

Recuerden poner a
grabar la clase



Clase 6

DOM

Diplomatura UNTREF



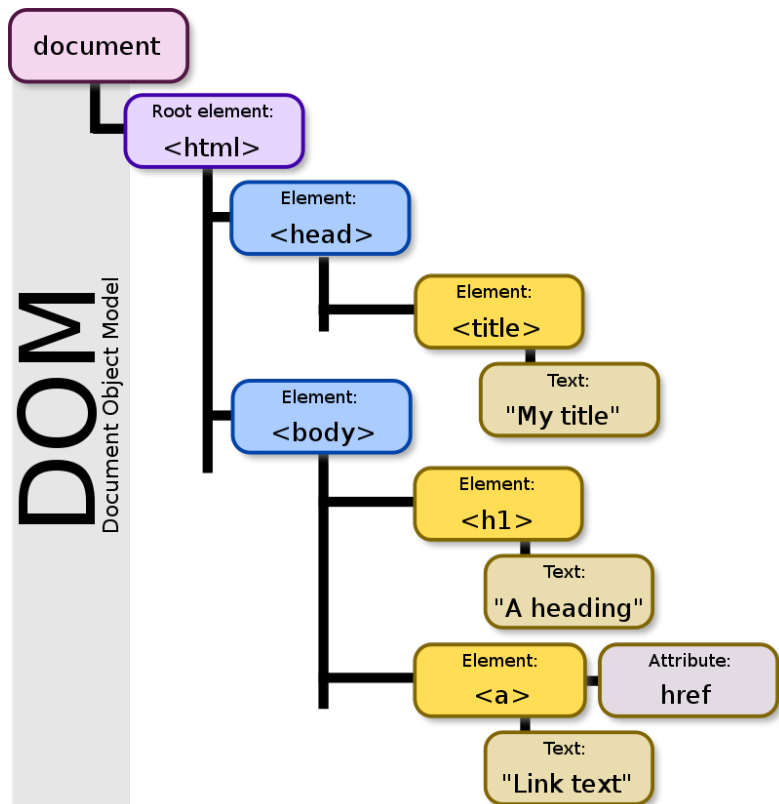
Temario

DOM:

- la importancia de defer
- El objeto Document
- El metodo getElementById()
- innertext
- innerHTML
- - El objeto Literal
- - Array de objetos literales

DOM

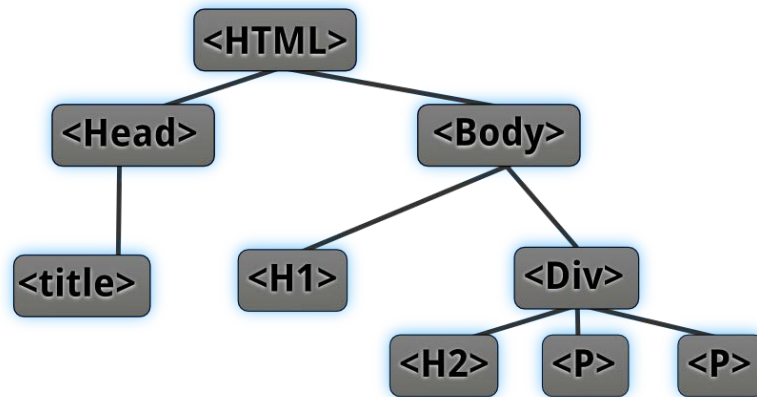
¿Qué es el DOM?



¿Definición de DOM?

