**21/9/2023-21/10/2023 Token github: ghp\_djN8vpWB3unfKEanyXcHlKCr6bqze03hEspp**

-Curso-JavaScript-DOM- 10/08/2023

Que es el DOM? Ese es el modelo que vamos a usar durante el curso y está estrechamente relacionado con el desarrollo web, nos va a permitir crear páginas interactivas que necesitamos. Cuando se crea una página web normalmente se crea un archivo HTML para definir la estructura de la página. Esa estructura en el navegador está representada también por una jerarquía. Un elemento está dentro de otro elemento y esos elementos se van a poder representar como ciertos obj en nuestro programa. Toda esa jerarquía de distintos elementos que se van agrupar y estructurar para poder mostrarlos en el navegador es lo que conocemos como DOM pero se va creando en base a lo que definimos en el archivo HTML. El navegador crea una representación de c/u de esos elementos y su ubicación en una jerarquía especifica. El significado de DOM→Modelo de Objetos del Documento. Formal mente podemos definir el DOM como la representación de los obj, los distintos elementos que conforman la estructura de un documento en la web.

A partir de estos elementos podemos generar un diagrama similar a este que representa su jerarquía HTMT



En el grafico vemos al inicio la etiqueta <html> que es el elemento raíz a partir de ahí continúan todos los otros elementos que van hacer descendientes del elemento principal. Luego tenemos <head> y <body> que están en la misma línea, esto nos dice que son hermanos porque están ubicados en el mismo nivel en la jerarquía. Seguidamente dentro de <head> encontramos el elemento <title> y por debajo tenemos otro nivel que representa el texto en sí. El DOM llega hasta ese ínfimo detalle de representar el texto de forma individual. Con <body> ocurre lo mismo, pero en este caso <body> contiene dos elementos. Un elemento <h1> y un elemento <p> están al mismo nivel en la jerarquía pero están contenidos en el elemento <body> y así se puede ir leyendo este diagrama hasta que llegamos al texto que corresponde a cada uno si es que contienen textos. Por ejemplo si tenemos un elemento de imagen no tendríamos texto, esto es básicamente la idea del diagrama. ¿Porque es importante representar el documento a través del DOM? Representar el documento HTML a través del DOM nos permite *acceder* a sus elementos en nuestro programa usando JS y también *manipularlos* que significa crear elementos nuevos, eliminar elementos que ya están en la página, modificar sus atributos, agregar atributos nuevos, cambiar el texto. Todas esas operaciones mencionadas son ejemplos de cómo podemos manipular elementos una vez que ya están representadas en el DOM en el navegador. Es uno de los aspectos más importante que se tiene que saber. El DOM nos permite representar los elementos como *NODOS* y *OBJETOS* con los cuales podemos trabajar *JS* con sus propiedades, métodos y con distintos elementos. Por ej. cuando se hace click sobre un botón todos esos eventos vamos a poder manejarlos y detectarlos en nuestro código gracias al DOM porque en si nuestro archivo HTML es solo eso un archivo pero el DOM es el que le da esa capacidad poderosa para poder trabajar con esos elementos en JS. hay que recordar estos dos conceptos *NODO y OBJETOS* son muy importantes para el DOM los Nodos son muy utilizados y prácticamente fundamentales en el área de estructuras de datos. También en el DOM vamos a tener Nodos pero esos Nodos en lugar de representar información como tal van a representar elementos con todos sus datos y toda esa información y los objetos van a estar presente cuando trabajemos con esos elementos en JS, van a tener propiedades y métodos.

Un Nodo es un punto específico del diagrama o del árbol de Nodos del DOM por ejemplo en el siguiente grafico tenemos la estructura del archivo HTML y el mismo diagrama anterior pero ves que todos tienen un borde amarillo es para indicarles que todos estos elementos son Nodos incluyendo el texto. Individualmente el texto en el DOM se representa como un Nodo y eso va ser importante cuando trabajemos con algunos métodos que nos puede permitir acceder a un elemento o a un Nodo como tal. Los Nodos en el DOM incluyen: El documento en sí, los elementos HTML que definamos en nuestro archivo, también cualquier texto que incluyas entre etiquetas se representa como un Nodo y también algo importante es que los comentarios que escribas en el archivo html también se representan como Nodos. Todo el proceso de transformar un archivo html a un DOM interactivo que podemos usar en nuestro archivo JS lo realiza el navegador o browser en inglés.



**Conceptos Importantes:**

Al referirte al DOM y a los elementos representados en el DOM dentro de su jerarquía, al igual que un árbol genealógico vamos a tener esos elementos representados como una jerarquía. Cual de esos elementos contiene a otros y cuales están contenidos dentro de otros elementos todo eso lo vamos a poder representar y cada tipo de Nodo va a tener un concepto diferente para referirnos a esa ubicación.



Por ejemplo como vemos en el diagrama lo primero que conseguimos en nuestra jerarquía en html es el Nodo raíz es el que contiene a todos los otros elementos. Luego comenzamos hablar sobre las distintas relaciones que puede haber. Primero tenemos parent node o (nodo padre) este nodo es el que contiene a otro nodo en la jerarquía del DOM.



