Frontend Alura

Flexbox

1. Introduccion al Flexbox y haciendo el encabezado.

Vamos a crear nuestro mini Alura.

Por defecto, tenemos la pagina hecha, pero sin los posicionamientos.

- Arreglando el header.

Se recomienda hacer un archivo css por separado para tener mas organizados los estilos y el posicionamiento en archivos distintos.

Adentro del header tenemos un contenedor con dos objetos, el logo y el nav.

En primer lugar vemos que el logo y la lista del nav están en bloque. Nosotros queremos que esten en linea.

Para que los dos objetos queden alineados verticalmente en el medio, vamos a utilizar en cada uno **vertical-align:middle;** . (esto es en los inline-block).

Para que los dos objetos queden separados horizontalmente (logo a la izq y nav a la der en este caso), podriamos usar margin o porcentajes con cada uno, pero aquí deberiamos adivinar la distancia. Ademas, con esto tenemos la desventaja de que si queremos reducir o ver en otra resolución, se romperia la visualizacion.

Podriamos usar float: left o right, pero esto complicaría el trabajo tambien. Asi que vamos a utilizar flexbox.

Con el flexbox, lo que hacemos es determinar el comportamiento de los objetos dentro de un contenedor (en este caso va a ser el contenedor dentro de header que dentro tiene los dos objetos a tratar). Vamos a hacer que el contenedor se comporte de manera flex con **display: flex**. Para centrarlos verticalmente, **align-items: center;** (siempre en el contenedor padre) y para separar los dos objetos para que quede uno a la izquierda y otro a la derecha, usamos **justify-content:space-between**.

El nav esta en forma de columna. Asi que lo podemos hacer flex tambien para modificar el comportamiento de los objetos en su interior. No hace falta alinear el nav en el centro porque ya lo alineamos antes en el contenedor padre del logo y nav. Y como en el nav no es tan ancho, no hace falta un space-between para separarlos. La separacion que tienen, viene de un margin puesto en el archivo style.css.

NOTA:

Distintos valores que le podemos agregar a **justify-content** ademas de space-between.

flex-end: Poner todo el espacio a la izquierda, lanzando el contenido a la derecha.

flex-start: Poner todo el espacio a la derecha, lanzando el contenido a la izquierda.(Esta por defecto)

center: Pone todo el espacio a la izquierda y a la derecha, lanzando el contenido al centro.

space-between: Pone el espacio entre los elementos.

space-around: Pone el espacio al rededor de los elementos.

1. Footer y controlando mejor los elementos

Acordemosnos de utilizar el contenedor padre para controlar a los hijos con flex. Deberiamos separar y alinear cada uno de los elementos.

Si hacemos un space-between entre los logos, veremos que no los separa, ya el navegador no toma como que haya una anchura suficiente como para acomodarlos. Esto es porque la lista de logos esta ocupando el ancho igual al espacio que esta utilizando (como inline-block). Asi que en esta situacion, podemos asignarle una anchura (como 70% por ejemplo) para que el space-between pueda actuar. Ademas, como detalle podemos poner un margin a su derecha (como 5% por ejemplo) para separarlo del campo form.

Otro detalle es que nosotros queremos que la lista de logos esté mas adentro y no alineado con los otros elementos de la pagina. Esto lo hacemos con space-around para separar cada objeto en su alrededor en lugar de space-between que los separa entre los mismos objetos entre si. Asi que deberiamos cambiarlo.

Tambien podemos ver que el formulario de la derecha deberia tener el boton a su lado y no abajo de el. Entonces deberiamos usar flex en su contenedor padre para manejarlos.

Ahora vamos a corregir el mapa de cursos. Tenemos que separar los elementos como columnas por cada tema. Eso se hace con **flex-direction: column;** Para que cada elemento dentro del contenedor se pongan uno abajo del otro (por defecto está el flex-direction:row). Pero ahora tenemos que poner una al lado de la otro cada columna. Esto lo podemos hacer dandole al contenedor un alto (como 250px por ejemplo). Pero esto no es suficiente. Habria que utilizar el **flex-wrap: wrap** para que los objetos se posicionen automaticamente al lado del otro respetando ese alto. Es decir, que el contenido al llegar a completar ese alto, salte a la columna siguiente.

1. Grid principal y limitaciones de Flexbox

Vamos a modificar la lista de Nuestros Cursos

Vamos a agarrar el ul que es el contenedor de la lista para que sea flex. Veremos que los elementos se superpondran, ya que no tienen un tamaño definido. En este caso, los links tienen posicion absoluta. Esto romperá el posicionamiento. Entonces a estos tambien le cambiaremos a flex. Deberiamosa a estos links, agregarles un ancho (23% aprox) y el wrap para que quiebre.