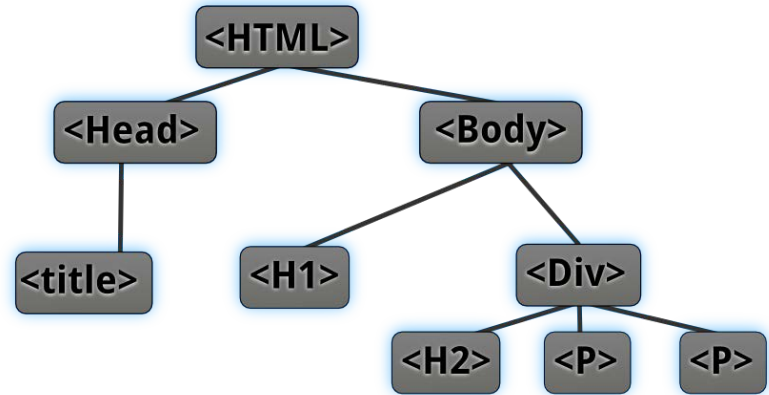
El **DOM (Document Object Model)** es una representación en memoria de un documento HTML/XML. Es una interfaz que proporciona a los programadores una forma de acceder y manipular los elementos de un documento web.

El **DOM** organiza los elementos del documento en una estructura de árbol, donde cada etiqueta HTML/XML se representa como un nodo en el árbol. Los nodos pueden tener hijos, padres y hermanos, formando una jerarquía que refleja la estructura del documento.



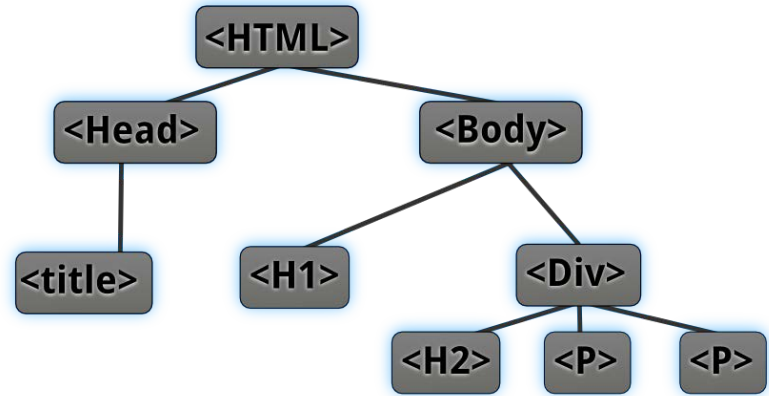
Estructura del DOM

La estructura del **DOM (Document Object Model)** es jerárquica y se organiza en forma de árbol. Cada elemento HTML o XML dentro del documento se representa como un nodo en este árbol.



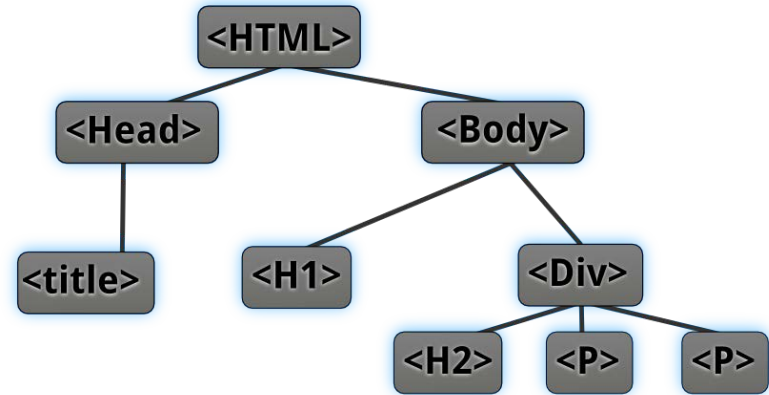
Conceptos claves del DOM

1- Nodo del DOM: Es una entidad básica en el DOM que representa un elemento individual en el árbol del documento. Hay diferentes tipos de nodos, como nodos de elemento, nodos de texto, nodos de atributo, etc.



