**14/10/2023-14/12/2023 Token github: ghp\_RZh59nvmcAfHuDtno1lVpdjel98REd34UQLf**

-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 así si en la pantalla o el monitor donde se está viendo es muy ancho el máximo se lo indicamos en 600px. El color de fondo **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.

\*/

Stop

/\*

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 especifico→.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 propiedad nos 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 Chrome aquí 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 porque 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 toppings 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 Chrome 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 Chrome consola mostramos el resultado vemos ese primer elemento del DOM que corresponde a aceitunas que tiene ese id eso es muy 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 Chrome 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:

Ahora veremos cómo podemos modificar los atributos de un elemento. Después de la lista no ordenada de toppings vamos agregar un enlace para modificar el atributo href no lo colocamos en una pestaña nueva para fines de la demostración <a href="https://www.freeCodeCam.org/Español/"> definimos una constante para el enlace y vamos a seleccionar los elementos por el nombre de su etiqueta. En este caso va a ser la etiquita →a Veamos el valor de enlace en consola, es una colección html con tan solo un elemento el elemento que tenemos para el enlace. Como es una collection html vamos a seleccionarla con su índice [0] y cuando lo seleccionamos vamos a poder trabajar con ella como si fuera cualquier otro elemento, cualquier otro objeto del DOM. Y si necesitamos verificar algunos de sus atributos podemos llamar el método .getAttribute("href") en este caso el valor de href en consola lo podemos ver→ <https://www.freeCodeCam.org/Español/> podemos trabajar con ese valor en nuestro programa, modificarlo o eliminarlo si lo necesitamos. Para eliminar ese atributo usamos el método→ .removeAttribute al ejecutar ese método seguidamente vemos en Console el valor undefinet y pasamos a Elements en la etiqueta de apertura ya no vemos el atributo href por que fue eliminado dinámicamente en el DOM ese atributo ya no es parte de ese elemento html. Si necesitamos obtenerlo usamos el método .getAttribute() y también si necesitamos actualizarlo podemos usar el método .setAttribute() Especificamos en el código como primer argumento el atributo que quiero personalizar→ href ya que es el único enlace que tenemos en la página y su índice es [0] y como segundo argumento paso el valor que quiero asignar .setAttribute("href", "https://www.freeCodeCam.org")); vemos en la herramienta de Chrome Elements el resultado, vemos como se actualizo el DOM ahora ese enlace tiene como referencia, como destino→ <https://www.freeCodeCam.org> ya no vemos /español así es como podemos actualizar los atributos y esto funciona con cualquier atributo valido en html para ese elemento.

\*/

/\*

Clases:

Otras operaciones importantes que puedes realizar en JS son agregar modificar y eliminar clases en forma dinámica con esta operación. Nuevamente vamos a seleccionar el primer toppings que consigamos en el DOM. Estamos usando los toppings solo para fines de demostración para que nos concentremos en las operaciones nuevas. Vamos a mostrarlo en la consola y ahí lo podemos ver **<li class="*toppings fondo-marron*" id="aceitunas" style="background-color: blue; color: rgb(109, 255, 0); text-transform: uppercase;">Aceitunas</li>**.

En JS cada uno de esos elementos que estamos seleccionando en el DOM va a tener una propiedad llamada **.classList** es una lista de todas las clases que tiene actualmente ese elemento “pirimerToppings” en el DOM, en consola lo puedes ver *DOMTokenList(2)* ese elemento tiene 2 clases, la clase **.toppings** y la clase **fondo-marron** puedes ver que es como un array con dos elementos cada uno tiene su índice [0][1] de igual forma puedes acceder a ellos. Vemos que tiene la propiedad lengts: que puedes usar en el programa y también tiene la propiedad value: en caso que necesites obtener toda esa cadena de caracteres con la lista de clases

{

"0": "toppings",

"1": "fondo-marron"

Lengts: 2

Value: “toppings fondo-marron”

}

**Agregar una Clase:**

Si necesitamos agregar una clase a un elemento escribimos el nombre de ese elemento luego que ya lo seleccionamos en el DOM luego class.list y luego llamamos al método .add()→(en español agregar) y agregamos en los patentices una clase espesifica **mi-Clase** que todavía no le hemos asignado ningún estilo es solo para fines de demostración. Es para verla aquí en la lista de clases *DOMTokenList(3)* vemos que se agregó una porque ahora vemos 3 y vemos mi-Clase en la lista que se retornó. También si vamos a Elements y seleccionamos ese elemento podemos ver que ahora tiene una tercera clase llamada mi\_Clase, si esa clase tuviera un estilo en el archivo Css se aplicaría ese estilo.

\*/

/\*

**Incluir la clase en el archivo Css:**

En otro ejemplo con el toppings-fonfo-Naranja, aplicando lo aprendido anterior mente vamos a nuestro archivo Css y agregamos una nueva clase que la llamaremos texto-verde y le asignamos el color verde neón. Estamos agregando esa clase al primer toppings de fondo-naranja que sería el elemento cebollay también mostramos la lista de clases para confirmar que existe. En Chrone vamos a Console y podemos ver que se aplicó el estilo texto-verde

\*/

/\*

**Verificar si una Clase existe:**

Otra operación importante también es verificar si un elemento posee una clase o no. Eso lo podemos confirmar con el método **contains()** este método nos permite verificar si una clase especifica existe en esa lista de clases en este caso <ul id="lista-toppings"> Digamos que queremos verificar si tiene la class="toppings fondo-marron". En ese caso como se va a retornar un valor, no es una acción si no que se va a retornar un valor boolean true o false. Vamos a mostrar ese valor con consol.log() y vemos el resultado que es true porque ese elemento si posee esa clase. Si pasamos una clase que no tiene ese elemento veremos que el valor es false. Esto es útil en caso que necesites usarlo en un condicional o en un bucle, más que todo en los condicionales son muy útiles.

**Eliminar una clase:**

También se puede eliminar una clase de un elemento, esa operación también es muy útil. Para eso usamos el método **remove()** y pasamos entre paréntesis el nombre de la clase, veamos qué ocurre si eliminamos la clase toppings a ese elemento. Vamos a Chrone y vemos que el topping ya no tiene el estilo asociado a los toppings porque es simplemente otro elemento más sin esa clase, si vemos la lista de clase en Console ya no la tiene solo tiene la clase fondo-marron.

Y así es cómo podemos agregar y eliminar una clase y como verificar si un elemento tiene una clase o no. Estas son operaciones muy útiles que vas a usar a menudo si trabajas con el DOM. Ahora como ya sabes cómo agregar clases y estilos entonces vamos a ver cómo puede crear agregar y eliminar elementos en el DOM.

\*/

/\*

**Crear un Elemento:**

Veamos cómo puedes crear un elemento desde cero, eso sí solo lo puedes realizar con JS y luego agregar ese elemento al DOM de forma dinámica, eso es muy útil como por ejemplo recibes información de un servidor o de una API recibes es información en un formato **json** que se usa para intercambiar datos en desarrollo web y necesitas transformar esa información en elementos HTML que los usuarios puedan ver con los cuales puedan interactuar, lo puedes realizar en forma dinámica. De alguna forma digamos que el usuario va a poder ingresar un toppings nuevo. En ese caso deberíamos tomar ese elemento nuevo y crear ese elemento en el DOM. Lo primero que debemos hacer es crear un elemento nuevo con el método **createElement();** Que tipo de elementos queremos crear? Un elemento (**li**) un elemento de lista. A ese elemento si lo agregamos desde cero sin personalizar nada veamos qué es lo que ocurre. Para agregarlo primero necesitamos obtener una referencia al elemento delDOM donde queremos agregarlo porque en este momento JS no sabe nada de nuestro documento, no tenemos referencia a ningún otro elemento más que el toppings nuevo, vamos a obtener alguna referencia a la lista de toppings que es donde vamos agregar ese elemento **li** y si vamos a nuetro archivo index.html vemos que el **id** de **ul** es lista-toppings.

