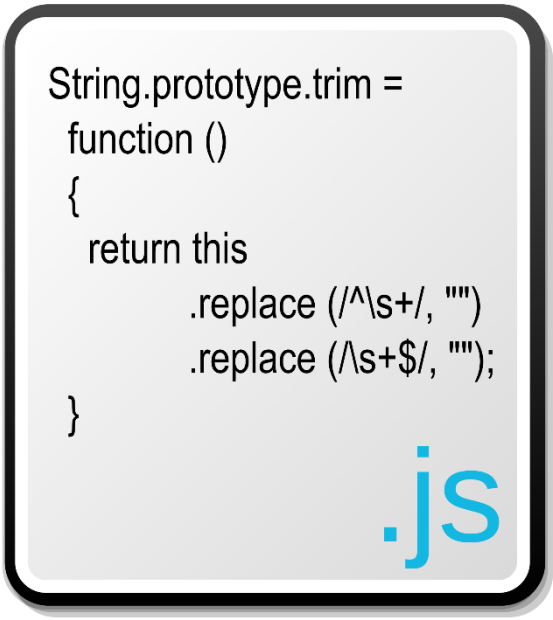
TEMA 2

INTRODUCCIÓN AL LENGUAJE JAVASCRIPT

OBJETIVOS DEL CAPÍTULO

- Conocer las principales características del lenguaje JavaScript .
- Dominar la sintaxis básica del lenguaje .
- Comprender y utilizar los distintos tipos de variables y operadores presentes en el lenguaje JavaScript.
- Conocer las diferentes sentencias condicionales de JavaScript y saber realizar operaciones complejas con ellas.



1. INTRODUCCIÓN

- Hemos visto lenguajes y tecnologías en el entorno del cliente.
- Algunos mejoran la interactividad con el usuario.
- El lenguaje más utilizados para este propósito es JavaScript.
- Comenzaremos con el "Hola Mundo", ejercicio de introducción cualquier lenguaje de programación.
- Estudiaremos la sintaxis del lenguaje.
- Conceptos relacionados con los tipos de datos.
- Variables y operadores usados por JavaScript.
- Veremos las sentencias condicionales, que nos permitirán aumentar la complejidad de los ejemplos y ejercicios propuestos durante todo el capítulo.

2. CARACTERÍSTICAS DE JAVASCRIPT

- JavaScript es un lenguaje de programación interpretado.
- Se utiliza para dotar de comportamiento dinámico a las páginas web.
- Cualquier navegador web actual incorpora un intérprete de JavaScript.

- El primer lenguaje de *scripting* para la Web fue LiveScript.
- Con él Netscape proporcionaba una alternativa "ligera" a Java para dar comportamiento dinámico tanto en el servidor como en el cliente.
- En 1995 Netscape y Sun aprovecharon la versión 2 de Navigator, para denominar al lenguaje JavaScript.
- Su éxito provocó que Microsoft lanzase su versión de JavaScript denominada JScript.
- Microsoft ya había liberado VBScript, basado en BASIC.
- jScript se integró en sus navegadores a partir de Internet Explorer 3.

- JavaScript y JScript no diferían mucho.
- Cada navegador soportaba su propio lenguaje.
- Los desarrolladores tenían que programar dos veces la misma funcionalidad (una con cada lenguaje).
- Había que usar sentencias condicionales para distinguir en qué navegador se ejecutaba la página.
- Para resolverlo, la ECMA *(European Computer Manufacturers Association)* estandarizó el lenguaje en el ECMAScript1.
- Actuálmente JavaScript y JScript son conformes a dicho estándar.
- El nombre JavaScript se impuso para denominar al propio lenguaje y al estándar.
- La última revisión del lenguaje es de junio de 2021. La décimo segunda edición.

Algunas de las características de JavaScript son:

- Sintaxis semejante a la de C++ y la de Java.
- JavaScript está **basado en el concepto de objeto**, pero no es un lenguaje orientado a objetos al uso.
- Los objetos JavaScript utilizan **herencia basada en prototipos**.
- Muy diferente de la herencia basada en clases propia de los lenguajes como Java.
- Los objetos heredan propiedades de otros objetos sin necesidad de que sean instancias de una misma clase (o de clases que participen en una misma jerarquía).
- Una característica de las más importante de JavaScript es que es un lenguaje **débilmente tipado**.
- Una variable puede contener valores de distintos tipos en diferentes momentos de la ejecución del programa.
- Muchos errores de programación no aparecen hasta que es ejecutado, ya que el compilador es incapaz de detectarlos.
- En JavaScript tiene por defecto, **todas las variables son globales**.
- Las variables que no se definen dentro de otra variable se ubican en un espacio de nombres común denominado *global object.*
- El uso variables globales no es una buena práctica de programación.

- ECMAScript 2016 fue la primera edición de ECMAScript lanzada bajo la nueva cadencia de lanzamiento anual de ECMA TC39 y el proceso de desarrollo abierto.
- Se construyó una fuente en texto plano a partir de ECMAScript 2015 como base para el desarrollo futuro de GitHub.
- ES2016 incluye soporte para un **nuevo operador de exponenciación** y añade un nuevo método al **Array.prototype** llamado `includes`.
- ECMAScript 2017 presentó **Async Functions**, **Shared Memory** y **Atomics** junto con pequeñas mejoras en el lenguaje y las librerías, correcciones de errores y actualizaciones editoriales.
- Las funciones Async mejoran la experiencia de programación asíncrona al proporcionar sintaxis para las funciones de devolución de **promesas**.

- Shared Memory y Atomics introducen un nuevo modelo de memoria que permite a los programas multiagente comunicarse mediante operaciones atómicas que garantizan un orden de ejecución bien definido incluso en CPUs paralelas.
- Incluye nuevos métodos estáticos en Object: Valores de objeto, entradas de objeto y descriptores de objeto `getOwnPropertyDescriptors`.
- ECMAScript 2018 introdujo soporte para la iteración asíncrona a través del protocolo `AsyncIterator` y los generadores de asíncronos.
- También incluía cuatro nuevas características de expresiones regulares: la bandera `dotAll`, los grupos de captura de nombres, los escapes de propiedades Unicode y las aserciones de mirada retrospectiva.
- Por último, incluye parámetros de reposo y soporte de operadores de **spread** para las propiedades de los objetos.

- Esta especificación, la décima edición, introduce algunas nuevas funciones incorporadas:
  - `flat` y `flatMap` en `Array.prototype` para aplanar matrices.
  - `Object.fromEntries` para convertir directamente el valor de retorno de las entradas `Object.entries` en un nuevo objeto
  - `trimStart` y `trimEnd` en `String.prototype` como alternativas mejor nombradas a las ampliamente implementadas pero no estándar `String.prototype.trimLeft` y `trimRight` `built-ins`.
- Además, incluye actualizaciones menores de la sintaxis y la semántica.
- La sintaxis actualizada incluye parámetros opcionales de encuadernación de capturas y permite que U+2028 (separador de línea) y U+2029 (separador de párrafo) en literales de cadena se alineen con JSON.

- Otras actualizaciones incluyen requerir que `Array.prototype.sort` sea un orden estable, requerir que `JSON.stringify` devuelva UTF-8 bien formado sin importar la entrada, y clarificar `Function.prototype.toString` requiriendo que devuelva el texto fuente original correspondiente o un marcador de posición estándar.
- ECMAScript 2020, la 11ª edición, introdujo:
  - El método `matchAll` para cadenas, que produce un iterador para todos los objetos que coincidan, generados desde una expresión regular global.
  - `import()`, una sintaxis para importar asíncronamente módulos con un especificador dinámico.
  - `BigInt`, una nueva primitiva numérica para trabajar con enteros de precisión arbitraria.
  - `Promise.allSettled`, un nuevo combinador `Promise` que no hace cortocircuito.
  - `globalThis`, una forma universal de acceder al valor global this.
  - sintaxis dedicada a exportar * como ns de 'module' para su uso dentro de los módulos.