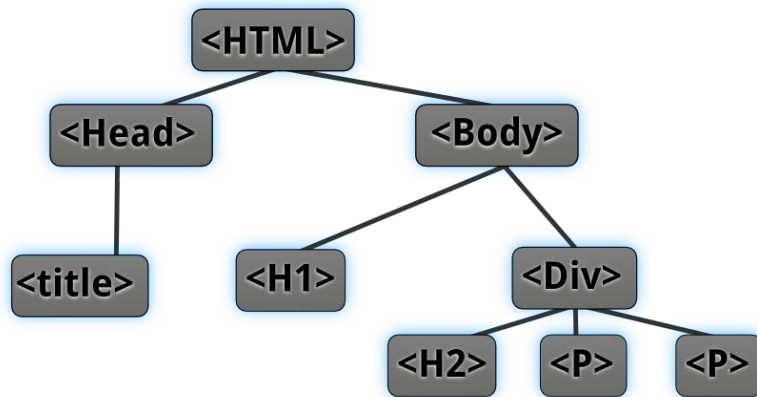
Conceptos claves del DOM

2- Árbol del DOM: Es la representación en forma de árbol de la estructura del documento. El nodo raíz del árbol del DOM es el nodo `<html>`, que contiene todos los demás nodos del documento. A partir de ahí, se ramifica en nodos hijos y se forma una jerarquía.



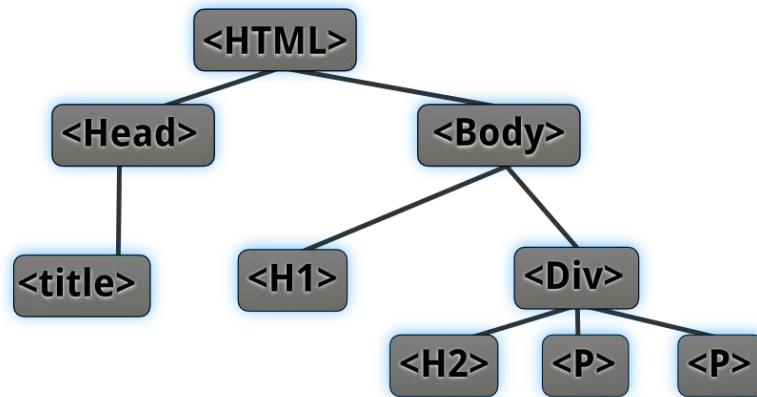
Conceptos claves del DOM

3- Nodos hijos, padres y hermanos: Los nodos en el árbol del DOM pueden tener relaciones entre sí. Un nodo hijo es un nodo directamente descendiente de otro nodo. Un nodo padre es el nodo inmediato superior de un nodo hijo. Los nodos que tienen el mismo nodo padre se llaman nodos hermanos.



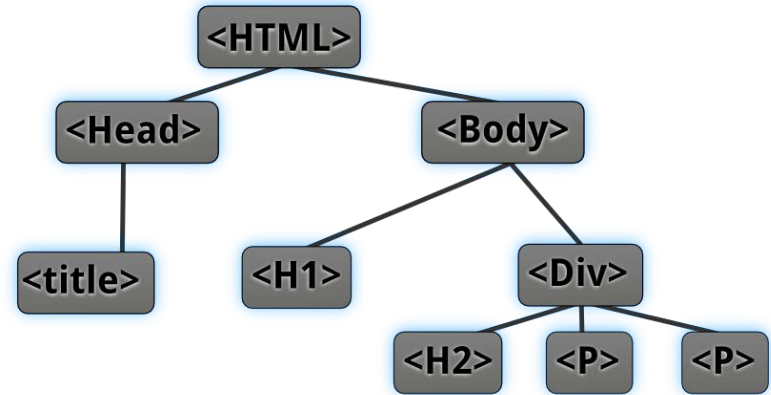
Conceptos claves del DOM

4- Elementos HTML/XML: Cada etiqueta HTML o XML se representa como un nodo de elemento en el DOM. Por ejemplo, un elemento `<div>` en el documento se convierte en un nodo de elemento `<div>` en el árbol del DOM.