NOTA: Tener cuidado con space-between y space-around, ya que si se hace un grid de varios elementos, como hicimos anteriormente. Estos, nos podran dar problemas en la ultima fila de los elementos, ya que si este contiene solo un par de elementos en la ultima fila, el valor de la propiedad los separará mucho en contra de nuestra voluntad.

1. Arreglando los elementos con Flex para mobile

Primero pondremos al navegador en modo mobile para visualizarlo.

En el css, para que los cambios sean de hasta en pantallas de un maximo de 768px utilizamos utilizamos la @media

Lo primero que vamos a modificar va a ser el header. Queremos que el menu este en una columna con el texto centrado.

Primero cambiamos del contenedor dentro de header la flex-direction a column. Asi el logo y el nav estan uno arriba del otro.

Ahora lo hacemos con el nav. Si cambiamos la flex-direction a column, lograremos que sea columna, pero mantiene el align-items. Este align-items, en row centraba los objetos de manera vertical (espaciaba arriba y abajo). Mientras que en column, centra de manera horizontal (espacia derecha e izquierda). Por ende, nosotros lo pondremos de manera **initial**. Tambien nos faltaria alinear solo el texto del nav al centro, esto lo haremos con **text-align:center;**.

Vamos con el Nuestros cursos. Primero, tambien los vamos a pasar a columnas. Y ademas, a los links le vamos aponer el ancho de 100%.

Tambien vamos amodificar el mapa de cursos, que ya esta en formato columna. Aquí solo quitamos la limitacion del alto, entonces pondremos **height:auto;** y quitarle el wrap con **flex-wrap:nowrap;**.

Al footer le vamos a poner column para que la lista de logos y el form esten uno arriba del otro. Y darles un ancho del 100% y darle align-items:center a los logos. Al form le podemos hacer justify-contect:center, para que se haga el espacio en los costados y los elementos de este se muevan al centro.

Para la version movil, queremos que el link de “Nuestra APP” quede primero. Esto lo hacemos agregandole a su respectiva clase el **order:-1;**. Asi pasa a ser el primero dentro del contenedor.

Algo que nos faltó es acomodar la section que tiene el video de youtube. Para esto, tambien haremos flex su contenedor. Y centrar el texto de “ventajas”. Tambien ese contenedor de ventajas lo haremos flex y luego ponerlo en formato columna para que el texto y el boton esten uno abajo del otro.

Al video, lo haremos mas grande con **flex-grow:2;**. Esta propiedad va a agarrar el espacio que sobre de lado a lado y lo va a dividir en la cantidad de secciones que indiquemos (en este caso 2). Si hacemos tambien flex-grow:2; en la lista de ventajas, pondra 2 partes del espacio y lo pondrá entre los elementos.

El responsive de esta section de video.

Mientras que el flex-grow hace que el que tenga mayor numero, va a crecer y ocupará ese numero mas veces que los otros de forma responsiva, el **flex-shrink** hace lo contrario. Es decir que el objeto que tenga esa propiedad, se va a encoger el numero dado. Si le damos de valor 0, ese elemento no se encogerá nunca. (el flex-shrink por defecto es 0).

Y haremos column los elementos de la section entera. Asi el video y las ventajas estarán una arriba de la otra.

NOTA:

Si tenemos varios items y queremos que un solo item ocupe el resto del espacio que queda dentro del contenedor, utilizaremos en el objeto flex-grow:1.

Ejemplo:

Elemento 1: 200 px.

Elemento 2: 200 px.

Espacio vacío que restó del flex container: 600 px.

Total: 1000 px.

Se ponemos flex-grow: 1 en el primer elemento, este pasa a tener 800 px de ancho, es decir:

Espacio vacío + Elemento 1: 800 px.

Y el segundo elemento sigue teniendo 200 px de ancho.

Ahora, si ponemos flex-grow: 1 en ambos elementos, tendrian 500 px cada uno. Ya que como ambos elementos tienen flex-grow: 1, la suma de flex-grow que tenemos dará 1 + 1 = 2.

El navegador toma este espacio vacío y lo divide por el número de flex-grow que tenemos: 600 px : 2 = 300 px.

Ahora distribuye ese espacio a cada uno de los elementos que hemos colocado flex-grow.

Elemento 1: 300 px + 200 px = 500 px.

Elemento 2: 300 px + 200 px = 500 px.

