciones se ejecuta a la vez en un único hilo de ejecución.

- Características:
 - JavaScript es un lenguaje de programación single-threaded
 - Opera en un único hilo en el entorno de ejecución (navegador o Node.js)
 - Las tareas se ejecutan secuencialmente y no en paralelo
- Modelo de ejecución basado en Event Loop: El bucle de eventos (Event Loop) es el mecanismo utilizado para manejar tareas asíncronas y eventos en JavaScript
- Procesa tareas en la pila de llamadas (Call Stack) y en la cola de tareas (Task Queue)
- Permite que JavaScript maneje operaciones asíncronas sin bloquear el hilo principal

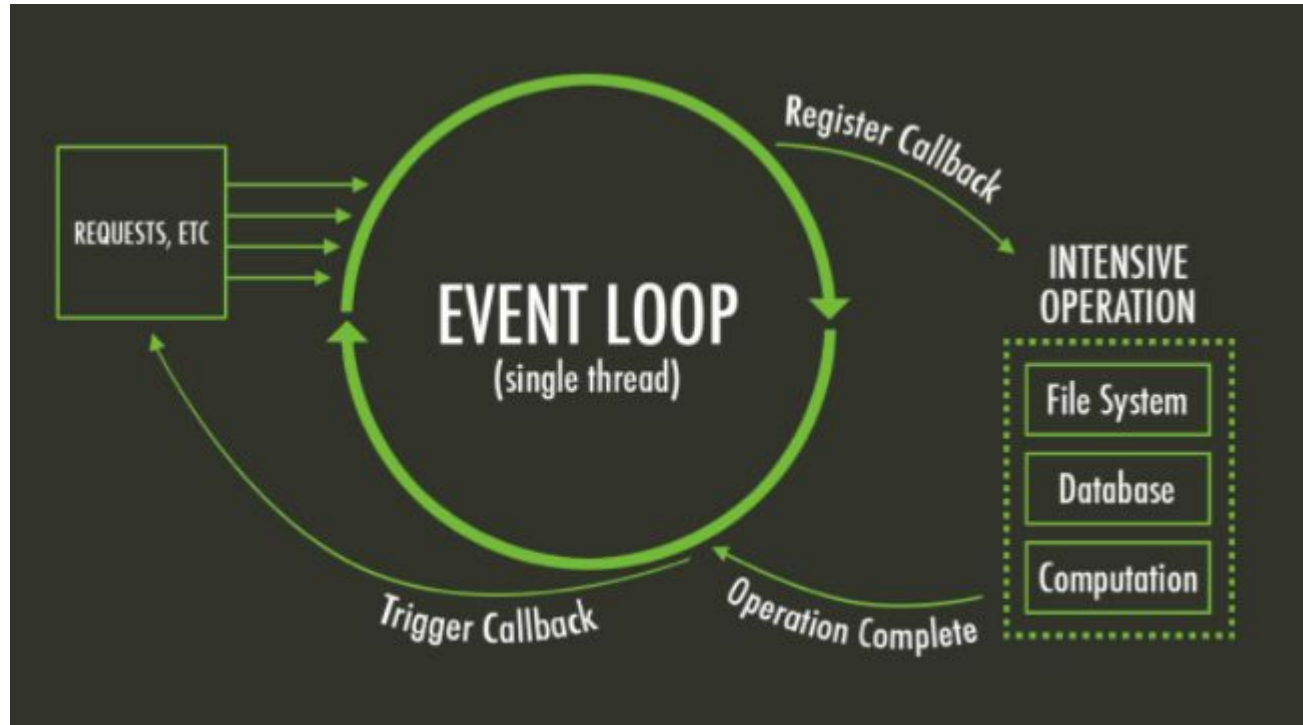


JavaScript Asíncrono

- Concepto de programación asíncrona
- Ventajas de la programación asíncrona: mejor rendimiento, experiencia de usuario y capacidad de respuesta
- Ejemplos de situaciones que requieren programación asíncrona: solicitudes de API, lectura de archivos y temporizadores
- Mecanismos asíncronos en JavaScript
 - **Callbacks**
 - **Promises**
 - **Async / Await**



¿Cómo funciona con un solo thread javascript de manera asincrónica?



Función asíncrona setTimeout

- **setTimeout**: función global de JavaScript que permite ejecutar una función o bloque de código después de un período de tiempo especificado

```
let idTimeout = scope.setTimeout(funcion[, retraso]);
```

- Recibe dos argumentos principales: una función (callback) y un tiempo de espera en milisegundos
- Registra el callback en un "temporizador"
- Después del tiempo de espera especificado, el callback se agrega a la cola de tareas
- El callback se ejecuta cuando el hilo principal de JavaScript está disponible (JavaScript es de un solo hilo)



Ejemplo de llamada asíncrona con setTimeout

```
function mensaje() {  
    console.log('Proceso asíncrono que se ejecuto a las ' +  
                (new Date()).toLocaleTimeString());  
}  
  
console.log("Inicio a las " + (new Date()).toLocaleTimeString());  
setTimeout(mensaje, 2000);  
console.log("Fin a las " + (new Date()).toLocaleTimeString());
```

Callbacks

- Definición de callback: función que se pasa como argumento a otra función y se ejecuta una vez que se completa una tarea.

```
function leerArchivo(callback) {  
  // Simulando lectura de archivo  
  setTimeout(() => {  
    const resultado = 'Contenido del archivo';  
    callback(resultado);  
  }, 1000);  
}
```

Realizamos un callback

```
leerArchivo((contenido) => {  
  console.log(contenido);  
});
```

Pasamos una arrow function como parámetro



GIRL : ARE YOU A JAVASCRIPT DEVELOPER?