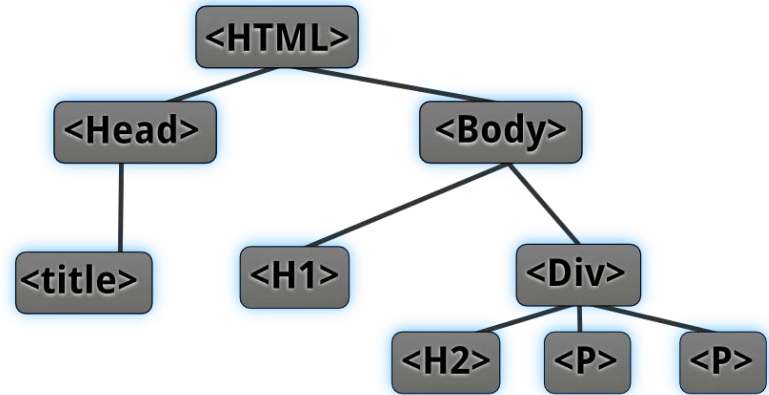
Conceptos claves del DOM

5- Nodos de texto: Los nodos de texto representan el contenido de texto dentro de un elemento. Por ejemplo, el texto "Hola, mundo" dentro de un elemento `<p>` se convierte en un nodo de texto en el DOM.



Conceptos claves del DOM

6- Atributos: Los elementos en el DOM pueden tener atributos, como id, class, src, etc. Los atributos se representan como nodos de atributo en el DOM y se adjuntan a los nodos de elemento correspondientes.



Manipulación del DOM

Manipulación del DOM

La manipulación del DOM (Document Object Model) se refiere a la capacidad de modificar y actualizar los elementos del DOM de un documento web utilizando JavaScript u otras tecnologías de programación.

```
// Manipulación del DOM
// Ejemplo de código JavaScript interactuando con el DOM
const elemento = document.getElementById("miElemento");
elemento.textContent = "¡Hola, mundo!";
elemento.style.color = "red";
```

Manipulación del DOM

La manipulación del DOM (**Document Object Model**) se refiere a la capacidad de modificar y actualizar los elementos del DOM de un documento web utilizando JavaScript u otras tecnologías de programación.

Selección de elementos en el DOM

La selección de elementos es el primer paso en la manipulación del DOM.

Podemos utilizar métodos como **getElementById()** o **querySelector()** para seleccionar elementos específicos.

Por ejemplo, podemos seleccionar un botón con **getElementById("boton")** o un párrafo con **querySelector("p")**.

```
// Selección de elementos
// Ejemplo de código para seleccionar elementos específicos del DOM
const boton = document.getElementById("miBoton");
boton.addEventListener("click", function() {
  // Acción a realizar al hacer clic en el botón
});
```

Modificación de atributos y contenido en DOM

Una vez que hemos seleccionado un elemento del DOM, podemos modificar sus atributos y contenido. Utilizando el método `setAttribute()`, podemos cambiar los atributos de un elemento.

Por ejemplo, podemos cambiar el atributo `src` de una imagen con `elemento.setAttribute("src", "imagen.jpg")`.

Además, podemos modificar el contenido de un elemento utilizando las propiedades `innerHTML` o `textContent`.

```
// Modificación de atributos y contenido
// Ejemplo de código para cambiar atributos y contenido de un elemento
const imagen = document.querySelector("img");
imagen.setAttribute("src", "nueva-imagen.jpg");

const parrafo = document.querySelector("p");
parrafo.textContent = "Texto modificado";
```

Creación y eliminación de elementos DOM

La manipulación del DOM también nos permite crear y eliminar elementos. Utilizando el método `createElement()`, podemos crear un nuevo elemento y configurar sus atributos y contenido.

Luego, podemos agregar el nuevo elemento al DOM utilizando métodos como `appendChild()` o `insertBefore()`.

Para eliminar elementos, se utiliza el método `removeChild()`.

```
// Creación y eliminación de elementos
// Ejemplo de código para crear y agregar un nuevo elemento al DOM
const nuevoElemento = document.createElement("div");
nuevoElemento.textContent = "Nuevo elemento";
document.body.appendChild(nuevoElemento);

// Ejemplo de código para eliminar un elemento existente
const elementoEliminar = document.getElementById("elementoEliminar");
elementoEliminar.parentNode.removeChild(elementoEliminar);
```

Cambios de estilos y manipulación de eventos DOM

Otra capacidad importante de la manipulación del DOM es cambiar los estilos de los elementos y manipular eventos.

Utilizando la propiedad `style`, podemos modificar las propiedades CSS de un elemento.