En el gráfico de arriba tenemos señalado el elemento *head* ese nodo es el nodo padre de *title* porque esta contenido por *head* lo mismo ocurre con *body* es el nodo padre de *h1* y de p. Eso también nos lleva a que tenemos otra relación reciproca si un Nodo es el nodo padre el otro nodo contenido por este es el nodo hijo. Digamos entonces que el nodo hijo es el que esté conectado directamente un nivel por debajo el nodo padre. Y en este ejemplo podemos ver que en la misma jerarquía esta h1 y p que son contenidos por su nodo padre body



Ahora que ocurre si la jerarquía es más elaborada como indica el siguiente gráfico. Digamos que como muestra el grafico anterior body contiene a h1 y p y en lugar de eso como vemos en el siguiente grafico en lugar de h1 tenemos un div y dentro de ese div tenemos un h1 y un a, en ese caso ya sería relevante el concepto de Nodo descendiente. Al igual que una familia tendrías a tu tátara abuelo tendría muchos descendientes en la jerarquía que no son hijos directos pero si son sus descendientes porque son sus bisnietos y nietos. Ese mismo concepto del árbol genealógico también aplica aquí al DOM. En resumen: html es el nodo padre de toda la estructura en la cual body es hijo de html a su vez body es padre de div y p y div es padre de h1 y a



Además otro concepto importante que ya hemos mencionado pero aquí tenemos el concepto formal. Esos nodos que están al mismo nivel como podemos observar en el siguiente diagrama se denominan Nodos hermanos y comparten el mismo Nodo padre.



**NODOS Vs. ELEMENTOS**

El concepto de Nodo es más amplio que el concepto de Elementos en el DOM, es importante saber cuál es la diferencia entre ambos. Estos conceptos son fáciles de intercambiar y es fácil confundirse entre un Nodo y un Elemento cuando ya estamos trabajando en el contexto del DOM. Un Nodo puede contener más tipo de información que los Elementos. Usualmente nos referimos a los Elementos del documento HTML como “elementos” es decir toda esa estructura que escribimos en el archivo html una vez que ya se traduce al DOM nos vamos a referir a ellos como elementos. Pero un Nodo puede ser un documento HTML puede representar o ser texto o comentarios que hayas escrito en el documento. Asique el Nodo como tal no solamente representa esas etiquetas html o los elementos como tal, también incluye el texto que es un Nodo aparte para cada uno de esos elementos que contengan texto o también pueden ser comentarios que hayas escrito en el documento html que aunque no tengan una función como tal en la estructura de la página web si son tomados en cuenta al momento de crear el DOM y crean nuevos nodos en el DOM por eso es la diferencia. El Nodo es un concepto más amplio que el Elemento

**Herramientas de Desarrollo de Chrome**

Veamos cómo usar las herramientas de google Chrome con la cual vamos a trabajar en el curso, en estas herramientas vamos a poder visualizar el DOM la estructura en el archivo html con todos su nodos. Primero que nada en VSC instalamos la extensión live server y puedes ver que se nos abre el archivo HTML en nuestro browser, en la barra superior aparece un elemento, un número (<http://127.0.0.1:5500/index.html>) Estos números 127.0.0.1 representan una dirección IP local a continuación de los dos puntos 5500 representa el puerto que es como un código para identificar un proceso específico dentro del servidor local, es como una ubicación y es único para ese proceso y luego vemos donde estamos ubicados→ /index.html ¡si en el proyecto! En el archivo que está abierto en VSC que también nos avisa que el servidor se abrió en el puerto 5500. Y si necesitas volver a entrar a esa página puedes escribir la URL que se describe más arriba en azul. Esa es la parte fundamental de live server y luego si quieres detener el servidor haces click en la barra inferior de VSC en port 5500 y eso lo detiene. Ahora vamos a ver cómo acceder al DOM con las herramientas de desarrollo de Chrome. Hacemos click derecho en cualquier lugar de la página web se abre un menú y hacemos click en inspeccionar en la parte derecha por defecto se abre una ventana de herramientas. En el menú aparece por defecto una pestaña ELEMENTOS esa pestaña es la que se necesita para poder visualizar el DOM allí vamos a ver toda la estructura del archivo html como el DOM y su jerarquía o sea que ya estamos dentro de las herramientas de desarrollo de Chrome. Esta es la representación en el browser del DOM que es lo que nos interesa. Se puede ver toda la estructura html pero con algo interactivo agregado. Si posamos el cursor en html veremos que se resalta todo el documento. Ese es el NODO padre, dentro de html vemos head pero este no es visible como tal en la estructura asique si lo seleccionamos no lo podemos encontrar pero está allí y existe y luego aquí vemos en body lo que si podemos observar en la página y a mediada que nos internamos podemos ver cómo se va construyendo esa estructura del DOM y también dentro de estos elementos tenemos textos. Además de esto como estamos usando la extensión live server para actualizar nuestros cambios en vivo vemos los elementos de comentarios que ha sido agregado por esa extensión esto nos dice que si son parte del DOM si se reflejan en las herramientas de Chrome.

