Glosario 02 y 03

- **Arrow Functions:** Es la nueva forma de crear funciones que viene con las especificaciones EcmaScript 5.

  Su sintaxis es la siguiente:

  ```
  1 const crearUsuario = () =>{...}
  ```

  En las Arrow Functions se omite la palabra function y se agrega un igual y un signo de menor después de los argumentos (en forma de flecha).

- **Async/Await:** Async/Await es una nueva forma de manejar promesas en Javascript. Disponemos de dos métodos para trabajar con este tipo de funciones, async y await. Su uso es bastante sencillo, utilizaremos async para declarar las funciones asíncronas, y dentro de estas funciones utilizaremos await para suspender la ejecución hasta que la expresión que le sigue devuelva un valor. Veamos un ejemplo sencillo.

  ```
  1 async function asyncFunction () {
  2  const foo = await asyncOperation()
  3  return foo
  4 }
  ```

  Al ejecutar el código anterior, la función asyncFunction() bloqueará la ejecución del resto de código del programa hasta disponer del valor devuelto por su función interna asyncOperation().

- **Bind:** La función `bind()` crea una nueva función (función ligada) con el mismo cuerpo (propiedad interna call en términos de ECMAScript `5`) como la función que será llamada con la referencia this asociada al primer argumento de bind `()`, el cual no podrá ser sobrescrito. `bind()` también acepta parámetros

predeterminados que anteceden al resto de los parámetros específicos cuando la función objetivo sea llamada. Una función ligada también puede ser construida utilizando el operador the`new`al hacerlo, actuará como si en su lugar hubiera sido construida la función objeto.

- **Bucles:** Bloques de código que se ejecutan mientras se cumpla una condición. Tres elementos controlan el flujo del `bucle`:

```
1 Inicialización: Fija los valores con los que iniciamos el bucle.
2
3 Permanencia:Fija el tiempo en el que se permanece en el bucle.
4
5 Actualización:_ determina cuando se actualizan las variables de
control al ejecutarse la iteración.
```

- **Callback:** Es la manera más común de controlar la asincronía en JavaScript, Es una función que se establece para realizar una llamada de vuelta en un momento del tiempo indeterminado, usada como parámetro de función en algún evento o función que presente asincronía.
- **Closures:** Los closures o funciones de cierre, son un patrón de diseño muy utilizado en`JavaScript.`Es una función que encapsula`variables`y definiciones`locales`, únicamente accesibles si son devueltas con el operador return, este patrón hace posible modularizar nuestro código.

```
1 function foo(lenguaje) {
2  const nombretrans = 'Babel'
3  return function(){
```

```
4    console.log(lenguaje +' es un lenguaje de programacion y
'+nombretrans+' un transpilador poderoso')
5  }
6 }
7
8 let bar = foo('Javascript')//foo devuelve una función que es
almacenada en bar
9bar() //Debido a que cuando foo fue llamada su parámetro
10     //era 'Javascript' entonces recuerda ese mismo valor
11 let baz = foo('JS')
12 baz()//Acá recuerda JS, Babel siempre se mantiene ya que es una
constante en la función
13
14//Lo importante es que una función interna recuerda
15//las variables de una función externa aun cuando la externa
16//no se está ejecutando
```

- **Click:** Se activa cuando se presiona el botón del mouse y ejecuta una función que recibe como parámetro, dicha función recibe el evento.

```
1 const button = document.getElementById("test")
2
3button.addEventListener("click",function(event{
4 console.log('hola mundo')
5 }
```

- **Const:** Define una constante que no va a cambiar su valor durante el tiempo.

```
1 const nombre  = 'Juan'
```

- **Constructor:** El método constructor es un método especial para crear e inicializar un objeto creado a partir de una clase.

```
Sintaxis: constructor([arguments]) {...}
```

- **Currying:** Una técnica que utiliza una evaluación parcial,**Currying** se refiere al proceso de tomar una función con argumentos y transformarla en funciones que cada una toma un solo argumento.
- **Document.querySelectorAll:**Devuelve una lista de elementos ,que son iguales al grupo de selectores. El NodeList contendrá todos los elementos que correspondan con cualquiera de los selectores especificados.

```
let elem = document.querySelectorAll("div.ejemplo1, div.ejemplo2");
function getElements(el) {
  return document.querySelectorAll(el);
 }
 getElements("div.ejemplo");
```

- **Document.getElementById:** Devuelve el elemento por su id; id que se usa para identificar de forma única el elemento, que se encuentra en el html con el atributo id.

```
<div id="ejemplo"></div>

 let element = document.getElementById('ejemplo');
```

- **Document.getElementsByClassName:** Devuelve una colección que contiene todos los elementos que contenga la clase del atributo, por ejemplo.

```
<div class="ejemplo"></div>
<div class="ejemplo"></div>
```

```
3<div class="ejemplo"></div>
4<div class="ejemplo"></div>
5
6 var ejemplo= document.getElementsByClassName("ejemplo")
7
8 ejemplo[0].innerHTML = "Soy la clase ejemplo del primer div!"
```