Por ejemplo, podemos cambiar el color de fondo con `elemento.style.backgroundColor = "red"` o el tamaño de fuente con `elemento.style.fontSize = "16px"`.

También podemos agregar y eliminar event listeners utilizando los métodos `addEventListener()` y `removeEventListener()`.

```
// Cambio de estilos y manipulación de eventos
// Ejemplo de código para cambiar estilos de un elemento
const elemento = document.getElementById("miElemento");
elemento.style.backgroundColor = "blue";
elemento.style.fontSize = "20px";

// Ejemplo de código para agregar un event listener a un elemento
const boton = document.getElementById("miBoton");
boton.addEventListener("click", function() {
  // Acción a realizar al hacer clic en el botón
});
```

La importancia del defer

La importancia del defer

Defer es un modificador que se puede utilizar en la etiqueta de **script** `<script>` en HTML. Se utiliza para indicar al navegador que el archivo de script externo se debe descargar en segundo plano mientras se sigue procesando el documento HTML, y que la ejecución del script debe retrasarse hasta que se haya cargado por completo el documento.



La importancia del defer

Cuando se utiliza el atributo **defer**, el script se ejecutará después de que se haya analizado y construido el árbol del DOM, pero antes de que se dispare el evento DOMContentLoaded.

Esto significa que el script no bloqueará el análisis y la representación del documento, lo que puede resultar en una carga más rápida de la página.



```
<script src="archivo.js" defer></script>
```

Consideraciones importantes defer

El atributo **defer** solo se aplica a scripts externos que se cargan mediante la etiqueta `<script src="archivo.js"></script>`.

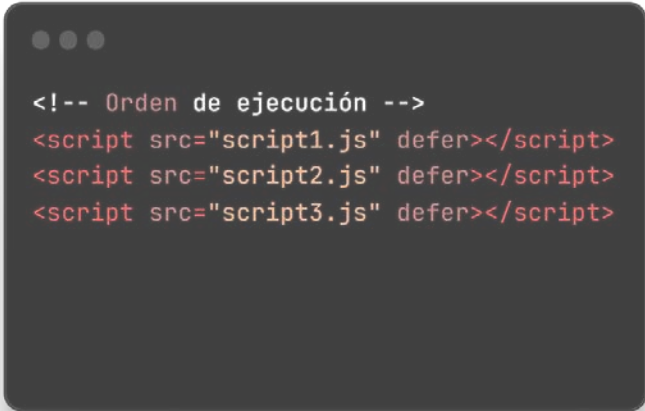
No se puede utilizar en scripts incrustados directamente en la página con contenido JavaScript.



Consideraciones importantes defer

Si hay varios scripts con el atributo defer, se mantendrá el orden de aparición de los scripts en el documento.

Es decir, los scripts se ejecutarán en el orden en que aparecen en el código HTML.

A dark-themed code editor window with three window control buttons (red, yellow, green) in the top-left corner. It contains HTML code for three deferred scripts.

```
<!-- Orden de ejecución -->  
<script src="script1.js" defer></script>  
<script src="script2.js" defer></script>  
<script src="script3.js" defer></script>
```

Consideraciones importantes defer

El atributo **defer** no es compatible con el atributo **async**.

Si ambos atributos están presentes en la etiqueta de script, se dará prioridad al atributo **async**, y el script se ejecutará en el orden en que se descargue, sin esperar a que el documento esté completamente analizado.



Consideraciones importantes defer

El atributo **defer** es compatible con la mayoría de los navegadores modernos, pero es posible que no funcione correctamente en versiones antiguas de algunos navegadores.

```
<!-- Compatibilidad y consideraciones -->
<script>
  if ('defer' in document.createElement('script')) {
    console.log('El atributo defer es compatible');
  } else {
    console.log('El atributo defer no es compatible');
  }
</script>
```

Nodo elementó

Representa un elemento **HTML** o **XML** en el árbol del DOM. Por ejemplo, un **<div>**, **<p>**, ****, ****, **etc.** Los nodos de elemento contienen otros nodos, como nodos de texto, atributos, nodos hijos (otros elementos o nodos de texto), y más.

