**Consulta JavaScript**

**Análisis y Desarrollo de Software**



**Ficha: 2502319**

**Presentado por:**

**David González Escobar**

**Presentado a :**

**David Lozada**

**2 de Marzo de 2022**

1. **Introducción**

JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado justo a tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat. JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo programación funcional). Lee más en acerca de JavaScript.

Esta sección está dedicada al lenguaje JavaScript en sí, y no a las partes que son específicas de las páginas web u otros entornos host. Para información acerca de APIs específicas para páginas Web, consulta APIs Web y DOM.

1. **Versiones**

| **Año** | **Versión** |
| --- | --- |
| Mayo 1995 | Mocha |
| Septiembre 1995 | Live Script |
| Diciembre 1995 | Java Script |
| 1996-1997 | ECMA-262 Ed.1 ES1 |
| 1998 | ES2 |
| 1999 | ES3 Moderno js baseline |
| 2005 | Ajax Revolución |
| 2008 | ES4 RIP |
| 2009 | ES5 |
| 2012 | ES6 Under way |
| 2015 | ES6 |
| 2016 | ES7 |
| 2017 | ES8 |
| 2018 | ES9 |
| 2019 | ES10 |
| 2020 | ES11 |

1. **Sintaxis**

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

* No se tienen en cuenta los espacios en blanco y las nuevas líneas: como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)
* Se distinguen las mayúsculas y minúsculas: al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.
* No se define el tipo de las variables: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
* No es necesario terminar cada sentencia con el carácter de punto y coma (;): en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter ;. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (;).
* Se pueden incluir comentarios: los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

JavaScript define dos tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Ejemplo de comentario de una sola línea:

*// a continuación se muestra un mensaje*

alert("mensaje de prueba");

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

Ejemplo de comentario de varias líneas:

*/\* Los comentarios de varias líneas son muy útiles*

*cuando se necesita incluir bastante información*

*en los comentarios \*/*

alert("mensaje de prueba");

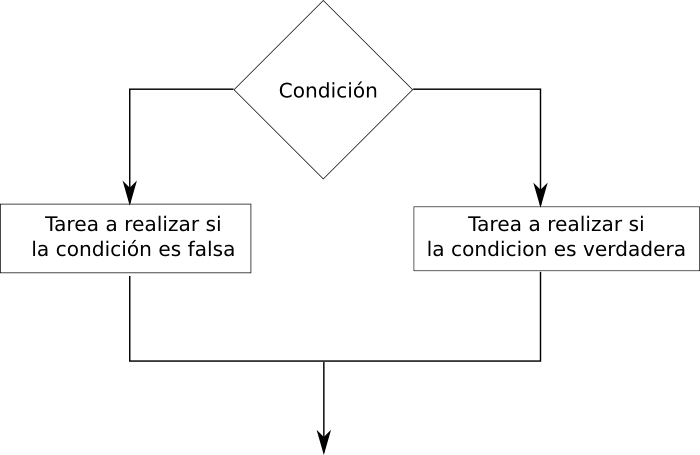
Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos /\* y \*/.

1. **Tipos de datos**

* Undefined
* Boolean
* Number
* String
* BigInt
* Symbol
* Null
* Object
* Funcion

1. **Estructuras de control**

En javascript disponemos de 2 tipos de estructuras de control:

* **Estructuras condicionales.** Este tipo de estructura de control tiene como objetivo realizar una bifurcación del flujo de instrucciones. Cuando el programa llega a un punto, nosotros establecemos una condición en en función de la misma el programa seguirá ejecutando unas instrucciones u otras; tal y como hemos visto en el anterior ejemplo cuando veíamos un caso de control de acceso sencillo.   
  
* **Estructuras de control de errores**. Estas estructuras son clave en el mundo de la seguridad informática. Son aquellas que **permiten controlar los errores que el usuario final comete de forma fortuita o intencionada** y poder seguir trabando de forma normal. Los errores más típicos a tener en cuenta se producen cuando pedimos al usuario que introduzca un número pero el usuario nos introduce una letra. Esto producirá un error en nuestro programá y hará que se comporte de forma inadecuada si no hemos controlado mediante una estructura de control de errores este caso.

A continuación pasamos a ver en detalle la sintáxis de cada una de las **estructuras de control javascript con ejemplos prácticos**.