**Agregar un Elemento:**

Ahora si tenemos una referencia de esa lista const lista\_Toppings = document.getElementById("lista-toppings"); donde queremos agregar el topping nuevo porque todo funciona dentro de un contexto y es importante que recuerdes que al agregar un elemento tenemos que especificar donde queremos agregarlo. Eso lo podemos hacer con el método **.append**

Este método nos va a permitir agregar un Nodo en esta lista. En resumen: vamos agregar un toping nuevo a la lista de toppings, veamos qué es lo que ocurre porque en este momento no tiene ningún contenido ni ningún estilo. Si vamos a Chrome en Console vemos que no pasa nada, no ves nada pero en Element vemos que el elemento agregado está allí lo podemos ver en el DOM si lo expandimos ▼ podemos ver que está pero no tiene ningún estilo, tampoco ningún contenido. Antes de agregarlo en realidad deberíamos asignarles esas propiedades y esos estilos importantes, entre esos vamos agregar una clase, tomando la lista de clases y el método  **.add** dos clases importantes ¡Sí! Con el método **.add** puedes agregar dos clases a la vez solo tienes que separarlas con comas. Abrimos Chrone, Element y podemos ver en el DOM que este último elemento **li** ahora si tiene un estilo asignado porque tiene la clase toppings y tiene la clase fondo-marron. Lo único que nos falta es el texto, para eso podemos personalizar lo que vimos anteriormente el método **.innertext**  agregando un texto “Queso extra” vamos a Chrone en Element vemos que se agregó a la lista de la clase topping y fondo marrón.

**Remover un Elemento:**

Así como podemos crear, agregar elementos también podemos eliminarlo, escribimos el nombre del topping nuevo y luego llamamos al método **.remove()** vamos a Chrone en Element vemos que el elemento nuevo ya no existe removió ese elemento del DOM, lo mismo puedo aplicarlo a cualquier elemento por ejemplo si quiero eliminar toda la lista de toppings puedo llamar al método .remove() vamos a Chrone en Element vemos que la lista de toppings ya no existe tan sencillo como eso. Lo único que necesitamos es una referencia en el DOM, digamos que sería como darle una dirección donde quereos que se desarrollen los acontecimientos

const lista\_Toppings = document.getElementById("lista-toppings"); También existe otro método llamado **.appendChild** también funciona y funciona para elementos html vamos a Chrone en Element vemos que en este caso es equivalente. Esto funciona con elementos html y si usamos **.append** podemos agregar texto **lista\_Toppings.append(¡Hola Mundo!);** vamos a Chrone en Element vemos que se agregó un Nodo ¡Hola Mundo! Que no es un elemento html como tal se agrega el texto pero es un texto sencillo no es un texto que es parte de otro elemento html.

\*/

/\*

**Recorrer el DOM:**

Ya sabes como como crear, agregar y eliminar elementos del DOM con JavaScript esas operaciones son muy útiles. Ahora veamos cómo puedes recorrer el DOM porque JS nos permite aprovechar esa jerarquía con Nodos padres, nodos hijos, nodos hermanos toda esa jerarquía nos va a permitir recorrer el DOM con nuestro código en caso de que lo necesitemos. Por ejemplo digamos que tenemos el elemento listaDeTtopping en caso de que ocurra algo, realizar alguna operación con su elemento o Nodo padre simplemente podemos seleccionar una de estas dos opciones **parentElement** “elemento padre” o **parentNode** “el Nodo padre” hay que recordar la diferencia entre elemento y Nodo. Nodo es más amplio e incluye también textos o comentarios pero el elemento padre solo incluyen los elementos html asique si queremos obtener el **parentElement** de esa lista veamos que obtenemos vamos a Chrome en Console vemos el elemento contenedor el **div** principal ese es el elemento que contiene a la lista de toppings. Es el elemento padre del elemento **ul** de la lista y también es el elemento padre de título asique podemos obtener una referencia a ese contenedor sin obtenerlo directamente si no indirectamente a través de la lista de toppings, eso va a ser importante

porque por ejemplo en nuestro código a veces vamos a necesitar realizar operaciones con el padre cuando ocurra algo en el hijo por ejemplo digamos si hago click en algunos de estos ingredientes de algunos de estos topings quiero cambiar el color de fondo del contenedor solo por decir un ejemplo en ese caso yo podría decir que cuando ocurra algo en esa lista, en esos ingredientes, en esos toppings quiero que el contenedor cambie de color de fondo. Indirectamente tenemos que afectar el contenedor puedo simplemente usar esta alternativa **.parentElement** con la notación de punto y selecciono a su padre es más sencillo. Ahora si tenemos el Nodo padre **.parentNode** veamos que obtenemos vamos a Chrone en Console vemos que obtenemos aquí **div►# contenedor** si ▼ expandimos puedes ver todas las características, todas las propiedades de ese Nodo siempre recordando que Nodo es más amplio. Si queremos ir todavía más arriba en la jerarquía podemos crear una cadena una secuencias de referencia al padre. Aquí estamos obteniendo el padre de la lista de toppigs→ **lista\_Toppings.parentElement** y luego usando la notación de punto obtenemos el padre del padre de la lista de toppings→ **lista\_Toppings.parentElement.parentElement);** y así sucesivamente vamos a Chrome en Console vemos que obtenemos ►body ya fuimos más arriba todavía en lugar del contenedor principal, eso es en el caso de los padres también podemos obtener los Nodos o elementos hijos. En ese caso necesitamos la propiedad **.children**→“En español hijo” esta propiedad nos va a dar todos los Nodos hijos en la lista de toppings en base a una HTMLCollection(5) en consola lo podemos ver, tiene 5 hijos esa lista de toppings expandimos ▼y estos 5 elementos los podemos usar de forma individual con sus [i] índices pero tenemos acceso a ellos gracias a esa propiedad **.children** es que no estamos accediendo a ellos directamente si no indirectamente a través de la lista de toppings y eso es útil también para realizar ciertas operaciones en nuestro código además de **.children** también tenemos la opción de seleccionar el primer hijo con la propiedad **.fistChild** En este caso inesperadamente vemos el ►#text ¿por qué? Esta propiedad trabaja con Nodos y desafortunadamente en este caso como estamos incluyendo espacios, indentación, ese espacio también se toma como un Nodo. Si quitáramos la indentación en nuestro archivo html en lugar de ►#text lo que se mostraría es el próximo elemento **<li class= "toppings fondo-marron" id="aceitunas">Aceitunas</li>** exactamente lo que esperábamos pero como trabaja con Nodos esta propiedad **.firstChild** normalmente obtendríamos textos en lugar de un elemento como tal por que por norma siempre nuestro código esta indectado adecuadamente para poder leerlo correctamente. En ese caso en vez de usar **.firstChild** usaríamos la propiedad **.children** y usamos el índice correspondiente [0] y allí tendríamos accesos a ese primer hijo directo del elemento, esa propiedad si nos retornaría elementos html. También podemos usar esta otra propiedad **.lastChild** que también usa Nodos y sucede lo mismo que con **.fistChild** nuevamente en Console vemos ►#text ya que en nuestro archivo usamos el código indectado en este caso no nos ayuda. Pero por fortunas esas dos propiedades también tienen su versión para obtener elementos html en lugar de usar **.firstChild** usamos la propiedad **.firstElementChild** y esto va a ser lo más práctico para nosotros si necesitamos obtener esos elementos html en Console podemos ver un elemento html el cual tiene propiedades, en este caso si verificamos con **typeof** vemos que es un obj de JS y también tenemos el equivalente para el último elemento hijo y en Console lo podemos ver es el toppings Queso extra. Asique tenemos las dos versiones, para trabajar con Nodos y para trabajar con elementos. Entre **.last** y **Child** agregamos la palabra **Element** para obtener un elemento html que no incluya ni texto ni los comentarios. Y también además de padres e hijos tenemos hermanos ¿Cómo podemos obtener los hermanos de toppings? En este caso si nos vamos a Chrone en Element venos que sería **h1** **título** que están en el mismo nivel en jerarquía que **ul** que pertenecen a contenedor que están dentro de ese **div** ¿Cómo podemos obtener titulo indirectamente a través de lista de toppings? Eso lo haremos con la propiedad **.previousElementSibling**