  - mayor estandarización del orden de enumeración `for-in`.
  - `import.meta`, un objeto poblado por el host disponible en los módulos que puede contener información contextual sobre el módulo.
  - Dos nuevas características sintácticas para mejorar el trabajo con valores "`nullish`" (nulos o indefinidos): `nullish coalescing`, un operador de selección de valores.
  - `optional chaining`, un operador de acceso a propiedades y de invocación de funciones que se cortocircuita si el valor a acceder/invocar es `nullish`.
- La especificación duodécima (ECMAScript 2021), introduce:
  - El método `replaceAll` para cadenas.
  - `Promise.any`, un combinador de `Promise` que se acorta cuando se cumple un valor de entrada.
  - `AggregateError`, un nuevo tipo de error para representar múltiples errores a la vez; operadores de asignación lógica (?? =, &&=, ||=).

- `WeakRef`, para referirse a un objeto de destino sin preservarlo de la recolección de basura, y `FinalizationRegistry`, para gestionar el registro y el desregistro de las operaciones de limpieza realizadas cuando los objetos de destino se recolectan de la basura.
- Separadores para literales numéricos (1_000).
- `Array.prototype.sort` se hizo más preciso, reduciendo la cantidad de casos que resultan en un orden de clasificación definido por la implementación.

3. "HOLA MUNDO" CON JAVASCRIPT

- JavaScript sólo necesita un explorador web y un simple editor de textos como entorno de desarrollo.
- No necesitamos comprar ni descargar ningún software especial para empezar a experimentar con JavaScript.
- Para ejecutar JavaScript tenemos que embeber el código en una página HTML.
- Lo haremos según alguno de los tres método vistos en el capítulo anterior.
- Usaremos las etiquetas <script></script> para marcar el principio y el fin del bloque de código JavaScript.

- El navegador sabrá que el texto contenido entre la etiqueta de inicio y fin no se corresponde con código HTML.
- Ese código debe ser procesado antes de mostrar el resultado en pantalla.
- El intérprete de JavaScript del navegador, se encargará de ejecutar el código cuando cargue la página.
- También mostrará el efecto de la ejecución sobre la página.

- Hay dos formas de embeber el código JavaScript utilizando la etiqueta <script>:

  1. Incluirlo directamente en la página HTML mediante la etiqueta <script>. Crea un fichero HolaMundo.html y copia y pega el siguiente código en el fichero:

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv= “content-type”
        content=“text/html”;
        charset=utf-8”>
        <title>Hola Mundo</title>
    </head>
    <body>
        <script>
            alert('Hola mundo en JavaScript’);
        </script>
    </body>
</html>
```

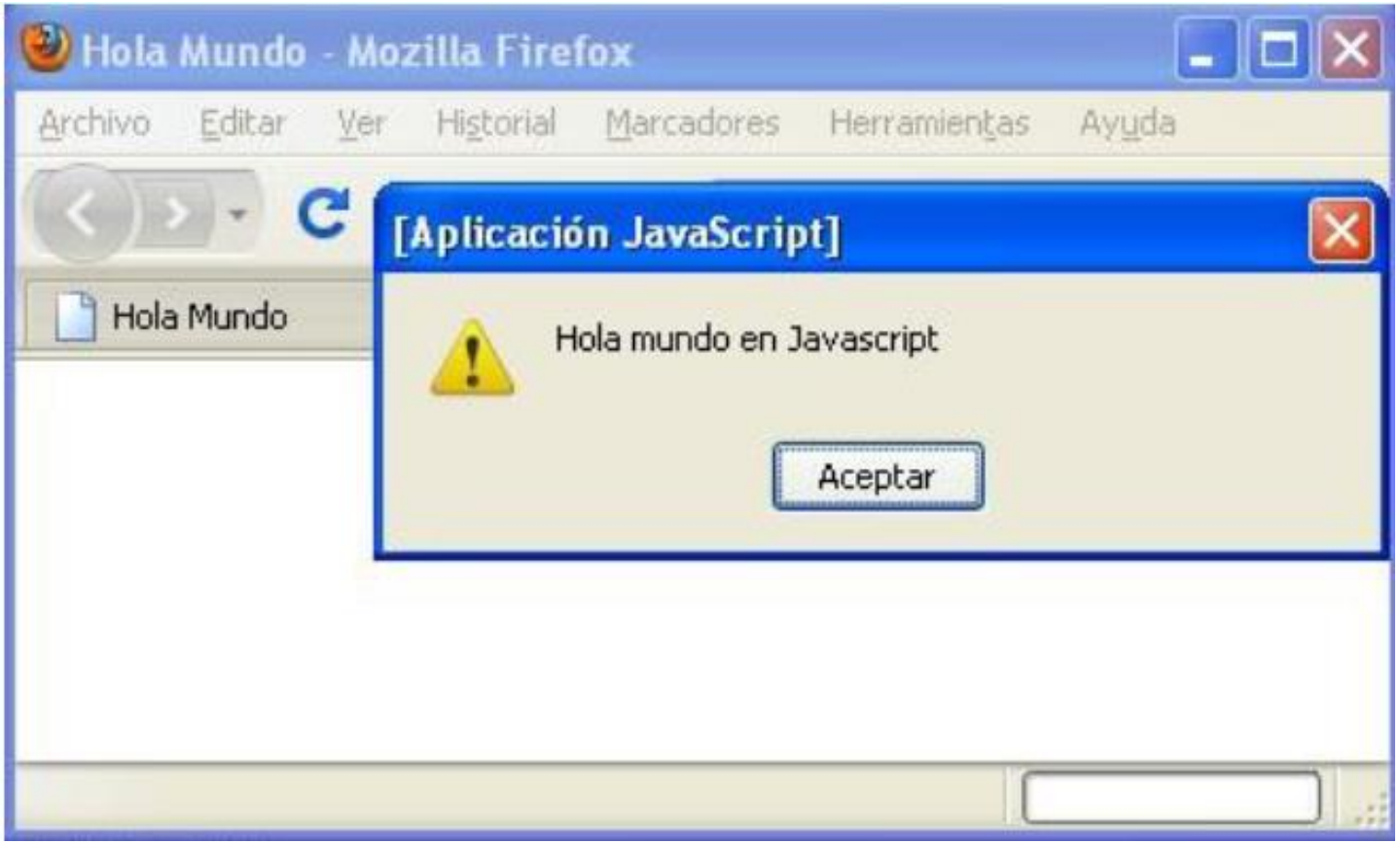
2. Utilizar el atributo src de dicha etiqueta para especificar el fichero que contiene el código JavaScript.

```
<!DOCTYPE html>
<html>
    <head>
       <meta http-equiv=“content-typeH content=Htext/html;
       charset=utf-8”>
       <title>Hola Mundo</title>
       <script type=“text/javascript” src=“HolaMundo.js”>
       </script>
    </head>
    <body></body>
</html>
```

El fichero HolaMundo.js debe contener esta línea de código:

```
alert('Hola mundo en JavaScript’);
```

- Si abrimos los dos ficheros HolaMundo.html con un navegador web, suponiendo que antes hayamos creado el fichero HolaMundo.js en el mismo directorio, el resultado sería exactamente el mismo: el navegador muestra el mensaje "Hola mundo en JavaScript".



- No obstante, la forma recomendada de proceder es la segunda.
- Así se facilita la reutilización y modularización de dicho código.
- Dado que hay distintos lenguajes de *script,* el atributo **type** nos permite decirle al navegador cuál es el usado en el *script.*
- Así el navegador sabrá que intérprete debe utilizar para ejecutar el *script.*
- Los navegadores más conocidos utilizan JavaScript como lenguaje de *script* por defecto.
- Es buena práctica especificarlo y obligatorio de acuerdo a la norma de la W3C, la cual es la encargada de la especificación del estándar.

ACTIVIDADES

1. Crea un nuevo fichero HTML vacío (puedes tomar el fichero HolaMundo2.html como ejemplo y eliminar la etiqueta <script>. Añade el siguiente código JavaScript en el cuerpo de la página (entre las etiquetas <body> y </body>:

