# Java Script - CheckPoint7

## ¿Qué diferencia a Javascript de cualquier otro lenguaje?

JavaScript es un lenguaje de programación, de secuencias de comandos, aporta soluciones eficaces en la mayoría de los ámbitos de la tecnología. Por este motivo se ha convertido en una de las herramientas más populares y esenciales para el desarrollo web. Permite crear contenido de actualización dinámica, controlar multimedia, animar imágenes, etc...

La gran diferencia con otros lenguajes de programación es que podemos ejecutar javascript directamente en cualquier navegador, no es necesario ningún programa, aplicación o herramienta para comenzar a programar en JavaScript.

Ejemplo de un mensaje por consola:

```
console.log('Mensaje Consola')
Mensaje Consola
```

A continuación las ventajas y desventajas de JavaScript:

### Ventajas de JavaScript

- Interactividad en tiempo real: Se refiere a la capacidad de una página web para responder y adaptarse a las acciones del usuario en el momento en que ocurren, de forma inmediata, sin la necesidad de recargar la página por completo. Esto crea una experiencia de usuario más dinámica, atractiva y fluida.
- Detección y respuesta a eventos: Los eventos son acciones (hacer clic en un botón, mover el ratón sobre un elemento o escribir en un campo de texto) todos ellos ocurren en una página web. Este lenguaje permite detectar y responder a estos eventos mediante controladores de eventos. Esto sirve, por ejemplo, para mostrar información adicional cuando un usuario pasa el ratón sobre un elemento o para ejecutar una acción específica al hacer clic en un botón.
- Manipulación sencilla del DOM: El DOM (Document Object Model) es una representación en la memoria de la estructura y contenido de una página web. JavaScript agiliza el acceso y la manipulación de elementos del DOM, es decir, te ayuda a cambiar el contenido, estilo y atributos de los elementos en respuesta a eventos. Por ejemplo, puedes cambiar el texto de un elemento, modificar su estilo visual o agregar elementos nuevos al DOM.
- Creación de formularios interactivos: Puedes mejorar la interacción con formularios al validar datos ingresados por el usuario. De esta manera, mostrarás mensajes de error instantáneos cuando un

usuario ingresa datos incorrectos o incompletos en un campo del formulario.

- Innovación en la carga de contenido dinámico: Mediante la técnica de AJAX (Asynchronous JavaScript and XML), esta herramienta carga y muestra contenido adicional en una página sin recurrir a una recarga. Esto es útil para mostrar comentarios, resultados de búsqueda o actualizaciones de forma rápida.
- Animación y efectos visuales para los sitios web: JavaScript se utiliza para crear animaciones y efectos visuales en una página web. De esta manera, puedes hacer que los elementos se muevan, cambien de tamaño, se desvanezcan y realicen otras transiciones de manera suave y atractiva.
- Desarrollo de juegos y aplicaciones interactivas: También es una gran plataforma para desarrollar juegos y aplicaciones interactivas dentro de los navegadores. Muchos videojuegos en línea, simuladores y apps educativas utilizan este lenguaje de programación para crear experiencias inmersivas y entretenidas.
- Comunicación inmediata entre el cliente y el servidor: Con tecnologías como WebSockets, esta herramienta permite establecer conexiones de comunicación bidireccional entre el cliente y el servidor. Esto es esencial para aplicaciones en tiempo real, como chats en línea y colaboración en tiempo real.
- Amplia adopción en el desarrollo web: JavaScript es uno de los lenguajes de programación más populares en el mundo del desarrollo web. Esto significa que hay una gran cantidad de recursos, tutoriales y comunidades en línea disponibles para sus usuarios.
- Ejecución constante en el lado del cliente: Este lenguaje de programación puede realizar acciones sin necesidad de comunicarse de manera constante con el servidor. Esto ayuda a reducir la carga del servidor y mejora la eficiencia de las aplicaciones.
- Comunicación asíncrona con el servidor: Asimismo, permite la comunicación asíncrona con el servidor a través de tecnologías como AJAX (Asynchronous JavaScript and XML). Es decir, los datos pueden cargarse o enviarse al servidor en segundo plano.
- Uso versátil para desarrolladores: Con la introducción de Node.js, JavaScript también se puede utilizar en el lado del servidor, lo que permite a los desarrolladores utilizar el mismo lenguaje en toda la pila de tecnología.
- Amplio repertorio de bibliotecas y frameworks: Existen diversas bibliotecas y frameworks de JavaScript (como jQuery, React, Angular y Vue.js) que simplifican tareas comunes como la manipulación del DOM (Document Object Model), el desarrollo de interfaces de usuario y la creación de acciones automatizadas programables.
- Facilidad de aprendizaje: JavaScript es fácil de aprender desde cero, ya que cuenta con una sintaxis clara y flexible.
- Evolución continua en cada nueva versión: Este lenguaje de programación evoluciona y mejora de forma constante. Sus nuevas versiones, como ECMAScript 6 (ES6) y posteriores, introducen características más modernas y poderosas que permiten a los desarrolladores escribir un código más limpio y eficiente.