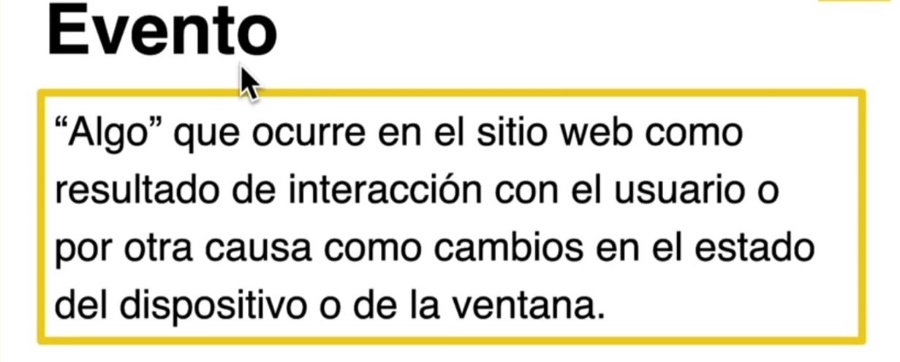
“En español .anteriorElementoHermano”y en Chrone en Console allí lo que vamos a ver es el elemento **titulo** porque obtuvimos el hermano anterior de la lista de toppings. También podemos ver el hermano posterior de la lista de toppings con la propiedad **.nextElementSibling** pero en este caso no tenemos ninguno en el mismo nivel de la jerarquía por lo tanto veremos **nul**l y también tenemos la versión para trabajar Nodos si omitimos la palabra Element vamos a obtener un Nodo **.nextSibling** en este caso es **texto** como vimos anteriormente por los espacio o indectado y **.previousSibling** también nos va a retornar **texto** asique normalmente vamos a trabajar con la versión que contiene **Elements** y así es como recorrer el DOM con notación de punto usando estas propiedades. Recuerda para **.parentElement** y **.parentNode** también tenemos **.fistChildren** para todos los hijos. Tenemos el **.fistChild** y **.firstElementChild** tenemos el **.lastChild** y **.lastElementdChild** (*last significa último en español*) y también tenemos **.nextElementSibling** y **.nextSibling** y también tenemos **.previousElementSibling** y **.previousSibling** para padres, hijos y hermanos y también puedes encadenar esa notación de punto para ir más arriba, más abajo o más profundo en el DOMcomo lo hicimos en el primer ejemplo **console.log(lista\_Toppings.firstElementChild.firstElementChild);** con esto se obtiene “hijo del primer hijo de la lista de toppings” o **console.log(lista\_Toppings.parentElement.parentElement);** obtenemos el padre del padre de la lista de toppings, Con todas estas propiedades puedes hacer exactamente lo mismo siempre deberías verificar que estas propiedades no sean Nodos que en verdad existe y no sea nulo el elemento y así se recorre el DOM.

\*/

/\*

**Eventos del DOM:**

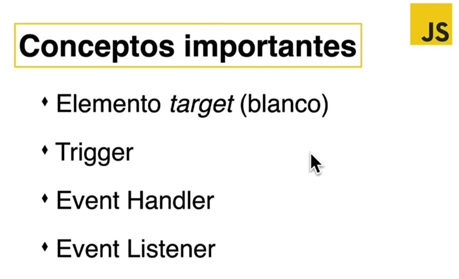
En esta parte vamos hablar un poco de los eventos, xq los eventos son muy importantes al momento de interactuar con el DOM.



En el grafico vemos una definición no muy formal como “Algo” Todo eso que describe el grafico, todos esos evento que ocurren en el sitio WEB van a desencadenar eventos formalmente en el sistema, en el navegador y eso nos va a permitir saber cuándo ocurren esos eventos cuando el usuario este interactuando con distintos elementos de la página y también nos permite definir cómo manejar esos eventos en caso de que ocurran por ejemplo uno de los más comunes es el evento de click y cada interacción con el ratón se va a registrar como un evento en el navegador y el navegador va a saber sobre cuál de esos elementos se hizo el click y vamos a poder decir en caso de que ocurra el click que es lo que vamos hacer en nuestro código. ¿Cómo podemos manejar ese evento? todo eso lo podemos definir con JS y también otro evento común es la iteración con el teclado, presionar alguna tecla. Podemos lograr que ciertos elementos que nos interesan en el DOM sepan cuando ocurren uno de esos eventos cual de esos elementos fue afectado debería reaccionar en base a ese evento y también cuál de esas teclas fue presionada para saber cómo manejar ese evento adecuadamente por ejemplo un caso común sería un juego, si estas desarrollando un juego interactivo en JS y el usuario presiona alguna de las teclas xej izquierda o derecha para avanzar y retroceder o arriba y abajo para saltar, todos eso eventos estarían registrados y podríamos manejarlos de acuerdo a la tecla que fue presionada todo eso es parte del DOM también cambios en la ventana de la pestaña o del navegador también pueden se registrados, se registran automáticamente y puedes manejarlos. Otro evento bastante común en JS es el evento de arrastrar xej en una página web te pide que arrastres una imagen para subir y asignarla como tu imagen de perfil. Ese evento de arrastrar un archivo asía un elemento en particular y que luego ocurra algo se maneja gracias a la combinación entre el DOM y JS a través de esos eventos y también recientemente eventos que deberías manejar son eventos que pueden ocurrir en dispositivos táctiles puedes manejar distintos gestos que realice el usuario de distintas formas así que puedes ver que los eventos son bastantes dinámicos y se pueden adaptar a distintas necesidades como desees manejarlos. Existen muchos eventos que puedes manejar y puedes encontrarlos en distintos artículos o en recursos en línea. En este curso solo veremos algunos de los más comunes como los elementos del teclado y del cursor pero una vez que sepas manejar estos eventos manejar otros tipos de eventos es exactamente igual o muy parecidos lo que se necesita saber es la sintaxis de cómo escribir ese código para manejar el evento y luego lo único que vas hacer es cambiar el nombre del evento y que es lo que quieres hacer con el elemento.

**Conceptos Importantes:**

Cuando hablamos sobre eventos en el DOM y al momento de manejarlos con JS es importante que sepas algunos conceptos importantes estos cuatros que se presentan en el gráfico.



Primero el Elemento **targuet** que significa “blanco” pero no blanco del color sino el blanco que va a recibir algo o donde queremos enviar algo nos referimos al elemento que va a recibir este evento. **Trigger** también es un concepto que lo vas a recibir muy a menudo es como un desencadenante y luego **Event Hander** y

**Event Listener** Esto es una lista general para que tengas una idea. Veamos c/u en más detalle: Primero tenemos el elemento **target** el elemento blanco, xej si tenemos algunas páginas web y hacemos click sobre algunos de esos elementos que podemos ver, como una imagen, un botón algo que tenga una capacidad interactiva ese elemento en el cual se va a generar ese evento es a lo que nos referimos como el **target element** el blanco de ese evento xq el usuario quiso interactuar con ese elemento asique hay que reaccionar a esa interacción y manejarla adecuadamente. Ese elemento, el elemento blanco va a decirle al navegador que se produjo una acción por eso es que lo denominamos el elemento blanco allí se desencadeno ese evento. Eso nos lleva al próximo termino **trigger** (en español desencadenar) **trigger** sería como un desencadenante, esa acción que va a desencadenar un evento xej hacer click que luego se va a poder manejar a travez del DOM y JS eso también nos lleva a un concepto que va a ser fundamental para trabajar con eventos también con librerías y framewrks como xej React. **Event Handler** es una función que se ejecuta cuando ocurre un evento **Handler** significa manejar es una función que va a manejar un evento cuando ocurra, pero no podemos saber cuándo va a ocurrir el evento asique va a estar allí como pendiente. Vamos asociar esto con nuestro código con una función. **Código←→Función** para decirle al navegador que cuando ocurra un evento especifico en un elemento especifico vamos a llamar a una función específica de manera tal que estamos asociando todo esto, el elemento, el evento y la función que lo va a manejar si ocurre.