NOTA:

Ejemplo de flex-shrink.

Si hay 4 elementos y al primero le ponemos flex-shrink:2, los demas tendran por defecto 1. Entonces la cuenta será 2+1+1+1=5.

Entonces si queremos disminuir la pantalla 200 px del tamaño total, el navegador tomara este valor y lo divide por el resultado de la cuenta anterior. Queda 200 px / 5 = 40 px.

Así el navegador toma 2 partes del primer elemento como 200 px - 40px\*2 = 120 px. Y toma 1 parte para cada uno de los otros, es decir 200 px - 40px\*1 = 160 px.

NOTA:

Las propiedades flex para un contenedor padre, pueden ser display:flex, justify-content, align-items, flex-direction, y modificaran instantaneamente a los objetos dentro de ese contenedor. Mientras que order, flex-grow, flex-shrink, van a ser solo para los objetos y se deberán poner a cada uno si los queremos modificar.

NOTA:

Otro mas para hablar, el flex-basis. Este lo que hace es asignar la medida base del objeto (puede ser en px o lo que sea), pero si tiene que ocupar menos de esa cantidad base asignada, se reducirá proporcionalmente. Basicamente es, si puede ocupar ese tamaño asignado, lo ocupa, sino se reduce. Pero como maximo a donde puede llegar es la medida basis asignada.

flex-basis sirve para definir un ancho para el objeto que este dentro de un contenedor con row container o row. Y en cada uno el width se invierte de eje.

Tambien otra nomenclatura para el flex de un objeto seria:

flex: 0 1 30% (grow shrink basis).

CSS Grid: Simplificando layouts

1. Definiendo el diseño base

Tenemos el html el cual solo tiene etiquetas y clases que vamos a utilizar, pero ningun contenido.

Vamos a ir al archivo style para comenzar. Para que el body de clase app se comporte como grid, le asignamos **display:grid**. Con esto, debemos asignar el template, las columnas y filas.

El template con **grid-template-areas:** y con strings declaramos las partes de este en orden.

**“encabezado”**

**”contenido”**

**”rodapie”;**

Columnas con **grid-template-columns:auto;** para que se ajuste al total de la pantalla.

Lineas con **grid-template-rows:50px 100vh auto;** puse propiedades para asignarles a cada una de las strings del template.

Ahora que estan definidas las templates areas, ahora hay que decirle cual es cual. Esto va a ser asignando **grid-area:nombreTemplate;** a cada clase correspondientes.

1. Diseñando la seccion de destacados

Ya con el nuevo archivo destacados.css creado y linkeado al index.html, vamos a hacer grid la section destacados. Ahora podemos definir las columnas y filas. Nosotros queremos que haya 4 columnas en el ancho del contenedor. Entonces tenemos que hacer **grid-template-column: 25% 25% 25% 25%;** ya que son las 4 en el 100% del ancho. Y con las filas, queremos que haya 3 por el alto del contenedor, eso seria **grid-template-column: 33.33% 33.33% 33.33%;** por las 3 filas en el 100% de alto.

Ahora queremos cambiar el tamaño de la seccion de destaques para que ocupe el 100% de la pantalla. Con **height: calc(110vh -50px);**.

Para que asignemos el tamaño que corresponda a cada uno, como a fornite, whatsapp, pubg etc. Hacemos lo siguiente. Primero nos vamos a enfocar en los destaques principal. Los asignamos en css y le damos el background correspondiente. A fornite por ejemplo, le vamos a decir en que columna empieza y termina. Esto con **grid-column-start:1;** y termina en **grid-column-end:4;**. La 4 la ponemos porque es inclusiva. Tambien lo hacemos con las filas con **grid-row-start:1;** y **grid-row-end:3;**.

Ademas, nos faltaria poner un **background-size:cover;** para que escale la imagen al mayor tamaño posible sin estirarla. Si las proporciones de la imagen difieren de las del elemento, es recortada vertical u horizontalmente para que no quede un espacio vacío.

Tambien nos faltaria un **background-position:center;** y **background-repeat:no-repeat;**.

Ahora vamos con los destaques secundarios. Hacemos lo mismo. Asignarlo al css y darle background y propiedades como al anterior. Pero si agregamos todo a un selector css .destaque\_\_secundario, va a hacer que todos los cuadros con esa clase tengan la misma foto y no queremos eso. Entonces vamos a asignar al selector .destaque\_\_secundario:nth-child(2){...}. Para los siguientes va a ser lo mismo pero cambiandoles el numero que corresponda al orden y darle su background y posicion start y end.