```
<script type="text/javascript">
      document.bgColor = "red";
      alert('Cambiamos el color de la p\xE1gina’);
</script>
```

2. Guarda la página con otro nombre (Actividad2-CambiarColor.html), ábrala con su navegador y compruebe el resultado.

Podemos observar que en la primera actividad hemos utilizado una cadena de escape para mostrar correctamente los caracteres tildados. Para evitar este problema debemos emplear la cadena *\xdd,* donde *dd* corresponde al carácter especial especificado por dos dígitos hexadecimales según el estándar Unicode. Existen diferentes secuencias de escape que son útiles a la hora de introducir no solo caracteres tildados, sino también saltos de línea (\n), tabuladores (\t), etc.

4. EL LENGUAJE JAVASCRIPT: SINTAXIS

- Su sintaxis muy similar a la de Java o a la del lenguaje *C++.*
- La sintaxis especifica los caracteres que se deben utilizar para los comentarios, la forma de los nombres de las variables, el modo de separar las diferentes instrucciones del código …

4.1 MAYÚSCULAS Y MINÚSCULAS

- JavaScript distingue entre mayúsculas y minúsculas, a diferencia de HTML. (por tanto es case sensitive).
- Es especialmente importante cuando utilizamos variables, objetos, funciones o cualquier otro símbolo del lenguaje.
- Por ejemplo, no es lo mismo utilizar la función alert () que Alert ().
- Como hemos visto en el ejemplo de HolaMundo.html, la primera muestra un texto en una ventana emergente del navegador.
- La segunda no existe a menos que la defina el programador.

4.2 COMENTARIOS EN EL CÓDIGO

- JavaScript permite insertar comentarios dentro del código.
- Este código no es interpretado por el navegador.
- Facilitan la lectura del código al programador.
- Existen dos formas de insertar comentarios:
  - ❖ Una de ellas es a través de la doble barra (//).
  - ❖ Esto comenta una sola línea de código desde el punto donde se coloca.
  - ❖ La otra forma es a través de los signos /* al inicio del comentario y los signos */ al final del mismo.
  - ❖ Podemos comentar varias líneas de código. En el siguiente código vemos un ejemplo de las dos formas que existen para insertar comentarios:

```
<script type="text/javascript">
    // Este modo permite comentar una sola línea
    /* Este modo permite realizar
       comentarios de
       varias líneas */
</script>
```

- Nota: Es posible que también vea un tercer tipo de sintaxis de comentarios al principio de algunos archivos JavaScript, en este sentido: `#!/usr/bin/env node`.
- Esto se denomina sintaxis de comentarios **hashbang** y es un comentario especial que se utiliza para especificar la ruta a un intérprete de JavaScript en particular que deseamos utilizar para ejecutar el script. Vea los comentarios de **Hashbang** para más detalles.

4.3 TABULACIÓN y SALTOS DE LÍNEA

- JavaScript ignora los espacios, las tabulaciones y los saltos de línea presentes entre los símbolos del código.
- La única restricción acerca de los saltos de línea la vemos en el siguiente apartado.
- Cuando los programas empiezan a ser complejos, resulta útil emplear las tabulaciones y los saltos de línea para mejorar la presentación y la legibilidad del código.
- Podemos observar la diferencia entre un código bien estructurado que facilita la lectura y un código que no está bien estructurado.

Versión 1:

```
<script type="text/javascript">var i,j=0;
      for (i=0;i<5;i++) { alert("Variable i: "+i);
      for (j=0;j<5;j++) { if (i%2==0) {
            document.write (i + “-” + j + "<br>");}}}
</script>
```

Versión 2:

```
<script type="text/javascript">
var i,j=0;
for (i=0;i<5;i++) {
    alert("Variable i: "+i);
    for (j=0;j<5; j++) {
          if (i%2==0) (
                document.write(i + “-” + j + "<br>");
          }
    }
}
</script>
```

- En la segunda versión es mucho más fácil comprender a qué sentencia corresponde cada instrucción.
- En el ejemplo se han usado sentencias condicionales y otros conceptos que estudiaremos a lo largo del capítulo.

4.4 EL PUNTO Y COMA

- Se suele insertar un signo de punto y coma (;) al final de cada instrucción de JavaScript, al igual que se hace en otros lenguajes.
- ; tiene la utilidad de separar y diferenciar cada instrucción.
- Podemos omitir el signo si cada instrucción de JavaScript se encuentra en una línea independiente. JavaScript tiene un mecanismo de inserción automática cuando se necesita.
- Por ejemplo, las siguientes instrucciones podrían escribirse de este modo:

```
pi_egipcio = 3.16
pi_griego = 3.14
```

- Si queremos definir dos valores en una misma línea, es necesario utilizar al menos el primer punto y coma:

```
pi_egipcio = 3.16; pi_griego = 3.14;
```

4.5 PALABRAS RESERVADAS

- JavaScript tiene una serie de palabras que no se pueden usar para definir nombres de variables, funciones o etiquetas. (No se pueden usar como identificador)
- Según la versión 12 de ECMAScript,(2021) las palabras reservadas son las que vemos a continuación:

await	break	case	catch	class	const
continue	debugger	default	delete	do	else
enum	export	extends	false	finally	for
function	if	import	in	instanceof	new
null	return	super	switch	this	throw
true	try	typeof	var	void	while
with	yield				

- Además el estándar ECMAScript contempla el uso de otras palabras reservadas en futuras versiones en el ***modo estricto***, como por ejemplo ***`implements`, `interface`, `package`, `private`, `protected`*** y ***`public`.***
- Aunque los interpretes actuales de JavaScript permiten el uso de estas posibles futuras palabras clave, se aconseja revisar la versión del estándar y averiguar cuáles son en el momento actual, con el fin de evitar su uso en la realización de programas.

Futuras Palabras Reservadas en estandares antiguos

Las siguientes estan reservadas como palabras futuras por la antigua especificación ECMAScript (ECMAScript 1 hasta 3).

- `abstract`
- `boolean`
- `byte`
- `char`
- `double`
- `final`
- `float`
- `goto`
- `int`
- `long`
- `native`
- `short`
- `synchronized`
- `transient`
- `volatile`

Adicionalmente, los literales "`null`", "`true`", y "`false`" estan reservadas en ECMAScript para usos normales.

Otras palabras reservadas

alert	all	anchor	anchors
area	assign	blur	button
checkbox	clearInterval	clearTimeout	clientInformation
close	closed	confirm	constructor
crypto	decodeURI	decodeURIComponent	defaultStatus
document	element	elements	embed
embeds	encodeURI	encodeURIComponent	escape
event	fileUpload	focus	form
forms	frame	innerHeight	innerWidth
layer	layers	link	location
mimeTypes	navigate	navigator	frames
frameRate	hidden	history	image
images	offscreenBuffering	open	opener
option	outerHeight	outerWidth	packages
pageXOffset	pageYOffset	parent	parseFloat
parseInt	password	pkcs11	plugin
prompt	propertyIsEnum	radio	reset
screenX	screenY	scroll	secure
select	self	setInterval	setTimeout
status	submit	taint	text
textarea	top	unescape	untaint
window			

ACTIVIDADES

- Visite la página web *http://www.ecmascript.org y consulte todas las palabras reservadas para las futuras* versiones del estándar.
- También se puede visitar http://www.w3im.com/es/js/js_reserved.html para tener una idea clara del estándar al respecto.
- Otra página interesante es https://www.ecma-international.org/ecma-262/8.0/index.html#sec-intro

5. TIPOS DE DATOS

- Los tipos de datos especifican qué tipo de valor se guardará en una determinada variable.
- Es importante a la hora de determinar cómo podemos combinar ciertas variables o si es posible hacerlo.
- Todos los elementos definidos en el lenguaje pertenecen al objeto Global que se crea al inicio de cada programa de manera automática.
- Los tipos de datos primitivos de JavaScript son: números, cadenas de texto y valores booleanos. `Number, String, Boolean`.
- Además existe el tipo de datos compuesto llamado objeto (`Object`).
- Dentro del objeto global tenemos propiedades que retornan un valor simple:
  - `Infinity`
  - `Nan`
  - `undefined`
  - `Literal null`
  - `globalThis`

- Tenemos también funciones del objeto global:
  - `eval()`
  - `uneval()`
  - `isFinite()`
  - `isNan()`
  - `parseFloat()`
  - `parseInt()`
  - `decodeURI()`
  - `decodeURIComponent()`
  - `escape()`
  - `unescape()`
- A continuación tenemos objetos fundamentales, objetos básicos sobre los que construimos otros objetos. Esto representa objetos generales, funcionas y errores.

5.1 NÚMEROS

- El objeto **Number** en un objeto envuelto que permite trabajar con valores numéricos. Se crean con el constructor **Number()**.
- El tipo primitivo de objeto number se crea usando la función **Number()**.
- Todos los números que utilicemos se representarán a través del formato de punto flotante de doble precisión de 64 bits definido por el estándar IEEE 754.
- Recientemente se soporta enteros de precisión arbitraria usando el tipo **BigInt**.