Esa asociación que hay en un evento especifico en un elemento y la función que lo va a manejar es lo que denominamos **Event Listener**, (En español *Listener* significa escuchar) es esa conexión que vamos a crear en nuestro código tenemos que crearla explícitamente en JS para asociar un evento en un elemento con una función específica que va a manejar ese evento en ese elemento cuando en ese elemento especifico ocurra un evento se va a llamar esa función pero esa asociación la tenemos que crear nosotros mismos en el programa y lo vamos hacer con unas pocas líneas de codigo. Cuando creamos Event Listener estamos escuchando ese evento, estamos diciéndole al elemento que cuando ocurra este evento llama a esta función y ejecuta su código. Pero como el DOM o el navegador no saben cuándo va a ocurrir ese evento tiene que estar pendiente allí escuchando en caso de que ocurra. Estos serían los conceptos más importantes para trabajar con eventos. El elemento **targuet** que es el que recibe esa acción que desencadena el evento, el **trigger** es el desencadenante o es la acción que desencadena el evento, el **Event Handler** es la función que va a manejar lo que ocurra cuando se desencadena el evento y **Event Listener** es más o menos como esa conexión que hay entre el evento el elemento y la función que lo va a manejar. Normalmente vas a conseguir estos dos términos **Event Handler** y **Event Listener** usados en forma equivalente y los términos a veces se intercambian. En la mayoría de los contextos puedes usarlos de forma prácticamente equivalente, hay ciertas fuentes que distinguen el **Event Listener** como esa asociación que hay entre el evento y la función y el **Event Handler** lo espesifican como la función en sí que maneja el evento.

\*/

/\*

**Elementos HTML:**

Ahora vamos a ver un ejemplo práctico en nuestro inde.html de toppings de pizza sobre los **Event Listener** y cómo puedes ejecutar código cuando ocurra un evento en algún elemento del DOM vemos un ejemplo que sirve para ilustrar este concepto y luego en los proyectos vamos a implementar este concepto de forma más avanzada. Una opción para poder ejecutar código cuando ocurra un evento especifico es indicar toda esta asociación en html digamos que cuando se haga click en el elemento albahaca quiero que se muestre un mensaje en la consola, por eso en html tenemos atributos que representan distintos eventos En este caso tenemos el atributo **onclick** vamos a manejar un evento de click, si el usuario hace click sobre ese elemento vamos a ejecutar una función específica esa función la vamos a definir en nuestro archivo JS ese archivo va a tener una función que se va a llamar **mostrarClick()** es importante que incluyas los paréntesis vacíos o con los valores que requieran y dentro de la función vamos a mostrar un mensaje con console.log(“Click”). En el archivo JS definimos la function mostrarClick() y dentro de la función vamos a mostrar este mensaje (“Click”) cuando el usuario haga click en el toppings que especificamos en la lista ul en el index.html este sería en cuanto a los términos que vimos en las diapositivas.Esto→**mostrarClick()** sería el **Event Handler** esa es la función que va a manejar ese evento y esta asociación que hay entre la función y el evento que esta ya predeterminada y lista para ejecutarce cuando ocurra el evento es lo que denominamos también un **Event Listener** pero ya también ese concepto lo vamos a ver en JS. Veamos cómo funciona como tenemos live server activado ya debería haberse actualizado todo nuestro código. Vamos a Chrone abrimos console y no vemos nada pero si hacemos click en el elemento albahaca nos aparece 1Click anteponiéndose el número 1 y cada vez que hagamos click ese número se irá incrementando el contador. Si hacemos click en alguno de los otros toppings vemos que no se incrementa el contador. Ahora asignemos a c/u de los topipings **onclick con la función** o sea que reutilizamos la misma función y la asociamos con el resto de los elementos de la lista y de igual forma obtenemos el mismo resultado que con el toppings Albahaca. Y también podemos personalizarlo, digamos que queremos personalizar el valor. Colocamos un parámetro en la funcion→(topping) de igual manera en console.log(topping); y en index.html en la función mostrarClick() los valores que van entre los paréntesis en esos string van comillas simple si no hay **error.** Lo visto hasta ahora es una alternativa pero la realidad cuando vas a crear aplicaciones más complejas no es tan practico escribir esto→ **mostrarClick(`Aceitunas`)"** en html porque tendrías que definir este atributo→ **onClick** y llamar a la función con sus argumentos fijos **`Aceitunas`** en html. Tenemos otra forma más dinámica de hacerlo a travez de un método llamado **.addEventListener** y allí entra el concepto que vimos Event Listener, (En español Listener significa escuchar) Ahora veremos cómo podemos usar este método para evitar usar oneClick en c/u de los topping en html es funcional para proyectos sencillos pero normal mente usamos la alternativa que veremos a continuación→ **.addEventListener()** Veamos como normalmente agregamos un **Event Listener** a un elemento del DOM, lo hacemos con JS vamos a mantener la función mostrarClick() pero vamos a eliminar toda esa asociación con el elemento onClik en html. Lo primero que vamos hacer en el archivo app.js es obtener una referencia al elemento en el DOM, digamos que queremos el topping albahaca y lo vamos a obtener a través de su **id** **const albahaca = document.getElementById(`albahaca`);** una vez que tenemos el elemento y tenemos la función tenemos que crear una asociación entre el elemento y la función, cuando se va a llamar esa función y que evento debe ocurrir para que se llame esa función ese es el **Event Listener** lo que estamos creando, asociando. Para agregar un **Event Listener** escribimos el nombre de la variable que contiene ese elemento y luego escribimos el nombre del método **.addEventListener();** ese métodos toma dos argumentos. Primero el nombre del evento en este caso vamos a usar el más común “Click” (*pero debes tener en cuenta que hay muchos eventos posible que pueden ocurrir en el documento que en el navegador hay toda una lista y es conveniente investigar cada uno de ellos*). Y el segundo argumento el nombre de la función que va a manejar ese evento→ mostrarClick, es importante que la escribas sin paréntesis. También a la función la podríamos crear dentro del método pero en este caso ya está declarada en nuestro archivo app.js

Esto→ **albahaca.addEventListener(`click`, mostrarClick);** es lo que crea la asociación entre el elemento, el evento y la función a la cual se va a llamar. Vamos a Chrone abrimos console y cada vez que hacemos click en albahaca vemos un *PointerEvent, ¿*Por que vemos un *PointerEvent?* Porque esta función→ **mostrarClick** por defecto va a recibir un argumento→ **topping** pero no va a ser el topping como queríamos anteriormente va a ser un evento como tal, ese evento se va a representar como un objeto y ese es el objeto que va a recibir como argumento **mostrarClick(e);** Si cambiamos el nombre a→ **e** que es lo que normalmente usamos para representar ese objeto de evento y hacemos click en albahaca vemos en console que es un *PointerEvent* ese evento también se representa como un objeto. Todos los eventos en el DOM se representan como objetos en JS si expandimos ▼vemos que tienen sus propiedades ahí puedes ver xej las coordenadas X clientX:216 e Y

clientY:323 donde ocurrió ese evento también podemos ver el tipo de evento que es importante type:"click" algo importante en estas propiedades es que registran cual fue el elemento blanco el target de ese evento→