- Compatibilidad con múltiples navegadores: Los navegadores modernos han mejorado de manera significativa en términos de soporte para el JavaScript estándar, lo que facilita la creación de aplicaciones consistentes en diferentes plataformas.

### Desventajas de JavaScript

- Falta de compatibilidad entre navegadores: Pese a que la compatibilidad entre navegadores ha mejorado, todavía puede haber diferencias en cómo interpretan y ejecutan el código JavaScript. Esto puede requerir ajustes y pruebas adicionales para garantizar un funcionamiento uniforme en diferentes navegadores.
- Nivel de seguridad deficiente: Como su código se ejecuta en el navegador del usuario, existe el riesgo de que algunos exploten sus vulnerabilidades para llevar a cabo ataques, como inyección de código o cross-site scripting (XSS). Por ello, los desarrolladores deben tomar medidas para asegurar su código y prevenir estos ataques.
- Bajo rendimiento en aplicaciones complejas: Esta herramienta es un lenguaje interpretado, lo que puede afectar el rendimiento en comparación con lenguajes compilados. Aunque los navegadores modernos han mejorado su velocidad de ejecución, el rendimiento puede ser un problema en aplicaciones web muy complejas o en dispositivos con recursos limitados.
- Funcionalidad sujeta al uso del cliente: Si un usuario deshabilita este recurso en su navegador o este no lo admite, la funcionalidad de la aplicación puede verse afectada.
- Dificultad para indexar en motores de búsqueda: Los motores de búsqueda suelen tener dificultades para indexar contenido generado dinámicamente por este lenguaje de programación. Aunque algunos han mejorado en este aspecto, todavía puede haber desafíos para garantizar que el contenido sea indexado de forma correcta.
- Mantenimiento y legibilidad complicados: A medida que las aplicaciones web se tornan más complejas o grandes, el código JavaScript puede volverse difícil de mantener y entender. Esto puede llevar a problemas de legibilidad, errores y dificultades para trabajar en equipo con proyectos grandes.
- Carga inicial prolongada y rendimiento deficiente en dispositivos móviles: Si no se optimiza de manera adecuada, los códigos creados con este tipo de lenguaje pueden aumentar los tiempos de carga de la página, lo que afecta la experiencia del usuario. En dispositivos móviles con conexiones más lentas, esto puede ser problemático.
- Obsolescencia de las versiones antiguas: Algunas características de las versiones antiguas de JavaScript pueden volverse obsoletas o no recomendadas. Esto puede ocasionar problemas de compatibilidad y dificultades en la migración de código.

## ¿Cuáles son los tipos de datos JS?

- Undefined : Representa una variable que no ha sido declarada o a la cual no se le ha asignado un valor.