**CAMBIAR EL DOM:**

En las herramientas de Chrome podemos hacer cambios, si haces doble click en el texto de algún a los elemento puedes modificarlo y esto en realidad no es html es una versión interactiva del DOM puedes confirmarlo si haces click derecho se despliega un menú de opciones que nos permite editar el elemento como si fuera html. Veamos la de diferencia de editar texto o como html se recuerda que el texto se representa de forma individual como un Nodo separado del elemento como tal. Si editamos el texto automáticamente nos selecciona el texto y podemos modificarlo pero si seleccionamos editar como html nos selecciona todo el elemento, todo el nodo del Dom incluyendo el texto que contiene. <h1>texto</h1> y te permite modificar no solo el texto si no también la etiqueta en vez de h1 puedes llevarlo al valor que desees h2, h3, etc. Todo eso va a ser temporal, los cambios que hagas en las herramientas de Chrome son temporales hasta que vuelvas a cargar la página. Es muy útil para tener una vista previa de modificaciones que planeamos hacer en la página web. También podemos seleccionar o inspeccionar algún elemento en especial.

En el vértice superior izquierdo veremos una flechita tipo puntero que si hacemos click al seleccionarla se pondrá de color azul. El cursor se va a tornar como un cursor especial que nos va a permitir seleccionar un elemento en la página para inspeccionarlos y se podrá apreciar como el elemento o el nodo correspondiente en el DOM se va resaltando, eso es muy útil cuando ya tienes una página más elaborada y quieres inspeccionar algún elemento especifico. También tenemos otras herramientas interactivas como por ejemplo una consola donde se puede ejecutar código de JS para ver distintos resultados o para interactuar con los distintos elementos.

**Nodos que no son elementos:**

Veamos un poco más sobre los elementos que son Nodos pero no son como tal elemento html, mencionamos que el texto era uno de ellos y que los comentarios era uno de ellos. Si agregamos un comentario dentro del body

</--Comentarios--> Como tenemos la extensión live server activada se va actualizar automáticamente y en las herramientas de Chrome lo podremos ver el comentario como parte del DOM, no es un elemento html como el texto de h1 pero si es parte del DOM y se mantiene allí como un nodo, eso va a ser importante al momento de seleccionar los métodos que vamos a usar. Porque hay algunos que seleccionan todos los Nodos incluyendo comentarios y textos y hay otros que seleccionan comentarios html y es importante que se comprenda esa diferencia y también como estamos trabajando con live server puedes hacer cualquier cambio, lo guardas con ctrl + s y veras ese cambio reflejado en vivo en el DOM, también en las herramientas de Chrome tenemos la pestaña de styles en este caso aún no hemos agregado styles pero más adelante lo haremos.

**Proyecto: Toppings de Pizza**

**HTML**

**CSS**

/\* Estilos generales Css\*/

/\* Con el selector universal vamos asignar por defecto un margin: 0; y padding: 0; esto se recomienda y lo vas a ver comúnmente para que ningún elemento tenga un margin o padding asignado por defecto en el Browser pero cuando iniciamos esto de la forma mencionada estamos reseteando todo eso. También vas a encontrar esta propiedad box-sizing: border-box; esto le dice al Browser como va a calcular el tamaño en base a su margin y a su padding, border-box es una de las propiedades específicas de los valores específicos. Y luego el tipo de fuente font-family: "lato", sans-serif;

\*/

/\*

Para el body vamos a usar (flexbox) **display: flex;** **justify-content: center;** y otras propiedades, vemos aquí **align-items: center;** vemos la altura→**height: 100vh;** va a ser 100% la altura de la ventana va a tener toda la dimensión así que vamos a poder centrar lo relativo al tamaño completo total de la ventana donde se está viendo el contenido **text-align: center;** alineamos en el centro y el fondo va a ser una imagen **background: url(imagenes/pizza.jpg)** Tambien además de la img vamos a especificar algunos valores para ese fondo. No vamos a repetir el fondo→**no-repeat** lo vamos a centrar→**center center/cover;** y de esta forma vamos a lograr que se vea de una forma peculiar el fondo, se va a poder ver centrado independientemente del dispositivo puedes ver en el proyecto como se reajusta el fondo La imagen seleccionada va a estar ubicada en la carpeta de imágenes de nuestro proyecto

\*/

/\*CONTENIDO: Vamos asignar estos estilos, primero tenemos el **id** que funciona como un contenedor entonces vamos asignarle un estilo a ese contenedor con un **id**→**#contenedor** ese contenedor va a ser flex

**flex-wrap: wrap;** para que todo se reajuste de acuerdo a cuanto espacio está disponible. **justify-content: center;** **align-items: center;** vamos alinearlos en el centro vertical y horizontalmente. **min-height: 80vb;** y la altura mínima de ese contenedor va a ser 80% de la altura de la ventana o donde se está viendo el contenido también el ancho de la ventana **width: 80vh;**  va a ser el 80% pero va a tener un ancho **max-width: 600px;** de 600px asi si en la pantalla o el monitor donde se está viendo es muy ancho el máximo se lo indicamos en 600px. El color de fonfo **background-color: white;** va a ser blanco y el borde **border: 4px solid black;** va a ser de 4px sólido y negro. Se puede ver en la aplicación un ancho máximo pero en dispositivos móviles se reajusta bien eso es para el contenedor principal.