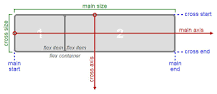
target:li#albahaca.toppings.fondo-marron asique si podemos tener acceso a ese elemento con la propiedad target podemos tener acceso a todas sus propiedades y usarlas. Xej si queremos mostrar el target de elemento→**console.log(e.target);** ahí puedes ver que el target **li** fue con el texto elemento albahaca y lo obtuvimos solamente con el objeto→**e** el objeto del evento. Pero digamos que en vez de mostrar ese target quiero mostrar solamente su texto→ **console.log(e.target.innerText );** retorna albahaca. Resumiendo: Usamos dos veces notación punto se obtiene primero el blanco de ese evento donde ocurrió y eso retorna un elemento html y luego usamos la segunda notación de punto con el método innerText para obtener su texto interno albahaca y al hacer click lo comprobamos por que se incrementa el contador. Eso es muy útil para personalizar ese texto dependiendo del elemento blanco y recuerda que todas estas propiedades se pueden usar en el programa. Pero vemos que si hacemos click en otro topping no vamos a ver nada entonces vamos hacer unos cambios para poder ver el texto correspondiente y pdemos manejar todo eso con una misma función general y eso es lo que buscamos en el código reusarlo lo más que se pueda accediendo al texto interno del texto blanco podemos mostrarlo. Vamos a modificar lo anteriormente hecho para que podamos asignarle un **Event Listener** a todos los toppings y mostrar su texto correspondiente. Si queremos seleccionar todos los toppings vamos a usar el método **.getElementsByClassName()** eso nos va a retornar una colección HTML→ **const todosLosToppings = document.getElementsByClassName(`toppings`)** una vez que tenemos la colección html podemos usar un bucle **for(of){ }** para iterar sobre todos esos elementos usamos el bucle fot(of) que también es útil para todosLostopping de la colección toppings, dentro del bucle definimos una constante de tal forma que el código correcto sería→ **for(const todosLosToppings of toppings)** y entre las llaves { } **console.log(todosLosToppings);** Y si vamos a Chrone en console podemos ver cada uno de los elementos en el bucle, esta es otra técnica muy poderosa al trabajar con los elementos del DOM. Si obtenemos un colección html podemos iterar sobre esa colección como usaríamos los bucles como xej con un array podemos usar ese bucle para iterar sobre todos esos elementos de la colección y hacer algo con ellos uno por uno para cada iteración en este caso lo estamos mostrando pero también podemos usar un **Event Listener** para c/u→ toppings.addEventListener(`click`, mostrarClick); a c/u se le va a pasar el evento el objeto del evento **(e)** que corresponde a su evento en particular y luego se va a extraer su texto **e.target.innerText** para poder personalizar donde se hizo Click. Con solo una función podemos lograr algo muy poderoso que va a funcionar con cualquiera de estos elementos gracias al objeto del evento que nos da toda esa información Si vamos a Chrone en console vamos si hacemos click en aceituna, efectivamente vemos ese topping y si hacemos lo mismo en distintos topping de la lista, sucederá lo mismo veremos los nombres de cada topping donde demos click y todo eso lo hacemos de forma dinámica en JS no hay nada fijo cien por ciento como lo teníamos anteriormente en htm. Este método→**.getElementsByClassName()** va a seleccionar todos los elementos de la clase que le pedimos, en este caso “toppings” y luego vamos a poder asignarle a cada uno los **Event Listener.** Otra alternativa que también es muy usada es que en lugar de definir una función aparte una función como tal **function mostrarClick() { }** podemos definir la función si es pequeña o precisa dentro del bucle **for** como una alternativa, se recomienda tenerla separada. Tomaríamos de la anterior función solo el objeto entre paréntesis **(e)** y definiríamos una función **=> { }** flecha como una alternativa defines la función flecha como el segundo argumento y eso es un concepto de JS intermedio, avanzado las funciones flecha que es recomendable repasar si te gustaría usar esto de esta forma es más conciso y lo vas a conseguir muy a menudo, básicamente este es el parámetro→**(e),** el evento y la flecha nos dice que está la relación entre los parámetrosy el cuerpo de la función que en este caso es console.log()↓ y se obtiene la misma funcionalidad **todosLosToppings.addEventListener(`click`, (e) => { console.log(e.target.innerText) });** Para manejar otros eventos la única deferencia seria personalizar el nombre del evento en vez de **click** puedes colocar el nombre de otro evento y esa función va a manejar el cuerpo de ese evento y en el cuerpo de la función decides cómo manejarlo, vamos a aplicar todos estos conceptos al momento de crear nuestros proyectos.

\*/

/\*

**Proyecto Colores Aleatorios:**

Comenzaremos ahora a crear proyectos interactivos trabajando con HTML, CSS y JS para manipular el DOM. En nuestro primer proyecto vamos a crear una aplicación sencilla en la que vamos a tener un código hexadecimal para un color. Recuerda que en Css para un color tenemos dos opciones hexadecimal o RGB son dos modelos diferentes, en el modelo hexadecimal tenemos 16 posibles opciones 0-9 A-F para cada carácter y en total el color tiene seis caracteres. En nuestra aplicación si hacemos click en el botón vamos a cambiar el color del cuerpo de la página web a un color hexadecimal aleatorio. Empecemos, abrimos VSC creamos una carpeta color-aleatorio que contiene una carpeta imágenes y los archivos index.html, styles.css y app.js con primero abrimos index.html usamos la abreviación M→**!** para obtener esa estructura inicial, cambiamos el idioma, dentro de **head** personalizamos el título, conectamos con la hoja de estilo con **estylesheet** y también vamos agregarle un **favicon** para que tenga un icono en la pestaña. Ya tenemos el contenido de head listo, ahora para el cuerpo de la aplicación, para el **body** vamos a tener una estructura sencilla, al cuerpo del <body aquí> le vamos asignar una clase centrar-flex, vamos a usar flexbox para poder centrar todo el contenido del cuerpo dentro de body colocamos un **div** principal que también podría ser **main** ese **div** va a tener un **id** = “contenedor” y una **class** = “centrar-flex” una clase que nos va a permitir centrar el contenido completo si vamos a reusar ese estilo solo para mostrar alternativas que puedes usar y luego tenemos una etiqueta→**p** dentro incluimos un **id** porque vamos a usar ese párrafo para mostrar el color hexadecimal #2B807G que actualmente se está asignando al fondo a continuación agregamos un botón el cual va a decir cambiar color y le asignamos un **id** botón-color y luego antes de cerrar la etiqueta **body** vamos agregar etiqueta, script un elemento **script** que nos va a permitir usar el código que definamos en nuestro archivo app.js el cual se encuentra en la misma carpeta que se aloja index.html y styls.css por eso en código escribimos solo su nombre si estuviera en otra ubicación tendíamos que escribir su path correspondiente. **CSS** Ahora vamos a trabajar con styls.css para asignar los estilos, iniciamos con los estilos generales: Con el selector universal→**el asterisco** reiniciamos el **margin** y el **padding**  en cero y vamos asignar por defecto el tipo de letra **verdana** y de respaldo **sans-serif** ahora para el body vamos asignar una altura→height del cien por cien→**vh** *Nota al pie*: **vh** significa viewport height (altura del dispositivo). También funciona como porcentaje, especificar 10vh es equivalente a ocupar 10% de la altura visible de la pantalla o 50vh significa 50%, lo que cubrirá el 50% de la altura de la pantalla. El color de fondo→**background-color** va a ser el código hexadecimal #2B807B que va a ser consistente con el prestablecido en el botón en html, seguidamente vamos alinear el texto ubicado en el body en el centro→**text-align: center;** Recuerda que en html tenemos una clase→.centrar-flex en esa clase vamos a definir que el elemento va a ser flex→*Nota al pie:* La propiedad de display flex, ayuda a colocar las cajas como si fueran celdas de tablas, con un comportamiento predecible y adaptable, flex soluciona problemas de control del diseño y se adapta fácilmente a los diferentes dispositivos



Y va a tener un justify-content: center; *Nota al pie:* La propiedad CSS justify-content define cómo el navegador distribuye el espacio entre y alrededor de los elementos de contenido a lo largo del eje principal de un contenedor flexible y el eje en línea de un contenedor de cuadrícula.

