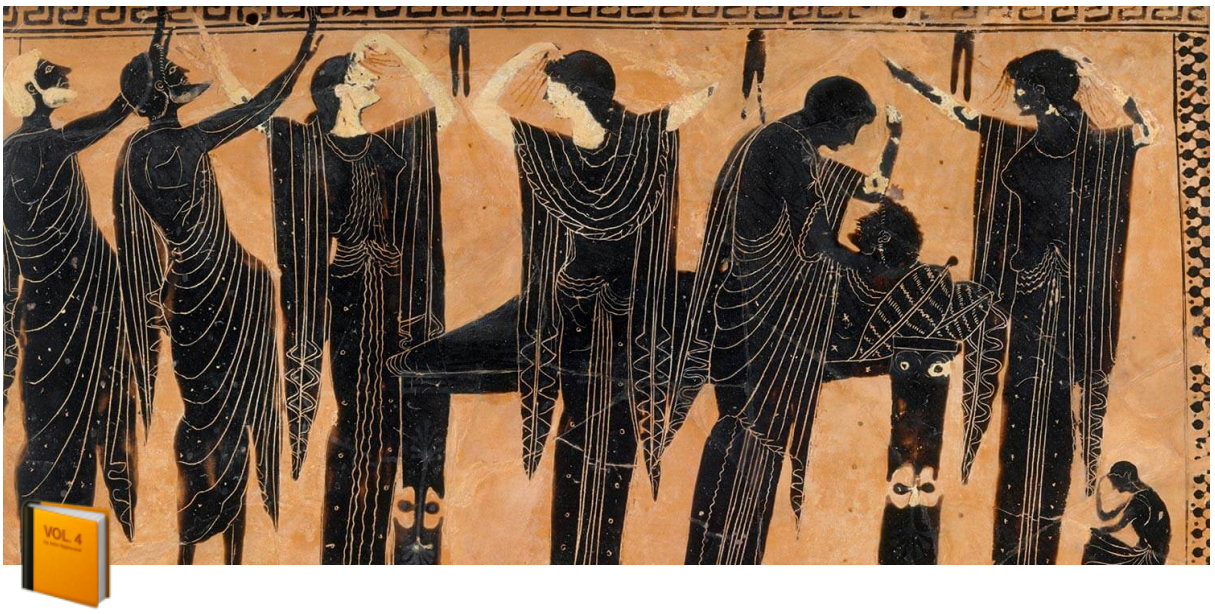

Guía de Estudio de JavaScript

- Es un lenguaje interpretado, es decir, no requiere compilación.
- Se ejecuta en el navegador del usuario. O sea se encarga de interpretar las sentencias JavaScript contenidas en una página HTML.
- Es un lenguaje orientado a eventos.
- Es un lenguaje basado en prototipos. Utiliza el concepto de prototipos para implementar o simular aspectos de la orientación a Objetos.

¿Cómo incluir JavaScript?

- Mediante la etiqueta <Script>. Escribiendo código dentro de las etiquetas o importando un archivo externo con el atributo “src”.
- El atributo “type” es opcional. Por defecto es js.
- Se puede incluir en el <head> o en el <body>. “Tener en cuenta el orden de ejecución”.

```
<script> type="text/javascript">
        function popup() {
            alert("Hello World")
</script>
```

```
<script src="principal.js"></script>
```

- También se puede incluir de manera externa:

```
/*Creo el archivo myScript.js*/
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}

/*Después lo llamo en HTML*/
<script src="myScript.js"></script>
```

Definición y uso

- El `Defer` Es un atributo booleano solo para scripts externos.
- Hay varias formas de ejecutar un script externo:

1. Si `async` esta presente (INTERNO), antes del **body** y en el mismo archivo. (El guion es descargado en paralelo al análisis de la página y ejecutando tan pronto como está disponible (antes de que se complete el análisis).
2. Si `defer` esta presente y no `async` (Si lo usas externo). El script se descarga en paralelo al análisis de la página y se ejecuta después de que la página haya terminado de analizarse.
3. Si ninguno `async` o `defer` está presente: El script se descarga y se ejecuta inmediatamente, bloquear el análisis hasta que se complete el script.

Sintaxis - Resumen

- Cuerpo de funciones **{ }**
- las sentencias opcionalmente terminan con **;**
- Cadenas de texto con **“...”, ‘...’ o `...`.**
- Comentarios: **/* */ o**
- Operadores: **+ - / * ** % ++ —**

- Asignación: **= += -= *= /= **= %=**
- Relaciones: **== === != !== < <= > >= ?**
- Lógicos: **&& || !**
- typeof instanceof

Operadores de asignación (=)

```
//EJEMPLO
let x = 5;
let y = 6;
let sum = x + y;
```

Operadores aritméticos (`+` `-` `*` `/`)

```
//EJEMPLO
5 * 10
```

JavaScript distingue entre mayúsculas y minúsculas

- sus identificadores son **sensible al caso**.