NOTA: Una forma de dar un background + posicion centrado + size cover + no-repeat:

**background:url(‘imagen.jpg’) center / cover no-repeat;**

Con esto nos ahorramos algunas lineas de codigo.

NOTA: para decir que una columna empieza y termina de forma mas resumida, podemos usar **grid-column: 1 / 4**. En lugar del start y end. Tambien se puede hacer para el grid-row.

Ahora vamos con los destaques categorias. Lo ponemos en el css. Le asignamos la cantidad de espacios de columnas y filas que vaya a ocupar. Las primeras 2 columnas y la ultima fila. A la lista de categorias que contiene, lo vamos a hacer flex y que esten uno al lado de otro, le haremos wrap para que pase a la siguiente linea si corta, ademas de decirle que va a ocupar el 100% de alto de su contenedor y darle un space-between.

A los items dentro de la lista los vamos a centrar con un align-items:center y darle un background fdfdfd, un display flex a los mismos y un color de texto 333333 (tenemos que aclarar que el color va a ser inherente). Les vamos a dar un ancho del 50% y padding 1rem.

Ahora vamos a modificar los titulos de las imagenes. Vamos a seleccionarlas en el css. Le damos color fdfdfd. Un background negro con 50% de opacidad. Que este en todo el ancho del cuadrado y centralizarlo y padding de 0.6rem. Y a las cajas seleccionarlas en el css y hacerlo flex, decirle align-items:flex-end para que este abajo y justificarlo al centro.

1. Diseñando la seccion populares

Despues de haber linkeado el nuevo archivo populares.css, vamos a diseñar.

Le podemos dar un padding 3rem 0 a los costados para que no este encima de la seccion destacados.

Primero vamos a las cartas y darle un background fdfdfd. Le vamos a dar un grid y hacer el template area “card-imagen” “card-base” y darle al template de la columna un 100% con grid-template-column:100% y un width de 20% - 1rem ya que van a entrar 5 columnas en el ancho de la pantalla menos el padding entre ellas. Tambien le vamos a dar un grid-template-row de 173px auto, para que ocupen ese tamaño. Y margin right de 1rem.

A la carta dentro de la imagen le vamos a decir que pertenece a la parte del grid correspondiente y darle un ancho y alto del 100% para que ocupe todo su cuadrado. Tambien vamos a hacer lo mismo con la base, pero a este le hacemos grid para acomodar el texto, le vamos a hacer un template de rows ya que son solo dos filas en una misma columna con nombre “card-header” “card-descripcion”. Tambien le vamos a hacer un grid-row-gap 0.5rem para que haya ese pequeño espacio entre los cuadrados de las filas. Vamos a declarar cuanto espacio ocupan estas dos filas con grid-template-row:50px auto; y un padding:0.75rem.

Ahora vamos a ver su descripcion. Le haremos un flex y flex-wrap. Le vamos a asignar el nombre grid area que le corresponde y padding 1rem 0rem.

A la clase populars contenido, le vamos a hacer flex, wrap y decirle que corresponde al grid area “contenido” y padding de 1rem 0rem.

Layouts Responsivos

1. Herramientas, unidades e imagenes.

EMMET es un plugin que va a generar codigo predefinido de manera abreviada. Entonces haremos menos esfuerzo y gastaremos menos tiempo al escribir codigo.

En VSC, Sublime y Atom, ya tienen este plugin de forma nativa.

Despues de haber creado el Index.html. Voy a comenzar. Primero con la version mobile. Podemos verlo en el Figma: [Figma](https://www.figma.com/file/Cv9OIfwW20qbM2ywcSXOnK/Apeperia-Mobile-First-preparando-el-ambiente?node-id=15:198)

Empezamos a crear el header con EMMET.

header>img+nav>ul>li\*6>a

Dentro del header tenemos una imagen y un nav. Dentro del nav tenemos una ul con 6 elementos. Y cada uno de estos 6 elementos, tienen una etiqueta ‘a’.

El em. Es una medida relativa a la font-size de la etiqueta padre de los elementos que estamos utilizando. El rem es una medida relativa tambien, pero con mayor nivel de jerarquia.

Es decir, si utilizamos estas medidas y luego cambiamos el tamaño de la fuente de texto, los objetos con rem o em van a cambiar su tamaño.

En cambio cuando usamos pixeles y hacemos zoom, quedan en ese tamaño de manera estatica. Con rem y em hace que la pagina sea capaz de adaptarse a diferentes resoluciones.