BOY: WHY ARE YOU ASKING FOR?

GIRL: THEY ALWAYS CALLBACK. 😊



Callbacks - Problemas


Callback Hell: término utilizado para describir el anidamiento profundo y excesivo de funciones callback en el código, lo que puede resultar en una estructura de código difícil de leer, mantener y depurar.

Causas:

- Ejecución de múltiples operaciones asíncronas en secuencia o en paralelo
- Manejo de errores en cada nivel de anidamiento
- Falta de buenas prácticas de programación y estructuración del código



Problemas del callback - Pirámide de la Perdición



```
pan.pourWater(function() {  
  range.bringToBoil(function() {  
    range.lowerHeat(function() {  
      pan.addRice(function() {  
        setTimeout(function() {  
          range.turnOff();  
          serve();  
        }, 15 * 60 * 1000);  
      });  
    });  
  });  
});
```

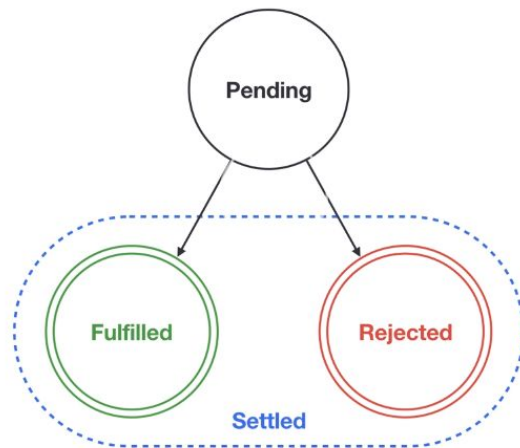
pyramid of doom

Promesas en Javascript

- Una Promesa es un objeto que representa el resultado eventual de una operación asíncrona. Permite organizar y gestionar callbacks de manera más estructurada y legible, facilitando el manejo de errores y el control del flujo de ejecución

- Estados:
 - Pendiente (Pending).
 - Resuelta (Fulfilled).
 - Rechazada (Rejected).
 - Arreglada (Settled) ha sido resuelta o rechazada

Promise States



Promesas - ejemplo

```
function leerArchivo() {  
  return new Promise((resolve, reject) => {  
    // Simulando lectura de archivo  
    setTimeout(() => {  
      const resultado = 'Contenido del archivo';  
      resolve(resultado);  
    }, 1000);  
  });  
}  
  
leerArchivo()  
  .then((contenido) => {  
    console.log(contenido);  
  })  
  .catch((error) => {  
    console.error(error);  
  });
```

Ventajas de las promesas:
mejor manejo de errores,
composición más sencilla y
código más legible



Consumir datos de un servidor - Protocolo HTTP

- Definición de HTTP (Protocolo de Transferencia de Hipertexto): protocolo de comunicación de la capa de aplicación utilizado para transmitir información en la World Wide Web.
- Características:
 - Basado en el modelo cliente-servidor: el cliente (navegador) envía solicitudes y el servidor responde con recursos o datos
 - Protocolo sin estado: cada solicitud es independiente y no mantiene información entre múltiples solicitudes
 - Texto plano y legible por humanos: fácil de interpretar y depurar
 - Soporta múltiples métodos de solicitud (verbos HTTP) como GET, POST, PUT, DELETE, etc.
 - Estructura de las solicitudes y respuestas: método, encabezados, URL y cuerpo (opcional)



Solicitud GET en HTTP

GET: método de petición HTTP utilizado para solicitar datos de un recurso específico en un servidor.

- Características:

- Método seguro e idempotente: no modifica recursos ni produce efectos secundarios en el servidor
- Solicita información: obtiene datos sin cambiar el estado del recurso
- Datos de la solicitud incluidos en la URL: parámetros de consulta, lo que hace que la solicitud sea fácilmente cacheable y se pueda guardar en marcadores
- Limitaciones en la cantidad de datos enviados: la longitud máxima de la URL puede estar restringida por el navegador o el servidor