Ejemplo:

```
let x;
```

- Boolean : Representa un valor lógico, dos posibles valores; true o false.

Ejemplo:

```
let is_true = true;
let is_false = false;
```

- Number : Permite representar y manipular valores numéricos.

Ejemplo:

```
let age = 48;
```

- String : Representa datos de texto (cadenas de caracteres).

Ejemplo:

```
let name = "Sonia";
```

- BigInt : Representa valores numéricos que son demasiado grandes para ser representados por el tipo de dato number.

Ejemplo:

```
const number = 9007199254740993n;
```

- Symbol : Es un valor primitivo único e inmutable.

Ejemplo:

```
let sym = Symbol("foo");
```

- Null : Representa la ausencia intencional de cualquier valor, un valor nulo o «vacío».

Ejemplo:

```
let is_null = null;
```

- Object : Representa una colección de datos definidos y entidades más complejas.

Ejemplo:

```
let person = {"name": "Sonia", "age": 48, "isAdmin":true};
```

- Function : Es una forma abreviada para funciones, aunque cada constructor de funciones se deriva del constructor Object. Son objetos con la capacidad de ser ejecutables.

Ejemplo:

```
const Hola = (a) => {
   return "Hola " + a;
};
```

Para hacer la llamada:

```
Hola("Sonia")
```

## ¿Cuáles son las funciones de strings más comunes en JS?

### Length

Encontrar la longuitud de una cadena.

Sintaxis:

```
str.length
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
textoTipico.length;
```

Devuelve:

```
43
```

### CharAt

Extrayendo un caracter específico de la cadena.

> ***NOTE:*** Recuerde las posiciones empiezan contando desde el 0.

Sintaxis:

```
str.charAt(indice)
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
```

Dependiendo de la posición que pidamos nos mostrará el caracter de esa posición:

```
textoTipico.charAt(4);
q
```

```
textoTipico.charAt(0);
T
```

Si pasamos una posición mayor entonces nos devuelve una cadena vacía.

```
textoTipico.charAt(300);
''
```

## Concat

Concatena, une dos o más cadensa o arrays, es decir, se puede usar con cadenas o arreglos

Sintaxis:

```
str.concat(str2 [, ...strN])
```

- Cadenas

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let textoPlus = " again and again";
```

```
textoTipico.concat(textoPlus);
```

Devuelve:

```
'The quick brown fox jumps over the lazy dog again and again'
```

- Arrays

Ejemplo:

```
let aTipico =  ["Cecilie", "Lone"];
let arr2 = ["Emil", "Tobias", "Linus"];
let arr3 = ["Robin"];

aTipico.concat(arr2, arr3);
```

Devuelve:

```
(6) ['Cecilie', 'Lone', 'Emil', 'Tobias', 'Linus', 'Robin']
```

## Includes

Comprueba si una cadena esta incluida en otra cadena de texto, devolviendo true ó false según corresponda.

> ***NOTE:*** Recuerde JavaScript es un lenguaje que distingue entre mayúsculas y minúsculas..

Sintaxis:

```
str.includes(searchString[, position])
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let word = 'fox';

textoTipico.includes(word);
```

Devuelve:

```
true
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let word = 'foxy';

textoTipico.includes(word);
```

Devuelve:

```
false
```

## StartWith

El método startsWith() indica si una cadena de texto comienza con los caracteres de una cadena de texto concreta, devolviendo true o false según corresponda.

> ***NOTE:*** Recuerde JavaScript es un lenguaje que distingue entre mayúsculas y minúsculas..

Sintaxis:

```
str.startsWith(stringBuscada[, posicion])
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let texto = 'The';

textoTipico.startsWith(texto);
```

Devuelve:

```
true
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let texto = 'the';
```

```
textoTipico.startsWith(texto);
```

Devuelve:

```
false
```

## EndsWith

El método endsWith() determina si una cadena de texto termina con los caracteres de una cadena indicada, devolviendo true o false según corresponda.

> ***NOTE:*** Recuerde JavaScript es un lenguaje que distingue entre mayúsculas y minúsculas..

Sintaxis:

```
str.endsWith(searchString[, position])
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let texto = 'lazy dog';

textoTipico.endsWith(texto);
```

Devuelve:

```
true
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let texto = 'lazy doggy';

textoTipico.endsWith(texto);
```

Devuelve:

```
false
```

## Replace

El método replace() reemplaza palabras de una cadena por otra que le pasemos.

> ***NOTE:*** Este método no cambia el valor de la cadena sobre la que se realiza la llamada. Devuelve una nueva cadena.

Sintaxis:

```
str.replace(patrón, reemplazo)
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
textoTipico.replace("dog", 'cat');
```

Devuelve:

```
'The quick brown fox jumps over the lazy cat'
```

## Repeat

El método repeat() construye y devuelve una nueva cadena que contiene el número especificado de copias de la cadena en la cual fue llamada, concatenados.

Sintaxis:

```
str.repeat(count)
```

Ejemplo:

```
let greeting = "Hi World";
greeting.repeat(2);
```

Devuelve:

```
'Hi WorldHi World'
```

## Match

El método match() devuelve todas las ocurrencias de una expresión regular dentro de una cadena.

Sintaxis:

```
str.match(regexp)
```

Ejemplo que obtiene solo las letras mayúsculas según una expresión regular:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";
let regex = /[A-Z]/g;
let found = textoTipico.match(regex);
```

Devuelve:

```
['T']
```

### IndexOf

El método indexOf() devuelve el índice ó posición, dentro del objeto String que realiza la llamada, de la primera ocurrencia del valor especificado, comenzando la búsqueda desde indice que puede ser un entero entre 0 y la longitud de la cadena. El valor predeterminado es 0. Devuelve -1 si no se encuentra dicho valor.

Sintaxis:

```
str.indexOf(valorBusqueda[, indiceDesde])
```

Ejemplo:

```
let textoTipico = "The quick brown fox jumps over the lazy dog";

textoTipico.indexOf("quick"); // returns 4
textoTipico.indexOf("pink"); // returns -1
textoTipico.indexOf("quick", 0); // returns 4
textoTipico.indexOf("quick", 5); // returns -1
```

## ¿Qué es un condicional?

Al hacer un programa necesitaremos establecer condiciones o decisiones, donde buscamos que se realice una acción A si se cumple una condición o una acción B si no se cumple. Este es el primer tipo de estructuras de control que encontraremos.

Tenemos varias estructuras de control condicionales:

| Syntax | Description |
|---|---|
| If | Condición simple: Si ocurre algo, haz lo siguiente... |
| If/else | Condición alternativa: Si ocurre algo, haz esto, sino haz esto otro. |
| ?: | Operador ternario: Equivalente a If/else. |
| switch | Esctructura para casos específicos: Equivalente a If/else. |

> ***NOTE:*** Cuando escribimos código, por lo general, se lee de forma secuencial, es decir, una línea detrás de otra, desde arriba hacia abajo.

A continuación vamos a centrarnos en las condiciones simples y alternativas.

## If

Condición simple: Si ocurre algo, haz lo siguiente...

Ejemplo:

```
let age = 19;

// Condición (si soy mayor de edad)
if (age >= 18) {
  console.log("¡Puede entrar a la disco!");
}
```

Devuelve:

```
'¡Puede entrar a la disco!'
```

## If/else

Condición alternativa: Si ocurre algo, haz esto, sino haz esto otro.

> ***NOTE:*** Es posible que necesitemos crear un condicional múltiple con más de 2 condiciones. Para ello, podemos anidar varios if/else uno dentro de otro. Sin embargo, anidar de esta forma varios if suele ser muy poco legible y produce un código repetitivo. En algunos casos se podría utilizar otra estructura de control llamada switch.