```
new Number(value);
var a = new Number('123'); // a === 123 is false
var b = Number('123'); // b === 123 is true
a instanceof Number; // is true
b instanceof Number; // is false
```

`value` es el valor numérico del objeto creado.

Este formato es el llamado **double** en los lenguajes Java o C++.

- Además de los valores en base 10, en JavaScript utiliza valores hexadecimales, (base 16).
- Es necesario iniciar el valor con 0x seguido por una secuencia de dígitos hexadecimales.
- La mayoría de las implementaciones de JavaScript admite la representación de valores en base 8, aunque el estándar ECMAScript no.
- Es aconsejable evitar el uso de los valores en formato octal.
- En el siguiente capítulo veremos los métodos y las propiedades de los objetos Number.

Más información sobre el tipo Number

• **BigInt** es un objeto incorporado que proporciona una manera de representar números enteros más grandes que 2^{53} - 1, que es el número más grande que JavaScript puede representar de manera confiable con la primitiva **Number**.

• **BigInt** puede ser usado para enteros arbitrariamente grandes.

• Un **BigInt** se crea añadiendo **n** al final de un entero literal - 10n - o llamando a la función BigInt().

```
const theBiggestInt = 9007199254740991n;
const alsoHuge = BigInt(9007199254740991);
// ↪ 9007199254740991n
const hugeString = BigInt("9007199254740991");
// ↪ 9007199254740991n
const hugeHex = BigInt("0x1fffffffffffff");
// ↪ 9007199254740991n
const hugeBin =
BigInt("0b11111111111111111111111111111111111111111111111111111");
// ↪ 9007199254740991n
```

- **BigInt** es similar a **Number** pero difiere en algunos aspectos clave - no puede ser usado con métodos en el objeto **Math** incorporado y no puede ser mezclado con instancias de **Number** en operaciones; deben ser promovidos al mismo tipo.
- Ten cuidado promoviendo valores hacia adelante y hacia atrás, ya que la precisión de un **BigInt** puede perderse cuando es promovido a un Number.
- Si probamos el tipo de los BigInt nos dará “bigint”.

```
typeof 1n === 'bigint'; // true
typeof BigInt('1') === 'bigint'; // true
```

- Si lo empaquetamos en un objeto, será considerado un **object** normal.

```
typeof Object(1n) === 'object'; // true
```

Más información sobre BigInt

5.2 CADENAS DE TEXTO

- JavaScript usa `string` para representar cadenas de texto.
- Así podemos representar una secuencia de letras, dígitos, signos de puntuación o cualquier otro carácter de Unicode.
- La debemos definir entre comillas dobles o simples (" o ’) o bien usando el objeto global **String**.

```
String(valor)
```

- Para representar estos signos necesitaremos utilizar secuencias de escape haciendo uso de la barra invertida (\).
- La combinación de la barra con otros caracteres permite al navegador representar un carácter, que sin dicha barra invertida no podría representarse.
- En la Tabla 2.2 podemos ver algunas combinaciones de secuencias de escape.

Secuencia de escape	Resultado
\XXX (XXX = 1 - 3 octal digits; range of 0 - 377)	ISO-8859-1 character / Unicode code point between U+0000 and U+00FF
\’	Comilla simple ‘
\”	Comilla doble “
\\	Barra invertida \
\n	Nueva línea
\r	Retorno de carro
\v	Tabulador vertical
\t	Tabulador
\b	Espacio atrás
\f	Avance de página
\uXXXX (XXXX = 4 hex digits; range of 0x0000 - 0xFFFF)	UTF-16 code unit / Unicode code point between U+0000 and U+FFFF
\u{X} ... \u{XXXXXX} (X…XXXXXX = 1 - 6 hex digits; range of 0x0 - 0x10FFFF)	UTF-32 code unit / Unicode code point between U+0000 and U+10FFFF
\xXX (XX = 2 hex digits; range of 0x00 - 0xFF)	ISO-8859-1 character / Unicode code point between U+0000 and U+00FF

Tabla 2.2: *Secuencias de escape*

```
`hello world`
`hello!
 world!`
`hello ${who}`
tag `<a>${who}</a>`
```

Más información sobre Strings

- Para cadenas largas se puede usar la concatenación (+) o usar la barra invertida (\) asegurándonos que de no haya otros carateres detrás de dicha barra. En el ejemplo ambos resultados son identicos.

```
let longString = "This is a very long string which needs " +
                 "to wrap across multiple lines because " +
                 "otherwise my code is unreadable.";
```

```
let longString = "This is a very long string which needs \
to wrap across multiple lines because \
otherwise my code is unreadable.";
```

5.3 VALORES BOOLEANOS

- El tipo booleano, conocido como valores lógicos, es el más simple.
- Sólo admite los valores: `true` o `false` (verdadero o falso).
- Estos valores van sin comillas.
- Es muy útil a la hora de evaluar expresiones lógicas o verificar condiciones particulares en nuestros programas.
- Tambíen existen los objetos **Boolean**. `new Boolean([valor])`
- Si se omite el valor en su constructor, o es `0`, `-0`, null, false, `NaN`, `undefined` o la cadena vacía “”, el objeto se inicializa a falso.
- Cualquier otro valor, incluido un array vacío ([]) o el string “false” serán inicializados a verdadero.
- No hay que confundir los valores primitivos **Boolean** `true` y `false` con el `true` y `false` de los objetos booleanos.

- Cualquier objeto cuyo valor no sea `undefined` o `null`, incluyendo un objeto booleano cuyo valor sea falso, se evalúa a verdadero cuando se pasa a una expresión condicional. Por ejemplo, la condición en el siguiente enunciado si se evalúa a verdadero:

```
var x = new Boolean(false);
if (x) {
    // this code is executed
}
```

- Este comportamiento no se aplica a los tipos primitivos booleanos. Por ejemplo, la condición en el siguiente enunciado si se evalúa como falsa:

```
var x = false;
if (x) {
  // this code is not executed
}
```

- No utilices un objeto booleano para convertir un valor no booleano en un valor booleano. Para realizar esta tarea, en su lugar, utilice Boolean como una función, o un operador doble NOT:

```
var x = Boolean(expression);     // use this...
var x = !!(expression);          // ...or this
var x = new Boolean(expression); // don't use this!
```

- Si se especifica cualquier objeto, incluido un objeto booleano cuyo valor es falso, como valor inicial de un objeto booleano, el nuevo objeto booleano tiene un valor verdadero.