Ejemplo método GET

The screenshot shows a web browser with the address bar displaying `pokeapi.co/api/v2/pokemon`. The page content is a JSON array of 20 Pokémon objects, starting with `bulbasaur` and ending with `raticate`. The right sidebar shows the Network tab with a single request selected. The request details panel shows the request URL as `https://pokeapi.co/api/v2/pokemon` and the request method as `GET`.

```
{
  "count": 1281,
  "next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
  "previous": null,
  "results": [
    {
      "name": "bulbasaur",
      "url": "https://pokeapi.co/api/v2/pokemon/1/"
    },
    {
      "name": "ivysaur",
      "url": "https://pokeapi.co/api/v2/pokemon/2/"
    },
    {
      "name": "venusaur",
      "url": "https://pokeapi.co/api/v2/pokemon/3/"
    },
    {
      "name": "charmander",
      "url": "https://pokeapi.co/api/v2/pokemon/4/"
    },
    {
      "name": "charmeleon",
      "url": "https://pokeapi.co/api/v2/pokemon/5/"
    },
    {
      "name": "charizard",
      "url": "https://pokeapi.co/api/v2/pokemon/6/"
    },
    {
      "name": "squirtle",
      "url": "https://pokeapi.co/api/v2/pokemon/7/"
    },
    {
      "name": "wartortle",
      "url": "https://pokeapi.co/api/v2/pokemon/8/"
    },
    {
      "name": "blastoise",
      "url": "https://pokeapi.co/api/v2/pokemon/9/"
    },
    {
      "name": "caterpie",
      "url": "https://pokeapi.co/api/v2/pokemon/10/"
    },
    {
      "name": "metapod",
      "url": "https://pokeapi.co/api/v2/pokemon/11/"
    },
    {
      "name": "butterfree",
      "url": "https://pokeapi.co/api/v2/pokemon/12/"
    },
    {
      "name": "weedle",
      "url": "https://pokeapi.co/api/v2/pokemon/13/"
    },
    {
      "name": "kakuna",
      "url": "https://pokeapi.co/api/v2/pokemon/14/"
    },
    {
      "name": "beedrill",
      "url": "https://pokeapi.co/api/v2/pokemon/15/"
    },
    {
      "name": "pidgey",
      "url": "https://pokeapi.co/api/v2/pokemon/16/"
    },
    {
      "name": "pidgeotto",
      "url": "https://pokeapi.co/api/v2/pokemon/17/"
    },
    {
      "name": "pidgeot",
      "url": "https://pokeapi.co/api/v2/pokemon/18/"
    },
    {
      "name": "rattata",
      "url": "https://pokeapi.co/api/v2/pokemon/19/"
    },
    {
      "name": "raticate",
      "url": "https://pokeapi.co/api/v2/pokemon/20/"
    }
  ]
}
```

Network Tab:

- Request:** `pokemon`
- General:**
 - Request URL:** `https://pokeapi.co/api/v2/pokemon`
 - Request Method:** `GET`



Promesa - Método fetch para obtener datos con HTTP

- fetch: API moderna de JavaScript para realizar solicitudes HTTP y gestionar respuestas, basada en Promesas.
 - Reemplaza XMLHttpRequest para simplificar y mejorar la experiencia de trabajo con solicitudes asíncronas.
- Características:
 - Sintaxis más clara y sencilla
 - Basada en Promesas

```
let pokemones = fetch("https://pokeapi.co/api/v2/pokemon/1");
pokemones.then(res => {
    if (res.ok)
        return res.json();
    else
        throw new Error(`HTTP error! Status: ${res.status}`);
})
.then(data => {
    console.log(data.name);
}).catch(error => console.log(`Ocurrio un error ${error}`));
```



Método fetch - Llamadas anidadas

```
function obtener_pokemon(id){  
    let url = "https://pokeapi.co/api/v2/pokemon/" + id;  
    return fetch(url).then(res => {return res.json()});  
}  
obtener_pokemon(1).then(data => {  
    console.log(data.name);  
    return obtener_pokemon(2);  
}).then(data =>{  
    console.log(data.name);  
    return obtener_pokemon(3);  
}).then(data =>{  
    console.log(data.name);  
    return obtener_pokemon(4);  
}).then(data =>{  
    console.log(data.name);  
})
```

A close-up photograph of a woman with dark hair, her eyes closed and a tearful expression. She is holding her right hand up to her forehead, with her fingers partially covering her face. The lighting is soft and warm, highlighting the texture of her skin and the intensity of her emotion. The background is a plain, light-colored wall.

WHEN JAVASCRIPT....

... BREAKS ITS PROMISE

Async - Await

- Async/Await: muy utilizado para trabajar con promesas en JavaScript de forma más sencilla y legible.
- **Async:**
 - Palabra clave utilizada para definir una función asíncrona
 - Indica que la función siempre devuelve una promesa
 - Facilita el uso de await dentro de la función
- **Await:**
 - Palabra clave que se utiliza para esperar la resolución de una promesa de tal forma que podamos volver nuestro código síncrono.
 - Sólo puede usarse dentro de funciones async
 - Detiene la ejecución de la función hasta que se resuelva la promesa, sin bloquear el hilo principal

Ejemplo async/await

```
async function fetchData(url) {  
  try {  
    const response = await fetch(url);  
  
    if (!response.ok) {  
      throw new Error(`Error en la solicitud:  
${response.status}`);  
    }  
  
    const data = await response.json();  
    console.log('Datos recibidos:', data);  
  } catch (error) {  
    console.error('Error al realizar la solicitud:',  
error);  
  }  
}  
  
fetchData('https://pokeapi.co/api/v2/pokemon/');
```