Ejemplo:

```
let age = 13;

// Condición (si soy mayor de edad)
if (age >= 18) {
```

```
  console.log("¡Puede entrar a la disco!");
} else {
  console.log("No puede entrar")
}
```

Devuelve:

```
'No puede entrar'
```

## Switch

La estructura de control switch permite definir casos específicos a realizar cuando la variable expuesta como condición sea igual a los valores que se especifican a continuación mediante cada case:

> ***NOTE:*** El switch comienza evaluando el primer case, y continua con el resto, hacia abajo.Observa que algunos case tienen un break. Esto hace que deje de evaluar y se salga del switch. El caso especial default es como un else. Si no entra en ninguno de los anteriores, entra en default.

Ejemplo:

```
let nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");

switch (nota) {
  case 10:
    calificacion = "Sobresaliente";
    break;
  case 9:
  case 8:
    calificacion = "Notable";
    break;
  case 7:
  case 6:
    calificacion = "Bien";
    break;
  case 5:
    calificacion = "Suficiente";
    break;
  case 4:
  case 3:
  case 2:
  case 1:
  case 0:
    calificacion = "Insuficiente";
    break;
  default:
    // Cualquier otro caso
    calificacion = "Nota errónea";
    break;
```

```
}

console.log("He obtenido un", calificacion);
```

## ¿Qué es un operador ternario?

El operador ternario es una forma alternativa y compacta de escribir una condición.

Sintaxis:

```
condición ? valor verdadero : valor falso;
```

Ejemplo:

```
let age = 13;
age >= 18 ? console.log("¡Puede entrar a la disco!") : console.log("No
puede entrar");
```

Devuelve:

```
'No puede entrar'
```

## ¿Declaración de función o expresión de función?

Las funciones son valores. Se pueden asignar, copiar o declarar en cualquier lugar del código. Si la función se declara como una declaración separada en el flujo del código principal, eso se llama Declaración de función. Si la función se crea como parte de una expresión, se llama Expresión de función.

Ejemplo declaración de función:

```
function gretting(name) {
  console.log( `Hola, ${name}` );
}
```

Llamada:

```
gretting("Sonia"); // Hola, Sonia
```

Ejemplo expresión de función:

```
let gretting = function(name) {
  console.log( `Hola, ${name}` );
};
```

Llamada:

```
gretting("Sonia"); // Hola, Sonia
```

## ¿Qué es la palabra clave 'this'?

This es una palabra clave muy utilizada dentro de funciones y clases, pues tiene un valor flexible. Cómo su nombre indica, hace referencia al objeto en cuestión. Es decir, si estamos creando cualquier función, la palabra clave this se usará para representar o llamar al objeto que dicha función está modificando.

### This como objeto global

Si se coloca código fuera de cualquier función e intentamos ver el contenido de this, entonces hablamos del contexto global.

Ejemplo:

```
console.log(this);
```

Si está en un navegador nos devolverá el objeto "window".

### This en el contexto de una función

> ***NOTE:*** No se puede saber el valor que tendrá la variable "this" al ver el código de una función, porque depende de cómo se haya invocado a esa función.

A continuación algunos ejemplos:

- Si la función es el método de un objeto y se invoca, this es el objeto en sí mismo.

```
Ejemplo:
const me = {
  name: 'Sonia D.',
  sayMyName() {
    return this.name
  },
}

console.log(me.sayMyName()) // 'Sonia D.'
```

- Cuando se define una función y luego se asigna como método de un objeto, this dentro de la función se refiere al objeto al que se ha asignado la función.

```
function sayHello() {
  console.log(`Hola, soy ${this.name}.`)
}

const person = {
  name: 'Sonia',
  greet: sayHello, // asignamos 'sayHello' a la propiedad 'greet'
}

person.greet() // Hola, soy Sonia.
```