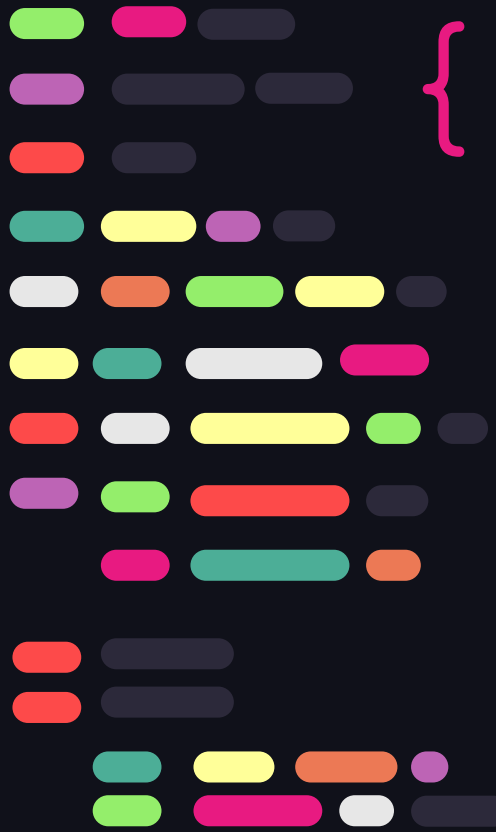
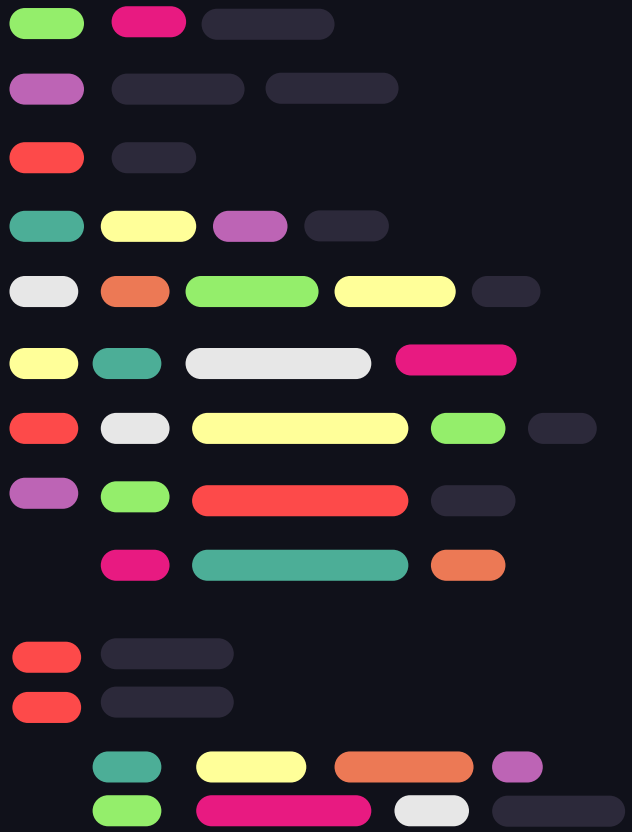


Javascript Development

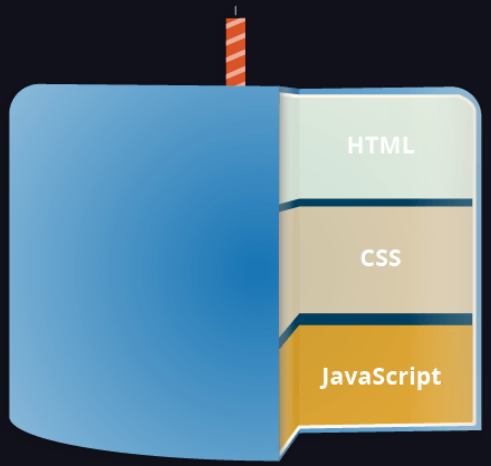
Juan Carlos Trejo



Modulo 1 }

Introducción a HTML

Introducción



Componentes de una pagina web.

HTML

¿Qué es HTML?

HTML (HyperText Markup Language) es un lenguaje de marcado utilizado para crear la estructura y contenido básico de las páginas web.

HTML

¿Qué es HTML?

La esencia de HTML radica en el uso de etiquetas (tags) para definir el contenido y la estructura de una página web. Cada etiqueta se compone de elementos angulares **("<>" o corchetes)** y, en ocasiones, tiene atributos que proporcionan información adicional sobre el elemento.

} . .

HTML

Elemento HTML

Un elemento HTML se define mediante una etiqueta de inicio, algo de contenido y una etiqueta de finalización.

```
<tagname>
    El contenido va aquí...
</tagname>
```

HTML

Página HTML

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>¡Mi primera página web!</title>
</head>
<body>
    <h1>Bienvenido a mi página web</h1>
    <p>Esta es una introducción básica a HTML.</p>
    <img src="imagen.jpg" alt="Descripción de la imagen">
    <a href="https://www.ejemplo.com">Enlace a otro sitio web</a>
</body>
</html>
```

Una página HTML básica comienza con la etiqueta <!DOCTYPE html> que especifica el tipo de documento y define la versión de HTML que se está utilizando (generalmente HTML5).

Luego, dentro del documento, se utiliza la etiqueta <html> para indicar el inicio y fin del contenido HTML.

HTML

Página HTML

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>¡Mi primera página web!</title>
</head>
<body>
    <h1>Bienvenido a mi página web</h1>
    <p>Esta es una introducción básica a HTML.</p>
    <img src="imagen.jpg" alt="Descripción de la imagen">
    <a href="https://www.ejemplo.com">Enlace a otro sitio web</a>
</body>
</html>
```

Dentro del elemento <html>, se encuentran dos secciones principales:

- **La sección <head>:** Aquí se incluyen metadatos y enlaces a recursos externos como hojas de estilo CSS, código JavaScript etc.
- **La sección <body>:** En este bloque, se coloca el contenido visible de la página, como texto, imágenes, enlaces, videos y otros elementos multimedia.

HTML

Página HTML

```
<html>
  <head>
    <title> Titulo de la pagina </title>
  </head>
  <body>
    <h1> Titulo 1 </h1>
    <p> Párrafo 1 </p>
  </body>
</html>
```

HTML

Atributos

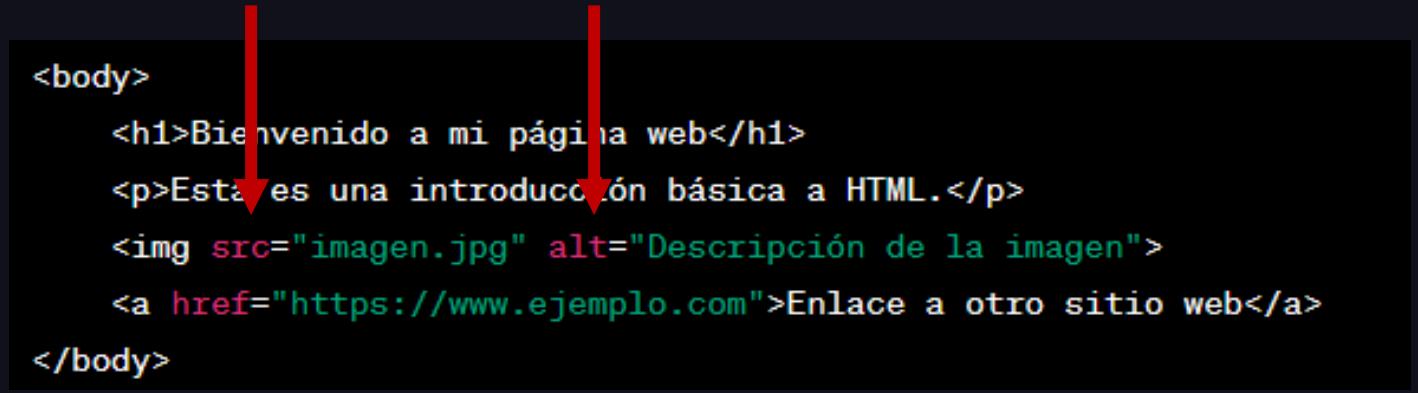
Los atributos HTML proporcionan información adicional
sobre los elementos o etiquetas HTML.

- Todos los elementos HTML pueden tener atributos
- Los atributos proporcionan información adicional sobre los elementos.
- Los atributos siempre se especifican en la etiqueta de inicio.
- Los atributos generalmente vienen en pares de nombre/valor como:
  nombre="valor"

} . .

HTML

Atributos



HTML

Explorador

El propósito del navegador web al abrir un documento HTML es interpretar y renderizar la página web para que los usuarios puedan visualizar y navegar por su contenido de manera interactiva.

HTML

Etiquetas

Las etiquetas de HTML se pueden dividir en diferentes categorías según su función y propósito.

