**Javascript**

**Clase1**

Crear un index.html para que el archivo JS se haga visible en el navegador

JS es key sensitive y no se tienen en cuenta espacios en blanco o nuevas líneas (como html). Comentarios similares a html

**Variables**:

Pueden declararse con var, let y const.Const: no puede ser reasignado con otro valor.

Para declararlas, no puedo usar espacios, ñ, guión medio, números por delante, etc.

, pero si camelcase, snake, $ por delante.

**Valores:**

Number y string(puede ir con “ o ´)

**Prompt** es lo mismo que el input de python

**Alert**devuelve un mensaje

En el html(!Doctype) tenemos que agregar

<scriptsrc="index.js"></script>

Antes de cerrar el body

**Clase2**

**Condicionales**.

Las palabras true y false son reservadas por js

**If** (**condición**) {

Console.log(“ bla bla bla”)

}

Si la condición se cumple, se ejecuta el console.log

**=** asigna

**==** compara **!=**distinto

**===**estrictamente igual. 10 === “10” es false **!==** estrictamente distinto

**&&** 🡪 and

**||** 🡪or

**!** 🡪not

**Clase3**

**Bucles**

Estructuras que hacen que nuestro código se repita hasta que se cumpla una condición preestablecida por el dev.

3 instancias dentro de la declaración del bucle for:

* Desde: valor inicial de las variables
* Hasta: elemento que determina la finalización
* Actualización: valor que se asigna después de cada repetición

Luego del for, siempre hay un ciclo de código dentro de llaves {}

for (let i = 0; i <10; i++){

console.log(i);

}

Aca se ven las 3 instancias, comienza en 0, va a hasta que i sea menor que 10 y suma 1 numero por cada i. para subir de a 2 se escribe i+=2

**Turnero:**

For (let**i** = 1; i<=5;i++){

let nombre = propmt (“ingrese…bla bla”)

if (nombre == “”){

console.log(“no ingresaste nombre, vuelva”)

**i** = **i -1**🡪borra un turno que no fue asignado

continue; 🡪 sigue el siclo y no hace break

}

Console.log (“hola, “ + nombre + “turno: “ + **i**)

}

**SI quiero cerrar el ciclo, corto con break**

**Si quiero que el ciclo continue, escribo continue y puedo borrar una iteracion**

**WHILE**

El código se repite hasta que la condición aparece.

Se puede generar una variable (**let = condition**) con valor **false** para evitar un bucle infinito y la condición **condition = true** se actualiza dentro del entorno de la condicion

Letcondition = true;

While (condition) {

Console.log(“la condición es verdadera);

Condition = 10 ==20; 🡪 se rompre porque 10==20 es falsa

}

**Clase4**

**Funciones**

**Conjunto de instrucciones que se agrupan para realizar una tarea y se pueden reutilizar**

**0:41hs:**  explicación de **parámetro** y  **argumento**

**1:29hs**: el **let**para las variables, corta con el hoisting, el **var** habilita el hoisting. El hoisting es una forma de lectura de js que lleva a todas las variables declaradas al inicio del documento, pero sin atributo. Si intentamos llamar a una, antes de asignarle un valor, nos da error de **undefined.**

Si declaramos **var**dentro de una función, la misma es tomada como una variable global, **let**respeta los scopes (ámbito).

**1:37:**

FUNCIONES DECLARADAS: las iniciales de la clase

**function suma(a,b){**

**Return a + b};**

FUNCIONES ANÓNIMAS: se crea una función sin nombre y se le asigna a una variable, en este tipo se puede reasignar el valor de la variable que contiene la función

**let suma = function(a,b){**

**Return a + b};**

FUNCIONES DE FLECHA:

similar a las anónimas, pero la función se cambia por paréntesis y el bloque de códgo por una flecha **=>**

**let suma = (a, b) => a + b**elresturn queda implícito en la flecha

**Clase5**

**Objetos**

Colección de datos relacionados que contienen claves y valores, que se pueden repetir muchas veces en el código

se declara de la siguiente manera: **nombre = {y entre llaves- propiedad:valor}**