REM: Se usa cuando no desea relacion con la etiqueta padre. Se basa en la etiqueta actual.

EM: El valor del elemento padre se usa en el calculo, creando una jerarquia en el calculo.

Formatos de imagenes.

PNG (Es posible fondo transparente). Hay mucha informacion en la imagen. Mantiene calidad de la imagen. Tiene tamaño elevado.

JGP (No es posible fondo transparente). Es muy compacto (Pierde informacion de la magen). Inferior calidad de imagen. Tamaño leve.

GIF Calidad de imagen muy baja. Uso limitado. Tamaño varia de la imagen.

SVG (imagenes muy utilizadas en desarrollo web) Imagenes vectoriales 2D. Es mas leve. Tamaño variable. Con este, se puede acercar a la imagen sin perder calidad porque guarda mas informacion, cosa que con el PNG se pierde calidad. Una desventaja del SVG es que pierde los colores.

EL SVG es muy utilizado para logotipos, iconos, ilustraciones mas sencillas.

El PNG se suele utilizar para imagenes que necesitan mas detalles.

1. Layout mobile y variables CSS

Mobile First o Desktop First. Cada uno tiene ventajas y desventajas.

Motivaciones para usar Mobile First: -Diseño minimalista y simplificado. -Enfoque en el contenido. -Gran parte de los accesos y ventas son de dispositivos mobiles.

Motivaciones para usar Desktop First: -La interfaz es mas rica. -Mayor capacidad de ejecucion de las instrucciones. -El producto es mas optimizado para desktop.

En HTML <address> una etiqueta orientada para mostrar una direccion de algun lugar. Ayuda a los buscadores cuando queremos encontrar un lugar. Se recomienda ponerla junto al numero de telefono y al correo electronico.

Cuando ponemos un mail o telefono dentro de una etiqueta <a>, en el src=”” se recomienda llenarlo con el “mailto:...” y para telefonos “tel:...”. Tambien está la etiqueta <video> para poner un video y poner su src o url correspondiente. Recordar de poner luego la propiedad controls para que el video se pueda controlar.

Arquitectura CSS: Descompilando los problemas

Nada que agregar.

Javasccript: Primeros pasos con el lenguaje

1. Introduccion al lenguaje

En este curso vamos a empezar a usar Node JS para el lado del servidor.

Que son las variables? Son espacios de memoria donde ponemos datos. Sus valores pueden variar en el tiempo o no.

Para ejecutar un archivo JS en VSC. Abrimos una nueva terminal, escribimos node y el nombre del archivo que queremos ejecutar, le damos a enter y nos dará la respuesta.

Los template strings (plantillas de texto) lo que me permite es poner variables dentro de un string. Se declara con comillas invertidas, signo dolar y llaves. Por ejemplo ´El nombre completo es ${nombrePasajero} ${apellidoPasajero}´; Y todo esto es una alternativa a la concadenacion.

Como filtrar un elemento de un array? Es decir, que nos devuelva solo los elementos que cumplan una condicion dada. Ejemplo:

Const nuevaVariable = arrayFiltrado.filter((elemento)=> elemento == ‘PalabraAFiltrar’);

Console.log(nuevaVariable);

Devolverá un nuveo arreglo con los elementos que cumplan la condicion.

Para unificar elementos de una lista en una cadena de caracteres con un guion por ejemplo, tengo que usar la siguiente funcion.

arrayaSeparar.join(‘-’);

Nos devolvera: elemento0-elemento1-elemento2-...etc.

Recordar que el resultado de esto no es un array, sino que es un string.

Para ordenar un array, uso: arrayaOrdenar.sort();

Para conocer la posicion de un elemento de un array puedo usar:

arrayElementoBuscado.indexOf(elemento);

Nos devolvera la posicion index del elemento que pusimos en el parentesis.

Unificar dos listas.

Const arrayNuevo = arrayUno.concat(arrayDos);

Con esto creamos un nuevo array con los dos arrays unidos.

JS: Manipulando el DOM con Javascript

Si vamos a cualquier pagina y le damos F12 podemos ir a la console, si escribimos ‘document’, nos dará como respuesta todo el HTML.

Si queremos en esa pagina cambiar un texto de un elemento ‘li’ por ejemplo; deberiamos usar el comando de JS document.querySelector(“li”); En los parentesis podemos poner la etiqueta HTML asi como tambien la class o id. Con la clase tendriamos que poner en los parentesis (“.nombreDeClase”) y para el id (“#nombreDeId”).