```
var myFalse = new Boolean(false);   // initial value of false
var g = Boolean(myFalse);           // initial value of true
var myString = new String('Hello'); // string object
var s = Boolean(myString);          // initial value of true
```

ACTIVIDADES

Crea un fichero HTML que permita mostrar el siguiente texto en una ventana emergente, a través de la función alert ():

I‘m = I am

I don't = I do not

En la Figura 2.2 puedes ver el resultado esperado. Ten en cuenta que debes usar secuencias de escape para poder representar las comillas simples y el salto de línea.

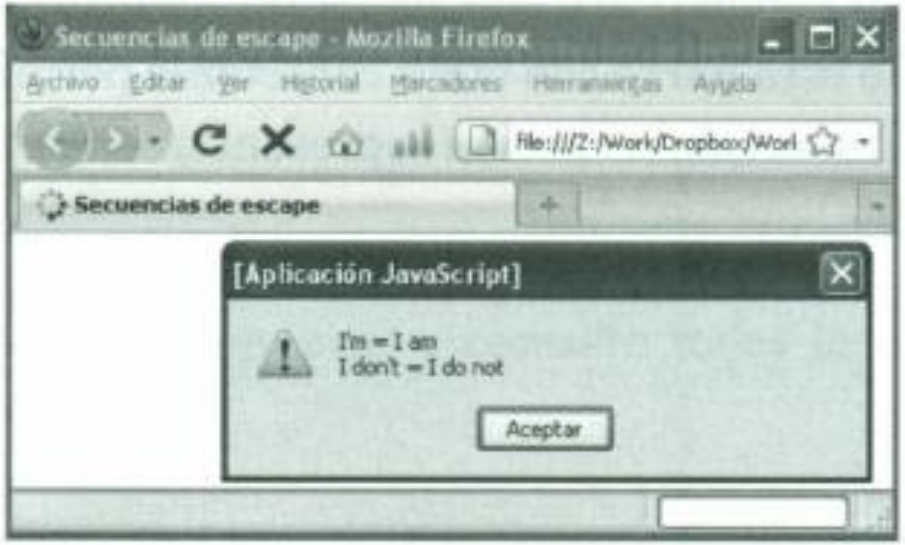


Figura 2.2. *Ejemplo del uso de las secuencias de escape*

6. VARIABLES

- Las variables se definen como zonas de memoria que se identifican con un nombre y en las cuales se almacenan ciertos datos.
- Nos permiten manipular y guardar datos.
- A veces no sabremos su valor a priori.
- Desarrollar de un *script* conlleva dos aspectos relacionados con las variables:

  - la ***declaración*** de variables.
  - la ***inicialización*** de variables.

- A continuación veremos los detalles de cada uno de estos dos aspectos.

6.1 DECLARACIÓN DE VARIABLES

- Lo primero será definirla.
- En JavaScript se definen las variables con tres tipos de declaraciones.
- ***var*** declara una variable y opcionalmente la inicializa a algún valor.
- Ejemplo:

```
var mi_variable_1;
var mi_variable_2;
```

- Es posible declarar más de una variable en una sola línea de código por medio de comas:

```
var mi_variable_1, mi_variable_2;
```

- El contenido de las variables estará **indefinido** hasta que una parte del código almacene un valor en ellas.
- Este proceso lo describimos en el siguiente apartado.

- ***let*** declara una variable en un ámbito de bloque, variable local y la inicializa opcionalmente.

```
let x = 1;
if (x === 1) {
     let x = 2;
     console.log(x);
      // expected output: 2
}
console.log(x);
// expected output: 1
```

- ***const*** declara constantes en un ámbito de bloque.

```
const number = 42;
try {
  number = 99;
}  catch(err) {
    console.log(err);
   // expected output: TypeError: invalid assignment to const `number'
   // Note - error messages will vary depending on browser
}
console.log(number);
// expected output: 42
```

- Las variables se utilizan como nombres simbólicos para valores en la aplicación.
- Los nombres de las variables, llamados identificadores, se ajustan a ciertas reglas.
- Un identificador JavaScript debe comenzar con una letra, guión bajo (_) o signo de dólar ($); los caracteres siguientes también pueden ser dígitos (0-9).
- Dado que JavaScript distingue entre mayúsculas y minúsculas, las letras incluyen los caracteres de la "A" a la "Z" (mayúsculas) y los caracteres de la "a" a la "z" (minúsculas).
- Puede utilizar la mayoría de las letras ISO 8859-1 o Unicode como å, ü o ñ en los identificadores.

- También puedes utilizar las secuencias de escape Unicode como caracteres en los identificadores.
- Algunos ejemplos de nombres legales son Number_hits, temp99, $credit y _name.
- Puedes declarar una variable de dos maneras:
  - Con la palabra clave var. por ejemplo, var x = 42. Esta sintaxis puede utilizarse para declarar variables tanto locales como globales, dependiendo del contexto de ejecución.
  - Con la palabra clave const o let. Por ejemplo, y = 13. Esta sintaxis puede ser usada para declarar una variable local de block-scope.
- También puedes asignar simplemente un valor a una variable Por ejemplo, x = 42.

- Esta forma crea una variable global no declarada.
- También genera una advertencia estricta de JavaScript.
- Las variables globales no declaradas a menudo pueden conducir a un comportamiento inesperado.
- Se desaconseja el uso de variables globales no declaradas.
- Una variable declarada mediante la sentencia ***var*** o ***let*** sin valor asignado especificado tiene el valor de ***undefined***.
- Si se intenta acceder a una variable no declarada, se lanza una excepción de ReferenceError:

```
var a;
console.log('The value of a is ' + a); // The value of a is undefined

console.log('The value of b is ' + b); // The value of b is undefined
var b;
// This one may puzzle you until you read 'Variable hoisting' below

console.log('The value of c is ' + c); // Uncaught ReferenceError: c is not defined

let x;
console.log('The value of x is ' + x); // The value of x is undefined

console.log('The value of y is ' + y); // Uncaught ReferenceError: y is not defined
let y;
```

- Puedes usar ***undefined*** para determinar si una variable tiene un valor. En el siguiente código, a la variable de entrada no se le asigna un valor, y la sentencia `if` se evalúa a true.

```
var input;
if (input === undefined) {
  doThis();
} else {
  doThat();
}
```

- El valor no ***undefined*** se comporta como falso cuando se utiliza en un contexto booleano. Por ejemplo, el siguiente código ejecuta la función `myFunction` porque el elemento `myArray` no está definido:

```
var myArray = [];
if (!myArray[0]) myFunction();
```

- El valor ***undefined*** se convierte en `NaN` cuando se utiliza en un contexto numérico.

```
var a;
a + 2;  // se evalua a NaN
```

6.2 INICIALIZACIÓN DE VARIABLES

- La asignación de un valor a una variable se puede hacer de tres formas:
  - ❖ Asignación directa de un valor concreto.
  - ❖ Asignación indirecta a través de un cálculo en el que se implican a otras variables o constantes.
  - ❖ Asignación a través de la solicitud del valor al usuario del programa.

- Ejemplos respectivos de cada caso:

```
var mi_variable_1 = 30;
var mi_variable_2 = mi_variable_1 + 10;
var mi_variable_3 = prompt('Introduce un valor:');
```

- En todas debemos utilizar el operador de asignación (el signo igual, "=").
- Si intentamos utilizar una variable que no haya sido inicializada, JavaScript generará un error.

- Una vez declarada y después de asígnale un valor, podemos utilizarla en otras instrucciones del programa.
- En el momento de usar una variable, el navegador accede a la memoria del ordenador, recuperando su valor y sustituyéndolo en la instrucción.
- Cuando se declara una variable fuera de cualquier función, se llama variable global, porque está disponible para cualquier otro código en el documento actual.
- Cuando se declara una variable dentro de una función, se denomina variable local, porque sólo está disponible dentro de esa función.
- JavaScript antes de ECMAScript 2015 no tiene alcance de declaración de bloque; más bien, una variable declarada dentro de un bloque es local a la función (o alcance global) en la que reside el bloque.

- Por ejemplo, el siguiente código registrará 5, porque el alcance de x es el contexto global (o la función si los siguientes códigos son parte de la función), no el inmediato si se bloquea la sentencia.

```
if (true) {
  var x = 5;
}
console.log(x);  // x is 5
```

- Este comportamiento cambia cuando se utiliza la declaración let introducida en ECMAScript 2015.

```
if (true) {
  let y = 5;
}
console.log(y);  // ReferenceError: y is not defined
```

- ***Hoisting*** de variables
- Otra tema inusual acerca de las variables en JavaScript es que podemos referirnos a una variable declarada más tarde, sin obtener una excepción.
- Este concepto se conoce como *elevación*; las variables en JavaScript son, en cierto sentido, "elevadas" o empujadas a la parte superior de la función o sentencia.
- Sin embargo, las variables que son izadas devuelven un valor de indefinido.
- Por lo tanto, incluso si declara e inicializa después de usar o referirse a esta variable, ésta retorna indefinida.