**Nombre** = { contiene pares de

propiedad: valor ,<- coma

propiedad: valor ,<- coma

}

Si yo quiero hacer console.log de una propiedad que esta dentro del scope va a dar undefined. Debería llamarse al producto y a su propiedad

Consol.log (**Nombre) ->**imprime todas las categorías del ítem

Consol.log (**Nombre. valor) ->**imprime el valor

Console.table**->** devuelve los datos en una tabla

**Tambien:**

Consol.log (**Nombre[ “valor”])**

**Para agregar propiedades:,**

**Producot1.pepe =** “soy pepe”

Del mismo modo puedo modificar algunas propiedades ya existentes

No se pueden modificar objetos a nivel estructura pero si sus propiedades

**Objetos contructores**

Moldes para crear objetos a partir de ellos

Se crea una **funciónProducto**(con los parámetros adentro**id**, **nombre**)**{**

**This.id** = atributo de producto

**This.nombre** = otro atributo etc.

**}**

Const producto\_1 = new **Producto (“abc023”, Arroz”)**

Const producto\_2 = new **Producto (“abc024”, Leche”)**

Console.log(producto\_1);

Console.log(producto\_2);

**Mètodos : 1:30hs**

Los métodos son comportamientos que tienen asociados los objetos

Const = mensaje = “Hola, soy un mensaje”

Console.log(mensaje.length) -> cantidad de caracteres

Console.log(mensaje.toUppercase()) ->devuelve la cadena en mayus

Al mismo tiempo puedo crear **funciones**:

Function Hero (name, lastname, alias, ){

This.name = name;

This.lastname = lastname;

This alias = alias;

]

This.sayHi = function () {

Console.log(“hola, mi nombre es: “ this.name + “ “ + thiis.lastname + ….

}

**FOR IN**

For (porperty in hero\_1){

Console.log(hero\_1[property])

}

**El objeto tiene propiedas, las propiedades son descriptivas del objeto y los métodos interactúan con ellas para modificarlas de acuerdo al interés del usuario, dev, etc.**

**Clase6**

**Arrays**

Los arrays son listas, arreglos u otros sinónimos. Se agrupan por corchetes **[]**, y dentro de los corchetes se ponen los nombres separados por coma.

En los arrays pueden ir muchos elementos, pero **NO** se puede declarar variables dentro

Para llamarlos se puede escribir **console.log [0],** por ejemplo, y accede al elemento que esta en ese lugar del índice.

Para recorrer un array:

**For (let i = 0, i<5, i++)**

**Console.log((i+1) + “. “ +simpsons[1])**;

**Simpsons.length ->** devuelve la cantidad de elementos del array. Evita espacios undefined en el console.log

Simpsons[6] = “Bob Patiño” -> agrega a Bob pero si no conozco el índice, puedo pisar elementos

Simpsons**.push** (“bobpatiño”); -> agrega a Bob al final de la lista //**MUTABLE**

Simpsons**.unshift** (“bobpatiño”); -> agrega a bob al principio // **MUTABLE**

Simpsons**.shift** (“bobpatiño”); -> elimina al primero //**MUTABLE**

Simpsons**.pop** (“bobpatiño”);-> elimina al ultimo // **MUTABLE**

Simpsons**.splice(3,2);** -> primer parámetro es donde arranca a eliminar, el segundo marca cuantos va a eliminar. En una lista de 6 elementos me deja el **0,1,2,6**//**MUTABLE**

Console.log (rest); -> devuelve los elementos eliminados del array original

Console.log (simpsons); ->devuelve lista modificada

**---------------------------------------------------------------------------------------------------------------**

Simpsons**.join(“ – “)** -> junta todos los elementos en una cadena de texto separado por **espacio, guion, espacio**. Para que pueda ser visualizado hay que guardarlo en una variable y mostrarla con console.log //**INMUTABLE**

Let frutas = [“frutilla”, “mango”, “pera”]

Let verduras = [“papa”, “zanahoria”, “cebolla”]

Let inventario

**1:55: array con objetos**

**Forof**

For (empleado ofempleados) **empleado = i**

Console.log(empleado)

Console.log(empleado.name)

**Clase7**

**Funciones de orden superior (HOF)**

Función de orden superior es aquella que puede devolver una función o recibir una función como parámetro.