Ahorasi. Si queremos pararnos en el texto de la etiqueta, podemos hacer document.querySelector(“li”).textContent; Si ejecutamos esto, nos dará como respuesta el contenido de texto dentro de esa etiqueta. Para cambiar ese texto podemos hacer document.querySelector(“li”).textContent= “Texto modificado”; Luego de esto, se actualizará el contenido de la pagina.

Otras formas de seleccionar.

document.getElementById(‘id’) selecciona el elemento por el id

`document.getElementsByClassName(‘clase’) retorna un arreglo de los elementos por el nombre de la clase

document.getElementsByTagName(‘tag’) retorna un arreglo de los elementos por el nombre del tag

document.querySelectorAll(“selector”) regresa todos los elementos con el mismo nombre

El arbol del DOM. Podemos empezar desde el objeto window. Adentro se encuentra document. Y adentro de esta está el HTML. Y del HTML estan sus hijos.

Data atributes

Esto nos va a ayudar a evitar errores por si alguien cambia el nombre de alguna clase o id en el HTML.

Se escriben de la siguiente manera: data-loquequieras

Lo que vaya despues del data puede ser lo que sea a tu gusto.

Y cuando queramos seleccionarlo con JS, deberiamos hacer lo siguiente.

Const nombreVariable = document.querySelector(“[data-loquequieras]”);. Recordar ponerlo entre corchetes dentro de las comillas.

Todos los elementos en nuestro árbol de DOM son nodos y todos los nodos pueden ser accedidos vía JavaScript. Los nodos pueden ser eliminados, creados o modificados. Durante el curso utilizamos el método appendChild que siempre es implementado al final del nodo, para colocar un nodo hijo dentro del nodo padre.

Existen otros métodos que podemos utilizar para manipular nodos:

insertBefore(padre, hijo): Coloca un nodo antes del otro

replaceChild(elemento1, elemento2): Sustituye el nodo del elemento 1 por el nodo del elemento 2

removeChild(elemento): Remueve un nodo del árbol

Explicacion Arrow function

Siempre hemos utilizado funciones como:

function suma(a, b){

return a + b ;

}

Console.log(suma(2, 4)); //Da como resultado 6

Pues, las arrow functions nos permite ahorrarnos utilizarla palabra reservada ‘function’ con solo declarar el nombre de la function que queramos hacer. Como por ejemplo:

Const suma = (a, b)=> a + b;

//Ademas podriamos enriquecer el codigo con algo como:

Const resultado = suma(2, 4);

Console.log(resultado); //Da como resultado 6

En esta arrow function de arriba podemos ver ademas de que ya no usamos ‘function’, tambien vemos que tiene un return implicito (es decir un return por defecto), ya que la accion es de una sola linea, si es mas larga tendriamos que usar las llaves como hacemos normalmente:

Const suma = (a, b)=>{

Return a + b;

}

Como hemos visto. Las arrow function nos ayudan a hacer el codigo mas legible. Pero hay que tomar unas advertencias. Como por ejemplo para cuando queremos poner una arrow function en una propiedad de un objeto, ademas de poder confundirnos en la lectura, puede confundir al mismo javascript. Entonces en este caso mejor usar functions ordinarias.

Otra desventaja que tienen es que ocupan un poco mas de memoria que las function ordinarias, pero es una diferencia despreciable.

IIFE Immediately invoked function expression (Funcion que se invoca inmediatamente).

Es cuando creamos una funcion y se invoca inmediatamente. Por ejemplo:

//Primero con una function ordinaria

function saludar(){

Console.log(“Hola. Yo soy Agustin”)

}

Para que funcione, tenemos que hacer:

Saludar(); y dará como resultado ‘Hola. Yo soy Agustin’

//Function ordinaria con IIFE. Tiene que estar encerrada por parentesis.

(function saludar(){

console.log(“Hola. Yo soy Agustin”)

})();

Para que funcione, ya no tenemos que hacer Saludar(); ya que los ultimos ‘()’ del final hará que se invoque inmediatamente sin necesidad de llamarla. Y dará como resultado ‘Hola. Yo soy Agustin’

//Ahora IIFE con arrow function

(const saludar= (nombre)=>{ //Creamos parametro nombre. Podriamos por defecto nombre=”Juan”

console.log(“Hola. Yo soy ${nombre}”) //Se ejecuta el mensaje con el parametro asignado

})(“Agustin”); //Aqui dijimos que contiene nombre

Da como resultado “Hola. Yo soy Agustin”

Lo que pongamos en los parentesis de abajo, tiene mas peso que si lo ponemos por defecto arriba

Otro Ejemplo:

(const saludar= (nombre=”Juan”)=>{ //Nombre por defecto

console.log(“Hola. Yo soy ${nombre}”) //Se ejecuta el mensaje con el parametro asignado

})(“Agustin”); //Aqui dijimos que contiene nombre

Da como resultado “Hola. Yo soy Agustin”

Va a seguir siendo Agustin.

