

# Javascript

## Resumen

# Lo básico

- JS es un lenguaje interpretado
- Tipado dinámico (tal como python)
- Orientado a objetos, todo es un objeto
- Se ejecuta nativamente en el navegador
- En constante evolución
- ES2015 introdujo varios cambios, incluida la sintaxis de clases

# Sintaxis básica

- `//` comentario de una línea
- `/*` comentario de  
Varias líneas `*/`
- Usar `;` para terminar las instrucciones (optional, pero RECOMENDADO)
- `{}` para definir bloques de código (condicionales, ciclos, funciones, etc)

```
let msg = "Hello";  
let y = 100;  
if (y > 10) {  
    str += ", world!";  
    y = y +4;  
}  
// imprimir valores  
console.log(msg);  
console.log(y);
```

# Variables

- **let, const** (ES6/ES2015, más parecido a otros lenguajes, respeta el scope, **RECOMENDADO**)
- **var** (vieja forma, el scope no es tan intuitivo como let y const. **NO USAR**)
- Usar una variable sin definirla (se hace global automáticamente. **NO USAR**)

# Tipos

- **Primitivos:**
  - String: Comillas dobles o simples (**elegir uno y mantenerlo**); templates `El valor es: `${var}``
  - Number: Tanto enteros como flotantes (Math tiene funciones para operar con números)
  - Boolean: true o false
  - null: Valor inexistente (se asigna intencionalmente)
  - undefined: no tiene valor (se asigna por defecto\*)
  - symbol
- **No primitivos**
  - Objetos, arrays y funciones
- **Typeof, instanceof**

```
const msg = "Hello";  
let y;  
console.log(msg);  
console.log(y); //undefined  
console.log(y+2); // NaN  
y == false; //false  
y = null;  
console.log(y); //null  
y==null; //true  
Math.log10(1000); //3  
console.log(y+2); //2
```

```
undefined == null; // true  
undefined === null; // false  
msg += "mundo"; // Error
```

```
/*undefined, null, 0, "",NaN se  
evaluan como false, cualquier  
otra cosa se evalúa como true */  
if(!y) console.log("verdad");  
let a = [];  
if(a) console.log("verdad");
```

# Operadores

- Aritméticos: (+, -, \*, /, %, \*\*, ++, --)
- Lógicos: (&&, ||, !, <, >, <=, >=, ==, !=)
  - Comparación estricta (tipo y valor): ===, !==
- Asignación: (=, +=, -=, \*=, /=, %=, \*\*=, ??=)
  - Ternario: condicion ? valor\_verdadero: valor\_falso



```
let x;  
x ??= 5; // 5  
let y = x < 10 ? x*2: x**2; //10  
console.log(x++); //5  
console.log(x); //6  
x/=2; //3  
console.log(x)  
if(x && y){  
  console.log("ambos tienen  
valor");  
}  
!(x > 10); //true
```

# Control de flujo

- **if-else:**
- **switch:** Evalúa una casos para el valor de una variable
- **for, for...of, for...in**
- **while:** pregunto antes de entrar al ciclo
- **do-while:** Ejecuto el bloque y luego pregunto para volver a entrar

# Strings

- `let str = 'soy un string';`
- `let str2 = "yo también";`
- `let str3 = `${str2} ${str}!`; // ${cualquier expresión JS}`
- `str.length;` // indica la longitud del string
- `str.toUpperCase();` // convierte a mayúsculas
- `str.toLowerCase();` // convierte a minúsculas

más en [https://www.w3schools.com/jsref/jsref\\_obj\\_string.asp](https://www.w3schools.com/jsref/jsref_obj_string.asp)

# Arreglos

- `const arr = [2, 4, 6, 8];`
- `arr.push(el1,el2,...);` // agregar elementos al arreglo
- `arr.indexOf(valor);` // el índice de valor en arr (-1 si no está)
- `arr.slice(inicio, fin);` // retorna un subarreglo
- `arr.forEach(callback);` // `callback(item, [index, [array]])`
- `arr.map(callback);`

más en [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array)