Puedes acceder a los nodos de elemento utilizando métodos como **getElementById()**, **getElementsByTagName()**, **querySelector()**, **etc.**

Nodo de texto

Representa el **contenido de texto** dentro de un elemento en el árbol del **DOM**. Por ejemplo, el texto "Hola" en `<div>Hola</div>`.

Un nodo de texto no tiene hijos y su contenido se almacena en la propiedad **nodeValue**. Puedes acceder a los nodos de texto a través de los nodos de elemento utilizando propiedades como **firstChild**, **nextSibling**, etc.

Nodo elementó y texto en acción

En este ejemplo, se selecciona un elemento `<div>` con el atributo `id="miDiv"`. Luego, se obtiene el primer hijo de ese elemento, que es un nodo de texto que contiene el texto "Hola ". Finalmente, se obtiene el contenido del nodo de texto utilizando la propiedad `nodeValue`.

Recuerda que el código JavaScript debe estar dentro de un evento `DOMContentLoaded` para asegurarse de que se ejecute después de que el documento HTML se haya cargado por completo.

```
<!-- HTML -->
<div id="miDiv">Hola <span>mundo</span></div>
```

```
// JavaScript
window.addEventListener("DOMContentLoaded", function() {
  // Obtener el nodo de elemento
  const miDiv = document.getElementById("miDiv");
  console.log(miDiv); // Devuelve el nodo de elemento <div id="miDiv">

  // Obtener el nodo de texto
  const nodoTexto = miDiv.firstChild;
  console.log(nodoTexto); // Devuelve el nodo de texto "Hola "

  // Obtener el contenido del nodo de texto
  const contenidoTexto = nodoTexto.nodeValue;
  console.log(contenidoTexto); // Devuelve "Hola "
});
```

El objeto document

El objeto document en JavaScript

- El objeto document es un objeto fundamental en JavaScript que representa el documento HTML cargado en el navegador.
- Proporciona una interfaz para interactuar con los elementos y contenido del documento, y realizar modificaciones dinámicas.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Title goes here</title>
  </head>
  <body>

  </body>
</html>
```


Accediendo a elementos y manipulando el contenido

El objeto document ofrece métodos y propiedades para acceder a los elementos del documento mediante identificadores únicos (como **getElementById()**, **getElementsByClassName()**, etc.).

Permite manipular el contenido de los elementos, como modificar el texto, cambiar atributos, agregar o eliminar elementos, etc.

También se puede utilizar para controlar eventos, como escuchar clics de botones o enviar formularios.

The image shows a registration form titled "Registration Form" with a close button (X) in the top right corner. The form contains the following fields and elements:

- Name:** A text input field.
- Email address:** A text input field.
- Country:** A text input field.
- Phone:** A text input field.
- Password:** A text input field with a blue eye icon for toggling visibility.
- Terms and Conditions:** A checkbox followed by the text: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat".
- CREATE ACCOUNT:** A large blue button.
- Link:** Below the button, the text "Already have an account? Sing in" is displayed.

El metodo getElementById()

getElementById()

El método `getElementById()` es una función esencial del objeto `document` en JavaScript.

Permite acceder a un elemento del documento HTML utilizando su identificador único (ID).

El ID se asigna a un elemento HTML mediante el atributo `id`.

```
// JavaScript
const miDivElement = document.getElementById("miDiv");
console.log(miDivElement);
```

getElementById()

Para utilizar **getElementById()**, se llama al método en el objeto `document` y se pasa el ID del elemento como argumento.

El método devuelve una referencia al elemento correspondiente o `null` si no se encuentra ningún elemento con el ID especificado.

```
// JavaScript
const miElemento = document.getElementById("miElemento");
if (miElemento) {
    miElemento.textContent = "Nuevo contenido";
}
```

getElementById()

Una vez obtenida la referencia a un elemento utilizando **getElementById()**, se puede manipular su contenido, atributos, estilos y más.

Por ejemplo, se puede cambiar el texto dentro del elemento, modificar atributos como src, href o class, o aplicar estilos mediante la manipulación de propiedades CSS.