**Ejemplo for of en el vsc**

Let frutas [“kiwi”, “banana”,”pera”,”melon”,”sandia”].. etc

**1:30hs Metodos de búsqueda de arrays**

Frutas.forEach((value) => console.log (value)); Metodo similar al for of

**FIND : busca elemento y es true o false. Devuelve un resultado específico de la primer coincidencia**

Const **resultado** = frutas.find ((fruta)=>fruta == “kiwi”);

Console.log (**resultado**)

**FILTER: devuelve todos los elementos que comparten un atributo**

Const **resultado** = frutas.filter ((fruta)=>fruta.kiwi == “semillas”);

Console.log (**resultado**)

**SOME: devuelve si alguno cumple con lo requerido**

Const hayCitrico = some ((fruta)=>fruta.kiwi == “citrico”);

**MAP: crea un nuevo array con los valores modificados**

Let preciosConIva = precios.map((precio) => precio\*1.21)

Console.log(preciosConIva);

**REDUCE: reduce el array en un único valor:**

let ventas = [100, 200, 300, 1, 2,32,44];

let total = ventas.reduce((valorA,valorB)=> valora+valorB, aca); puedo setear el valor inicial ¡=0 (el dinero con el que comenzo la caja antes de abrir el local)

console.log(total); ----> devuelve la suma de todos como ganancia del dia

**NOTA DEL TUTOR!**

*Profe para la siguiente pre entrega estaría bien pensado, que en carro de compras los ítem que seleccione la persona se pongan en un array y se vaya sumando cada ítem que coloque hasta que termine y le aparezca todo ese array con la suma total y los ítem que seleccionó? Disculpa la pregunta un poco fuera de lo que estamos viendo*

*[Tutor]Braian Silva 21:13*

*Si, eso sería ideal de hechoPodrías usar métodos que fuimos viendo para seleccionar el producto, y el reduce para lo que es el total de la compra, tomando solo el precio de cada uno de los objetos dentro del array*

**Clase8**

**HOF II**

**MATH**

**Min y max ():** devuelven el elemento máximo y minimo de una lista. Se debe pasar una lista de elementos, sin ninguna estructura particular (no array). Poniendo “. . . “ delante del array, nos lee el max o min

**Ceil, floor, round():**

Ceil: valor redondeado al entero mayor mas próximo

Floor: redondea para abajo

Round: redondeo mas común, va al entero mas ceracano

**Sqrt ():** devuelve la raíz cuadrada del número

**Random:** genera un número pseudo-aleatorio entre 2 valores

**Trunc:** corta con los decimales de un numero.

Se pueden apiilar los métodos **(Math.ceil(Math.sqrt(number))**

**DATE**

Es necesario crear una nueva instancia para poder utilizar la función

Let nuevafecha = new Date() . haces console.log y te da la hora de ese momento

**Javascript**

**Clase9**

**DOM (modelo de objetos del documento)**

El DOM es un mapa de cómo esta creado nuestra web y nos permite modificarla de manera dinámica a través del JS.

Desde cualquier parte de mi código puedo acceder al objeto “document” y el árbol de nodos del DOM

**Si quiero pedir un elemento: getElementById(“nombre”)**

**Const title = getElementById(“nombre”) -> al ser ID devuelve solo un elemento**

Title.innertext = “el texto que quiero reemplazar en html” **Agrega todo como texto, hasta lo que esta entre <>**

Title.innerHTML = “<span> con lo que quieras reemplazar<span>”

**Si quiero pedir un item: getElementByClassName(“nombre”)**

**al ser nombre de clase puede devolver varios items**

Const ítems = document.getElementsByClassName(“nombre de clase”)