# Objetos

- Almacena parejas clave-valor, parecido a los diccionarios en python
- Las claves deben ser strings y los valores cualquier cosa, incluso funciones

```
const car = {marca:"Fiat", color:"white"};
```

```
console.log(car.type, car["color"]);
```

```
car.modelo = "2015";
```

```
car.pitar=function(){console.log('piiiii')};
```

```
car.pitar(); // piiiiii
```

# Objetos

- `"clave" in objeto;` // verifica si "clave" está en el objeto
- `delete car.marca;` //elimina la clave "marca" y su valor asociado
- `Object.keys(car);` // retorna arreglo de clave
- `Object.values(car);` // retorna arreglo de valores
- `Object.entries(car);` // retorna arreglo de [clave, valor]

# Funciones

- `function nombre(params){}`
- `const nombre = function(params){}`
- Funciones flecha:
  - `const nombre = (params) => {}`
  - `var => {};` // si 1 solo parámetro, no necesita paréntesis
  - `var => var*2;` // si es una sola expresión, no necesita llaves y retorna implícitamente el resultado de la expresión
- Funcion constructor:
  - `function Plantilla(param1, ...){...}`
  - `const myObj = new Plantilla(val1,...);`

# Desestructuración

- `let [uno, dos] = arr; //primer y segundo elementos se asignan a uno y dos, respectivamente`
- `let {marca, color} = car; //los atributos marca y color se asignan a las 2 variables, respectivamente (los nombre importan)`
- `let [uno, ...otro] = arr; //uno es el primer valor y el resto del array queda en la variable (array) otro`



# Modulos (exportar)

- “use strict”
- Variables no son accesibles desde el exterior
- Hay que exportar lo que se requiera
  - `export let variable = ...;`
  - `export const fn = ()=>{...};`
  - `export default`
  - `export {...};`

# Módulos (importar)

- `import myName from "./modulo.js";` // importa default como myName
- `import { fn } from "./modulo.js";` // importa fn que se exportó en el módulo (en nombre debe coincidir)
- `import myName, { fn, variable } from "./modulo.js";` // se pueden combinar
- La ruta debe iniciar con `"./"` o `"../"` (rutas relativas, como en Linux)

# Módulos (usar en HTML)

- `<script type="module" src="RUTA"></script>`

# Clases

- Class Carro{

static cantidad = 0; // métodos y propiedades pueden ser estáticos (variables de clase)

constructor(marca, color){ // constructor no puede ser asíncrono

Carro.cantidad++;

this.marca = marca; // propiedades (variables de instancia)

this.color = color;

}

pitar(){ // método

console.log(`piiiita el \${this.marca} \${this.color}`);

}

const miCarro = new Carro("mazda", "verde");

miCarro.pitar(); //piiiita el mazda verde

# Clases

- Variables/métodos privados (su nombre inicia con #)
  - Pensar bien qué debe ser privado (cosas que no tengan sentido que se vean afuera de la clase) para no usar getter y setter triviales
- Getter y Setter
  - `get name(){...}; // → console.log(miCarro.name);`
  - `set name(val){...}; // → miCarro.name = nuevoValor;`
- Constructor no puede ser asíncrono
- `static` para definir atributos estáticos (de clase)

# Programación asíncrona

- Callbacks: función que se pasa como argumento de otra función para ser llamada más adelante → callback hell
  - `function(callbackOk, callbackError){}`
- Promise (algo que va a pasar en el futuro) → `then()`, `catch()`, `all()`, `any()`
  - Pending → fulfilled (success/result) → rejected (failed/error)
  - `New Promise((suc, rej) => {...});` → encadenamiento de promesas
- `async/await` → Sintaxis más moderna que parece código sincrónico
  - `async (...params) => {...await...};` // Retorna una promesa por defecto

# Excepciones

- try/catch

```
try{ // flujo "normal" esperado
```

```
...
```

```
}catch(err){ // si algo falla, ocurre algún error
```

```
  console.log(err.stack);
```

```
}
```