- Encabezados
- Párrafos y texto
- Listas
- Enlaces e imágenes
- Tablas
- Formularios
- Multimedia

HTML elements reference - HTML: HyperText Markup Language | MDN (mozilla.org)

HTML

Etiquetas (Encabezados)

Etiquetas que se utilizan para definir títulos y subtítulos en una página: <h1>, <h2>, <h3>, <h4>, <h5>, <h6>.

```
<body>
    <h1>Este es un encabezado de nivel 1</h1>
    <p>Este es un párrafo de texto.</p>

    <h2>Este es un encabezado de nivel 2</h2>
    <p>Este es otro párrafo de texto.</p>

    <h3>Este es un encabezado de nivel 3</h3>
    <p>Un tercer párrafo de texto.</p>
</body>
```

HTML

Etiquetas (Encabezados)

Etiquetas que se utilizan para definir títulos y subtítulos en una página: <h1>, <h2>, <h3>, <h4>, <h5>, <h6>.

```
<body>
    <h1>Este es un encabezado de nivel 1</h1>
    <p>Este es un párrafo de texto.</p>

    <h2>Este es un encabezado de nivel 2</h2>
    <p>Este es otro párrafo de texto.</p>

    <h3>Este es un encabezado de nivel 3</h3>
    <p>Un tercer párrafo de texto.</p>
</body>
```

HTML

Etiquetas (Párrafos y texto)

Etiquetas para formatear texto y párrafos. Ejemplos: <p>, , , , <blockquote>, <pre>.

```
<body>
    <p>Este es un párrafo de texto regular.</p>

    <blockquote>
        <p>Esta es una cita bloque.</p>
        <p>
            Las citas bloque suelen estar
            indentadas y resaltadas para
            distinguirlas del texto regular.
        </p>
    </blockquote>
</body>
```

HTML

Etiquetas (Listas)

Etiquetas para crear listas ordenadas y desordenadas. Ejemplos: <ul>, <ol>, <li>, <dl>, <dt>, <dd>.

```
<body>
    <h2>Lista desordenada (ul)</h2>
    <ul>
        <li>Manzanas</li>
        <li>Naranjas</li>
        <li>Plátanos</li>
    </ul>
</body>
```

HTML

Etiquetas (Enlaces e imágenes)

```
<body>
    <h2>Enlace de imagen</h2>
    <a href="https://www.ejemplo.com">
        <img src="imagen.jpg" alt="Descripción de la imagen">
    </a>

    <h2>Enlace para enviar un correo electrónico</h2>
    <p>Ponte en contacto con nosotros mediante
        <a href="mailto:info@ejemplo.com">info@ejemplo.com</a>
    </p>
</body>
```

Etiquetas para enlaces a otras páginas o recursos y para mostrar imágenes. Ejemplos: <a>, <img>

HTML

Etiquetas (Tablas)

Etiquetas para crear tablas de datos. Ejemplos: <table>, <tr>, <td>, <th>.

```
<body>
    <table>
        <tr>
            <th>Nombre</th>
            <th>Edad</th>
            <th>Ciudad</th>
        </tr>
        <tr>
            <td>Juan</td>
            <td>25</td>
            <td>Madrid</td>
        </tr>
    </table>
</body>
```

HTML

Etiquetas (Formulario)

```html
<body>
    <form action="/enviar_datos" method="post">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre" required>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>

        <label for="mensaje">Mensaje:</label>
        <textarea id="mensaje" name="mensaje" rows="4" required></textarea>

        <button type="submit">Enviar</button>
    </form>
</body>
```

Etiquetas para crear formularios interactivos.
Ejemplos: <form>, <input>, <textarea>, <select>, <button>.

HTML

Etiquetas (Video y audio)

Etiquetas para integrar contenido multimedia, como audio y video. Ejemplos: <audio>, <video>.

```
<body>
    <h1>Audio</h1>
    <audio controls>
        <source src="audio.mp3" type="audio/mpeg">
    </audio>

    <h1>Video</h1>
    <video width="640" height="360" controls>
        <source src="video.mp4" type="video/mp4">
    </video>
</body>
```

HTML

Etiquetas (Meta información)

La etiqueta <meta> en un documento HTML se utiliza para proporcionar metadatos, es decir, información adicional sobre el documento HTML.

Estos metadatos no se muestran directamente en la página web, pero son utilizados por los navegadores web, motores de búsqueda y otros servicios para comprender y procesar el contenido de la página de manera más efectiva.

} . .

HTML

Etiquetas (Meta información)

Los metadatos pueden incluir:

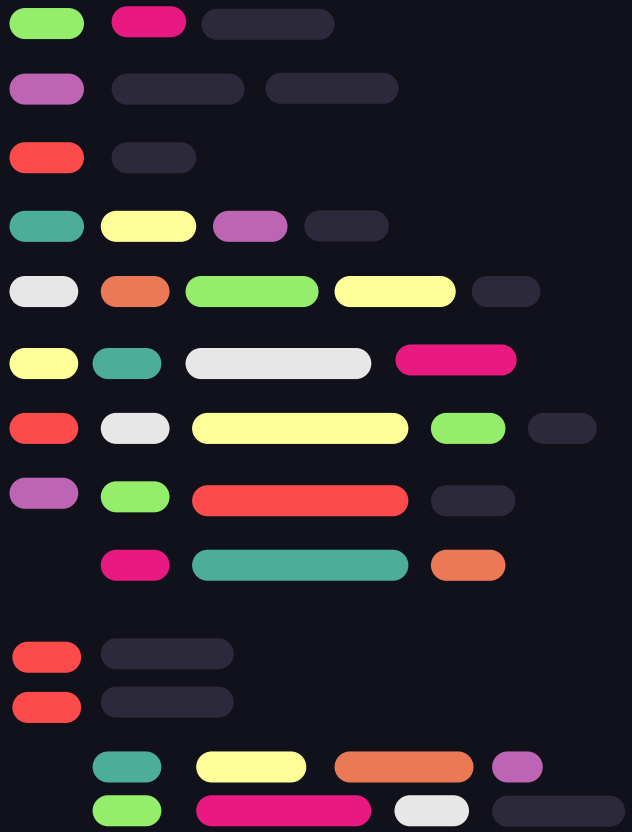
- Información sobre el juego de caracteres utilizado en el documento.
- La descripción del contenido.
- El autor del documento.
- Palabras clave relevantes.
- La configuración de la vista del portátil para dispositivos móviles.
- Otras propiedades importantes del documento.

HTML

Etiquetas (Meta información)

Etiquetas para proporcionar información sobre el documento o controlar su comportamiento. Ejemplos: <meta>, <title>, <link>, <base>.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <meta name="keywords" content="ejemplo, meta, etiquetas, HTML">
    <meta name="description" content="Este es un ejemplo del uso de etiquetas de meta en HTML.">
    <meta name="author" content="Tu Nombre">
    <title>Ejemplo de etiquetas de meta en HTML</title>
</head>
<body>
    <h1>Ejemplo de etiquetas de meta en HTML</h1>
    <p>Contenido de la página...</p>
</body>
</html>
```



Modulo 2 }

Introducción a CSS

CSS

CSS (Cascading Style Sheets) es un lenguaje de hojas de estilo utilizado para describir la presentación visual de un documento escrito en lenguaje HTML (HyperText Markup Language)

CSS

Selectores

Un selector es una parte de una regla que indica a qué elementos HTML se aplicará un conjunto específico de estilos. Los selectores se utilizan para seleccionar elementos individuales o grupos de elementos en un documento HTML y aplicarles un estilo determinado.