Luego va a tener un align-items center; *Nota al pie:* centra los elementos hijo de un contendor (las cajas :d) y text-align centra el contenido que está dentro de la cajas en concreto el texto, esta propiedad se declara dentro del elemento padre y todos los elementos hijos lo heredan. Ahora vamos a definir los estilos para el #contenedor, este es el contenedor blanco que va a contener el código del color y el botón. Primero vamos asignarle una **altura→height:** de 220 px, su ancho→**width** va hacer de 450 px, el color de fondo fondo→**background-color** va a ser blanco, su borde→**border** va a ser de 5 px sólido y negro. También queremos que los elementos se distribuyan en varias líneas si lo necesitan asique vamos agregar **flex-wrap: wrap** *Nota al pie:*→La propiedad flex-wrap de CSS especifica si los elementos "hijos" son obligados a permanecer en una misma línea o pueden fluir en varias líneas. Si la cobertura (wrap) está permitida, esta propiedad también te permite controlar la dirección en la cual serán apilados los elementos. Y queremos que estén uno debajo del otro verticalmente asique agregamos **flex-direction: column** *Nota al pie:*→Flex-direction determina la dirección en que se ordenarán los elementos en el espacio. Esto puede ser de izquierda a derecha (row), de derecha a izquierda (row-reverse), de arriba a abajo (column) o de abajo a arriba (column-reverse) ( flex-flow flex-direction ). Ejecutamos la aplicación en VSC y podemos apreciar el cambio, ahora tenemos un contenedor de fondo blanco y los elementos están alineados verticalmente. Ahora vamos a definir un estilo para el texto, ese texto está definido en html en el elemento **p** por el id="color">#2B807B vamos asignarle un estilo al código. Vamos asignar un tamaño de letra→font-size: 70px; el peso de la letra font-weight: bold negrita, ahora vamos a los estilo del botón, el tamaño de la letra→font-size: 30px; color blanco y bordes redondeados→ border-right: 20px; y el color de fondo→ background-color: #008CBA; y para el margen margin: 15px; y un padding: 10px 30px; *Nota al pie:*→La propiedad padding es la que crea el espacio o área alrededor del contenido de un elemento. Consiste en el rellenosuperior derecho, inferior izquierdo. Y un borde

border: 3px solid y el mismo código hexadecimal #008CBA para el color. Para que el usuario no pueda seleccionar el texto que está dentro del botón usamos la propiedad→user-select: none; también agregamos la propiedad→cursor: pointer; el cursor sobre el botón se vuelve una manito. Por ultimo asignarle un efecto, una transición para cambiar su estilo cuando el cursor este sobre ese botón vamos asignarle una duración de tiempo→ transition-duration: 0.4s; 0.4 segundos y para terminar colocamos el efecto→**button:hover** cuando el cursor se coloque sobre el botón sin hacer click en el vamos a cambiar el color de fondo a #05455A con esto logramos una transición sutil de 0.4 segundos en el que el fondo del botón cambia de color y se mantiene el borde más claro. Con esto ya tenemos todos los estilos principales y la estructura, ahora podemos comenzar a enfocarnos en la funcionalidad del archivo app.js para crear esa interactividad.

**JavaScipt**

Lo primero que debemos hacer es seleccionar los elementos del DOM eso lo debemos hacer con los métodos que aprendidos con anterioridad en el curso que ante cualquier duda puedes consultar, es aconsejable releer toda esa primer parte. ¿Cuáles de esos elementos en realidad que necesitamos en JS? Bueno nuestra meta es lograr que el botón sea interactivo asique debemos asignarle un→**EventListener** a ese botón y para eso necesitamos obtener una referencia en el DOM y también vamos a seleccionar este elemento→**p** el párrafo donde mostramos el color→#2B807B para poder actualizarlo cuando ágamos click en el botón vamos a cambiar ese código dinámicamente que se genera aleatoriamente y se va a cambiar el color de fondo del cuerpo asique con un evento de click van a ocurrir bastantes cosas. Para seleccionar el boton vamos a usar **.querySelector(“button”)**; eso nos va a retornar el primer elemento con la etiqueta “button” y también vamos a obtener el **id** “color” de este elemento→**p** color→#2B807B en este caso vamos a usar **.getElementById();** Lo primero que necesitamos en verdad es definir una función que maneje el proceso de generar un color hexadecimal aleatorio. Vamos a crear una función y en su cuerpo vamos a definir una cadena de caracteres con todos los dígitos posibles para ese color Hexadecimal, xej. en el sistema decimal podemos usar los números entre cero y nueve pero en el sistema hexadecimal los caracteres son mixtos números entre cero y nueve y las letras entre A y F, en este caso vamos a implementarlo en una cadena de caracteres con todas esas posibles opciones `0123456789ABCDEF` 16 opciones es lo que realmente necesitamos. Para cada digito vamos a seleccionar cada uno de esos caracteres, hay muchas formas de implementarlo en este caso para fines de simplificar el ejemplo en el proyecto se escogió ser explícito y crear esta cadena de caracteres. También vamos a definir una variable para guardar el color que vamos generando el color Hexadecimal. En este caso tenemos dos variables con el mismo nombre→color pero const es una variable global y let es una variable local para esa función, no va haber ningún conflicto si tienen el mismo nombre pero para evitar cualquier posible confusión vamos a definir la variable local let colorEx. Como un color hexadecimal tiene seis dígitos se pueden abreviar en tres si se repiten de a pares pero vamos asumir que el color va a tener seis posibles dígitos porque no sabemos si son iguales uno de los pares, entonces vamos a tener un bucle **for** con seis iteraciones para generar un índice aleatorio con esos seis dígitos dentro de la cadena de caracteres que nos va a permitir seleccionar de forma aleatoria uno de estos dígitos para agregarlo al color hexadecimal, para eso vamos a usar el método **.random()** que nos va a retornar un número seudo aleatorio entre cero y uno vamos a multiplicar ese número por 16 ¿por que multiplicamos por 16? Porque la variable contiene 16 caracteres y en este caso **Math.random** no va a incluir ese valor 1 por que no es una opción no está incluido el 1 en sí asique el resultado nunca va a ser 16 el mayor valor que vamos a obtener cuando usemos **.floor** va a ser 15 que es el último índice vamos a ir del indice 0 a 15 `0123456789ABCDEF` y vamos a escoger un indice aleatorio y ese es el indise que luego vamos a usar para agregar ese digito al color. Luego como queremos tener un entero vamos a usar el método **.floor** que nos va a permitir redondear el numero para obtener el mayor entero que es menor o igual a este valor por ejemplo si retona 14.25 **Math.floor()** nos va a entregar 14 y ese es el índice que vamos a usar. *Nota al pie:* Tanto los metodos **.random** y **.floor** van precedida pot Math ¿que es→Math? Es un objeto incorporado que tiene propiedades y métodos para constantes y funciones matemáticas. No es un objeto de función. Math funciona con el tipo Number. Proporciona constantes y métodos estáticos para operaciones trigonométricas, logarítmicas y otras funciones matemáticas comunes.