- throw <expresion>; // puedo lanzar cualquier cosa
- new Error(mensaje); // Mejor si es un error con un mensaje descriptivo. Se pueden crear subclases de error y se crea una traza del error (stack trace)

# Document Object Model (DOM)

- Modela el documento como un árbol de objetos
- **window**: Objeto global, información de la ventana del navegador
- **document**: DOM
  - document.body, document.head
  - .parentElement // elemento padre
  - .children // colección de elementos hijos. .length, [indice], .id



# Document Object Model (DOM)

- Los atributos HTML se pueden acceder como propiedades JS: src, href, id, style, className, classList, etc
- elemento.textContent: obtener o asignar el texto en el elemento
- Elemento.innerHTML puede conllevar riesgos de seguridad
- .getElementById(), .getElementsByClassName(), .getElementsByTagName()
- .querySelector(<selector css>) (1ro), querySelectorAll(<selector css>) (listado)

# Document Object Model (DOM)

- `document.createElement(tag);`
- `padre.append(hijo);`
- `padre.prepend(hijo);`
- `padre.remove();`

# DOM Eventos

- `.addEventListener(tipo, manejador)`
  - `button.addEventListener("click", (ev)=>{...})`
  - `ev.preventDefault();` // Evitar el comportamiento por defecto
  - `ev.stopPropagation();` // Evitar que se propague a los padres
  - `ev.stopImmediate();` // Parar en este manejador
- `.removeEventListener(tipo, manejador)`
  - Manejador debe ser la misma función que cuando se agregó

# DOM Eventos

- Tipos:
  - Mouse: click, mouseenter, mouseleave
  - Teclado: keydown, keyup, keypress
  - Interacción: change, input, focus, blur, submit, reset

Más en: [https://www.w3schools.com/tags/ref\\_eventattributes.asp](https://www.w3schools.com/tags/ref_eventattributes.asp)

- `ev.currentTarget`; // elemento donde se está manejando el evento
- `ev.target`; // elemento donde se generó el evento

# JSON

- JavaScript Object Notation
  - Como un objeto JS, pero las claves tienen que ir entre comillas dobles. No incluye los métodos
  - `JSON.stringify(objeto);` // Convierte un objeto JS a string JSON
  - `JSON.parse(texto);` // Convierte un string JSON a objeto JS

```
let obj = {a: 78, b: "hola", c: function(a,b){return a+b;}};
```

```
JSON.stringify(obj); // '{"a":78,"b":"hola"}'
```

```
let str = '{a: 65}';
```

```
JSON.parse(str); /*VM350:1 Uncaught SyntaxError:
```

```
    Expected property name or '}' in JSON
```

```
    at JSON.parse (<anonymous>)
```

```
    at <anonymous>:1:6
```

```
    (anonymous) @ VM349:1*/
```

```
let str = '{"a": 65}'
```

```
JSON.parse(str); //{a: 65}
```

# APIs, fetch, async/await

- API: Conjunto de mensajes que pueden intercambiar 2 sistemas:
  - Define qué y cómo puede preguntar el cliente (browser)
  - Define qué y cómo puede responder el servidor (web server)
- Permite actualizar la información que se ve sin necesidad de cargar toda la página de nuevo
- El cliente actualiza las partes del DOM que corresponda según los datos recibidos

# APIs, fetch, async/await

- `fetch(url[, options])`:
  - Lee contenido desde la URL (puede ser una ruta relativa, un archivo).
  - Retorna una promesa con la respuesta (`response`)
- `response.status`: Código de estado HTTP de la respuesta
- `response.text()` - `response.json()`
  - Interpreta el body de la respuesta
  - Retorna una promesa con los datos interpretados

[https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side\\_web\\_APIs/Fetching\\_data](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data)



# APIs, fetch, async/await

- `fetch(url[, options])`
  - Url: puede contener query parámetros
  - Options : {method: ..., headers: {...}, body:{...} }
    - Method: string con el métodos HTTP (**GET**,POST,PATCH,PUT,DELETE, etc)
    - Headers: objeto con encabezados HTTP (content-type, user-agent, etc)
    - Body: Los datos a enviar. String JSON
      - NO se usa cuando el método es GET

