/TALLER DE JAVASCRIPT

https://t.me/javascript_uno

/in/ignagarcia

ORGANIZA GNUNO

/CONTENIDOS

/01 /INTRODUCCIÓN

Variables, Tipos de Datos, Operadores y Estructuras, Funciones, Objetos y Arrays

/02 /JS FUNCIONAL

Expresiones, Anónimas, Callbacks

/03 /JS Y EL DOM

Nodos, Acceder al DOM, Manipular el DOM, Template String, Event Listeners

/04 /JS ASINCRÓNICO

Promesas, Fetch, Async/Await

/INTRODUCCIÓN

/VARIABLES

Para la Definición:
* let
* const
* var (no recomendado)

EJ:

```
let algo
let contador = 0
const URL = 'https://…'
```

/TIPOS DE DATOS

El tipado es dinámico y por inferencia

number
string
boolean
bigint
object
null
undefined

/INTRODUCCIÓN

/OPERADORES Y ESTRUCTURAS

Operador Ternario

```
let soyMayor = (edad ⩾ 18) ? true : false
```

Nullish Coalescing

```
console.log(user ?? ‘Anonimo’)
```

For Of

```
for(let item of list) console.log(item)
```

/INTRODUCCIÓN

/FUNCIONES

```
function nombre(arg1, arg2) {
   let contador = 0
   …
   return contador
}
```

/OBJETOS

```
let obj = {
   nombre: 'Igna',
   apellido: 'Garcia',
   edad: 32,
   soyMayor: function(){
       return this.edad ⩾ 18
   }
}

obj.nombre
obj.soyMayor()
obj['edad']
```

/INTRODUCCIÓN

/ARRAYS

```
let lista = [‘elemento1’, 2, true, …, ‘elementoN’]

let n = lista.length

let elemento = lista[2]

lista.push(‘nuevo’)

let ultimo = lista.pop()
```

/FUNCIONAL

En JavaScript las Funciones son Objetos, por ende:

Anónimas:

```
let hola = function() {
    console.log('Hola')
}
let chau = () => {
    console.log('Chau')
}
```

Es importante notar que
hola ≠ hola()

```
hola    // function

hola() // Hola
```

/FUNCIONAL

En JavaScript las Funciones son Objetos, por ende:

Callbacks:

```
let suma = (n1, n2) ⇒ n1 + n2

let calc = (n1, n2, op) ⇒ op(n1, n2)

calc(7, 3, suma) // 10
```

/FUNCIONAL

Cuidado!

Caso 1:

```
suma(3, 5) // error undefined

let suma = (n1, n2) ⇒ n1 + n2
```

Caso 2:

```
suma(3, 5) // 8

function suma(n1, n2) {
    return n1 + n2
}
```

/DOM

El DOM es una estructura de datos con forma de Árbol donde el elemento mínimo es llamado NODO, es usado para representar a una página web.

Un nodo puede ser un objeto HTML, un simple Texto o un Comentario.

La importancia de este es que podemos manipularlo fácilmente a través de JS dinámicamente.

/DOM

.querySelector

```
// único obj o Null
document.querySelector(“#titulo”)
document.querySelector(“p.lead”)

// array de obj
document.querySelectorAll(“li.item”)
```

.getElementBy

```
// único obj o Null
document.getElementById(“titulo”)

// array de obj
document.getElementByClassName(“lead”)
document.getElementByName(“item”)
document.getElementByTagName(“li”)
```

/DOM

Una vez accedemos a nuestros elementos podemos acceder a sus propiedades y editarlas

```
// contenido
elemento.innerText = “hola”
elemento.innerHTML = “<b>hola<b>”

// atributos
elemento.id = “saludo”
elemento.className += “lead”
elemento.placeholder = “pepe@gmail.com”
```

```
element.remove()
element.removeAttribute(“placeholder”)

// agregar contenido
element.append(otroElemento)
element.prepend(otroElemento)
```

/DOM

Podemos registrar procedimientos a seguir luego de que ocurra una determinada acción sobre determinado elemento con los EventListeners

```
const alertar = (event) => {
    event.target.className += “clickeado”
}

let boton = document.querySelector(“button”)
boton.addEventListener(“click”, alertar)
```

/ASINCRONISMO

Hay ciertas acciones que no sabemos cuánto van a demorar en resolverse y no podemos esperarlas.

El Asincronismo nos permite continuar con la ejecución por más que la petición aún se esté resolviendo.

Cuando esta petición se resuelva se notificará y pasará a ejecutar un Callback.

/ASINCRONISMO

Las Promesas son un mecanismo para tratar con el Asincronismo

```
let miPromesa = new Promise((resolve, reject) => {
    /* ... */
    if(/* ... */) resolve() else reject()
})

miPromesa // Pending || Fulfilled || Rejected

miPromesa
    .then(res => console.log(res))
    .catch(err => console.log(err))
```

/ASINCRONISMO

Usualmente cuando queremos consumir un servicio externo, necesitamos controlar su asincronismo con la función fetch

```
fetch(url)
   .then(res ⇒ res.json())
   .then(data ⇒ hazAlgo(data))
   .catch(err ⇒ console.log(err))
```

/ASINCRONISMO

A veces, la sintaxis del encadenamiento se vuelve confusa y difícil de leer; es por eso que podemos utilizar async/await, para que el código sea similar a uno Sincrónico

```
async function requestX() {
   let res = await fetch(url)
   let data = await res.json()
   hazAlgo(data)
}
```

/FIN

Linkedin: /in/ignagarcia

ORGANIZA GNUNO