\*/

/\*

Ahora para el título principal h1 el tamaño de la letra **font-size: 3rem;** va a ser 3 rem para que sea grande va a tomar el 100% del ancho de su contenedor y va a tener un margen  **margin: 20px 10px;** de 20px arriba y abajo y 10px a los lados

\*/

/\*

Ahora vamos a señalar un estilo a la clase toppings, en html se asigna a cada uno de los elementos de la lista. El tipo de letra **font-size: 2rem;** va a ser 2rem, el peso de la letra **font-weight: bold;** va a ser negrita color **color: white;** blanco luego va a ser **display: flex;** **flex-wrap: wrap;** para que nos permita que los distintos toppings estén en distintas líneas, también vamos a centrar **justify-content: center;** alineado **align-items: center;** vamos a centrar, el ancho→**width: 100%;** cien por ciento, altura→**height: 80px;** y un margen→

margin: 15 0; y cero

\*/

/\*

Clases fondo marrón que va asignar ese color en la ul para los que tienen ese fondo **background-color: #aa5822;**

Clases fondo naranja que va asignar ese color en la ul para los que tienen ese fondo **background-color: #ff7b0f;**

\*/

/\*

El estilo a la lista→ul no ordenada el ancho de esa lista va a ser cien por ciento de su contenedor→width: 100%; y el alto va ser cien por ciento→height: 100%;

\*/

/\*

**Vincular JavaScript a HTML**

Vamos a comenzar a usar JS para acceder a los elementos de este DOM, para eso vamos a tener que conectar el archivo **html** con el de **JavaScript**. Nos dirigimos al final de la etiqueta **body** pero dentro de body vamos a especificar donde se encuentra el **archivo.JS**. Como esta en la misma carpeta y al mismo nivel simplemente escribimos el nombre del archivo.js y ya eso nos va a permitir que el elemento **script** importe es archivo de JS y lo vincule con html de modo tal que lo podremos usar en nuestra página web. Ya tenemos la estructura y el maquetado y podemos ver como se adapta a nuestros dispositivos. Otra herramienta muy útil de Desarrollo de Chrom en el vértice superior izquierdo veremos una flechita tipo puntero, justo al lado hay otro icono que es cómo una tableta y un teléfono si hacemos click ahí al seleccionarla se pondrá de color azul y vamos a entrar en el modo de vista previa responsiva si arrastras con el cursor estos indicadores puedes simular estar en un dispositivo de un ancho diferente y podrás apreciar cómo se ajustan los elementos.

\*/

/\*

**Seleccionar Elementos**

En VSC abrimos nuestro archivo js y solo vamos a trabajar con html y JS al estar vinculados index.html con app.js vamos a poder tener acceso a su DOM en la aplicación. En esta parte veremos cómo podemos seleccionar elementos del DOM para tener accesos a sus propiedades y en general a todos sus métodos, el primer paso es seleccionar ese elemento con JS. Al tener live server activado vemos el proyecto en el Browser. De esta forma vamos a ver el resultado de lo que escribamos en JS en la consola del Browser. La consola es una herramienta que nos permite ejecutar código de JS pero también nos va a permitir ver el resultado de nuestro código JS si usamos console.log, cuando abrimos la consola para que sea más fácil poder verla mientras tengo todas las herramientas a la misma vez hacemos click en los tres puntos y luego seleccionamos el primer icono de la izquierda Dock side significa que vamos a separar esas herramientas de chrom para tener todo abierto al mismo tiempo, luego vamos a console y se abre la herramienta la cual ubicamos convenientemente. Sorprendentemente puedes seleccionar un elemento del DOM simplemente usando un método espesifico→**.getElementByld()** en español significa obtener elemento por el **id.** Si nosotros tenemos en nuestro archivo html un elemento div que contiene un id espesifico→contenedor según las reglas de html debería ser único por página web por archivo html asique vamos a poder seleccionar cada uno de sus elementos a partir de su **id.** Es muy similar a definir una constante, si queremos obtener ese elemento del contenedor definimos una constante porque una vez que lo obtenemos no lo vamos a cambiar. Le Definimos un nombre y luego usamos un objeto especial llamado **document** ese obj especial que nos da acceso el navegador nos va a permitir llamar al método .**getElementByld()** y luego dentro de los paréntesis tenemos que especificar entre comillas el id que queremos seleccionar→contenedor en este caso. Así que lo vamos asignar a esta constante que definimos y vamos a usar console.log (); para poder ver el valor de la constante contenedor. Como estamos mostrando el valor de contenedor, el obj retornado por el método.getElementByld(), si vamos a las herramientas de chrome podemos ver en la consola que tenemos un resultado <div contenedor> ese es el elemento html que pedimos con ese método es un div que tiene el id contenedor y puedes ver que tiene muchas propiedades diferentes entre esas propiedades podemos ver un elemento familiar **childElementCount: 2**→el número de elementos hijos, en este caso tiene dos directos. Como estamos obteniendo ese contenedor vamos a poder acceder a sus propiedades y métodos. Una de sus propiedades es→**innerHTML** esa propiedadnos da acceso a la estructura HTML que está contenida dentro de ese elemento. Veamos ese valor con console.log(contenedor.innerHTML); es mostrado como una cadena de caracteres que está contenido dentro del **div** desde el título hasta la lista no ordenada. En este caso es una cadena de caracteres y lo puedes ver con console.log(**typeof** contenedor.innerHTML); typeof devuelve una string. \*/