- `await` espera por la promesa. Solo puede usarse en funciones `async`
  - Si el estado es `rejected`, se lanza una excepción (usar `try-catch`)
- `async` indica que se va a usar `await`
  - La función retorna una promesa de lo que normalmente retorna

```
const makeRequest = async () => {  
  let response = await fetch("./data.json"); // Espera por la promesa  
  console.log(response.status);  
  let text = await response.json(); // Espera por la promesa  
  console.log(text);  
};
```

# REST API

- REpresentational State Transfer
  - Recurso: cada cosa que queremos enviar o recibir
    - Se identifica con una URI. Ej: /users/emanuel, /cursos/daw
    - El servidor retorna una representación del estado del recurso
    - El cliente envía la representación del estado del recurso para actualizarlo (o parte de el)
  - Sin estado: El servidor no recuerda al cliente
    - Cada petición lleva la URI y demás información necesaria

# HTTP (petición)

## HTTP Request

Method      URL      Protocol Version

↓            ↓            ↓

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: ISO-8859-1,utf-8
Connection: keep-alive
blank line
```

Headers {

Body (optional) {

# HTTP (petición)

- Método: Indica qué quiero hacer
  - GET: obtener información de un recurso
  - POST: enviar información
  - PUT: actualizar información (recurso completo)
  - PATCH: actualizar parte de la información del recurso
  - DELETE: eliminar recurso

# HTTP (petición)

- METODO **URI**?**QUERY\_STRING**

Ejemplo:

GET **/cursos/daw**?year=2024&student=25

- **METODO**: Qué quiero hacer
- **URI**: dirección del recurso
- **QUERY\_STRING**: información adicional acerca del recurso
  - Lo que hablamos antes sobre query params
  - Clave-valor, separados por &

# HTTP (petición)

- Ejemplo:

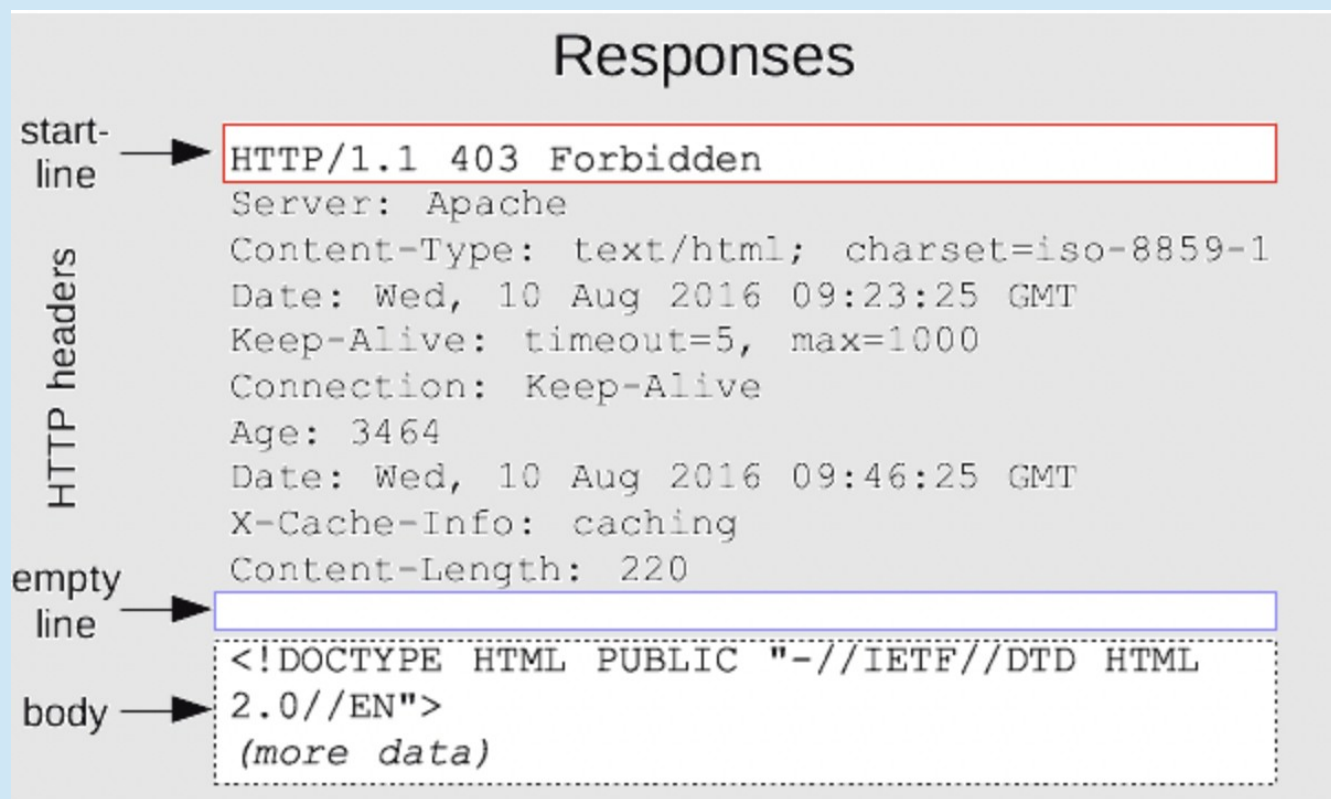
POST /cursos/daw/matricular

Content-Type: application/json

{"nombre": "nuevo", "apellido": "estudiante"}

- Encabezados (Headers): Información sobre la petición
- Body: Datos que se envían al servidor (no en GET)

# HTTP (respuesta)





# HTTP (respuesta)

- Ejemplo:


HTTP/1.1 200 OK

Content-Type: application/json