Las variables `lastName` y `lastname`, son variables diferentes:

```
//Ejemplo
let lastName = "Doe";
let lastname = "Peterson";
```

JAVASCRIPT LET

- Variables declaradas con `let` have **Alcance del bloque**
- Variables declaradas con `let` debe ser **Declarado** antes de usar
- Variables declaradas con `let` no puede ser **Redeclarado** en el mismo ámbito

EJEMPLO:
No se puede acceder a las variables declaradas dentro de un bloque {} Desde fuera del bloque:

```
{
  let x = 2;
}
```

EJEMPLO:

Variables declaradas con `var` Dentro de un bloque { } se puede acceder desde Fuera del bloque:

```
{
  var x = 2;
}
```

Tipos de Datos, , y Conversión de Tipos

JavaScript utiliza **tipos dinámicos**. Esto significa que la misma variable puede almacenar diferentes tipos de datos a lo largo de su ciclo de vida.

Los tipos de datos que encontramos en JS incluyen: `String`, `Number`, `Boolean`, `Object`, `Array`, `Date`, y `undefined`.

Tipos Primitivos (String, Number, Boolean, undefined)

1. String (Cadena de Caracteres)

◦ Las cadenas de texto se pueden definir utilizando comillas dobles (`"`), comillas simples (`'`) o acentos graves (`` ` ``).

◦ **Funciones de String:** Las funciones `search` y `replace` de String pueden utilizar expresiones regulares como parámetros.

2. Number (Número)

◦ Los literales numéricos se representan como 10 o 99.99.

◦ **Funciones de Number:** Existen funciones específicas para manejar el tipo `Number`.

3. Boolean

◦ Representa valores lógicos `true` o `false`.

4. undefined

◦ Uno de los tipos que puede albergar una variable. Los parámetros de función ausentes, por ejemplo, son considerados `undefined`.

El Operador

El operador `typeof` se utiliza para determinar el tipo de dato de una variable.

```
let nombre = "Juan";
console.log(typeof nombre); // ⇒ "string" [4]

let edad = 30;
console.log(typeof edad); // ⇒ "number"

let esActivo = true;
console.log(typeof esActivo); // ⇒ "boolean"

let miFuncion = function() {};
console.log(typeof miFuncion); // ⇒ "function" [9]
```

Conversión de Tipos

JS soporta la **conversión de tipos**. Esto a menudo ocurre implícitamente, pero también se puede realizar de forma explícita. Por ejemplo, el motor de JavaScript transforma un índice de *array* numérico en un *string* a través de una conversión implícita usando `toString`.

Estructuras de Control

Las estructuras de control definen el flujo secuencial de un programa.

1. Condicionales

- **if, else**: Permiten tomar decisiones basadas en una condición.

```
let temperatura = 25;
if (temperatura > 30) {
    console.log("Hace calor.");
} else {
```

```
    console.log("Temperatura agradable."); // Se ejecuta esta línea.
}
```

- **switch**: Evalúa una expresión contra múltiples casos posibles.

```
let dia = 3;
switch (dia) {
    case 1:
        console.log("Lunes");
        break;
    case 3:
        console.log("Miércoles"); // Se ejecuta esta línea.
        break;
    default:
        console.log("Otro día");
}
```

2. Bucles (Iteraciones)

Los bucles se utilizan para repetir tareas hasta que se cumple una condición.

- **for**: Se utiliza para iterar un número específico de veces.

```
for (let i = 0; i < 3; i++) {
    console.log(i); // 0, 1, 2
}
```

- **for in**: Itera sobre las propiedades enumerables de un objeto.
- **for of**: Itera sobre los valores de objetos iterables (como Arrays o Strings).
- **while**: Repite un bloque de código mientras una condición sea verdadera.
- **do while**: Ejecuta el bloque de código al menos una vez, y luego repite mientras la condición sea verdadera.

Manejo de Errores:

El manejo de errores en JavaScript mediante `try-catch-finally` es similar al de Java.

- **Bloques:**

◦ **try**: Contiene el código que se intenta ejecutar.

◦ **catch**: Si ocurre un error en el bloque `try`, el control pasa al bloque `catch`.

◦ **finally**: El código en este bloque se ejecuta siempre, haya ocurrido un error o no.