```
// JavaScript
const miBoton = document.getElementById("miBoton");
miBoton.addEventListener("click", () => {
  miBoton.style.backgroundColor = "red";
});
```

getElementById()

getElementById() es una herramienta poderosa para interactuar con elementos específicos de un documento HTML.

Es importante asegurarse de asignar identificadores únicos a los elementos para evitar conflictos y errores.

Recuerda que los elementos deben existir en el documento antes de llamar a **getElementById()** para evitar obtener null como resultado.

```
// JavaScript
const miImagen = document.getElementById("miImagen");
miImagen.setAttribute("src", "nueva-imagen.png");
```

El método innerText

InnerText

El método **innerText** es una propiedad del objeto **HTMLElement** en **JavaScript**.

Permite acceder y manipular el contenido de texto visible de un elemento HTML.

```
<div id="myDiv">HOLA UNTREF </div>
<script type="text/javascript">
  document.getElementById("myDiv").innerText="hola de nuevo";
</script>
```


Uso básico

Para utilizar **innerText**, se accede al elemento HTML deseado utilizando métodos como **getElementById()** o **querySelector()**.

Una vez obtenida la referencia al elemento, se puede acceder y modificar su contenido de texto utilizando la propiedad **innerText**.

```
const miElemento = document.getElementById("miElemento");  
console.log(miElemento.innerText);
```

Modificación del contenido

La propiedad `innerText` permite cambiar el contenido de texto de un elemento.

Se puede asignar un nuevo valor a `innerText` para reemplazar el texto existente en el elemento.

```
const miElemento = document.getElementById("miElemento");  
miElemento.innerText = "Nuevo texto";
```

Importancia y consideraciones

innerText es una forma sencilla de acceder y modificar el contenido de texto visible de un elemento HTML.

A diferencia de **textContent**, **innerText** no muestra el texto oculto por CSS o elementos hijos ocultos, lo que puede tener implicaciones en la manipulación del contenido.

```
const miElemento = document.getElementById("miElemento");  
miElemento.innerText = "Texto visible"; // No mostrará texto oculto por CSS o  
elementos hijos ocultos
```

El método innerHTML

El metodo innerHTML

El método **innerHTML** es una propiedad del objeto **HTMLElement** en JavaScript.

Permite acceder y manipular el contenido HTML dentro de un elemento.

```
const miElemento = document.getElementById("miElemento");  
console.log(miElemento.innerHTML);
```

Uso básico

Para utilizar **innerHTML**, se accede al elemento HTML deseado utilizando métodos como **getElementById()** o **querySelector()**.

Una vez obtenida la referencia al elemento, se puede acceder y modificar su contenido HTML utilizando la propiedad **innerHTML**.

```
const miElemento = document.getElementById("miElemento");  
console.log(miElemento.innerHTML);
```

Modificación del contenido

La propiedad **innerHTML** permite cambiar el contenido HTML de un elemento.

Se puede asignar un nuevo valor a **innerHTML** para reemplazar el contenido existente en el elemento.

```
const miElemento = document.getElementById("miElemento");  
miElemento.innerHTML = "<p>Nuevo contenido HTML</p>";
```

Importancia y consideraciones

innerHTML es una forma poderosa de acceder y manipular el contenido HTML dentro de un elemento.

Al utilizar **innerHTML**, es importante tener en cuenta la seguridad y evitar la inyección de código malicioso o no confiable en el contenido.

```
const miElemento = document.getElementById("miElemento");  
miElemento.innerHTML = '<script>alert(";Hola, soy un script malicioso!");</script>';
```


El objeto literal

El objeto literal en JavaScript

En JavaScript, un **objeto literal** es una colección de pares **clave-valor**.

Se utiliza para representar y organizar datos relacionados en una estructura fácil de leer y acceder.

Creación de un objeto literal

Un **objeto literal** se define utilizando llaves {} y separando cada par clave-valor por dos puntos :

Las claves representan los nombres de las propiedades y los valores son los datos asociados.