## **Estructura de control javascript condicional (IF ELSE)**

Las estructuras condicionales en javascript nos sirven para **tomar decisiones en función de una condición** que nosotros establecemos. Su sintáxis es así:

*if (condicion)*

*{*

*instrucciones que se ejecutarán si se cumple la condición*

*}*

*else*

*{*

*instrucciones que se ejecutarán si NO se cumple la condición*

*}*

«If» es una sentencia que significa “si condicional”. La idea es que si sucede tal cosa, (si la condición es verdadera) se debe ejecutar la sentencia que le sigue, es decir la misma sólo se ejecutaría en caso de que la expresión de tipo Boolean sea verdadera. Si es falsa, el interprete pasará a la parte «else» que signfica «sino»; y ejecutará las instrucciones que existen dicha parte. La parte «else» no es obligatoria, pero aquí la mostramos para que sepáis que podemos ejecutar unas instrucciones en caso de que se cumpla la condición u otras en caso contrario. Os dejamos con un ejemplo práctico en el que solicitamos la edad al visitante y le dejamos entrar o no en base a su mayoría de edad:

*<script>*

*var edad = prompt(«Dime tu edad»);*

*if (num >17) {*

*alert(‘Eres mayor de edad, puedes acceder’);*

*}else {*

*alert(‘Eres menor de edad; NO puedes acceder’);*

*}*

*</script>*

## **Estructura de control javascript condicional (SWITCH)**

No en todos los casos nos es suficiente una estructura de control que nos permita realizar una acción si se cumple una condición u otra acción si no se cumple. A veces nos ocurrirá que debemos hacer unas acciones si una variables tiene un valor, otras si tiene un valor distinto y otras si tiene otro valor distinto al anterior. Es el caso típico de los menús de elección de opciones. En función de la opción elegida por el usuario nosotros debemos hacer lo que nos pide. Para estos casos entre otros muchos se crearon las estructuras de control SWITCH; cuya sintáxis es así:

*switch (expresion){*

*case valor1: sentencia1;*

*break;*

*case valor2: sentencia2;*

*break;*

*…*

*case valorN: sentenciaN;*

*break;*

*default: sentenciaFinal;*

*break;*

*}*

Como véis según el valor de la expresión se ejecutarán unas sentencias u otras y en el caso de que la expresión de un valor distinto a aquellos que tenemos previstos se ejecutará aquello que tengamos en la parte final (default). Vamos a ver un ejemplo mediante el cual nosotros convertimos un mes dado en formáto numérico (del 0 al 11) en formato texto (enero, febrero….diciembre):

*<script>*

*var hoy = new Date()*

*var nombremes*

*var mes=hoy.getMonth()*

*switch (mes) {*

*case 0 :*

*nombremes=» Enero»*

*break*

*case 1 :*

*nombremes=» Febrero»*

*break*

*case 6 :*

*nombremes=» Julio»*

*break*

*case 9 :*

*nombremes=» Setiembre»*

*break*

*default : nombremes= «… no se el mes»*

*}*

*alert(«Estamos en el mes llamado» + nombremes)*

*</script>*

Faltarían añadir los 12 meses, pero para no extender mucho el ejemplo te dejamos a ti la tarea de completar los meses que faltan y así probar el ejemplo completo en tu ordenador. Como habrás podido observar podrías haber logrado el mismo objetivo creado 12 «IFs» y habrías obtenido el mismo resultado pero esa solución a parte de denotar que no conoces la estructura de control SWITCH solo conseguría alargar tu código; hacerlo menos legible y por ende más difícil de mantener.

## **Estructura de control javascript bucle WHILE**

Pasamos a ver la primera estructura de repetición en javascript; el bucle WHILE. La idea principal con la que se han creado estos bucles es: MIENTRAS se cumpla la condición REALIZAR estas acciones. Cuando la condición deje de cumplirse salimos del bucle y continúa el flujo del programa. Su sintáxis es:

*while (expresionBooleana) {*

*sentencia;*

*}*

Una característica a tener muy en cuenta a la hora de decidirse a utilizar este tipo de bucles es que **la condición es lo primero que se evalua**. En el siguiente ejemplo vamos a mostrar los números del 1 al 9 haciendo uso del bucle WHILE:

*<script>*

*var i = 0;*

*while (i<9) {*

*alert (i);*

*i++;*

*}*

*alert(‘»Salimos» del while porque i vale: ‘ +i)*

*</script>*

## **Estructura de control javascript bucle DO WHILE**

Este bucle es exactamente igual que el anterior pero con la diferencia de que **la condición se comprueba al final**. Este tipo de bucles son muy utilizados cuando creamos un menú de opciones; mientras el usuario no elija la opción de salir del programa seguiremos trabajando con el. Por ello necesitamos comprobar la opción que el usuario a elegido al final del programa, ya que al principio aún no sabemos la opción que ha escogido. Vamos a ver un ejemplo de cómo podemos calcular el factorial de un número con el uso de este bucle:

*<script>*

*var resultado = 1;*

*var numero = 5;*

*do {*

*resultado = resultado \* numero;*

*numero–;*

*} while(numero > 0);*

*alert(resultado);*

*</script>*

## **Estructura de control javascript bucle FOR**

En los bucles vistos hasta ahora nosotros no conociamos el número de iteraciones que se darían en el bucle, ya que continuar dentro del bucle o salir es algo que depende de una condición (elegir una opción de un menú, introducir una contraseña válida….). Pero a veces nosotros sabemos exactamente el número de iteraciones que queremos hacer; no dependen de una condición. Para estos casos existe el bucle «FOR» cuya sintáxis es así:

*for (inicio; test; incremento)*

*{*

*sentencia;*

*}*

Estos bucles son muy utilizados para recorrer listas de objetos. Si nosotros tenemos una lista de 10 variables y queremos hacer algo sobre cada una de las variables podemos hacer un bucle for que hará las 10 iteraciones que necestiamos. En el siguiente ejemplo a reescribir ejemplo anterior que mostraba los 10 primeros números con un blucle WHILE pero esta vez con un bucle FOR:

*<script>*

*for (i=0; i<10; i++)*

*{*

*alert (i);*

*}*

*</script>*

¿Por qué hemos reescrito el mismo ejemplo? Por una sencilla razón. No hay bucles mejores ni bucle peores, cada uno tiene sus características y se adaptan mejor o peor en cada caso en base a nuestras necesidades. Es muy fácil convertir un bucle en otro y que el programa siga manteniendo intacto su comportamiento. Un buen programador sabrá en cada caso qué bucle utilizar. Sabrá si necesita comprobar la condición al inicio o al final, si sabe el número de iteraciones que debe realizar es fijo o depende de una condición… etc.

## **Estructura de control de errores javascript TRY CATCH**

Estas sentencias se utilizan para ejecutar código de forma segura, ya que en el caso de que existan excepciones, podremos capturarlas y seguir ejecutando normalmente el código, **Una excepción es un error que se produce por algún motivo determinado durante la ejecución de un script**, ese error comúnmente es mostrado al usuario, en este caso será lanzado para «afuera», en donde la siguiente sentencia (match) lo capturará, impidiéndole salir de la manera nativa al usuario, permitiéndonos trabajar con el error o bien mostrar un mensaje más amigable y claro a los ojos del mismo. Veamos un ejemplo de uso de esta estructura de control:

*<script>*

*try*

*{*

*document.write(10/variable1) ; //Variable insegura del usuario (puede introducir un 0)*

*}catch(e)*

*{*

*alert(e.message); // Mensaje en caso de error*

*}*

*</script>*

Todas las instrucciones que tengamos en la sección TRY se ejecutarán de forma segura; si el usuario mete un 0 variable1 valdrá 0 y el programa intentará dividir por 0. Como esto no es posible si no hubieramos hecho esta instrucción dentro del TRY el programá fallaría y dejaría de funcionar. Como hemos tenido la precaución de utilizar el TRY solo se producirá una excepción que se recogerá dentro del apartado CATCH y se mostrará el error permitiendo al programa trabajar de forma de forma normal.

Ahora te recomendamos que hagas distintos ejemplos en los que puedas utilizar y ganar experiencia en el uso de todas un cada una de las estructuras de control javascript.

**Bibliografía**

[**https://developer.mozilla.org/es/docs/Web/JavaScript**](https://developer.mozilla.org/es/docs/Web/JavaScript)

[**https://lenguajejs.com/javascript/introduccion/ecmascript/**](https://lenguajejs.com/javascript/introduccion/ecmascript/)

**https://blog.hubspot.es/website/tipos-de-datos-javascript**