- **ECMAScript:** ECMA Script se refiere a las nuevas características del estándar ECMA-262 conocidos como JavaScript, introducido después de ECMAScript 2015. Las nuevas versiones de las especificaciones ECMAScript se liberan anualmente, ES 2016 será liberado y él ES 2017 es el proyecto actual ECMAScript.
- **Element.setAttribute:** Agrega el valor de un atributo en el elemento especificado. Si el atributo ya existe, el valor se actualiza, en el caso contrario se agrega el atributo con el nombre y el valor especificado.Elemento

```
1 element.setAttribute (nombre, value);
2
3 name // especifica el nombre del atributo que se va a establecer.
string
4 value // contiene el valor a asignar al atributo. string
```

- **Eventos:** Los eventos de **JavaScript** permiten la interacción entre las aplicaciones JavaScript y los usuarios. Cuando se toca un botón, cuando se pulsa una tecla determinada, se produce un evento. Sin embargo no todos los eventos los produce el usuario, cuando se carga una página también se produce un evento.

- **ForEach:** Es uno de los objetos de la clase Array. Ejecuta la función callback una vez por cada elemento presente orden ascendente. No es invocada para índices que han sido eliminados o que no hayan sido inicializados.
- **ForIn:** Podemos iterar sobre las propiedades del objeto.

```
let apple  = {
 name: 'apple',
 category: 'fruit',
 price: 100
}
 for(let prop in apple){
 console.log('la propiedad ' + prop + 'contiene '+ apple [prop\]);
}
// Devuelve las propiedades del objeto Apple, name, category, price.
```

- **Función:** Las funciones en JavaScript son un objeto, heredan sus propiedades de la clase padre object.Son bloques de código ejecutable, le podemos pasar parámetros y operar con ellos. Podemos modular nuestros programas y estructurar en bloque que realicen una tarea en concreto. **Más.**
- **Funciones Asíncronas:** Son funciones que al ejecutarse, devuelven una promesa. Cuando una función asíncrona devuelve un valor, la promesa será resuelta con el valor devuelto. Cuando una función asíncrona lanza una Excepción o un valor, la promesa será rechazada con el valor lanzado.
- **Generadores:** Es un objeto que sirve para decirle a JavaScript que nuestra función es un generador y se debe indicar con un asterisco de la siguiente forma:

```
function* generador() {
  yield 1;
  yield 2;
  yield 3;
}

var g = generador(); // "Generador { }"
```

Si creamos un generador debemos colocar la palabra clave yield la cual indica que cuando llamemos a la función después de la primera vez, esta iniciará en la línea después de yield. El generador convierte en objeto la función.

- **Hoisting o “elevamiento”:** Javascript primero busca las declaraciones de función (estas van primero que las declaraciones de variables ya que tienen prioridad),el hoisting en funciones es de 2 maneras las funciones como declaración son llevadas al tope del actual scope, por otro lado las funciones como expresión se eleva su variable al tope de ese scope de tal manera que esta no funciona por medio del hoisting, las pone en el tope del scope, cómo undefined, sólo hasta llega al lugar donde fueron anteriormente declaradas (la expresión) será inicializado su valor real de esta última forma ocurre con las variables.

```
// Ejemplo1
 var foo = "12"

 function foo(){

```

```
6         console.log("foo function");
7    }
8
9 console.log(foo) -> 12 ( foo is not a function )
10
11// Ejemplo2
12 console.log(foo) -> undefined
13
14 var foo = function(){
15
16      console.log("foo function");
17    }
18
19    //ocurre que
20
21    var foo = undefined;
22
23    foo = function(){...};
```

- **Immediately Invoked Function Expressions (IIFEs):** Existe un tipo de función llamado IIFEs, usadas comúnmente patrones de diseño de software cómo module pattern , factory pattern etc... ICFEs tiene diferentes implementaciones

```
1// Classic
2(function(){})();
3
4// Crockford's favorite
```

```
(function(){}());

// Unary versions
+function(){}();

// Facebook version
!function(){}();

// Crazy version
!1%-+~function(){}();
```

- ***Inmutabilidad***: Tiene por objetivo hacer que los parámetros de un objeto sean no modificados o inmutables.

  El operador`===`nos ayuda a comparar objetos, ejecutando la comparación no directamente a los datos del objeto sino, la referencia del objeto. Cuando asignamos un objeto a otro estamos haciendo que ambos apunten a la misma referencia, por eso al modificar un objeto el otro también se verá afectado, porque ambos tienen la misma referencia de memoria.
- **Iteradores:** Nos permiten hacer listas infinitas de elementos haciendo los distintos arreglos, los cuales tienen un número finito de elementos definidos, siempre debe tener el método next`()`

Para los iterados podemos obtener lo siguiente:

```
next()—itera los datos, es una function

value()—devolver el valor del dato
```

```
5 done()—sera un indicador para cuando la lista se haya terminado y
reciba true o false. Por default siempre es false
```

Con los iteradores es muy fácil sencillo realizar un for y obtener los datos..

```
1for (let value of fibo){
2 console.log(value)
3}
```

- **JSON:** Es un objeto JavaScript con unos métodos de implementación que nos permite serializarlo para poder utilizarlo como intercambio de datos entre servicios. Las propiedades del objeto deben ser strings para poder codificarlo y decodificarlo. Los tipos name infinity se codifican como null. Si queremos usar el intercambio de datos debemos usar la función.