```
const persona = {  
  nombre: "Juan",  
  edad: 25,  
  profesion: "Desarrollador",  
};
```

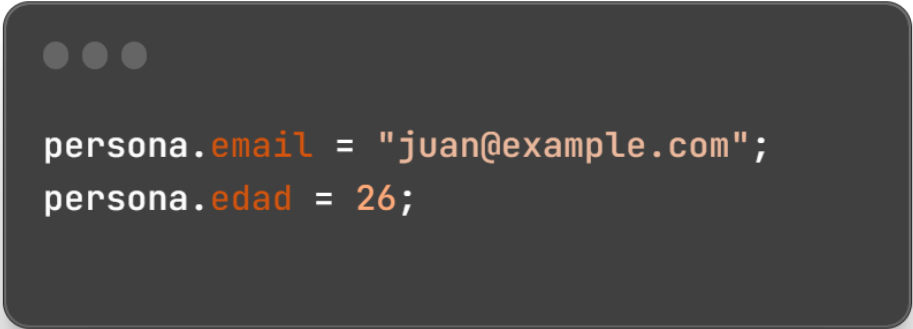
Acceso a las propiedades

Las propiedades de un objeto literal se pueden acceder utilizando la sintaxis de punto (`objeto.propiedad`) o la sintaxis de corchetes (`objeto["propiedad"]`).

```
console.log(persona.nombre); // Juan  
console.log(persona["edad"]); // 25
```

Agregar y modificar propiedades

Es posible agregar nuevas propiedades a un objeto literal o modificar las existentes.

A dark-themed code editor window with three small gray circles in the top-left corner. It contains two lines of JavaScript code. The first line is `persona.email = "juan@example.com";` and the second line is `persona.edad = 26;`. The text is in a monospaced font, with `email` and `edad` highlighted in orange.

```
persona.email = "juan@example.com";  
persona.edad = 26;
```

Uso de objetos literales en situaciones complejas

Los objetos literales son versátiles y se pueden utilizar para representar estructuras de datos más complejas, como objetos anidados o matrices de objetos.

```
const libro = {  
  titulo: "El Gran Gatsby",  
  autor: {  
    nombre: "F. Scott",  
    apellido: "Fitzgerald",  
  },  
  capitulos: [  
    { numero: 1, titulo: "Prólogo" },  
    { numero: 2, titulo: "La fiesta" },  
    // ...  
  ],  
};
```

El array de objeto literal

El array de objetos literales en JavaScript

Un array de objetos literales es una estructura que permite almacenar y acceder a múltiples objetos literales en secuencia.

Es útil cuando necesitas trabajar con conjuntos de datos relacionados.

```
const frutas = [  
  { nombre: "Manzana", color: "Rojo" },  
  { nombre: "Plátano", color: "Amarillo" },  
  { nombre: "Naranja", color: "Naranja" },  
];
```


El array de objetos literales en JavaScript

Un array de objetos literales se define utilizando corchetes [] y separando cada objeto por comas.

Cada objeto dentro del array es un objeto literal que contiene par clave-valor.

```
const personas = [  
  { nombre: "Juan", edad: 25 },  
  { nombre: "María", edad: 30 },  
  { nombre: "Pedro", edad: 28 },  
];
```

Operaciones con arrays de objetos

- Agregar nuevos objetos:
 - Utilizar `push()` para agregar un objeto al final del array.
 - Ejemplo: `personas.push({ nombre: "Laura", edad: 28, ciudad: "Valencia" })`.
- Longitud del array:
 - La propiedad `length` indica la cantidad de objetos en el array.
 - Ejemplo: `personas.length` devuelve 4 después de agregar a Laura.
- Utilidad:
 - Los arrays de objetos literales son comunes en aplicaciones para manejar datos relacionados.
 - Útil para almacenar colecciones de información con diferentes propiedades.

```
const personas = [  
  { nombre: "Juan", edad: 25 },  
  { nombre: "María", edad: 30 },  
  { nombre: "Pedro", edad: 28 },  
];
```

¿Dudas o consultas?





¡Muchas Gracias!