```
<selector> {
    property: value;
}
```

} . .

CSS

Selectores

Selector	Descripción	Ejemplo
Selector de elemento	Selecciona elementos HTML por su nombre de etiqueta.	**p** seleccionará todos los párrafos
Selector de clase	Selecciona elementos con un atributo de clase específico.	**.destacado** seleccionará elementos con la clase "destacado"
Selector de ID	Selecciona un elemento por su atributo de ID único.	**#encabezado** seleccionará el elemento con el ID "encabezado"
Selector universal	Selecciona todos los elementos en el documento.	* seleccionará todos los elementos

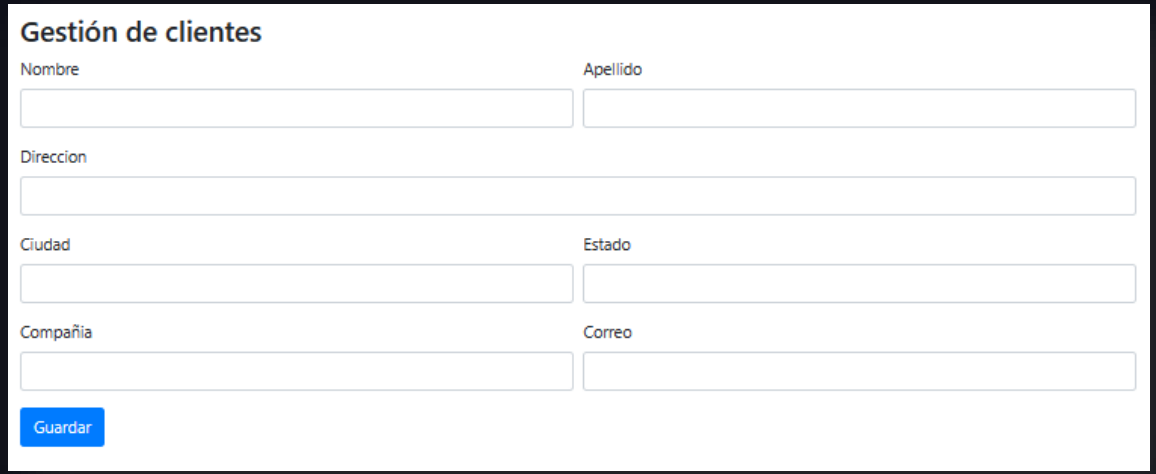
CSS

Selectores

Selector	Descripción	Ejemplo
Selector de atributo	Selecciona elementos que tienen un atributo específico.	**input[type="text"]** seleccionará todos los elementos input de tipo "text"
Selector de descendiente	Selecciona elementos descendientes directos o indirectos de otro elemento.	**div p** Seleccionará todos los párrafos que estén dentro de un elemento div
Selector de hijo	Selecciona elementos que son hijos directos de otro elemento.	**ul > li** seleccionará todos los elementos li que son hijos directos de un elemento ul

CSS

Ejercicio

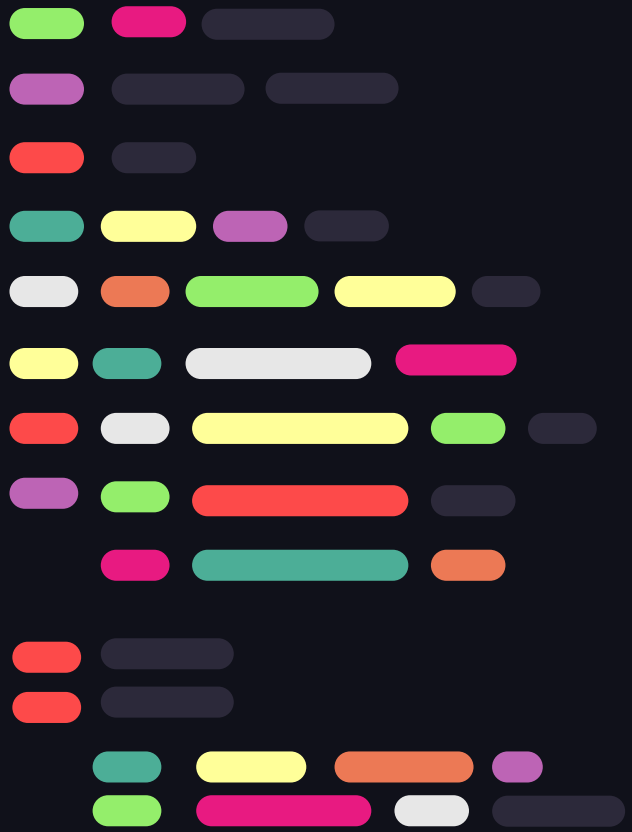


CSS

Ejercicio

Relacion de clientes

Nombre	Apellido	Direccion	Ciudad	Estado	Compañia	Correo electronico	Acciones
Pedro actualizado	martinez	Direccion conocida	CDMX	CDMX	Personal	null	Eliminar Actualizar
Andrea	Rosales	Direccion conocida	CDMX	CDMX	Personal	coreeo@correo.com	Eliminar Actualizar
Luara	Martinez	110 Raeburn Pl	Edinburgh	Georgia	Desconocida	correobueno@correo.com	Eliminar Actualizar
Juan	Trejo	Direccion conocida	CDMX	CDMX	Personal	coreeo@correo.com	Eliminar Actualizar
Diego	Gutiérrez	307 Macacha Güemes	Buenos Aires	null	null	diego.gutierrez@yahoo.ar	Eliminar Actualizar



Modulo 3 }

Introducción a Javascript

Javascript

¿Qué es Javascript?

JavaScript es un lenguaje de programación o de secuencias de comandos que te permite implementar funciones complejas en páginas web.

Javascript

Creación y primeros años (1995-1999)1995:

- Brendan Eich creó JavaScript en solo 10 días en Netscape. Inicialmente se llamaba Mocha, luego LiveScript, y finalmente JavaScript debido a la popularidad de Java.
- 1997: Se creó el primer estándar de JavaScript bajo ECMAScript (ES1), estableciendo una base para futuras versiones.

Javascript

Estándar ECMAScript (2000-2009)

- **1999**: Se lanzó **ECMAScript 3 (ES3)**, una versión clave que introdujo muchas de las características fundamentales como el manejo de excepciones y expresiones regulares.

- **2000**: Internet Explorer 5 introdujo el objeto **XMLHttpRequest**, permitiendo las primeras versiones de AJAX, que más tarde revolucionarían el desarrollo web interactivo.

Javascript

Renacimiento (2009-2015)

- 2009: Se lanzó ECMAScript 5 (ES5), una actualización importante que incluyó muchas mejoras como strict mode, JSON nativo, getters y setters, y métodos de arrays avanzados.
- 2009: Se creó Node.js por Ryan Dahl, permitiendo el uso de JavaScript del lado del servidor y ampliando su uso fuera del navegador.
- 2010: jQuery se convirtió en la biblioteca de JavaScript dominante, simplificando mucho la manipulación del DOM y el manejo de eventos.

Javascript

Era moderna (2015 en adelante)

- 2015: Lanzamiento de ECMAScript 6 (ES6/ES2015), la versión más significativa desde ES3. Introdujo clases, promesas, let/const, arrow functions, módulos y generadores, estableciendo a JavaScript como un lenguaje más moderno y robusto..
- 2017: ECMAScript 2017 trajo funcionalidades clave como async/await, facilitando el manejo de operaciones asíncronas.

Javascript

Tendencias actuales

- 2016: Se consolidan frameworks/librerías como React, Angular y Vue.js, permitiendo la creación de aplicaciones web más eficientes y escalables.
- 2019 en adelante: Las herramientas de desarrollo como TypeScript (un superconjunto de JavaScript que añade tipado estático).

Javascript

¿Qué se puede hacer en Javascript?

- Agregar nuevo HTML a la página, cambiar el contenido existente y modificar estilos.
- Reaccionar a las acciones del usuario, ejecutarse con los clics del ratón, movimientos del puntero y al oprimir teclas.
- Enviar solicitudes de red a servidores remotos, descargar y cargar archivos.

} . .

Javascript

¿Qué se puede hacer en Javascript?

- Obtener y configurar cookies, hacer preguntas al visitante y mostrar mensajes.
- Recordar datos en el lado del cliente con el almacenamiento local (“local storage”).

} . .

Javascript

Software a usar

- Visual Studio Code
Download Visual Studio Code - Mac, Linux, Windows

- Google Chrome
Google Chrome - Download the Fast, Secure Browser from Google

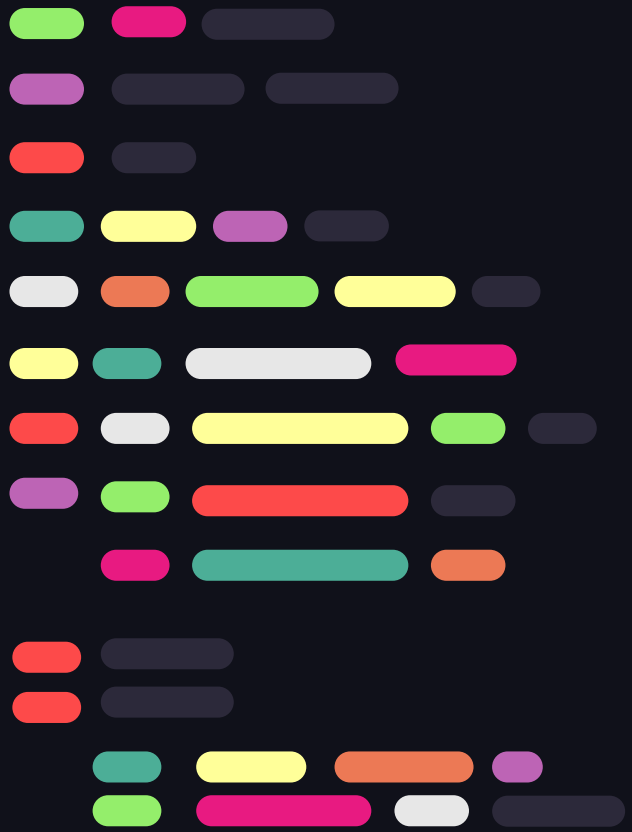
Javascript

Manuales

- La especificación EMAC
ECMAScript® 2021 Language Specification (ecma-international.org)

- MDN Mozilla Refrence
JavaScript reference - JavaScript | MDN (mozilla.org)

} . .



Modulo 4 }

Fundamentos de javascript:
Lo Básico

Lo básico

Tipos de datos

Hay ocho tipos de datos básicos en JavaScript. Podemos almacenar un valor de cualquier tipo dentro de una variable. Por ejemplo, una variable puede contener en un momento un string y luego almacenar un número.