/\*

En este caso estábamos trabajando con un contenedor, veamos qué ocurre si intentamos seleccionar un elemento como por ejemplo el título, el elemento **h1.** Veamos que obtenemos, si vamos a las herramientas de desarrollo de chromeaquí vemos en la consola Titulo un elemento del DOM llamado **h1** puedes ver ese elemento de una forma común poco interactiva porque lo puedes expandir ▼ o colapsar ▲ es algo similar a lo que obtenemos cuando usamos la pestaña de elementos para ver su contenido al igual que podemos verlo en consola también podemos trabajar con ese contenido nuestro programa en VSC . Primero por curiosidad vamos a ver el tipo de datos empleando console.log(**typeof** titulo); el tipo de datos que se retorna es **object** o sea que título es un objeto con los cuales podemos trabajar usando sus propiedades y sus métodos. Otra propiedad importante para los proyectos es la propiedad **innerTex** esa propiedad inner (en español significa interno y Tex texto) retorna el texto contenida en ese elemento en este caso como es un título tiene texto interno tiene un Nodo de texto así que vamos a poder tener acceso a ese texto→Topping de Pizza y también al emoji por que también es parte de ese texto y al salto de línea, si agregamos typeof se retorna un strings.

También otra propiedad que puede ser interesante es el nombre de esa etiqueta **.tagName** Si en algún momento necesitas implementar alguna lógica en base al tipo de etiqueta html la puedes obtener a partir de esta propiedad, en español tag es etiqueta y name es nombre en consola veras **h1** y así es básicamente cómo funciona el método .getElementByld() Nota: se supone que el **id** es único por documento HTML. En caso que existirá otro **id** en el mismo documento lo cual no seguiría las recomendaciones de html y en consecuencia JS selecciona el primero que encuentra. Otra cosa que suele suceder es un error de tipeo, en ese caso el id no existe, como es un objeto la ausencia de ese obj se representa **null.** Es asi como se selecciona un elemento en base a su id y también sabes cómo acceder a ciertas propiedades importantes usando la notación de punto y también esa misma notación la puedes usar para llamar a ciertos métodos que veremos más adelantes.

\*/

Seleccionar elementos por clases:

En el caso anterior vimos como seleccionar un id, solo podemos seleccionar un solo elemento que tenga ese id por que el id es único. Pero cuando seleccionamos con clase podemos seleccionar varios elementos que tengan esa clase. Con el método **.getElementsByClassName** podemos obtener todos esos elementos en base a su clase, todos los que tengan esa clase. Declaramos una constante y vamos a usar ese método, nuevamente usamos el objeto especial llamado **document** porque ese método→**.getElementsByClassName** le pertenece al obj especial **document** que representa en general al documento html a partir del cual se generó ese DOM.

Codigo→const toppings = document.getElementsByClassName("toppings"); y vamos a especificar la clase entre comillas y dentro de los patentices como está especificado en el código, la clase es toppings y luego vamos a mostrar ese resultado con console.log(); a ver lo que obtuvimos. Siempre en ese caso cuando selecciones por clases y sepas que hay más de un elemento el nombre de la constante debería estar en plural topping**s** para que sea más fácil de leer el código y más fácil de entender. Vamos a la consola y aquí vemos algo especial, una colección HTML vemos que es un tipo de dato especial que es similar a un arreglo en JS y tiene 4 elementos, son 4 elementos porque tenemos 4 toppings. Si expandimos ▼ el contenido puedes ver que tenemos algo parecido a índices en realidad si son índices, el primer elemento tiene el índice 0, el segundo 1 y así sucesivamente para cada elemento igual que un arreglo de JS pero en este caso en vez de tener elementos sencillos son elementos objetos tomados a partir del DOM y vemos que el tamaño o longitud es 4. Si usamos la propiedad .length retornara 4. Volviendo a la colección y veamos aquí que es lo que obtenemos. Si queremos expandir ▼ el primer elemento que es cero veras todas las propiedades un elemento del DOM así sea nulas o no y su valor para c/u de ellos. Puedes ver que el primero corresponde a aceitunas vemos su texto interior y también tiene un id aceitunas así también puedes obtener un id de un elemento porque es una propiedad de ese elemento que vamos a obtener. También puedes obtener todas esas listas de clases que tiene ese elemento pero básicamente esto funciona como un array. Veamos cómo puedes acceder a esos elementos individualmente por que no siempre vamos a tener que necesitar trabajar con ellos como un array por ejemplo si necesitas solo acceder a las aceitunas a ese elemento que corresponde a aceitunas usarías su índice correspondiente en este caso [0] si lo mostramos con console.log(toppings[0]); vemos el elemento seleccionado **li#aceitunas.toppings.fondo-marron** la primera secuencia de caracteres representa la etiquetas de caracteres luego el numeral que es un id y los puntos son clases que tiene ese elemento si expandimos ▼ podrás ver todo el contenido todas las propiedades y métodos de ese Nodo de ese elemento html por ejemplo si quiero ese id usando notación de punto console.log(toppings[0].id); obtendremos el id de ese elemento que es aceitunas y es exactamente lo que tenemos en el archivo html.

