¿QUE ES EL DOM?

El DOM (Document Object Model) es una representación en memoria del contenido de un documento HTML. Básicamente, es un árbol de nodos que representa la estructura del documento HTML, donde cada nodo corresponde a un elemento, atributo o texto del HTML.

Cuando un navegador recibe un archivo HTML, primero lo analiza y construye el DOM correspondiente. Luego, aplica las reglas de estilo CSS al DOM para determinar la apariencia de cada elemento en la página. Finalmente, ejecuta cualquier código JavaScript incluido en el documento, lo que puede alterar dinámicamente el contenido o el estilo del DOM.

Una vez que el DOM está completamente construido y renderizado, el navegador lo muestra en la pantalla, creando una representación visual de la página HTML. Cada vez que se actualiza el DOM, ya sea a través de cambios en el HTML, CSS o JavaScript, el navegador vuelve a renderizar la página para reflejar esos cambios.

La ubicación recomendada para incluir los scripts de JavaScript en una página web es justo antes de cerrar la etiqueta **body**.

Hay varias razones por las que esto es recomendado:

1. Para mejorar el rendimiento: El navegador comienza a cargar y renderizar la página HTML tan pronto como recibe los primeros bytes. Si se incluyen los scripts de JavaScript en la parte superior del documento, el navegador debe pausar la carga de la página hasta que se descargue y ejecute el script. Al colocar los scripts justo antes de cerrar la etiqueta **body**, el navegador ya habrá renderizado la mayor parte de la página y el script se cargará y ejecutará en segundo plano, lo que mejora el tiempo de carga de la página.
2. Para evitar problemas de ejecución: Al colocar los scripts de JavaScript al final del documento, se asegura que todo el contenido de la página se haya cargado completamente antes de que se ejecute cualquier script. Esto es importante porque algunos scripts pueden intentar manipular elementos HTML que aún no se han creado, lo que puede causar errores.
3. Para facilitar el mantenimiento del código: Al colocar todos los scripts de JavaScript en un solo lugar (es decir, justo antes de cerrar la etiqueta **body**), es más fácil encontrar y actualizar el código. Si los scripts están dispersos por todo el documento HTML, puede ser difícil rastrear y hacer cambios en el código.

JavaScript

JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, que se utiliza principalmente para crear páginas web interactivas. Es compatible con la mayoría de los navegadores web modernos y se ejecuta del lado del cliente (es decir, en el navegador del usuario) para agregar interactividad y dinamismo a las páginas web. Con JavaScript, se pueden manipular elementos HTML, crear animaciones, validar formularios, enviar y recibir datos del servidor y mucho más. En resumen, JavaScript es una herramienta poderosa para crear experiencias web dinámicas e interactivas.

Diferencia entre class y interfaz

Una clase es una estructura que define las propiedades y los comportamientos de un objeto. Es un plano o una plantilla para crear objetos específicos. Una clase puede contener variables (llamadas atributos) y funciones (llamadas métodos) que definen el comportamiento de los objetos creados a partir de esa clase. Puedes crear múltiples instancias (objetos) de una clase y cada objeto tendrá sus propios valores para las variables y métodos definidos en la clase.

Por otro lado, una interfaz es una colección de métodos abstractos (métodos sin implementación) que definen un contrato que una clase debe seguir. En otras palabras, una interfaz establece un conjunto de métodos que una clase concreta debe proporcionar. Las interfaces solo pueden contener métodos y no pueden tener atributos. Una clase puede implementar múltiples interfaces, lo que significa que la clase debe proporcionar la implementación (el código real) para todos los métodos definidos en esas interfaces.

La principal diferencia entre una clase y una interfaz radica en su propósito y uso. Una clase se utiliza para definir una estructura de objetos con atributos y comportamientos, mientras que una interfaz se utiliza para establecer un contrato que las clases deben cumplir implementando los métodos definidos en la interfaz.

Además, las clases pueden heredar de otras clases utilizando la herencia, lo que significa que pueden obtener atributos y comportamientos de una clase base. Las interfaces, por otro lado, no pueden heredar directamente de otras interfaces, pero una interfaz puede extender (heredar) otra interfaz, lo que significa que puede agregar más métodos al contrato establecido por la interfaz base.

En resumen, las clases definen la estructura y el comportamiento de los objetos, mientras que las interfaces establecen un contrato que las clases deben cumplir implementando los métodos definidos en la interfaz. Las clases pueden tener atributos y métodos con implementaciones concretas, mientras que las interfaces solo pueden tener métodos abstractos sin implementación.