```
1  // no hay error
2  let message = "hola";
3  message = 123456;
```

Lo básico

Tipos de datos

Number	Para números de cualquier tipo: enteros o de punto flotante, los enteros están limitados por $\pm(2^{53}-1)$.
Bigint	Para números enteros de longitud arbitraria.
String	Para cadenas. Una cadena puede tener cero o más caracteres, no hay un tipo especial para un único carácter.
boolean	Para verdadero y falso: true/false.

Lo básico

Tipos de datos

Null	Para valores desconocidos - un tipo independiente que tiene un solo valor nulo: null
Undefined	Para valores no asignados - un tipo independiente que tiene un único valor indefinido: undefined
Symbol	Para identificadores únicos.
Object	Para estructuras de datos complejas.

Lo básico

Variables

Una variable es un “almacén con un nombre” para guardar datos. Existen 3 formas de declarar una variable:

1. let
2. const
3. var

} . .

Lo básico

Variables

```
let message;
message = 'Hola!';
alert(message);
```

```
let user = 'John', age = 25, message = 'Hola';
```

```
let user = 'John';
let age = 25;
let message = 'Hola';
```

```
let user = 'John',
  age = 25,
  message = 'Hola';
```

Lo básico

Variables (const)

```
const COLOR_RED = "#F00";
const COLOR_GREEN = "#0F0";
const COLOR_BLUE = "#00F";
const COLOR_ORANGE = "#FF7F00";
```

Una variable de tipo const es inmutable, por lo cual solo se puede guardar una vez y sólo una vez un valor. Tales constantes se nombran utilizando letras mayúsculas y guiones bajos.

} . .

Lo básico

Variables (var)

Las variables declaradas con var pueden tener a la función como entorno de visibilidad, o bien ser globales. Su visibilidad atraviesa los bloques.

```
if (true) {
  var test = true;
}

alert(test)
```

```
if (true) {
  let test = true;
}

alert(test);
```

Lo básico

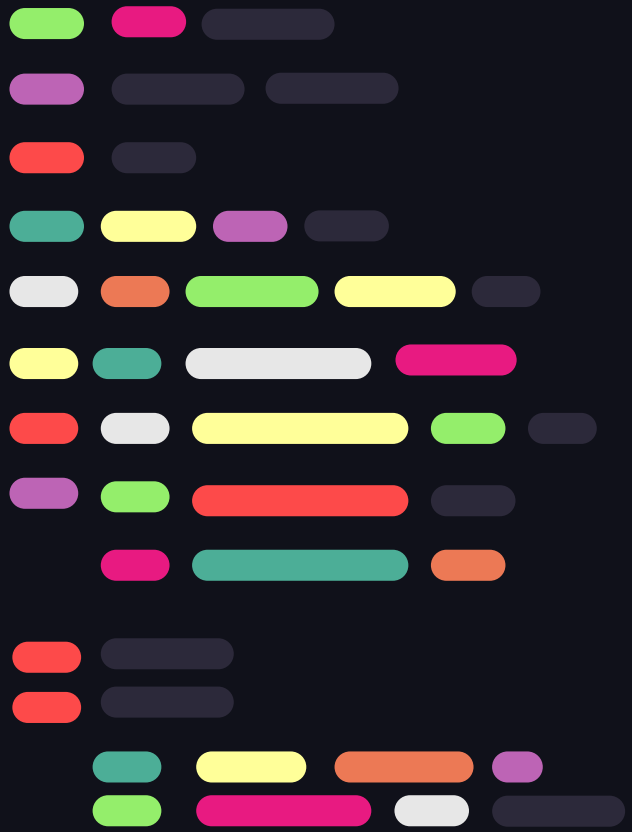
Operaciones aritméticas

Operación	Operador aritmético	Expresión en Javascript
Suma	+	variable1 + variable2
Resta	-	variable1 - variable2
Multiplicación	*	variable1 * variable2
División	/	variable1 / variable2
Residuo	%	variable1 % variable2

Lo básico

Operaciones relacionales

Operador estándar	Instrucción de javascript	Expresión en Javascript
=	x == y	x igual a y
!=	x != y	x es diferente de y
>	x > y	x es mayor a y
<	x < y	x es menor a y
<=	x <= y	x es menor o igual a y
>=	x >= y	x es mayor o igual a y



Modulo 5 }

Introducción a los objetos

Introducción a los objetos

Objeto

Los objetos son usados para almacenar colecciones de varios datos y entidades más complejas asociados con un nombre clave. En JavaScript, los objetos penetran casi todos los aspectos del lenguaje.

Introducción a los objetos

Objeto

Los objetos son usados para almacenar colecciones de varios datos y entidades más complejas asociados con un nombre clave. En JavaScript, los objetos penetran casi todos los aspectos del lenguaje.

```
// Sintaxis de "constructor de objetos"
let usuario = new Object();
// Sintaxis de "objeto literal"
let usuario = {};
```

Introducción a los objetos

Objeto

Se puede declarar un objeto y agregar propiedades. Una propiedad tiene una clave (también conocida como “nombre” o “identificador”) antes de los dos puntos ":" y un valor a la derecha.

```
// Objeto user
let user = {
    // En la clave "name" se almacena el valor "John"
    name: "John",
    // En la clave "age" se almacena el valor 30
    age: 30
};
```

Introducción a los objetos

Objeto (propiedades)

Se puede declarar un objeto y agregar propiedades. Una propiedad tiene una clave (también conocida como “nombre” o “identificador”) antes de los dos puntos ":" y un valor a la derecha.

```
// Objeto user
let user = {
    // En la clave "name" se almacena el valor "John"
    name: "John",
    // En la clave "age" se almacena el valor 30
    age: 30
};
```

Introducción a los objetos

Objeto

1. Agregar una propiedad

```
user.isAdmin = false;
alert(user);
```

2. Eliminar una propiedad

```
delete user.name;
alert(user);
```

Introducción a los objetos

Objeto

3. Acceder a una propiedad

```
// Objeto user
let user = {
    // En la clave "name" se almacena el valor "John"
    name: "John",
    // En la clave "age" se almacena el valor 30
    age: 30
};


console.log(user.age);
console.log(user['name']);
```

Introducción a los objetos

Objeto (Destructuring)

La sintaxis de asignación de desestructuración es una expresión de JavaScript que hace posible descomprimir valores de matrices o propiedades de objetos en distintas variables.

```
// Objeto user
let user = {
    // En la clave "name" se almacena el valor "John"
    name: "John",
    // En la clave "age" se almacena el valor 30
    age: 30
};

const {nombre, edad} = producto;
```

Introducción a los objetos

Array

Es una estructura o colección para insertar datos de forma ordenada.

Índice →	0	1	2	3	4	5	6	7	8	9
Valor →	10	20	30	40	50	60	70	80	90	100

Introducción a los objetos

Array

Es una estructura o colección para insertar datos de forma ordenada.

```
let arreglo = new Array();
let arreglo = [];
```

```
let frutas = ["Apple", "Orange", "Plum"];

alert( frutas[0] ); // Apple
alert( frutas[1] ); // Orange
alert( frutas[2] ); // Plum
```

Introducción a los objetos

Array

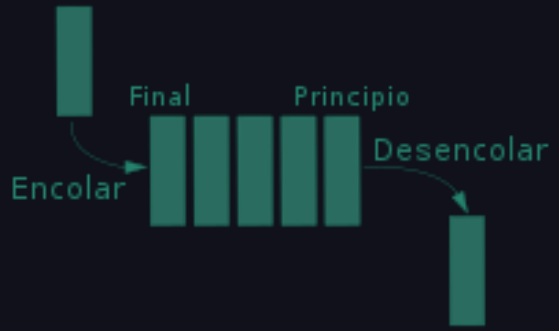
Operaciones comunes

```
let frutas = ["Apple", "Orange", "Plum"];

// Agregar un nuevo elemento
fruits[3] = 'Lemon';
// Reemplazar un elemento
fruits[2] = 'Watermelon';
// Obtener el tamanio del arreglo
console.log(fruits.legth);
// Eliminar un elemento por indice
delete fruits[1];
```

Introducción a los objetos

Array

Cola (push/shift)

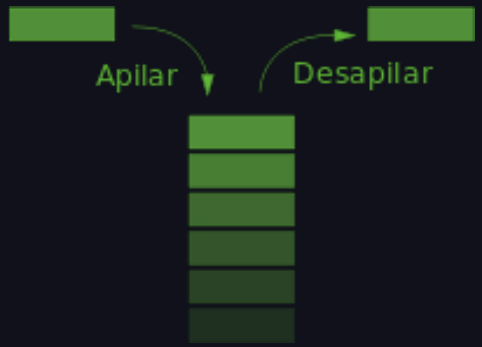



```
// Creamos un arreglo inicial
let miArreglo = [1, 2, 3, 4, 5];

// Utilizamos el método 'shift' para
// eliminar el primer elemento del arreglo
let primerElemento = miArreglo.shift();

// Utilizamos el método 'push' para
// agregar elementos al final del arreglo
miArreglo.push(6, 7);

console.log("Arreglo final:", miArreglo);
```

Introducción a los objetos

Array

Pila (push/pop)

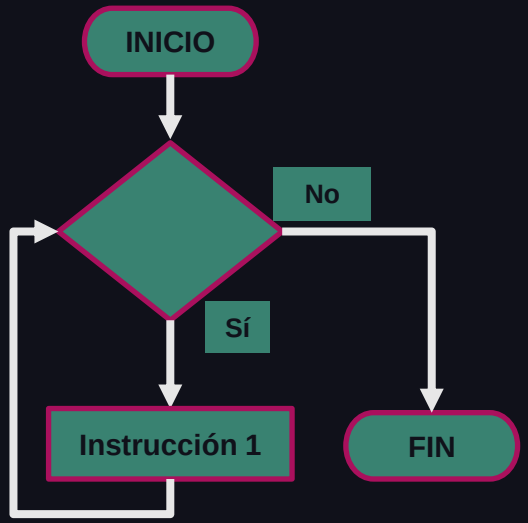



```
// Creamos un arreglo inicial
let miArreglo = ["manzana", "banana", "cereza"];

// Utilizamos el método 'push' para
// agregar un elemento al final del arreglo
miArreglo.push("durazno");

// Utilizamos el método 'pop' para
// eliminar el último elemento del arreglo
let ultimoElemento = miArreglo.pop();

// Salida final
console.log("Arreglo final:", miArreglo);
```

Introducción a los objetos

Array

Ciclos




```
let arreglo = ["Apple", "Orange", "Pear"];

// Iterar sobre indices del array
for (let i = 0; i < arreglo.length; i++) {
  alert( arreglo[i] );
}

// iterar sobre los elementos del array
for (let fruit of fruits) {
    alert( fruit );
  }
```

Introducción a los objetos

Array

Ciclos

El método forEach itera sobre cada elemento del arreglo y ejecuta la función proporcionada en cada iteración. En cada iteración, el valor del elemento actual se asigna automáticamente al parámetro de la función, lo que nos permite realizar alguna operación con él.