**Si quiero crear un nuevo elemento en el DOM: document.createElement(“nombre de etiqueta)**

Const newNode = document.createElement(“section”) **CREAR**

Lo guardo en una variable para que lo pueda inyectar en HTML **ASIGNAR CONTENIDO**

newNode.innerHTML = “<article><p>hola desde el nuevo nodo<p><article>”

**document**.**body**.**append**(**newNode**); **append siempre se agrega al final**

**aplico** el **nodo**, en el segmento **body.**

**Si quiero crear un elemento en el DOM: etiqueta.remove(“nombre)**

Title.remove(“nombre”)

**Si quiero obtener info de un elemento en el DOM: document.getElementById(“nombre).value**

Document.getElementById(“text”).value

**Traer info de u lugar particular del documento**

Document.queryselector(“title . span .p”) tiene que ser un solo elemento con esa especificidad o ponerle un id. Es lo mismo que cuando querías aplicar estilos.

QuerySelectorAll, trae todos los elementos de ese id, etiqueta html o ruta de etiquetas.

**2:00 ejemplo de personajes**

**Javascript**

**Clase10**

**Eventos**

Con Js podemos determinar que va a suceder al interactuar con un elemento de la pagina, dándole una función

Definir eventos en js:

* botón.onclick = () => pepe() ->seleccionas el elemento y asignas una funcion
* **Sobre la etiqueta HTML -> sobre la etiqueta HTML ->** no hay que mezclar codigo
* addEventListener -> recibe 2 parametros

-nombre de evento (click)

- función de callback (pepe)

Basicamente es elegir un elemento del Dom y traerlo con una cost por su ID, ponerle un listener, darle un evento para que responda y le pasamos una función

**0:40: APLICACIÓN DE EVENTO INVITAR A PEPE A LA FIESTA.** Hago con productos?

**EVENTOS MAS COMUNES**

* Mouse:
  + mousedown/mouseip: se oprime o suelta el botón sobre elemento
  + Mouseover/mouseout: puntero sobre elemento
  + Mousemove = hover (cada vez ue me muevo dentro, genero acción)
  + Click
* Teclado:
  + Keyup
  + Keydown : ambos similates a mousedown. Reaccionan al teclear y soltar
* Change:
  + Creas una etiqueta con id, usas const =X para hacer el getelementyID y le pones el adeventlistener a esa const. Change y el lo que quieras que haga al detectar el cambio.
* Input:
  + Similar a change, pero detecta los cambios mientras se ejecuta

**1:32:** MIXKIT pagina de videos, muestra de evento de play en videos

**1:40:** coordenadas X e Y

**1:50** ingresa valores por el input y se imprimen en la pagina. focus

**Javascript**

**Clase1**

**JASON Y STORAGE**

Storage es un **objeto** (API de almacenamiento web), almacena datos de manera local sin necesidad de conectarse al servidor. Los datos se almacenan en la pc y no pueden ser obtenidos de forma externa. **No es BD**

La info se almacena en clave-valor, similar a los objetos.

**localStorage** y SessionStorage son objetos globales, se pueden usar en cualquier parte del código, almacena datos hasta que se borra la cache.

**localStorage.**setItem(‘clave’,’valor´)

Para trer la info del **localStorage,** puedo crear una variable o un objeto y llamar con un getItem. **Funciona a nivel dominio**

**sessionStorage:** Los datos son guardados hasta que el user cierra la ventana, cada ventana tiene su sessionStorage. **Funciona a nivel pestaña**

A los storages no puedo pasarles objetos literales en arrays, para hacer eso, debo transformar a objeto JSON (**javascript object notation**). Se usa para representar datos estructurados con sinstaxis JS. Se utiliza para enviar y almacenar datos en apps web.

Stringify: toma un objeto literal y lo convierte en Json

Hago **localStorage.setItem** del objeto literal y le agrego **JSON. Stringify(‘jey’ , NOMBRE)**

Si quiero hacer console.log, hago el **localStorage.** **getItem(key)**