RECORDAR QUE EN IIFE SE TIENE QUE HACER ENTRE PARENTESIS.

Validación de formularios y HTML5

El siguiente es un input donde nos pide poner un email:

<input name="email" id="email" class="input" type="email" placeholder="Email" required data-tipo="email">

<label class="input-label" for="email">Email</label>

<span class="input-message-error">Este campo no es valido</span>

Se puede ver que en una de sus propiedades dice ‘required’, eso significa que tenemos que completarlo si o si para poder enviar el formulario.

Otra propiedad es el type=”email”. Este lo que hace es obligarle al usuario a que ponga el ‘@loquesea.com’. Ya que, sino podrian escribir cualquier cosa aunque no sea un email.

El siguiente es un campo para contraseña:

<input name="password" id="password" class="input" type="password" placeholder="Contraseña" required pattern="^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[0-9])(?!.\*[ !@#$%^&\*\_=+-]).{6,12}$" title="Al menos 6 caracteres, máximo 12, debe contener una letra minúscula, una letra mayúscula, un número y no puede contener caracteres especiales." data-tipo="password">

Si ponemos type=”password”. Hará que cuando escribamos algo, se pongan los asteriscos por privacidad.

Tambien podríamos poner la propiedad minlength=”6” significa que el formulario espera que pongas como minimo 6 caracteres para la contraseña. Tambien puede hacerse lo inverso con maxlength=”12” para decir que no tenga mas de esos caracteres. Con el maxlength, no te deja escribir mas en el campo cuando se supera el limite puesto.

Si queremos que la contraseña cumpla ciertos requisitos ademas del minimo y maximo de caracteres, vamos a usar regex pattern que son expresiones regulares y lo escribimos como pattern=””; ( [Regular Expression Library (regexlib.com)](https://regexlib.com/Search.aspx?k=password&AspxAutoDetectCookieSupport=1) ). En ese link podes buscar patrones que crearon otras personas. Con lo puesto en el patron del codigo mas arriba hay 4 reglas, quiero decir:

(?=.\*[a-z]) ‘?=’ Aceptar caracteres siempre y cuando...

‘.’ Que no acepta saltos de linea (Enter).

‘\*’ Que tiene que cumplirse al menos una vez el siguiente...

‘[a-z]’ Desde la a a la z minuscula.

(?=.\*[A-Z]) ‘[A-Z]’ Desde la A a la Z mayuscula.

(?=.\*[0-9]) ‘[0-9]’ Desde el numero 0 al 9.

(?!.\*[ !@#$%^&\*\_=+-])

‘?!’ No va a aceptar... Todos los caracteres dentro de los corchetes

.{6,12} Todo esto significa que vamos a querer que se cumplan esas reglas anteriores entre 6 y 12 veces.

Todo lo anterior citado, tiene que estar entre "^...$"

Y aquí el cartel que le diga al usuario los requisitos para este campo:

title="Al menos 6 caracteres, máximo 12, debe contener una letra minúscula, una letra mayúscula, un número y no puede contener caracteres especiales."

Para un campo de fecha de nacimiento como el siguiente:

<input name="birth" id="birth" class="input" type="date" placeholder="Fecha de nacimento" required data-tipo="nacimiento">

Vemos que es un type=”date” para que aparezca el formato para poner la fecha.

Por desgracia, si queremos poner un minimo para la edad de 18 años, necesitamos hacerlo si o si por Javascript.

Javascript orientado a objetos:

1. Repeticion de codigo

En esta seccion vamos hacer el ejercicios de cuenta bancaria con nombre, dni, numero de cuenta, saldo. Pero si tenemos 1000 clientes, copiar y pegar el codigo es algo inviable, entonces vamos a usar objetos.

Para crear un molde donde poner nuestros miles de datos, vamos a usar class nombredeclase{...};

Para crear las piezas para ese molde tenemos que usar new nombredeclase();

Las clases ayudan a organizar datos repetibles.

El ‘this’ define la clase actual donde lo ejecutamos.