```
// Creamos un arreglo de frutas
let frutas = ["manzana", "banana", "cereza"];

// Utilizamos el método 'forEach' para iterar
// sobre cada elemento del arreglo
frutas.forEach(function(fruta) {
  console.log(fruta);
});
```

Introducción a los objetos

Array

Ciclos

El método map crea un nuevo arreglo que contiene los valores resultantes de aplicar la operación a cada elemento del arreglo original.

```
// Creamos un arreglo de números
let numeros = [1, 2, 3, 4, 5];

// Utilizamos el método 'map' para realizar
// una operación en cada elemento del arreglo
let duplicados = numeros.map(function(numero) {
  return numero * 2;
});
```

Introducción a los objetos

Array

Filter

El método filter en JavaScript se utiliza para crear un nuevo arreglo con todos los elementos que cumplan cierta condición especificada en una función de filtrado.

```
// Creamos un arreglo de números
let numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Utilizamos el método 'filter' para obtener
// los números pares del arreglo
let numerosPares = numeros.filter(function(numero) {
  return numero % 2 === 0;
});

console.log("Números pares:", numerosPares);
```

Introducción a los objetos

Array

Destructuring

La destructuración es una característica de JavaScript que permite extraer elementos de arreglos o propiedades de objetos y asignarlos a variables individuales de forma más concisa.

```
let numeros = [1, 2, 3];

let [a, b, c] = numeros;

console.log(a); // Resultado: 1
console.log(b); // Resultado: 2
console.log(c); // Resultado: 3
```

Introducción a los objetos

Ejercicios

3.1 Acceder a las propiedades uno a uno de objeto JSON.
3.2 Iterar sobre el objeto JSON los pares clave - valor.
3.3 Convertir el objeto JSON en una cadena String en formato JSON.
3.4 Convertir una cadena en formato JSON en un objeto JSON.

```
let persona = {
    nombre: "Juan",
    edad: 25,
    ciudad: "Madrid"
};
```

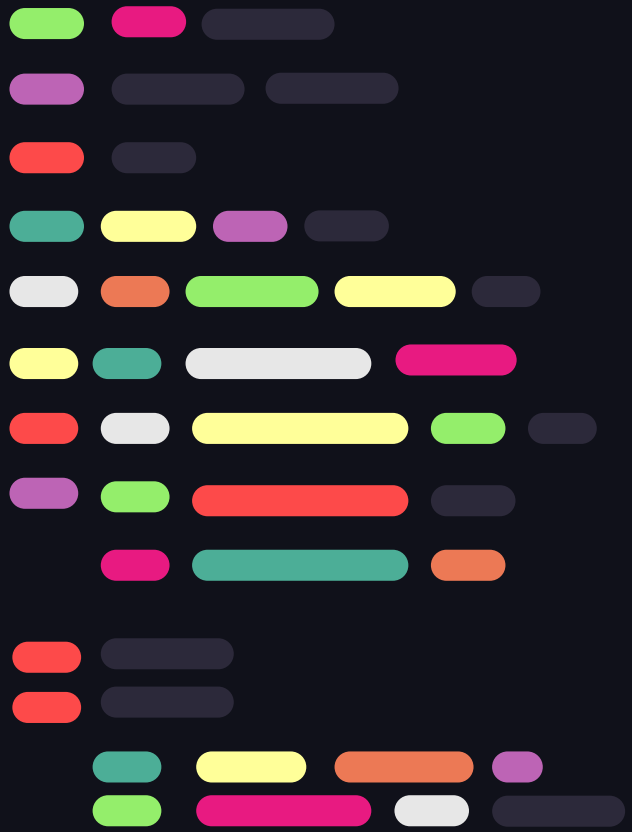
Introducción a los objetos

Ejercicios

3.5 Dados los siguientes objetos, hacer un destructuring de ambos.
3.6 Dados los siguientes objetos, crear un nuevo objeto usando ambos.

```
const cliente = {
    nombre: "Juan",
    edad: 25,
    ciudad: "Madrid"
};
```

```
const producto = {
    nombre: "Cellphone",
    precio: 2500,
    marca: "Motorola"
};
```



Modulo 6 }

Funciones

Funciones

Las funciones son los principales “bloques de construcción” del programa. Permiten que el código se llame muchas veces sin repetición.

```
function name(parameter1, parameter2, ... parameterN) {
    // body
}
```

Funciones

Las funciones son los principales “bloques de construcción” del programa. Permiten que el código se llame muchas veces sin repetición.

```
function showMessage() {
    alert( '¡Hola a todos!' );
}

showMessage();
showMessage();
```

Funciones

Variables

Si una variable con el mismo nombre se declara dentro de la función, le *hace sombra* a la externa.

```
let usuario = 'John';

function showMessage() {
  let usuario = "Bob"; // declara variable local

  let mensaje = 'Hello, ' + usuario;
  // ¿Cual es la salida?
  alert(mensaje);
}

// la función crea y utiliza su propia variable local usuario
showMessage();

// ¿Cual es la salida?
alert( usuario );
```

Funciones

Parámetros

Se pueden pasar N numero de parámetros a una función.

```
// parámetros: from, text
function showMessage(from, text) {
    alert(from + ': ' + text);
}
// Llamar la funcion
showMessage('Ann', '¡Hola!');
```

Funciones

Parámetros por default

Si una función es llamada, pero no se le proporciona un argumento, su valor correspondiente se convierte en undefined. Podemos especificar un valor llamado “predeterminado” en la declaración de función usando “=”

```
function showMessage(from, text = "sin texto") {
    alert( from + ": " + text );
}

showMessage("Ann");
```

Funciones

Expresiones de función

En JavaScript, una función no es una estructura del lenguaje, sino un tipo de valor especial. Existe otra sintaxis para crear una función que se llama una Expresión de Función.

```
let saludar = function() {
    alert( "Hola" );
};

typeof saludar;
```

Funciones

Expresiones de función

```
function saludar() {   // a) Crear la funcion
    alert( "Hola" );
}

let copia = saludar;    // b) Hacer una copia

copia(); // Hola        // c) Ejecutar la funcion copia
saludar(); // Hola      // d) Ejecutar la funcion original
```

Funciones

Funciones callback

```
function saludar(nombre) {
    alert('Hola ' + nombre);
}

function procesarEntradaUsuario(callback) {
    var nombre = prompt('Por favor ingresa tu nombre.');
    callback(nombre);
}

procesarEntradaUsuario(saludar);
```

Una función de callback es una función que se pasa a otra función como un argumento, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

Funciones

Funciones flecha

Una función de flecha (también conocida como "arrow function" en inglés). Proporciona una sintaxis más concisa y simplificada para crear funciones en comparación con las funciones tradicionales.