\*/

/\*

**.getElementsByTagName ()**

Digamos que en lugar de seleccionar en base a una clase específica quiero seleccionar todos los elementos **li** en mi página web de mi documento, como hacemos eso? No necesariamente tengo que asignarle la misma clase si no lo necesito. Si quiero seleccionarlos todos puedo usar el método→.getElementsByTagName()

“tag es etiqueta en español” es muy similar al método→.getElementsByClassName pero en este caso la búsqueda va a ser más general vamos a seleccionarlo en base al nombre de su etiqueta html. La etiqueta debe ir entre comillas veamos el código→ const misToppings = document.getElementsByTagName(**"li"**); y en la herramienta de chrom en consola puedes ver la colección con 4 elementos. De la misma forma que lo seleccionamos anteriormente con la clase ahora la estamos seleccionando de una forma más general, todos los elementos de ese tipo se van a seleccionar sin importar si tienen alguna clase o **li** especifico, es importante saber que si tenemos otra lista con elementos **li** también se seleccionarían esos elemento. Veamos la diferencia de esto si quisiera seleccionar todos los elementos con la clase fondo marrón.

Vemos el codigo→const toppingFondoMarron = document.getElementsByClassName("fondo-marron"); pasamos el nombre de la clase en lugar de la etiqueta y en la herramienta de chrom en consola puedes ver la colección con 2 elementos seleccionados a partir del DOM. También puedes seleccionar elementos en base a los elementos que cumplan un selector Css específico.

\*/

/\*

**.querySelector()**

¿Cómo podemos seleccionar un elemento que cumpla el criterio de un selector Css? Tenemos dos métodos diferentes, .querySelector() nos permite seleccionar el primer elemento que cumple ese criterio y .querySelectorAll() nos permite seleccionar todos los que cumplan ese criterio. Estos métodos son útiles cuando necesitamos un proceso de selección un poco más específico es como cuando escribes tus selectores en Css tienes por ejemplo puedes combinar **id, class** en distintos estados también por ejemplo activos o enfocado, esas variaciones más poderosas la de los selectores Css las puedes usar con el método .querySelector(). En cambio los métodos que aviamos visto hasta el momento habían sido más generales. Veamos cómo puedes usar este método. Vamos a seleccionar este método con el **#→id** aceitunas usando este método que toma como argumento una cadena de caracteres con el selector Css que quieres usar. En este caso como queremos encontrar un **id** el primer elemento con ese id escribimos ("#aceitunas") como vemos en el codigo→const aceitunas = document.querySelector**("#aceitunas")**; y en la herramienta de chrom consola mostramos el resultado vemos ese primer elemento del DOM que corresponde a aceitunas que tiene ese **id** eso es uy útil porque también podemos trabajar con sus propiedades y métodos. También podemos usar este método para seleccionar una class específica en el DOM en esa jerarquía **(".toppings")** seria aceitunas y si efectivamente vemos en consola aceitunas. ¿Que nos retorna este método? veamos con **typeof** vemos en consola que su retorno es un **object** con el cual podemos trabajar en JS.

\*/

/\*

Pero hasta el momento todo parece similar con lo que aviamos hecho anteriormente con los otros métodos, el poder de este método es que podemos combinar distintos criterios para el selector. Por ejemplo si necesitamos escoger un elemento que tenga dos **class** o con algún **id** especifico podemos hacer esas combinaciones para obtener lo que necesitamos. Digamos que queremos obtener el primer elemento que tenga la clase .toppings y la clase .fondo-naranja, eso es algo que no aviamos podido hacer anteriormente pero con el codigo→const primerToppingsNaranja = document.querySelector**(".toppings.fondo-naranja")**; vemos que el resultado es cebolla porque puedes ver que en realidad efectivamente si es el primer elemento del DOM que tiene la clase toppings como la clase fondo-naranja y con typeof vemos que retorna object. Veamos otro ejemplo: Una forma equivalente a seleccionar el primer toopings con fondo naranja para este caso sería con un selector más elaborado que tenga la etiqueta→ul dentro de una lista no ordenada que sea de tipo **li** y que tengan la clase .fondo-naranja. Aquí si vemos el contenido que nos retorna el método.

const primerToppingsNaranja = document.querySelector(".toppings.fondo-naranja"); toppings fondo-naranja, ya esto es un ejemplo más elaborado del tipo de selector que puedas escribir para encontrar elementos con el método .querySelector() que es muy útil.