Y luego en el cuerpo del bucle **for** ahí en el color vamos a tomar el valor actual del color los dígitos que ya hemos agregado y vamos agregar un digito nuevo ¿Cuál digito va hacer? Un índice de esa secuencia un digito que seleccionamos de forma aleatoria. Esto→**colorEx += digitos[indiceAleatorio]** lo que está haciendo es expandir esa cadena de caracteres con el color hexadecimal y luego aquí fuera del bucle **for** retornamos el color hexadecimal y con esto ya tenemos la función definida. Pero primero para poder ejecutar la función tenemos que decirle a nuestra aplicación que queremos asociar nuestra función con un click. Para hacer esa conexión vamos a usar el método→**.addEventListener()** y cual de esos eventos queremos manejar, el evento de click y como lo queremos manejar porque eso es muy importante como queremos manejar ese evento. Necesitamos realizar distintos pasos, no solamente necesitamos llamar esta función **generarColorExAleatorio()** por que la función solo va a generar un color aleatorio, no va actualizar nada en el DOM ningún color, ningún texto y ningún elemento, queremos llamar a la función obtener ese color retornado y luego hacer las actualizaciones necesarias. Para eso vamos a definir otra función si puedes definir la función directamente aquí pasándola como un segundo argumento o también puedes definirla de forma independiente en el código, son todas alternativas diferentes que se pueden realizar y es bueno saberlo. Lo primero antes de hacer cualquier cambio es obtener el color aleatorio, vamos a llamar a la función generarColorExAleatorio() y vamos a obtener ese color. ¿Qué es lo que queremos hacer con ese color?

El color va aparecer en el texto que vemos sobre el botón asique vamos actualizar el contenido del texto de ese elemento **p** en este caso si vamos a trabajar con el elemento del DOM este color que obtuvimos con el metodo **.getElementById(`color`);** a partir de su **id** este es el elemento **p** en el párrafo que mostramos su color actual, vamos actualizar su→**textContent** también se pueden utilizar distintas opciones para actualizar pero en este caso usamos textContent que va a funcionar perfectamente en este proyecto y vamos asignarle directamente colorAleatorio que ya es una cadena de caracteres y luego ¿Cómo podemos hacer para actualizar la etiqueta body? De esta forma→ **document.body** y luego que ya accedemos al body podemos acceder a sus propiedades como por ejemplo style, recuerda style cuando asignábamos estilos en línea en los ejemplos anteriores exactamente igual podemos actualizar su estilo y debemos especificar cuál de esas propiedades debemos actualizar en este caso es el color de fondo→backgroundColor ¿y cual valor queremos asignar? El color aleatorio que obtuvimos a partir de esta función→**generarColorExAleatorio()** y esas son las dos funciones. Es importante analizar todas las necesidades del proyecto para tener una idea de que es lo que necesitas en el código por que inicialmente podríamos a ver pensado que solo necesitábamos esta funcion→**generarColorExAleatorio()** pero también necesitamos actualizar estos valores el texto y el color de fondo propiedades en el DOM asique eso es muy importante para saber que referencias necesitas tener en el archivo app.js. Ahora ya para finalizar este proceso con las herramientas de Chrone vamos a comprobar fehacientemente que el color aleatorio hexadecimal y su texto lo vemos dentro del contenedor es el color que se presenta como como tal de fondo en la pantalla. Ya en Chrone en console vamos a body y vemos ahí ese estilo de fondo de pantalla puedes ver que automáticamente lo convierte a RGB pero si haces click sobre el color y luego haces click en las flechitas que están sobre la derecha podrás ver el código hexadecimal equivalente al código que muestra la aplicación. Y así ya completamos nuestra aplicación interactiva para manipular el DOM

\*/

**Proyecto Colores RGB**

En este proyecto vas aprender como extraer el valor de un elemento input en el DOM y trabajar con ese valor para personalizar un color, este es un proyecto similar a la aplicación hexadecimal que anteriormente hicimos pero en este caso no va hacer aleatorio, el usuario va a escoger distintas combinaciones del modelo RGB para generar un color y ese color es el que se va asignar como el color de fondo va ser similar el concepto al anterior proyecto pero muy distinto en la implementación y se van aprender cosas nuevas sobre JavaeScript para el DOM. Empecemos, abrimos VSC creamos una carpeta color-RGB que contiene una carpeta imágenes y los archivos index.html, styles.css y app.js primero abrimos index.html usamos la abreviación M→**!** para obtener esa estructura inicial, cambiamos el idioma, dentro de **head** personalizamos el título, conectamos con la hoja de estilo con **estylesheet** y también vamos agregarle un **favicon** para que tenga un icono en la pestaña. Ya tenemos el contenido de head listo, ahora para el cuerpo de la aplicación, para el **body** vamos a tener una estructura sencilla, primero creamos un **div** con la clase “contenedor-principal”, luego un **div** con la clase “contenedor-interno” y otro **div** con la clase “color” para cada uno de los colores. En el modelo RGB un color se define en base a un valor entre 0 y 255 de rojo, verde y azul *Nota al pie:* RGB, un acrónimo de los colores que utiliza: rojo, verde y azul (red-green-blue), son colores que se crean mezclando cantidades variables de luz en una pantalla y se consideran colores aditivos; a medida que se necesitan, se iluminan a diferentes intensidades para crear color. Para cada uno de esos colores vamos a tener una **etiqueta** y un elemento **input** donde vamos a poder escoger el valor para ese color. Por ejemplo para rojo vamos a tener un tipo de elemento rango porque vamos a poder seleccionar un valor en un rango. Ese rango va a ir desde 0 hasta 255 y el valor inicial va a ser 23 y el paso para incrementar de un valor a otro va a ser 1 vemos el código **<input id="rojo" type="range" min="0" max="255" value="23" step="1">** ese es el campo de **input.** Cada input debería tener una etiqueta asociada también. Entonces vamos asociarla con ese input rojo y le asignamos la letra R de RGB **<label for=">rojo" rojo (R) </label**> y luego también tenemos el párrafo en donde vamos a mostrar el valor actual del color. En este elemento→**<p id="texto-rojo"></p>** vamos a mostrar el valor actual de ese indicador. Eso mismo, esa misma estructura la vamos a repetir para los dos colores restante, verde y azul cambiando solo el valor inicial para verde es 41 y para el azul 56 esto va a ser importante porque vamos a manejar el contenido con JS. Y si vamos a Chrone podemos observar como se ve la aplicación donde tenemos tres indicadores de rango que van a tener estilos diferentes dependiendo del navegador y del sistema operativo y tenemos sus etiquetas correspondientes y por ultimo vamos a completar body con una etiqueta→**script src="app.js">** para asocia con nuestro archivo app.js con lo hecho ya tenemos esa estructura inicial.

**CSS** Estilos generales:

Con el selector universal→el asterisco reiniciamos el **margin** y el **padding** en cero ahora para el body

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

text-align: center;

background-color:rgb(23,51,46)

Para los contenedores vamos asignar un estilo, nota que estos valores→rgb(23,51,46) son los valores iniciales que asignamos en los indicadores en index.html que corresponden al color de fondo que le vamos asignar a body así lo que mostramos es consistente con el color que también estamos asignando, eso es en cuanto a body ahora vamos hablar sobre los contenedores. Tenemos el estilo para el contenedor principal va a rener un ancho de 50 por ciento de la pantalla, el ancho máximo va a ser de 600 px para que no se estire demasiado y la altura minima va a ser de 350 px va a ser flex, flex-wrap: wra, lo vamos a centrar→ justify-content: center; align-items: center; el color de fondo va a ser blanco y va a tener un radio de 50 px Y si vamos a Chrone podemos observar como se ve la aplicación donde ahora tenemos tres indicadores centrados y son parte del contenedor principal que ahora es blanco con los bordes redondeados. Ahora vamos asignarle un estilo a los distintos colores. Esta clase color corresponde al **div** que contiene cada uno de los indicadores con sus etiquetas cada uno de los elementos **input** donde vamos a indicar cada uno de los valores para rojo, verde o azul RGB vamos asignarle otro tipo de letra→font-family: Georgia, 'Times New Roman', Times, serif; queremos que todo este alineado verticalmente y también en lugar de centrarlo vamos asignarle el valor →**space-around**  para que añada espacios en los distintos indicadores y lo distribuya pero si lo vamos alinear el centro en el eje perpenticular horizontalmente y luego para las etiquetas y los párrafos vamos asignarle un margen de 20px puedes escribirlos en dos líneas o en una sola separado por una coma. Ya tenemos nuestro estilo lo único que nos falta es inmplementaar esa funcionalidad interactiva porque queremos mostrar el valor actual de esos elementos **input** y queremos actualizar el color de fondo que tenemos de body dinámicamente a medida que desplazamos y cambiemos estos valores vamos a generar un color nuevo.