- **Errores:** Los errores predefinidos son objetos que tienen la forma `{ name: <String>, message: <String> }`.
- **Arrojar Errores:** La palabra clave `throw` permite arrojar errores de forma explícita.

```
try {
    // Intenta ejecutar este código
    if (alerta === undefined) {
        throw { name: "ErrorPersonalizado", message: "La variable no existe"
}; // Arrojar un error [5, 15]
    }
} catch (error) {
    // Captura el error y lo muestra
    console.log("Se capturó un error:", error.name, error.message);
} finally {
    // Este código siempre se ejecuta
    console.log("Proceso de manejo de error finalizado.");
}
```

Funciones

Las funciones son bloques fundamentales para desarrollar aplicaciones.

- **Declaración:** Se definen con la palabra clave `function`. El cuerpo de la función se define entre llaves `{}`.
- **Retorno:** El uso de `return` es opcional.
- **Almacenamiento:** Las funciones pueden ser almacenadas en variables.

```
// Función básica
function sumar(a, b) { // Uso de la palabra clave function [17]
    return a + b;
```

```
}

// Función almacenada en una variable
let multiplicar = function(a, b) { // La función es almacenada en una variabl
e [17]
   return a * b;
}
let res = multiplicar(3, 2); // res ⇒ 6 [17]
```

Paso de Parámetros

- **Tipado y Cantidad:** No se especifican los tipos de los parámetros ni del retorno. Tampoco hay control estricto por la cantidad de valores pasados.
- **Valores Ausentes:** Los parámetros ausentes se consideran `undefined`, a menos que se les asigne un valor por defecto.
- **Paso de Argumentos:** Los tipos simples se pasan por valor, mientras que los objetos se pasan por referencia.

Métodos Especiales de Funciones

- **func.toString():** Aplicar el método `.toString()` a una función devuelve la función completa como una cadena.
- **El objeto arguments:** Es un objeto especial, similar a un `Array`, que es accesible dentro de las funciones. Contiene los valores de todos los argumentos que fueron pasados a esa función.

◦ **arguments.length:** Permite conocer la cantidad de parámetros recibidos en la invocación de la función.

```
function mostrarArgs(x, y) {
   console.log(mostrarArgs.toString()); // Devuelve la función como string
[9]
   console.log(arguments.length); // Muestra cuántos argumentos se recibi
eron [9]
   console.log(arguments); // Muestra los valores de los argumentos [9]
}

mostrarArgs(1, "dos", true); // arguments.length será 3 [9]
```

Arreglos () y Colecciones

El objeto `Array` de JavaScript es un objeto global utilizado para construir *arrays*, que son objetos tipo lista de alto nivel.

Características

- **Naturaleza:** Los arreglos son una colección de elementos, y son considerados objetos en JS.
- **Heterogéneos:** Pueden contener elementos de cualquier tipo (son un conjunto heterogéneo).
- **Indexación:** Están basados en índice cero (0, 1, 2, ...).
- **Tamaño:** No tienen un tamaño fijo; su longitud y el tipo de sus elementos son variables.
- **Validación:** Se puede verificar si una variable es un array usando el método estático `Array.isArray(arr)`.

```
const autos = ["Fiat", "Volvo", "BMW"]; [21]
const nombres = [];
nombres = "Pepe";
nombres[24] = "Juan"; // Los arrays no tienen tamaño fijo [21]
console.log(Array.isArray(autos)); // true [21, 23]
```

Operaciones Comunes (Referencia Completa - Métodos de Instancia)

Operación	Método	Descripción	Fuente
Longitud	`arr.length`	Indica el número de elementos.	
Añadir al final	`arr.push(e)`	Inserta uno o más elementos al final. Devuelve la nueva longitud.	
Eliminar al final	`arr.pop()`	Elimina el último elemento y lo devuelve.	
Añadir al inicio	`arr.unshift(e)`	Añade uno o más elementos al inicio. Devuelve la nueva longitud.	
Eliminar al inicio	`arr.shift()`	Elimina el primer elemento y lo devuelve.	
Recorrer	`arr.forEach(f)`	Llama a una función para cada elemento.	

Buscar índice	`arr.indexOf(e)`	Devuelve el índice del primer elemento igual a `e`, o -1 si no existe.	
Modificar/Borrar	`arr.splice(pos, num)`	Añade, borra o modifica elementos en el array. Puede eliminar elementos a partir de una posición.	
Copiar	`arr.slice()`	Extrae una porción del array y devuelve un nuevo array.	
Transformar	`arr.map(f)`	Devuelve un nuevo array con el resultado de llamar a la función en cada elemento.	

Nota sobre splice(): Al usar `splice(pos, 1)` se elimina un único elemento en la posición especificada.