\*/

/\*

Y por último veremos otro ejemplo para mostrar cómo puede ser elaborado. Digamos que queremos seleccionar el primer elemento, el primer toppings que no sea marrón el primer toppings que no tenga la clase fondo-marron. Esto te debería resultar interesante es como un operador en Css que nos permite descartar los elementos que tengan esa clase, esa característica. Con este codigo→const primerToppingsNoMarron = document.querySelector**("ul li:not(fondo-marron)")**; vamos a mostrar en consola y vemos el primer toppings que no tiene fondo-marron que es cebolla, así que puedes ver las variaciones que podemos usar para seleccionar exactamente lo mismo pero debemos saber cómo personalizar el selector para que cumpla exactamente lo que necesitamos para nuestra página web. En este caso estamos seleccionado el primer elemento que cumpla ese criterio que fuera seleccionado con este selector ("ul li:not(fondo-marron)");

Pero veamos qué es lo que ocurre con el método .querySelectorAll() que es una variación y lo veremos a continuacion

\*/

/\*

Y finalmente el cuarto método que vamos a ver para seleccionar los elementos del DOM es .querySelectorAll() veamos un ejemplos con los toppings de fondo naranja, es muy similar a.querySelector() y vamos a pasar como argumento un selector Css como una cadena de caracteres, pero en este caso como le estamos agregando la palabra **All** tenemos un método diferente que nos va a retornar una colección html como un arreglo con todos los elementos que cumplen ese criterio por ejemplo los elementos de la clase toppings y fondo-naranja veamos qué es lo que ocurre, usando la herramienta de chrom consola mostramos el resultado. En este caso tenemos algo diferente, tenemos una lista de Nodos que es diferente a una colección html Nodelist y tiene 2 elementos a los cuales podemos acceder con sus índices respectivos [0][1] además esa lista de Nodos tiene un tamaño→length además la lista de Nodos se considera un objeto en JS, el método retorna un objeto, puedes usar notación de punto para obtener sus propiedades y trabajar con sus métodos. Recuerda los 4 elementos básicos para seleccionar elementos del DOM son los que a continuación vemos en el gráfico.



Los métodos .querySelector(); y .querySelectorAll(); son más amplios son, más poderosos pero normalmente tratamos de usar los que son más específicos como .getElementById(); .getElementsByClassName(); depende de lo que necesites usar y también debes tener en cuenta el efecto que puede tener usarlos sobre el desempeño o la eficiencia de tu página web o de tu aplicación.

\*/

/\*

**Asignar Estilos:**

En este capítulo veremos cómo puedes personalizar estilos de un elemento con JS, eso es muy útil en caso si quieres cambiar el estilo o el valor de alguna propiedad Css en base a como el usuario a interactuado con la página web. Por ejemplo si hace click en un botón otro elemento puede estar afectado o si alguna operación se completa puedes querer mostrar algún elemento o esconderlo, depende de lo que necesites porque esto es muy versátil. Comencemos lo aprendido en la sección anterior como seleccionar un elemento, vamos a continuar con nuestros toppings como queremos solamente obtener nuestro primer topping de la lista vamos a usar el método .querySelector(); y vamos a encontrar ese primer elemento con la clase .toppings. Para confirmar que tenemos acceso a ese elemento, vamos a mostrar en consola y vemos el primer topping aceitunas. Una vez que ya tenemos ese elemento podemos asignar un estilo usando notación de punto. Todos los elementos que selecciones van a tener una propiedad llamada style, mostremos esa propiedad en la consola para ver el estilo actual de ese elemento veremos el contenido de ese estilo. Es una declaración de estilo Css “cssstyledeclaration” expandiendo ▼ en la consola podemos ver todas las propiedades css que normalmente podemos asignarle a un elemento. Todas esas propiedades que podemos asignarle en el archivo styles.css también podemos personalizarla en JS. Algo importante para tener en cuenta es que en css esos nombre cuando tienen más de una palabra lo separamos con un guion pero en JS vamos a seguir la convención de nomenclatura que usamos con las variables y las funciones →camelCase hay que recordar que son objetos y usando notación de punto puedes acceder a sus propiedades y atributos. Vamos a personalizar este topings, básicamente es como asignarle un valor a una variable. Primero seleccionamos el estilo de ese elemento→ primerTopping.style y luego usando notación de punto especificamos cuál de esas propiedades queremos personalizar en este caso es color de fondo→ .backgroundColor = "blue"; y veamos qué es lo que ocurre, usando la herramienta de chrom consola mostramos el resultado. El primer topping Aceitunas su color es de fondo azul se cambió el estilo con JS. Si cambiamos a la pestaña Elements en las herramienta de Chrome seleccionamos el icono superior izquierdo que permite que el cursor seleccione el elemento deseado en este caso el primer elemento de la clase .topping puedes notar que si revisas el DOM vas a encontrar un estyle en línea que se le asignó a ese elemento en específico, le agrega un atributo estyle a ese elemento y agrega esa propiedad Css con el valor que corresponde y vemos también como JS sabe automáticamente como transformar esa propiedad en la notación que normalmente usamos en Css con un guion para separar las palabras style = “background**-**color: blue;” y culmina esa línea con un punto y coma como hacemos en Css asique ese estilo va a tener la mayor prioridad porque es un estilo en línea. No es un estilo asignado a través de una clase o de un id. A continuación vamos a personalizar el estilo del color de letra usando color hexadecimal como lo harías normalmente en Css .style.color = "#6dff00"; y veamos qué es lo que ocurre, usando la herramienta de chrom consola, efectivamente vemos el color de la letra como un verde neón y el fondo es azul. Puedes ver que ese estilo se agregó a la lista de style en línea para ese elemento, si bien en JS lo especificamos como hexadecimal pero automáticamente se transformó en un modelo RGB. También otro ejemplo interesante es que podemos transformar el texto a mayúscula. .style.textTransform = "uppercase"; igual que lo haríamos con una propiedad en Css. Puedes ver que ese estilo se agregó a la lista de style en línea para ese elemento y recuerda que ese estilo va a tener la mayor prioridad porque es un estilo en línea.