```
(parametro1, parametro2, ....) => {
    // Cuerpo de la función
    // Puede contener una o varias instrucciones
    return resultado;
}
```

Funciones

Funciones flecha

```
//Funcion 1
const sumar1 = function(num1, num2) {
    return num1 + num2;
};
//Funcion 2
const sumar2 = (num1, num2) => {
    return num1 + num2;
};
//Funcion 3
const sumar3 = (num1, num2) => num1 + num2;
```

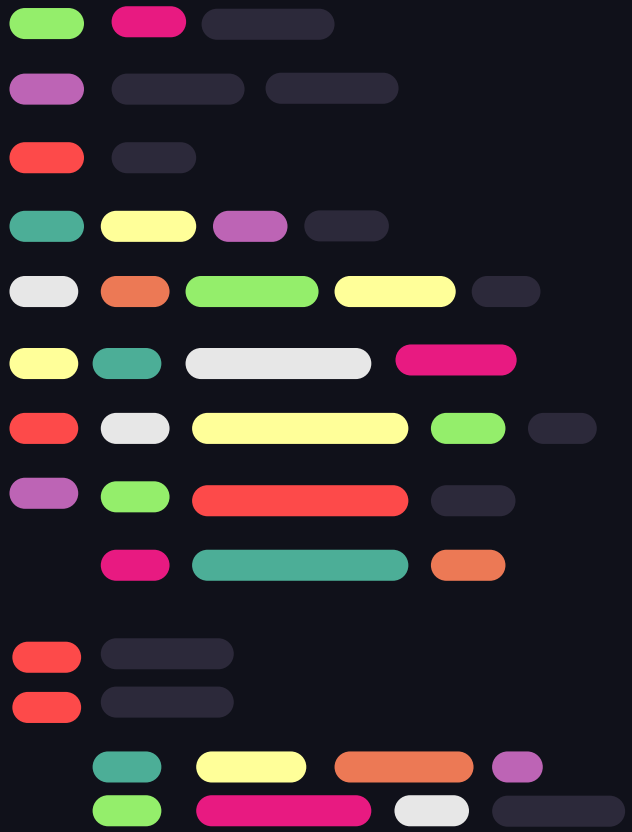
Funciones

Ejercicios

Dado el siguiente array y mediante arrow functions:

```
let numeros = [1, 2, 3, 4, 5];
```

4.1 Elevar al cuadrado todos los elementos de un arreglo.
4.2 Filtrar los números pares de un arreglo.
4.3 Sumar todos los elementos de un arreglo.
4.4 Verificar si todos los elementos de un arreglo son mayores que 0.

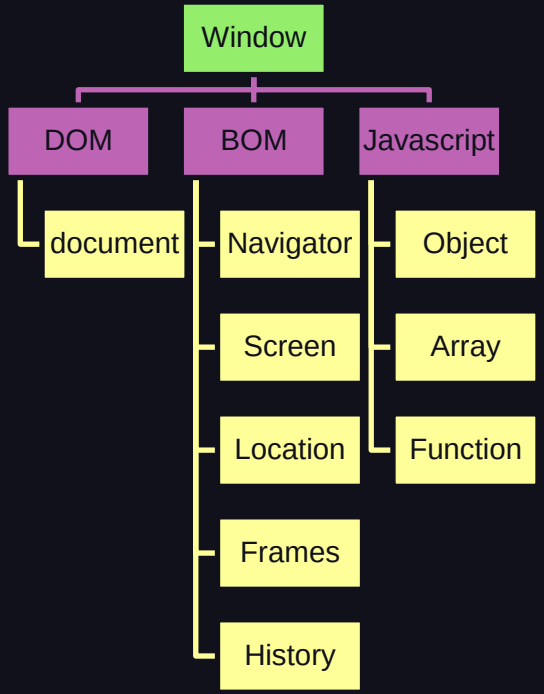


Modulo 7 }

DOM
(Modelo de Objetos del Documento)

DOM

Cuando se ejecuta el explorador, carga por detrás un entorno que da soporte a todas las operaciones mismas del explorador.



DOM

BOM (Modelo de Objetos del Navegador)

El BOM (Browser Object Model) son objetos adicionales proporcionados por el navegador (entorno host) para trabajar con todo excepto el documento.

Por ejemplo:

- El objeto navigator proporciona información sobre el navegador y el sistema operativo.
- El objeto location nos permite leer la URL actual y puede redirigir el navegador a una nueva.

DOM

DOM (Modelo de Objetos del Documento)

- La estructura de un documento HTML son las etiquetas.
- Según el DOM, cada etiqueta HTML es un objeto. Las etiquetas anidadas son llamadas “hijas” de la etiqueta que las contiene. El texto dentro de una etiqueta también es un objeto.
- Todos estos objetos son accesibles empleando JavaScript, y podemos usarlos para modificar la página.

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>
            Titulo del documento
        </title>
    </head>

    <body>
        Cuerpo del documento
    </body>
</html>
```

DOM

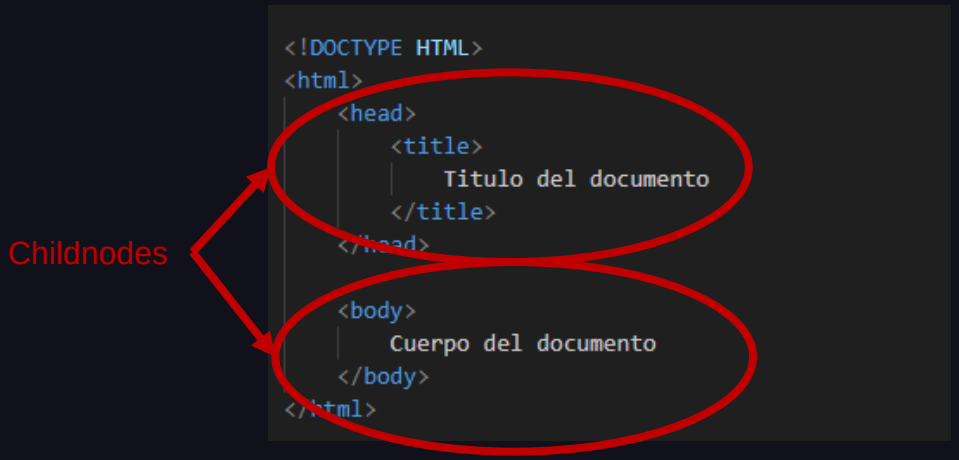
DOM (Modelo de Objetos del Documento)

- El DOM nos permite hacer cualquier cosa con sus elementos y contenidos, pero lo primero que tenemos que hacer es llegar al objeto correspondiente del DOM.
- Todas las operaciones en el DOM comienzan con el objeto **document**. Este es el principal “punto de entrada” al DOM.

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>
            Titulo del documento
        </title>
    </head>

    <body>
        Cuerpo del documento
    </body>
</html>
```

DOM

Recorriendo el DOM



Un childNode son hijos directos, es decir sus descendientes inmediatos. Por ejemplo, <head> y <body> son hijos del elemento <html>.

DOM

Recorriendo el DOM

```html
<!DOCTYPE HTML>
<html>
    <head>
        <title>
            Titulo del documento
        </title>
    </head>

    <body>
        Cuerpo del documento
    </body>
</html>
```

```js
for (let i = 0; i < document.body.childNodes.length; i++) {
    console.log( document.body.childNodes[i] );
}
```

DOM

Selectores

Los "Query Selectors" o "Selectores de consulta" en JavaScript son métodos que te permiten seleccionar elementos HTML en el documento utilizando una sintaxis similar a la de los selectores de CSS (hojas de estilo en cascada).

DOM

Selectores

getElementById()	Selecciona un elemento por su atributo "id".
getElementsByClassName()	Selecciona elementos por su atributo "class".
getElementsByTagName()	Selecciona elementos por su etiqueta.
querySelector()	Selecciona el primer elemento que coincida con el selector especificado.
querySelectorAll()	Selecciona todos los elementos que coincidan con el selector especificado y devuelve una lista (NodeList) de elementos.

DOM

Eventos

Un evento es una señal de que algo ocurrió. Todos los nodos del DOM generan dichas señales (pero los eventos no están limitados sólo al DOM).

DOM

Eventos

Eventos del mouse	
Click	Cuando el mouse hace click sobre un elemento (los dispositivos touch lo generan con un toque).
Contextmenu	Cuando el mouse hace click derecho sobre un elemento.
Mouseove	Cuando el cursor del mouse ingresa/abandona un elemento.
Mousedown	Cuando el botón del mouse es presionado/soltado sobre un elemento.
Mousemove	Cuando el mouse se mueve.

DOM

Eventos

Eventos del teclado	
Keydown/Keyup	Cuando se presiona/suelta una tecla.

Eventos del formulario	
submit	Cuando el visitante envía un <form>.
focus	Cuando el visitante se centra sobre un elemento, por ejemplo un <input>.

DOM

Registrar un evento

```
<input
    type="button"
    value="Haz click"
    onclick="evento()">
<script>
    const evento = function() {
        alert("Hola");
    }
</script>
```

A través del atributo de la etiqueta se registra el tipo de evento deseado y se le asocia una función ya existente.

DOM

Registrar un evento

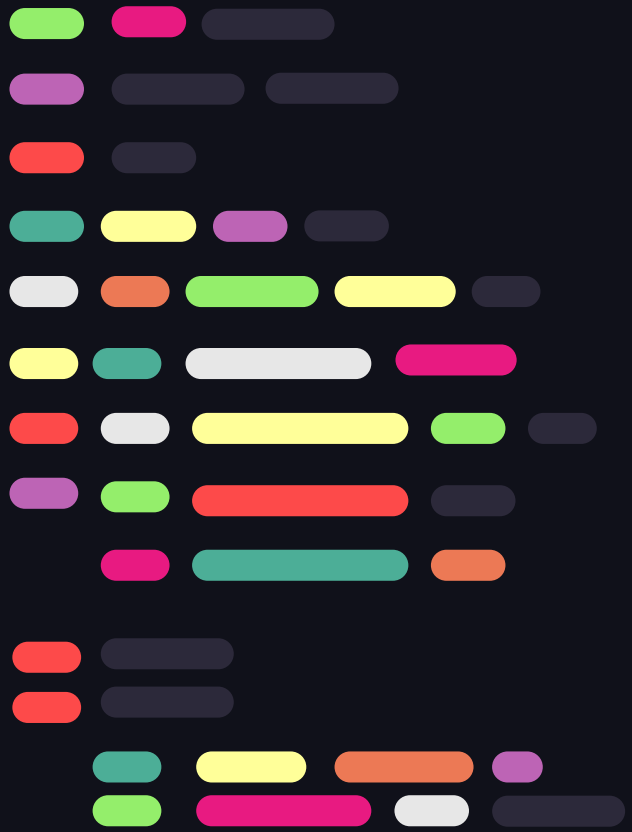
```
<input
    type="button"
    value="Haz click"
    id="miBoton">