```
{"id":"daw","year":"2024",...}
```

- Código (200) y mensaje de estado (OK)
- Encabezados (Headers): Información acerca de la respuesta
- Cuerpo de la respuesta (Body): El (estado) del recurso, mensaje de error, etc

# HTTP (estados)



|     |                     |  |
|-----|---------------------|--|
| 1XX | Informational codes | The server acknowledges and is processing the request.   |
| 2XX | Success codes       | The server successfully received, understood, and processed the request.   |
| 3XX | Redirection codes   | The server received the request, but there's a redirect to somewhere else (or, in rare cases, some additional action other than a redirect must be completed). |
| 4XX | Client error codes  | The server couldn't find (or reach) the page or website. This is an error on the site's side.  |
| 5XX | Server error codes  | The client made a valid request, but the server failed to complete the request.  |

<https://www.semrush.com/blog/http-status-codes/>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

# REST API/fetch/async

- Recibir datos desde el servidor

```
const getData = async () => {  
  const BASE_URL = "http://localhost:8080";  
  const URI = "/cursos/daw";  
  const PARAMS = "year=2024" ; //new URLSearchParams({year: 2024})  
  let response = await fetch(`${BASE_URL}${URI}?${PARAMS}`);  
  console.log(response.status);  
  let datos = await response.json();  
  console.log(datos);  
};
```

# REST API/fetch/async

- Enviar datos al servidor

```
const postData = async
```

```
  const BASE_URL = "http://localhost:8080";
```

```
  const URI = "/cursos/daw/matricular";
```

```
  let newSt= { nombre: "emanuel", apellido: "montoya" };
```

```
  let res = await fetch(`${BASE_URL}${URI}`, {
```

```
    method: "POST",
```

```
    headers: { "Content-Type": "application/json" },
```

```
    body: JSON.stringify(newSt)
```

```
  }
```

```
};
```

```
...
```

```
};
```

# REST API + Clases

- Componente: Gestiona DOM/ eventos
  - bind a this para los métodos que manejan eventos  
metodo = ()=>{...}
- Modelo: Gestiona los datos (llamados a la API)
- Para casos simples, se pueden mezclar