  `JSON.stringify()`// devuelve en un string la información del objeto que se le pasa por parámetro.
- **Let:** Define el ámbito de la `variable` en donde ha sido definida global o ámbito de una función.
- **Map:** Retorna un `array`, transforma los elementos de entrada y devuelve un`a array`, con los elementos transformados del mismo tamaño que el original. Además permite encadenar funciones, usar `sort, find, reduce,` etc.
- **Math:** Es un objeto de javascript que posee propiedades y métodos creados por constantes y funciones matemáticas. No es un objeto de función, no se puede editar, todos los métodos y propiedades son estáticos.

- **Math.floor:** Devuelve el máximo entero menor o igual a un número. Cómo `floor` es un método estático, siempre debe usarse como `Math.floor()`, en lugar de usarlo como método de un objeto.

```
Math.floor(5.7);  // Devuelve 5
Math.floor(5.4);  // Devuelve 5
Math.floor(5.1);  // Devuelve 5
```

- **Math.ceil:** La función devuelve el entero mayor o igual a un número dado. Como el ceil es un método estático, siempre debe usarse como `Math.ceil()`, en lugar de usarlo como método de un objeto.

```
Math.ceil(.60);       // 1
Math.ceil(-4.5);     // -4
Math.ceil(5.3);     // 5
```

- **Math.round:** La función retorna el valor de un número redondeado al entero más cercano. Si la porción fraccionaría del número es 0.5 o mayor, será redondeado al siguiente número entero superior. Si la porción de la fracción del número es menor a 0.5, será redondeado al siguiente entero inferior. Cómo round es un método estático, siempre debe usarse como `Math.round()`, en lugar de usarlo como método de un objeto.

```
Math.round(20.5);   // es igual a 21
Math.round(12.2.);   // es igual a 12
Math.round(-10.7);   // es igual a -11
```

- **Math.random:** Retorna un punto flotante, un número aleatorio dentro del rango `[0, 1]`.toma desde el número 0 (Incluido) hasta el 1 pero sin incluirlo, y se puede escalar hasta el rango deseado.

```
function random(){
  return Math.random()
}
```

- **MouseEvent:** Son eventos que se producen cuando el usuario interactúa con el mouse, algunos eventos comunes que se utilizan son:`click`,`dblclick`,`mouseup`,`mousedown`. MouseEvent deriva de `Event`. Aunque el método `MouseEvent.initMouseEvent()` se mantiene para la compatibilidad con versiones anteriores.
- **Mousedown:** Este evento se activa cuando se presiona el botón del mouse y se arrastra un elemento. Varía una operación de arrastrar y soltar; al iniciar una selección de texto; al iniciar una interacción de desplazamiento en combinación con el botón central del ratón, si es compatible. solo funciona en `<textarea> o <input/>`
- **Mouseup:** este evento ocurre cuando se deja de presionar el botón del mouse. También al invocar un menú contextual en combinación con el botón derecho del ratón, si es compatible. solo funciona en `<textarea> o <input/>`
- **NaN**: Un valor representando un Not a Number (No es un número).
- **Number:** Es uno de los 4 tipos de datos primitivos que podemos asignar a las variables y son utilizados a menudo para hacer cálculos matemáticos, comparaciones. Etc..
- **Parsear:** Significa transformar una entrada de texto a una estructura de datos que podamos usar para procesar dichos datos.

- **Scroll:** Se activa cuando la vista del documento se ha desplazado en forma vertical. Los eventos pueden dispararse a una velocidad alta, y el controlador de eventos no debería ejecutar operaciones muy costosas como modificaciones al DOM, para eso se recomienda usar requestAnimationFrame, setTimeout, customEvent.
- **Strings:** Son un tipo de `variable` primitiva en `JavaScript` y al igual que `number` tiene su propia `clase` y `métodos`. Se comporta como un `array`, es un conjunto de caracteres con índices que van desde el 0 para el primer carácter hasta el último.

Algunos métodos:

```
1"Ejemplo"[2] --- devuelve el tercer carácter "e"
2"Ejemplo".length --- devuelve el tamaño del string "7"
3"Ejemplo".charCodeAt() --- devuelve el carácter en formato UNICODE
4"Ejemplo".indexOf("em") --- devuelve el índice donde comienza el
carácter "em"
5"Ejemplo".substring(2, 4) --- devuelve la parte del string entre "e"
y "p"
6 Split() --- transforma el string en un array, pasándole como
parámetro el delimitador  (" ")
```

- **Variables:** Las variables en **JavaScript** son espacios de memoria donde almacenar temporalmente datos, que podemos acceder en algún momento de la ejecución de nuestro código.

```
var Numero;
var numero = 5;
```

JavaScript distingue entre mayúscula y minúsculas, así que var Numero; _es una variable diferente de var numero = 5;

- **While:** El bloque de código se ejercita mientras se cumpla la condición.

```js
let i = 1;

while(i < 10){

i++

console.log(i)

}
```