<script>
    const evento = document.getElementById("miBoton")

    evento.onclick = () => {
        alert("Hola Mundo");
    };
</script>
```

A través del atributo id de la etiqueta se obtiene la referencia y se le asocia el evento deseado y una función.

DOM

Ejercicios

1. Acceder a todas las etiquetas de enlaces (a).
2. Acceder a todas las etiquetas a través de su clase.
3. Acceder a todas las etiquetas de entrada (input).
4. Agregar un nuevo enlace (a)
5. Agregar una nueva caja de texto (input)
6. Cambiar el titulo de la pagina.
7. Agregar un valor a la caja de texto para el nombre.
8. Eliminar un elemento del documento.
9. Cambiar el color del fondo de la pagina.
10. Agregar una función para el botón del formulario para que recoja los valores del usuario y los muestre en consola.



Modulo 8 }

Solicitudes de red

Solicitudes de red

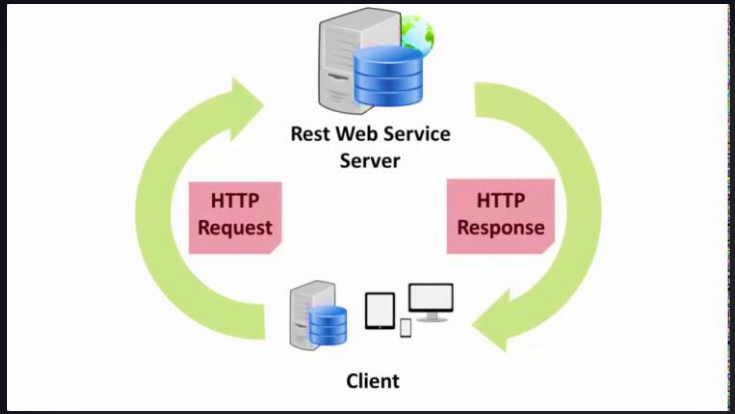
JavaScript puede enviar peticiones de red al servidor y cargar nueva información siempre que se necesite. Por ejemplo, se puede utilizar una petición de red para:

- Crear una orden,
- Cargar información de usuario,
- Recibir las últimas actualizaciones desde un servidor.

Solicitudes de red

Servicios Web

¿Qué es un servicio web? No es más que una invocación de método remoto (RMI) realizada a través de HTTP utilizando un mecanismo de transporte basado en texto (un documento XML en este caso).



Solicitudes de red

API REST

API

Application Programming Interface

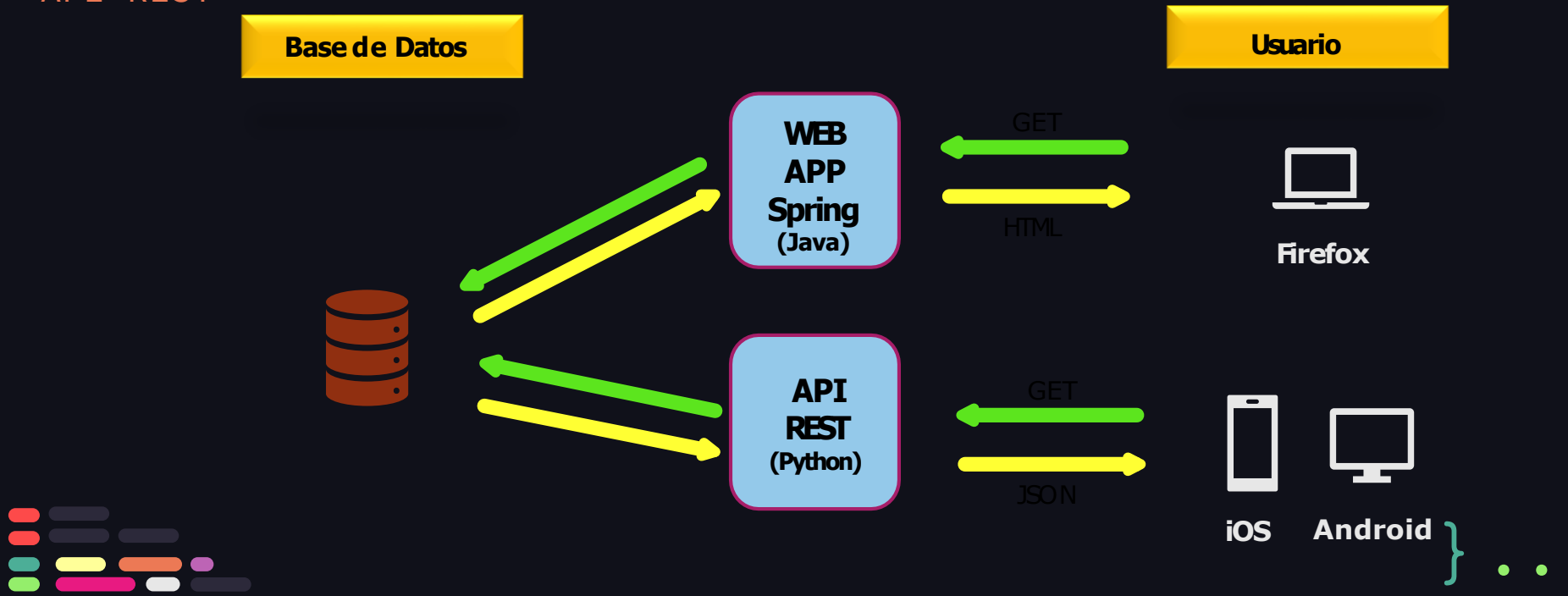
REST

Representational State Transfer

Interfaz de aplicaciones para transferir datos

Solicitudes de red

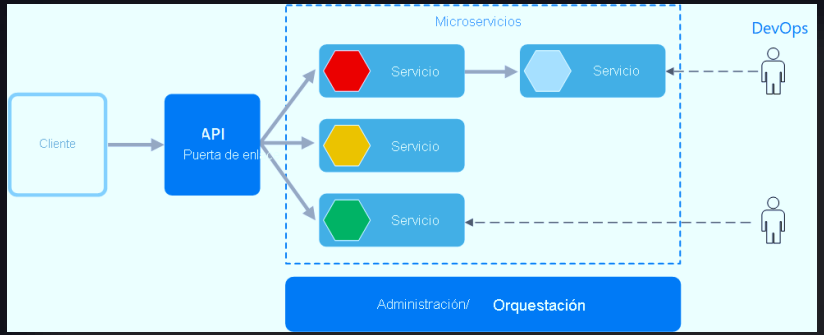
API REST



Solicitudes de red

Microservicios

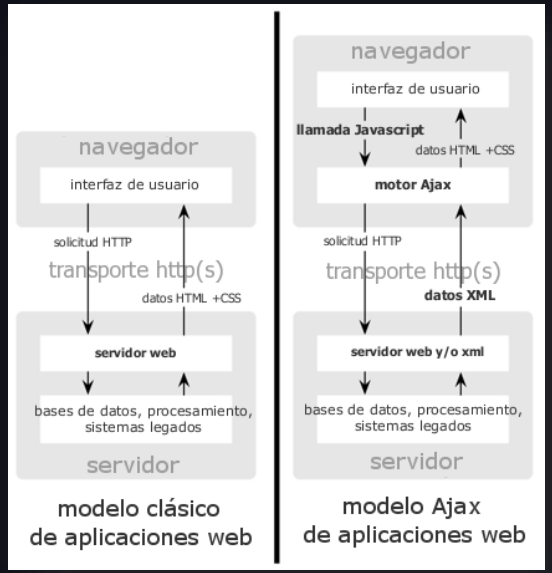
Una arquitectura de microservicios consta de una colección de servicios autónomos y pequeños. Cada uno de servicio es independiente y debe implementar una funcionalidad de negocio individual dentro de un contexto delimitado.



Solicitudes de red

AJAX

AJAX permite que una página web que ya ha sido cargada solicite nueva información al servidor.



Solicitudes de red

AJAX

AJAX permite que una página web que ya ha sido cargada solicite nueva información al servidor.

```
const xhr = new XMLHttpRequest();
const url = 'https://api.example.com/data';

xhr.open('GET', url, true);

xhr.onreadystatechange = function() {
  if (xhr.status === 200) {
    const response = JSON.parse(xhr.responseText);
    // Hacer algo con la respuesta recibida
  } else if (xhr.status !== 200) {
    // Manejar errores de la solicitud
  }
};

xhr.send();
```

Solicitudes de red

Promise

Una Promise (promesa) en JavaScript es un objeto que representa la eventual finalización (éxito o fracaso) de una operación asíncrona y la entrega de su resultado.