Para exportar/importar archivos js entre si, tenemos que hacer lo siguiente:

import {nombreDeClass} from './nombreDeArchivo.js'

Para hacer esto, tenemos que hacer module con el comando en node ‘npm’ que significa node package manager. Si hacemos npm init, creará el archivo package.json y nos preguntará el nombre del proyecto, puede ser cualquiera pero no puede tener mayusculas y una descripcion (todo esto puede recordar un poco a Git). Luego, nos mostrará el entry point con el archivo que node buscará para ejecutar. Tambien nos preguntara test command, git repository, keywords, author y license.

Ahora podremos ver el archivo package.json que tendrá todas las definiciones del proyecto anteriormente mostradas. Aquí, tenemos que escribir la propiedad ‘ ”type”:”module” ’.

Un paso importante es hacer visible la clase que quisimos importar. Para eso hay que ir a ese archivo y poner export antes de la clase. Como por ejemplo:

export class cuentaCorriente{

Volviendo al tema de los objetos, al crearlos podemos poner atributos con sus valores por defecto en el constructor. Por ejemplo:

constructor(){//valores base.

        this.cliente = null; //Referencias (a objetos en este caso)

        this.#saldoCuenta = 0;//numero.

        this.numeroCuenta=""; //cadena caracteres

        this.agencia=""; //cadena caracteres

    }

Es buena practica inicializar variables con su valor por defecto como se hizo arriba.

Cuando cuando creas una variable y no le das un valor inicial, JS por defecto asume que está undefined. Esto puede generar problemas sin querer. Por ejemplo, al querer acceder a una variable sin verificar que esta esté definida, el programa nos dará error. Así que siempre es mejor dejar un valor base para cada variable declarada.

Ahora, para hacer que un objeto sea private y poder entrar a sus propiedades. No es tan sencillo. Vamos a tener que utilizar las palabras reservadas get y set a diferencia de lo que pasó antes con #saldo, pudimos ingresar a ella a traves de las function utilizadas en el codigo.

Con las nuevas paabras reservadas, podemos ingresar a ellas como si fueran un objeto.

Con set podemos asignar valores a nuestra propiedad del objeto privado. La creamos dentro del objeto con:

set nomSetterProp (valor){

This.PropObjeto = valor;

}

Lo llamamos en el codigo con:

#Objeto.nomSetterProp = elNuevoValor;

Así se le dara el valor a esa propiedad del Objeto.

Con get podemos obtener/llamar el valor del objeto privado:

get nombreGetter(){

Return this.PropObjeto;

}

Lo llamamos con el codigo:

Console.log(#Objeto.nombreGetter); //Es un ejemplo como para tomar la propiedad y mostrarla en pantalla.

Ejemplo:

Const Persona = {

nombre:””,

apellido:””,

set darNombre(valor){

this.nombre=valor;

},

get traerNombre(){

return this.nombre;

}

}

Persona.darNombre=”Agustín”; //Le dimos al nombre el valor Agustin

Console.log(Persona.traerNombre); //Nos imprimirá el nombre asignado: El get podemos llamarlo para cuando queramos hacer una comprobacion if, etc.

Para fines practicos, el getter y setter para una propiedad, podrian tener el mismo nombre.

Constructores:

Un ejemplo de constructor es el siguiente

 constructor(){//Este es un constructor y se ejecuta al crear un nuevo objeto. Y en este caso nos sirve para dar valores base.

        this.#cliente = null;

        this.#saldoCuenta = 0;//Para que por defecto, cada vez que creamos un nuevo cliente, este tenga el saldo 0.

        this.numeroCuenta="";

        this.agencia="";

    }

Como dije, el constructor es una funcion que se genera cada vez que se genera una nueva instancia. Y si pasamos el mouse por encima de este constructor, nos dirá a quien pertenece, como por ejemplo:

constructor cuentaCorriente(): cuentaCorriente

Con el constructor, ademas de darle valores iniciales a las propiedades, podemos pasarle parametros.

 constructor(valor1, valor2, valor3){

        this.nombreCliente=valor1;

        this.dniCliente=valor2;

        this.rutCliente=valor3;

    }

Atributos estaticos dentro de clases u objetos:

Serian los atributos que quiero que se manejen de manera global para todas las clases creadas por el molde. Por ejemplo, que se sume 1 a un contador por la cantidad de objetos creados.

Para hacer ese atributo estatico, hay que primero declararlo en la class con static adelante, y luego en el constructor llamarlo con el nombre de la clase en lugar del ‘this’, ya que el this lo estaria haciendo para cada clase de manera individual, y nosotros lo queremos para la clase molde en general.

JS:Herencia e Interfaces en Orientacion a Objetos