Objetos ()

Los objetos en JavaScript son una colección de valores nombrados (pares clave-valor).

- **Contenido:** Pueden contener datos (atributos) y también funciones (métodos).
- **Orientación a Objetos:** JavaScript es un lenguaje basado en prototipos que utiliza este concepto para simular aspectos de la Programación Orientada a Objetos (POO). Las clases introducidas en ECMAScript 2015 son una mejora sintáctica sobre la herencia basada en prototipos.
- **Acceso a Propiedades:** Se puede acceder a las propiedades usando la notación de punto o la notación de corchetes.

```
const persona = { // Colección de valores nombrados [31]
    nombre: "Juan",
    edad: 50,
    saludar: function() { // Pueden contener funciones [12]
        console.log("Hola!");
    }
};

persona.edad += 10; // Acceso con notación de punto [12]
```

```
console.log("Nombre " + persona.nombre);
console.log("Edad " + persona["edad"]); // Acceso con notación de corche
te (válido) [12]
```

Objeto

El objeto `Date` representa un momento específico en el tiempo.

- **Base de Tiempo:** Contiene un número que representa los milisegundos transcurridos desde el 1 de enero de 1970 (Epoch).
- **Creación de Fechas:** Se utiliza su constructor para crear instancias de fechas.
- **¡Cuidado con el Mes!** Al crear una fecha, el mes va de 0 (Enero) a 11 (Diciembre).
- **Obtención/Establecimiento de Partes:** Posee funciones con el formato `getXYZ` y `setXYZ` para obtener o establecer partes específicas de la fecha (segundos, minutos, día, etc.).

```
// Creación de una fecha (Diciembre es 11) [33]
let d = new Date(2022, 11, 25);
console.log("Navidad: " + d);

// Se utilizan métodos get/set, como getFullYear() o setHours() [15]
```

Expresiones Regulares ()

Las Expresiones Regulares permiten la búsqueda de patrones dentro de cadenas de texto. Son útiles para la búsqueda y la validación de formatos.

Sintaxis y Modificadores

La sintaxis básica es `/patrón/modificadores`.

Modificador	Significado	Fuente
`i`	Búsqueda no sensible a mayúsculas/minúsculas.	
`g`	Global, busca todas las apariciones en lugar de detenerse en la primera.	

`m`	Multilínea.	

Metacaracteres Comunes (Patrones)

Símbolo	Significado	Ejemplo	Fuente
`^`	Comienzo de línea	`/^mo/` (ej. "moderna", "mono")	
`$`	Final de una línea	`/ión$/` (ej. "Avión", "Acción")	
`\w`	Letra, Número o `_`	`/^\w\w\w$/` (ej. "sol", "p2p")	
`\d`	Dígito	`/\d\d\d/` (ej. "123", "555")	
`\s`	Espacio en blanco	`/a\sa/` (ej. "a a")	
`( )`	Agrupar	`/^(sol)$/` (ej. "sol")	
`` ` ``	`` ` ``	Opciones (OR)	`/^(L
`[ ]`	Conjunto de caracteres	`/le[aeo]/` (ej. "lea", "lee", "leo")	
`?`	Opcional (cero o una vez)	`/profe(sor)?/` (ej. "profe", "profesor")	
`+`	Una o más veces	`/(a)+h/` (ej. "ah", "aah")	
`*`	Cero o más veces	`/(o)*/` (ej. "", "o", "oo")	
`{n,m}`	Entre n y m veces	`/(ja){1,3}/` (ej. "ja", "jaja", "jajaja")	
`{n,}`	Al menos n veces	`/Gracia(s){2,}$/` (ej. "Graciass")	

Uso con Métodos

- **test():** La función `test` permite ver si una cadena cumple con el patrón de la expresión regular. Devuelve `true` o `false`.

```
let patron_mail = /^\w+@\w+(\.\w{2,4})+$/;
let email = "info@algo.com.ar";
patron_mail.test(email); // ⇒ true [6]
```

- **search() y replace():** Las funciones de string `search` y `replace` pueden usar expresiones regulares como parámetros.

Cuadros de Diálogo (Bloqueantes)

Los cuadros de diálogo son una forma nativa de comunicación del navegador. Son **bloqueantes**, lo que significa que el código JavaScript se detiene hasta que el usuario interactúa con el cuadro. No permiten cambiar su estilo.

1. **alert():** Muestra un mensaje simple ("Cartel").

2. **confirm():** Muestra un mensaje y pide confirmación al usuario. Devuelve un valor booleano (`true` o `false`).

3. **prompt():** Muestra un mensaje y pide una entrada de texto al usuario. Devuelve el valor introducido por el usuario (un string).