**JavaScritp**

Primero necesitamos referencias a los elementos del DOM que queremos modificar o de las cuales queremos obtener esa información. Vamos a declar una contante porque normalmente cuando trabajamos con elementos del DOM a esa varible no la queremos modificar, esa información vamos a obtenerla por el **id** empleando el método→**.getElementById();** para rojo, verde y azul. Y también vamos a obtener esos elementos que corresponden al texto donde vamos a mostrar los valores de los **i**nput que en estos momentos están vacíos pero lo vamos a remplazar de forma dinámicacuando se ejecute este código. Primero necesitamos una referencia a esos elementos a los elementos de párrafo nuevamente usamos el método→**.getElementById();** textoRojo, textoVerde y textoAzul en estos casos no nos referimos al color del texto en si, sino el color RGB, una vez escrito el código equivalente a lo mencionado ya tenemos todas las referencias que corresponden a los **id** en el archivo index.html rojo, verde y azul son los input y luego si le agregamos el texto adelante obtenemos **p** su párrafo correspondiente. Ahora vamos a definir unas variables que nos van a ser útiles para poder especificar el color RGB cada valor debe estar entre 0 y 255, inicialmente el valor del color rojo debería ser el valor por defecto del indicador que muestra CHrone asique vamos a tomar ese valor inicial con la propiedad→**value** de igual forma podemos obtener el input verde y azul esto nos va a dar el valor de esos elementos del DOM el elemento input para el rango eso es muy útil y nos va a permitir escoger o remplazar ese valor inicialmente con los valores que asignamos por defecto en html vamos a tomar este valor inicial 23, 41 y 56 lo vamos a extraer en JS con estas propiedades let rojo = inputRojo.value;

let verde = inpuyVerde.value; let azul = inputAzul.value; y luego lo vamos a usar para generar el ese texto y remplazarlo y mostrárselo al usuario y vamos a remplazar el inerTex de ese elemento textoRojo con el valor de rojo de igual forma para textoVerde y textoAzul en las variables que definimos anteriormente pero esa no es la única vez que vamos a usar esas variables por eso es que las definimos como variables en nuestro código sino usaríamos los valores directos. Primero extraemos el valor de input y lo asignamos a variables y luego estamos usando esos valores para remplazar el texto de cada uno de esos párrafos eso es lo que estamos haciendo actualizar el texto de los párrafos. Actualizamos en Chrone la app nos muestra el texto de los párrafos 23, 41 y 56 son los valores iniciales que previamente indicamos a los input en html. Ahora lo que debemos manejar es como vamos actualizar el color una vez que se cambian los indicadores porque hasta el momento no ocurre nada si cambiamos el valor. Vamos a implementar esa funcionalidad para eso vamos a definir una función que denominaremos actualizarColor() la función va a tomar tres parámetros rojo, verde y azul y vamos a crear un color RGB como una cadena de caracteres en JavaScript recuerda que tenemos plantillas literales son muy útiles para poder remplazar valores de variables directamente dentro de las cadenas de caracteres eso nos puede permitir crear un color de esta forma. En Css para ese valor normalmente escribimos rgb() y dentro de los paréntesis los valores, vamos a escribir exactamente eso pero vamos a remplazar esos valores con los valores de las variables. Con el signo **$** y las llaves **{ }** vamos a remplazar el valor de esas variables "rgb(${rojo}, ${verde}, ${azul})" ese es el valor y ahora luego que tenemos ese color RGB lo que haríamos seria actualizar el color de fondo del cuerpo. Vemos el código posteriormente la explicación: **document.body.style.backgroundColor = colorRGB;** obtenemos la etiqueta body, luego su estilo y específicamente el color del fondo y lo vamos actualizar con ese color que vamos a generar. Esa cadena de caracteres se la vamos asignar y eso va actualizar el **body** en el DOM. Pero solo con definir esa función no estamos cambiando absolutamente nada porque no estamos escuchando ese evento de cambio. Necesitamos asignarle un→**EventListener** para cada uno de esos elementos input. Ese **EventListener** va a

escuchar un evento llamado→change que en español significa (cambio) en lugar de manejar el evento que eventualmente manejábamos como click ahora vamos a manejar change que va hacer un cambio en este elemento de **inputRojo** y ahora vamos a ver algo interesante que es el objeto del evento eso va hacer muy importante. Vamos a tomar el objeto→(**e**) como parámetro en la función flecha. Nota al pie: En la función flecha que ya es un concepto más intermedio/avanzado en JS son muy similares a las funciones pero primero toma su parámetro en este caso el objeto (e) y la flechita => separa sus parámetros del cuerpo **{** la secuencia de instrucciones que se van a ejecutar **}** de la función, lo que está entre las llaves. Como tenemos acceso al objeto del evento tenemos acceso a información útil donde ocurrió ese evento. Una forma de obtener ese valor es usar el objeto (**e**) del evento, obtener el **target** que es el elemento blanco y luego extraer su valor y así podemos asignarle a una variable→  **rojo = e.target.value;**  va a ser esa variable que definimos como **let** anteriormente. Podemos usar esa variable o podemos definir una variable local nueva, en este caso va a ser equivalente, vamos a mantener esta variable como una variable local y veamos que ocurre. Vamos asignarle ese valor **rojo** como el texto, podemos usar textContent o innerText en este caso va a ser equivalente como innerText de texto rojo de ese párrafo y luego vamos a llamar a la función actualizarColor() vamos a pasar en este caso como parámetro rojo, verde y azul. Estas variables verde y azul son las variables globales. Nota al pie: La variable en el cuerpo de la función flecha que definimos como local **let rojo** se recomienda mantenerla como global por si la necesitamos usar más adelante poder tener una forma de obtener esos valores de forma global en nuestro programa como estamos llamando a la función actualizarColor() con las variables de parámetros es mejor tenerlas actualizadas. Y ahora podemos actualizar el color **rojo** y así lo podemos apreciar en las herramienta de Chrone seguida mente vamos a implementar el código para verde y azul la sintaxis es muy similar asique solamente tenemos que cambiar los colores o valores de manera tal que estamos actualizando el DOM. Y así estamos cambiando el color de fondo en base a un color que nosotros seleccionamos en html. Otra opción en lugar de usar el evento (e) es usar directamente la variable global inputRojo, en el ejemplo de código lo vemos:

inputRojo.addEventListener('change', () => {

rojo = **intputRojo**.value;

textoRojo.innerText = rojo;

actualizarColor(rojo, verde, azul);

y comprobamos que el funcionamiento del programa no se ve afectado por que son variables globales. Esto es importante por si no tienes una referencia específica a ese elemento del DOM, si no sabes exactamente de dónde quieres extraer ese valor puedes optener ese elmento donde se hizo click o donde ocurrió el evento y extraer su valor correspondiente en el programa. Con esto ya sabes cómo obtener elementos input en html y como actualizar el estilo de otros elementos en base a esos valores.

**Proyecto Citas Aleatorias:**