```
function fetchData() {
    return new Promise((resolve, reject) => {
      // Simulando una operación asincrónica
      setTimeout(() => {
        const data = 'Datos obtenidos de forma exitosa';
        if (data) {
            // La promesa se cumple
            resolve(data);
        } else {
            // La promesa se rechaza
            reject('Error al obtener los datos');
        }
      }, 2000); // Espera de 2 segundos
    });
}
```

Solicitudes de red

Promise (estados)

Estados de una promise:

1. Pendiente (pending): Estado inicial. Se crea una promesa y la operación está en progreso.
2. Cumplida (fulfilled): La promesa se resuelve correctamente y se obtiene el resultado esperado.
3. Rechazada (rejected): La promesa falla y se obtiene un motivo de error.

```
function fetchData() {
    return new Promise((resolve, reject) => {
      // Simulando una operación asincrónica
      setTimeout(() => {
        const data = 'Datos obtenidos de forma exitosa';
        if (data) {
            // La promesa se cumple
            resolve(data);
        } else {
            // La promesa se rechaza
            reject('Error al obtener los datos');
        }
      }, 2000); // Espera de 2 segundos
    });
}
```

Solicitudes de red

Promise (estados)

Usando la promesa

```
// Utilizando la promesa
fetchData()
  .then(data => {
    // Manejar los datos obtenidos
    console.log(data);
  })
  .catch(error => {
    // Manejar el error
    console.error(error);
  });
```

Solicitudes de red

Fetch

El método fetch() no es soportado por navegadores antiguos pero es perfectamente soportado por los navegadores actuales y modernos.

```
let promise = fetch(url, [options])
```

- **url:** representa la dirección URL a la que deseamos acceder.
- **options:** representa los parámetros opcionales, como puede ser un método o los encabezados de nuestra petición, etc.

Solicitudes de red

Fetch

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
    return response.json();
})
  .then(data => {
    // Hacer algo con los datos obtenidos
})
  .catch(error => {
    // Manejar el error de la promesa
});
```

Solicitudes de red

Comunicación asíncrona

La llamada telefónica es comunicación sincrónica, porque el mensaje se recibe al mismo tiempo que se manda.

Un mensaje de texto es comunicación asincrónica, pues es una acción que se ejecuta en un momento, pero solo finaliza cuando la otra parte lee el texto.

Solicitudes de red

Asincronía en Javascript

Lo que hacemos en este lenguaje de programación es programar acciones que se ejecutarán en caso de que otra acción suceda. Sin embargo, no podemos controlar cuándo, ni quiera si sucederá la acción.

Solicitudes de red

Asincronía en Javascript

¿Para qué sirve?

La programación orientada a eventos es aquella que prepara la ejecución de código en función de los eventos que pueden ocurrir. Es decir, preparamos el código por si un evento sucede.

Solicitudes de red

async/await

async y await en JavaScript son dos palabras clave que nos permiten transformar un código asíncrono para que parezca ser síncrono.

- La palabra clave async se utiliza en una función para envolver el contenido de la función en una promesa.
- La palabra clave await en JavaScript nos permite definir una sección de la función a la cual el resto del código debe esperar.

```
async function fetchData() {
    try {
      const response = await fetch(
        'https://api.example.com/data');

      if (!response.ok) {
        throw new Error('Error en la solicitud');
      }
      const data = await response.json();
      // Hacer algo con los datos obtenidos
      return data;
    } catch (error) {
      // Manejar el error
      console.error(error);
      throw error;
    }
}
```

Solicitudes de red

async/await

Mandando a llamar la función fetchData con async/await

```
// Llamada a la función fetchData utilizando
//then y catch
fetchData()
.then(data => {
  // Manejar los datos obtenidos
})
.catch(error => {
  // Manejar el error
});
```

Solicitudes de red

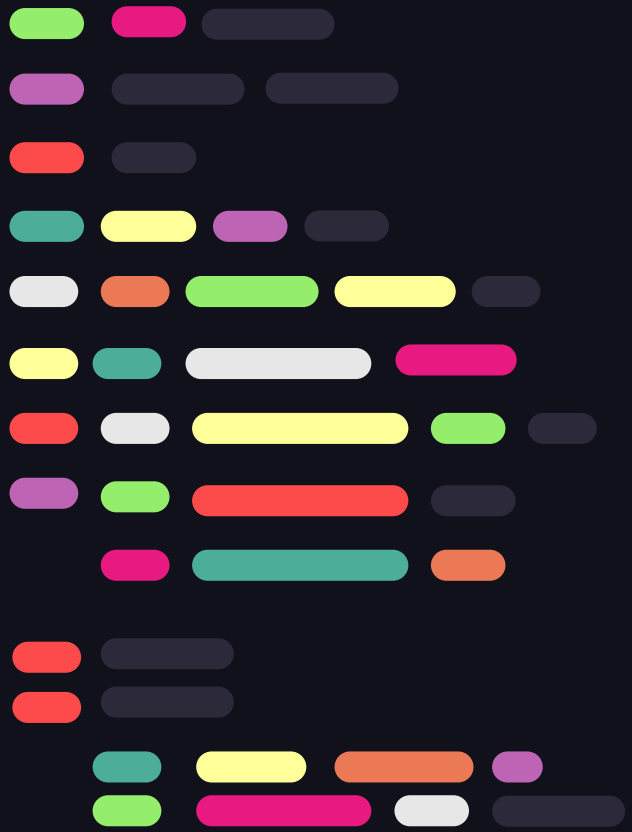
async/await

1. Consumir una API de la url JSONPlaceholder - Free Fake REST API (typicode.com) a través de un método que use una Promise y un método que use async/await.

2. Consumir dos APIs dentro de un mismo método usando async / await.

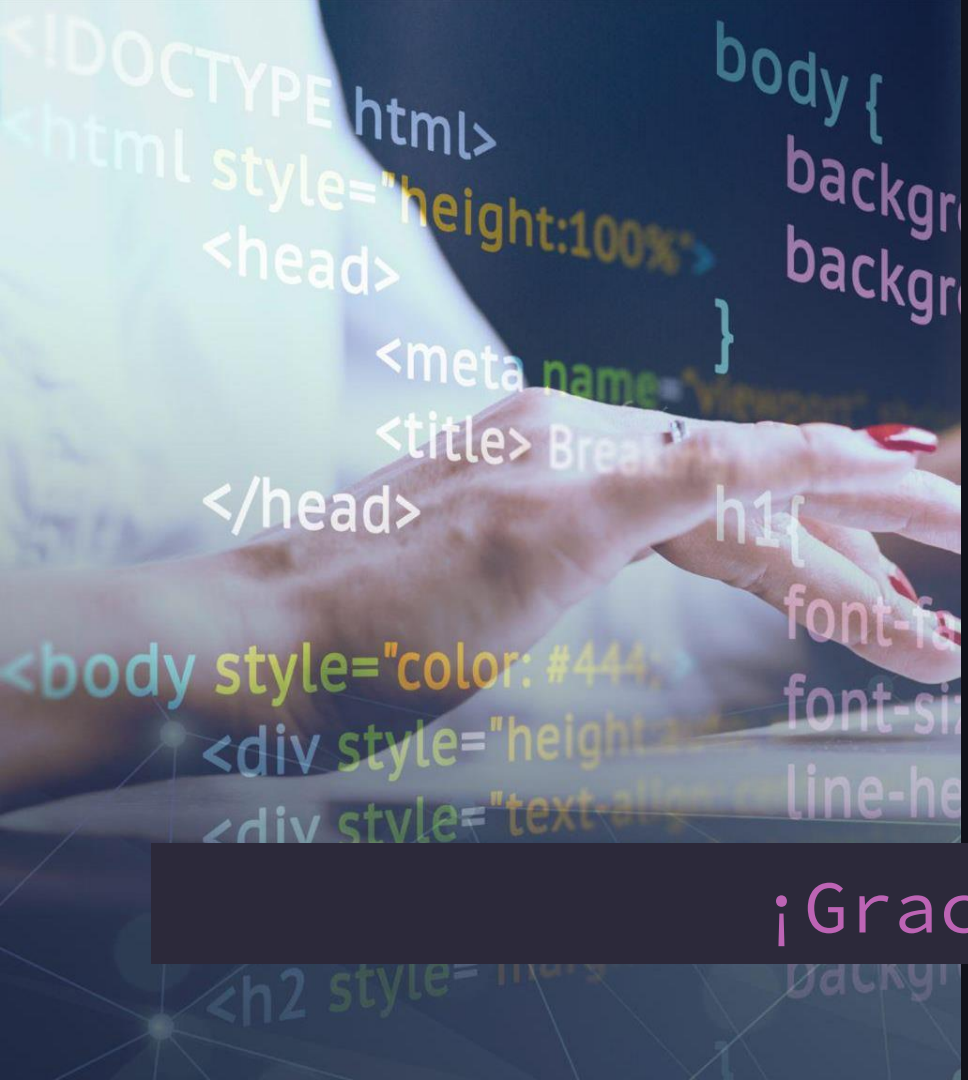
3. Consumir una API para que busque un elemento a través de su ID y llene un formulario con los datos encontrados.

} . .



Modulo 9 }

Proyecto final


<!DOCTYPE html>
<html style="height:100%">
<head>
<meta name="viewport"
<title> Brea
</head>
<body style="color: #444;
<div style="height
<div style="text-align
<h2 style="
body {
backgr
backgr
}
h1 {
font-fa
font-si
line-he
backgr

¡Gracias!