Una vez que ya obtuvimos del DOM el nombre del elemento, en este caso→ **primerTopping** puedes usar el mismo formato o misma sintaxis→**primerTopping.style.** seguido del nombre de cualquier propiedad que puedas personalizar. console.log(primerTopping.style); Si expandimos ▼ return muestra todas las propiedades

\*/

/\*

**Textos en el DOM, Acceder al texto:**

Forma y alternativa de acceder al contenido interno de un elemento que contiene texto. Primero que todo cualquier operación realizada con el DOM debemos obtener ese elemento usando uno de los métodos que vimos en anteriores capítulos. En este caso vamos a seleccionar la lista-toppings, la lista como tal el elemento **ul** con el método **.getElementById();** vamos a obtener esa lista con su **id**→("lista-toppings") lo mostramos en la consola para ver que tenemos acceso a esa lista. Efectivamente tenemos la lista de toppings con sus 4 elementos interno, esa lista también contiene Nodos específicos para el texto y hay varias formas para acceder a ese texto en caso que necesites usarlo en el programa o modificarlo. El primero es **.innerText.**

(Acotación: Texto interno) Vemos en consola el resultado ACEITUNAS, Cebolla, Albahaca, Champiñones. Si vemos la descripción nos dice que retorna una cadena de caracteres.

Ahora veamos el resultado con **.textContente** que puede ser similar a **.innerText** y vamos a mostrar el nombre de las propiedades para diferenciarlas solo para fines visuales. Vemos en consola que .innerText nos retorna cadenas de caracteres en el orden en que se encuentran en el archivo html. En contraste .tectContent aparece en forma similar pero incluye los espacios que tenía ese texto en el archivo html, si vemos el archivo html puedes ver que el texto de los títulos esta indentado. Este texto que muestra .textContent no está al mismo nivel que el elemento html principal, en el archivo index.html contamos 8 espacio de indentación para ese elemento **li** exactamente los mismos espacio vemos en consola. Esto es importante saberlo en caso que tengas que decidir cuándo usar estas propiedades .textContent o .innerText. Y final mente tenemos una tercera opción que se denomina .innerHTML esta propiedad es un poco diferente a las dos anteriores que solo retornaban textos en cambio .innerHTML va retornar la estructura html interna de ese elemento como una cadena de caracteres. En consola podemos ver toda esa parte de la estructura html como si fuera el archivo index.html incluyendo las etiquetas de apertura de cierre de cada uno de esos elementos incluyendo las class sus id. en contraste a las dos anteriores propiedades solo obteníamos el texto esa son las tres formas de poder acceder al texto de un elemento.

\*/

**Acceder al texto de un elemento:**

Al igual que podemos acceder a l texto al contenido de ese elemento, también podemos modificarlo. Vamos a obtener el elemento titilo y vamos a cambiar su contenido y verificamos con la herramienta de Chrome en la consola const modificarTitulo = document.getElementById("titulo"); y efectivamente tenemos el título **h1** y si obtenemos su valor innerText vamos a obtener toppings de pizza, digamos que queremos cambiar ese texto como tal, lo que hacemos en ese caso es asignar una cadena de caracteres a la propiedad .innerText digamos que optamos por “Mis toppings favoritos”Verificamos en consola y podemos ver el cambio realizado con el nuevo texto. Simplemente se modificó el texto de ese elemento en el DOM con una sola línea de código.

**Atributos:**