```
/**
 * Ejemplo 1
 */
console.log(x === undefined); // true
var x = 3;

/**
 * Ejemplo 2
 */
// devolverá undefined
var myvar = 'my value';

(function() {
  console.log(myvar); // undefined
  var myvar = 'local value';
})();
```

ACTIVIDADES

1. Crea un nuevo fichero HTML y, al igual que en las anteriores actividades, copia el siguiente código JavaScript dentro del cuerpo de la página:

```
<script type="text/javascript">
	var primer_saludo = "hola";
	var segundo_saludo = primer_saludo;
	primer_saludo = "hello";
	alert(segundo_saludo);
</script>
```

2. Antes de ver el resultado, ¿cuál crees que será el valor en la ventana emergente?

7. OPERADORES

- El uso de operadores permite construir expresiones con una colección de símbolos, palabras y/o números con los que podremos realizar cálculos complejos.
- JavaScript utiliza principalmente cinco tipo de operadores:

  - ❖ Operadores aritméticos.
  - ❖ Operadores lógicos.
  - ❖ Operadores de asignación.
  - ❖ Operadores de comparación.
  - ❖ Operadores condicionales.

- Los operandos son los elementos utilizados en una expresión y unidos a través de un operador.
- A continuación veremos los diferentes tipos de operadores.

7.1 OPERADORES ARITMÉTICOS

- Los operadores aritméticos permiten realizar cálculos elementales entre variables numéricas.
- En la Tabla 2.3 podemos ver las cuatro operaciones aritméticas elementales: suma, resta, multiplicación y división.
- Hay además otros tres operadores menos comunes: módulo, incremento y decremento.

Tabla 2.3 Operadores aritméticos

Operador	Nombre	Descripción
+	Suma	Efectúa la suma entre los operandos.
-	Resta	Efectúa la resta entre los operandos.
*	Multiplicación	Efectúa la multiplicación entre los operandos.
/	División	Efectúa la división entre los operandos.
%	Módulo	Extrae la parte entera del resultado de la división entre los operandos.
++	Incremento	Permite incrementar un valor.
--	Decremento	Permite decrementar un valor.

- El operador módulo divide un número entre otro y devuelve es el resto de dicha división.
- Por ejemplo, para comprobar que el año 2012 es un año bisiesto (teniendo en cuenta algunas excepciones, los años bisiestos son divisibles por 4) debemos realizar la siguiente operación:

```
var anyo = 2012;
var bisiesto = anyo % 4;
```

- Si la variable bisiesto almacena el valor 0,podemos afirmar que 2012 es un año bisiesto.
- Los operadores incremento y decremento permiten incrementar o decrementar un valor en una unidad.
- Podemos utilizar estos operadores antes o después de la variable, aunque su efecto es diferente en ambos casos.

- En el siguiente código almacena el valor 2 en ambas variables:

```
Variable_1= 1;
Variable_2= ++variable_1;
```

- En este otro ejemplo la primera variable contendrá al valor 2 y la segunda 1:

```
variable_1= 1;
variable_2= variable_1++;
```

- http://www.w3im.com/es/js/js_operators.html

7.2 OPERADORES LÓGICOS

- Los operadores lógicos permiten definir expresiones lógicas con el fin de evaluar su resultado que será verdadero o falso.
- Se utiliza para tomar decisiones dentro de un programa.
- Podemos ver los tres operadores lógicos que existen en la Tabla 2.4.

Tabla 2.4 Operadores lógicos

Operador	Nombre	Descripción
&&	Y	Ejecuta la operación booleana AND sobre los valores. Devuelve true solo si todos los valores son `true`. Devuelve `false` en caso contrario.
||	O	Ejecuta la operación booleana OR sobre los valores. Devuelve true en el caso en que al menos uno de los valores sea `true`. Devuelve `false` en caso contrario.
!	No	Invierte el valor booleano de su operando.

7.4 OPERADORES DE ASIGNACIÓN

- Tenemos el operador de asignación igual (=).
- JavaScript cuenta con otros operadores con características en común con los operadores aritméticos de incremento y decremento.
- Nos ahorran tiempo y disminuyen la cantidad de código a escribir.
- Podemos obtener métodos abreviados que evitan tener que escribir dos veces la variable que se encuentra a la izquierda del operador.
- La Tabla 2.5 muestra todos los operadores de asignación.

7.4 OPERADORES DE ASIGNACIÓN

Tabla 2.5 Operadores de asignación

Operador	Nombre	Descripción
+=	Suma y asigna	Ejecuta una suma y asigna el valor al operando de la izquierda.
-=	Resta y asigna	Ejecuta una resta y asigna el valor al operando de la izquierda.
*=	Multiplica y asigna	Ejecuta una multiplicación y asigna el valor al operando de la izquierda.
/*	Divide y asigna	Ejecuta una división y asigna el valor al operando de la izquierda.
%=	Módulo y asigna	Ejecuta el módulo y asigna el valor al operando de la izquierda.

- Podríamos usar el operador de restar y asignar de este modo:

```
var deudas = 1500; // tenemos unas deudas de 1500 euros
deudas -= 300; // nuestras deudas disminuyen de 300 euros
// esto es lo mismo que escribir deudas = deudas – 300
```

7.5 OPERADORES DE COMPARACIÓN

- Los operadores de comparación permiten comparar todo tipo de variables con el fin de verificar sus valores.
- El resultado de dicha comparación nos devolverá un valor booleano
- También determina el orden relativo de dos valores.
- La comparación ejecutada por estos operadores se puede realizar sólo sobre números y cadenas.
- Si utilizamos otro tipo de datos, se convertirán automáticamente para intentar que la comparación se pueda llevar a cabo.
- Esto se conoce como la promoción automática de tipos.

Tabla 2.6 Operadores de comparación

Operador	Nombre	Descripción
<	Menor que	Verifica si el operando a la izquierda del operador es menor que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.
<=	Menor o igual que	Verifica si el operando a la izquierda del operador es menor o igual que el operando de la derecha. Devuelve `true` en ese caso o false en caso contrario.
==	Igual	Verifica si los dos operandos son iguales. Devuelve `true` en ese caso o `false` en caso contrario.
>	Mayor que	Verifica si el operando a la izquierda del operador es mayor que el operando de la derecha. Devuelve `true` en ese caso o false en caso contrario.

>=	Mayor o igual que	Verifica si el operando a la izquierda del operador es mayor o igual que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.
!=	Diferente	Verifica si los dos operandos son diferentes. Devuelve `true` en ese caso o `false` en caso contrario.
===	Estrictamente igual	Verifica si el operando a la izquierda del operador es igual y del mismo tipo de datos que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.
!==	Estrictamente diferente	Verifica si el operando a la izquierda del operador es diferente y/o de tipo diferente que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.

7.6 OPERADORES CONDICIONALES

- Existe otro operador más complejo, pero muy útil para tomar decisiones en los programas. Es el operador condicional.
- Podemos indicarle al navegador que ejecute una acción en concreto después de evaluar una expresión.
- Consta de tres partes:
  - ❖ La primera es la expresión a evaluar.
  - ❖ La segunda es la acción a realizar si la expresión es verdadera
  - ❖ La acción a realizar si la expresión es falsa.

Tabla 2.7 Operadores condicionales

Operador	Nombre	Descripción
?:	Condicional	Si la expresión antes del operador es verdadera, se utiliza el primer valor a la derecha. En caso contrario se utiliza el segundo valor a la derecha.

- Por ejemplo, si queremos avisar al usuario que no se puede efectuar una división por cero, es posible hacerlo mediante un operador condicional:

```
<script type="text/javascript">
    var dividendo = prompt("Introduce el dividendo: ");
    var divisor = prompt("Introduce el divisor: ");
    var resultado;
    divisor != 0 ? resultado = dividendo/divisor:
    alert("No es posible la división por cero");
    alert("El resultado es: " + resultado);
</script>
```

ACTIVIDADES

La precedencia de los operadores determina el orden en que se ejecuta una expresión. Realice una búsqueda con el fin de encontrar alguna de las tablas que indiquen el orden de precedencia de los operadores de JavaScript.

Operador	Operación
`. []`	**Acceso a propiedad o invocar método; índice a array**
`new`	**Crear objeto con constructor de clase**
`()`	**Invocación de función/método o agrupar expresión**
`++ --`	**Pre o post auto-incremento; pre o post auto-decremento**
`! ~`	**Negación lógica (NOT); complemento de bits**
`+ -`	**Operador unitario, números. signo positivo; signo negativo**
`delete`	**Borrar propiedad de un objeto**
`typeof void`	**Devolver tipo; valor indefinido**
`* / %`	**Números. Multiplicación; división; modulo (o resto)**
`+`	**Concatenación de string**
`+ -`	**Números. Suma; resta**
`<< >> >>>`	**Desplazamientos de bit**
`< <= > >=`	**Menor; menor o igual; mayor; mayor o igual**
`instanceof in`	**¿objeto pertenece a clase?; ¿propiedad pertenece a objeto?**
`== != === !==`	**Igualdad; desigualdad; identidad; no identidad**
`&`	**Operación y (AND) de bits**
`^`	**Operación ó exclusivo (XOR) de bits**
`|`	**Operación ó (OR) de bits**
`&&`	**Operación lógica y (AND)**
`||`	**Operación lógica o (OR)**
`?:`	**Asignación condicional**
`=`	**Asignación de valor**
`OP=`	**Asig. con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=**
`,`	**Evaluación múltiple**

Tabla de operadores JavaScript y su precedencia de arriba a abajo

8. SENTENCIAS CONDICIONALES

- Hasta el momento hemos realizado ejemplos lineales en los que las sentencias se ejecutaban unas detrás de otras, desde la primera hasta la última.
- Se puede controlar la toma de decisiones y el resultado posterior por parte del navegador mediante el uso de sentencias condicionales.
- Permiten evaluar condiciones y ejecutar ciertas instrucciones si la condición es verdadera y ejecutar otras instrucciones distintas si la condición es falsa.

8. SENTENCIAS CONDICIONALES

- Existen tres tipos de sentencias condicionales:
  - `If`.
  - `Switch`.
  - Bucles `while` y `for`.
- La sentencia `if` indica al navegador si debe ejecutar una parte de código según el valor lógico de una expresión condicional.
- La sentencia `switch` compara el valor de una variable con una serie de valores conocidos. Si uno de ellos coincide con el valor de la variable, se ejecuta el código asociado a dicho valor conocido.
- Las sentencias en bucle permiten al navegador ejecutar un fragmento de código de forma repetida mientras la condición sea verdadera.

8.1 SENTENCIA IF

- La sentencia **`if`** es fundamental para realizar controles en la ejecución de los programas.
- El navegador tomará una decisión y ejecutará ciertas instrucciones de forma condicional. Su sintaxis es la siguiente:

```
if (expresión) {
   instrucciones
}
```

- La expresión debe ser una comparación o una expresión lógica que devuelve los valores *`true`* o *`false`*.
- Las instrucciones son el código que se ejecutará si la expresión devuelve *`true`*.
- JavaScript ignora las instrucciones en el caso en que la expresión devuelva *`false`*.

- El uso de las llaves {} no es obligatorio con una sola instrucción. De lo contrario, sí que es obligatorio su uso.
- La segunda forma de `if` es agregando la palabra clave `else`. De este modo podemos ejecutar instrucciones en el caso en que la expresión devuelva el valor *false*. Así es como definimos una sentencia `if-else`*:*

```
if (expresión) {
  instrucciones si true
}else {
  instrucciones si false
}
```

- El diagrama de flujo de control de la sentencia *`if-else`* lo vemos en la Figura 2.3:

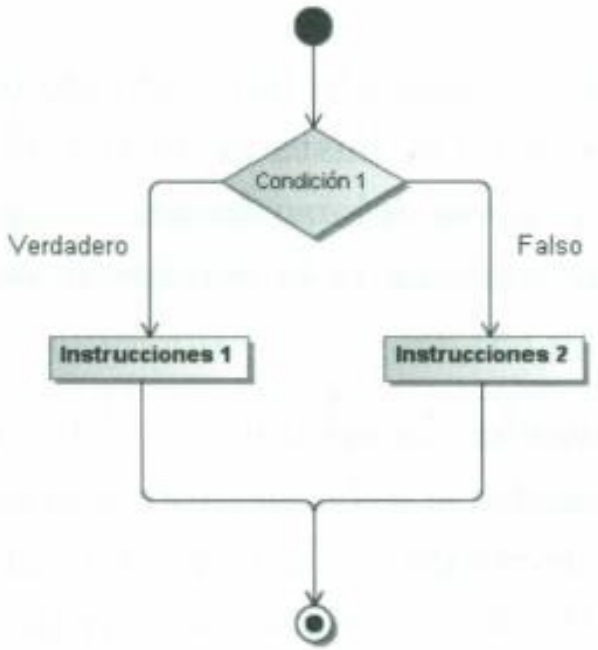


Figura 2.3. *Diagrama de flujo de control de la sentencia* `if-else`

- La tercera forma de utilizar la sentencia `if` es mediante la sentencia `if-else-if`.
- La Figura 2.4 muestra su diagrama de flujo de control.
- Así le pedimos al navegador que evalúe una expresión, y si ésta devuelve el valor `false`, el navegador evaluará una nueva expresión.
- Si ninguna de las dos o más expresiones utilizadas devuelven el valor `true`, se puede usar un último `else`, donde es posible escribir el código que se ejecutará en este caso.

La estructura de la sentencia `if-else-if` es la siguiente:

```
if(expresión_1) {
    instrucciones 1
} else if (expresión_2) {
    instrucciones 2
}else {
    instrucciones 3
}
```

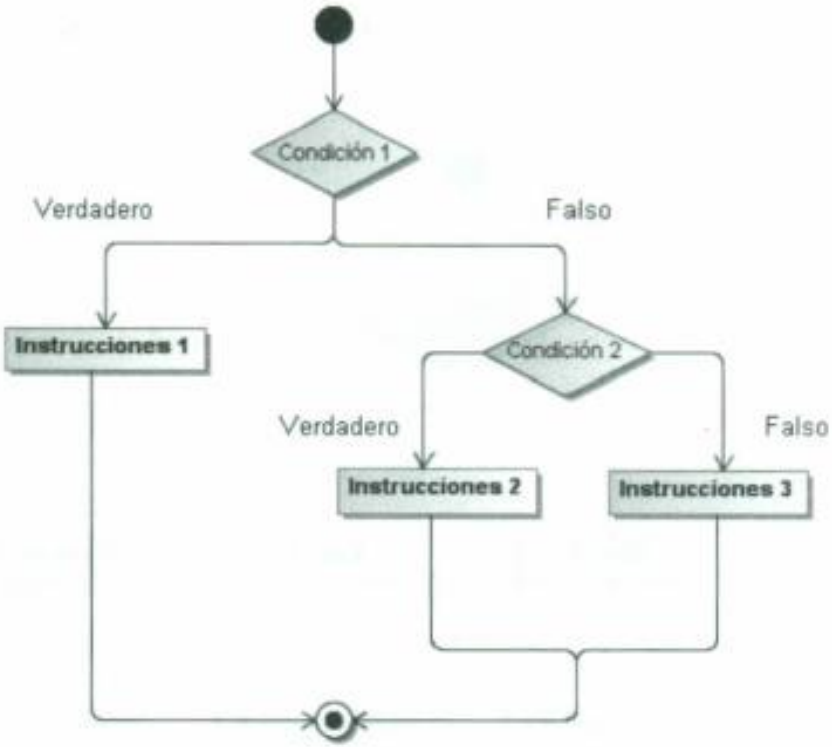


Figura 2.4. *Diagrama de flujo de control de la sentencia* `if-else-if`

- Con el uso de la sentencia `if`, será posible escribir código con tomas de decisiones más complejas a través de `if` anidados.
- Los `if` anidados nos permitirán solucionar los problemas más comunes, como por ejemplo tomar de decisiones basadas en decisiones precedentes.

8.2 SENTENCIA SWITCH

- La sentencia `switch` se usa para comparar el valor de una variable con algunos valores conocidos.
- `Switch` tiene una expresión a evaluar y una serie de posibles valores de dicha expresión.
- Los valores son los casos, en ellos están las instrucciones a ejecutar cuando coincidan el valor de la expresión y el del caso. La sintaxis de esta sentencia es la siguiente:

```
switch (expresión) {
case valor1:
     instrucciones a ejecutar si expresión = valor1
break;
case valor2:
     instrucciones a ejecutar si expresión = valor2
break;
case valor3:
     instrucciones a ejecutar si expresión = valor3
break
default:
     instrucciones a ejecutar si expresión es diferente a los valores
anteriores
```

- Cada caso termina con la palabra clave `break`.
- `Break` detiene la ejecución del `switch` cuando uno de los casos resulte verdadero.
- Esto evita que se evalúen los demás casos.
- El diagrama de flujo de la sentencia switch podemos verlo en la Figura 2.5:

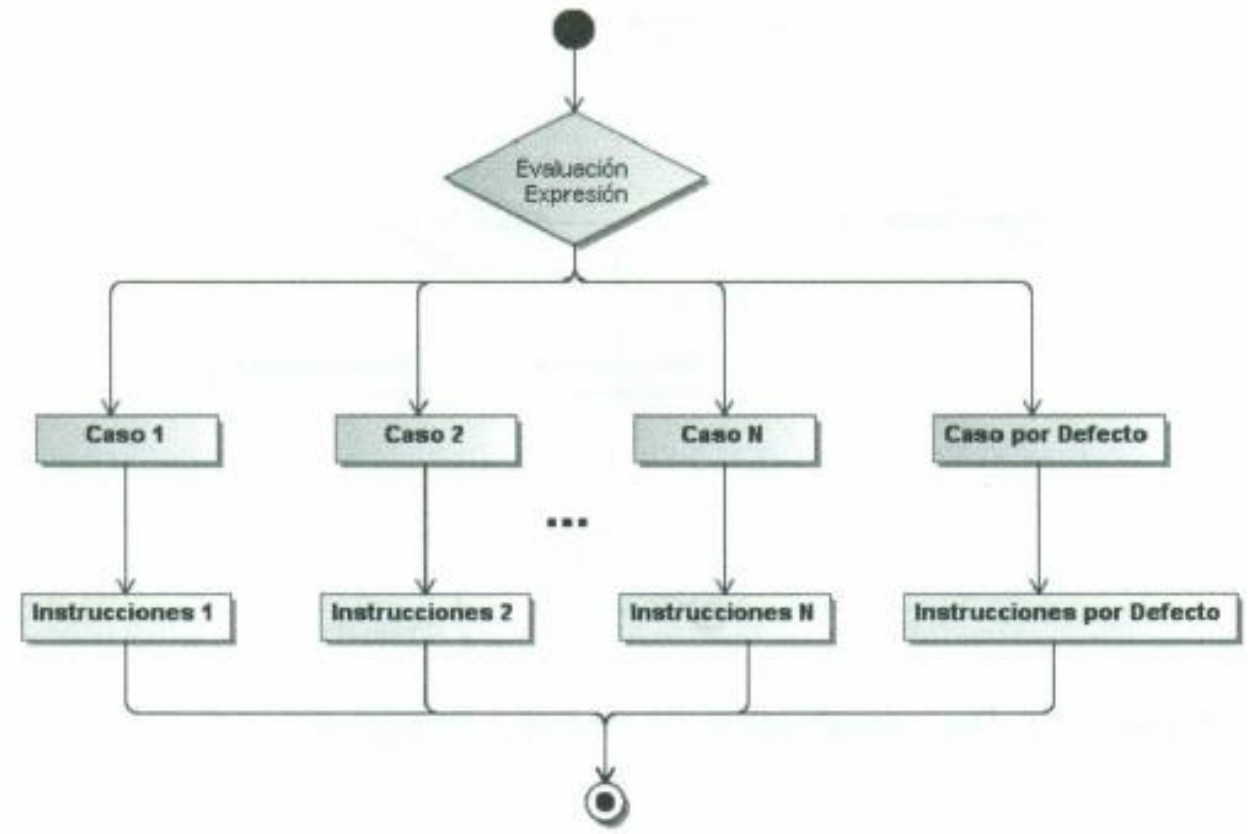


Figura 2.5. *Diagrama de flujo de control de la sentencia* `switch`

8.3 BUCLE WHILE

- Es el más sencillo en JavaScript.
- El navegador ejecutará una o más instrucciones hasta que una condición deje de ser verdadera.
- Es el más utilizado si no sabemos el número de repeticiones del bucle.
- El bucle `while` consta de tres partes fundamentales:
  - La palabra clave `while`.
  - La expresión condicional.
  - Las instrucciones que se ejecutan mientras la expresión condicional sea verdadera. Su sintaxis es la siguiente:

```
while (expresión) {
instrucciones}
```

- Una variación del bucle `while` es el denominado `do-while`.
- Su estructura es la siguiente:

```
do{
instrucciones
} while (expresión)
```

- El número mínimo de repeticiones de este bucle es al menos una vez.
- Al final, se evalúa si se debe seguir ejecutando o no.
- Sin embargo, se puede realizar este proceso a través de un bucle `while`.
- Es recomendable ya que el bucle `do-while` se introdujo sólo a partir de la versión 1.2 de JavaScript, con lo cual no todos los navegadores soportan este último bucle.
- La Figura 2.6 muestra su diagrama de flujo.

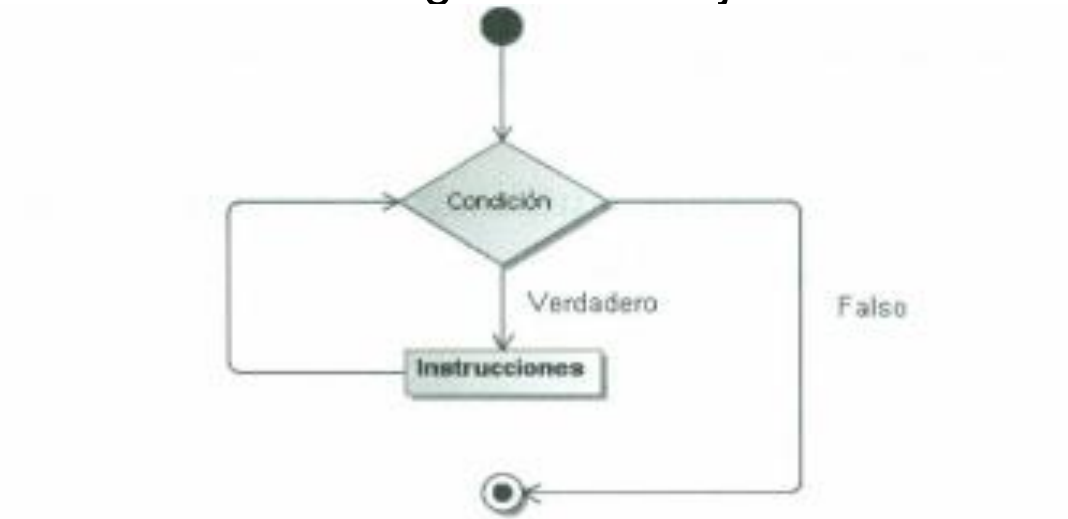


Figura 2.6. *Diagrama de flujo de control del bucle* `while`

8.4 BUCLE FOR

- El bucle **`for`** permite que un navegador ejecute las instrucciones que se encuentren dentro del mismo bucle hasta que la expresión condicional devuelva el valor falso.
- Este tipo de sentencia consta de cinco partes:
  - La palabra clave **`for`**.
  - El valor inicial de la variable de control.
  - La condición basada en dicha variable.
  - El incremento o decremento de la variable.
  - El cuerpo del bucle.
- Su sintaxis general es la siguiente:

```
for(valor_inicial_variable; expresion condicional;
    incremento_o_decremento_de la_variable) {
  cuerpo del_bucle
}
```

- Este bucle es bastante potente para realizar diferentes actividades con pocas líneas de código.
- El diagrama de flujo de este bucle los vemos en la Figura 2.7.
- En este diagrama, el incremento del contador se efectúa después de llevar a cabo las instrucciones del bucle, sin embargo, se puede realizar inmediatamente después de hacer la comprobación inicial de la condición o en alguna de las mismas instrucciones del bucle.

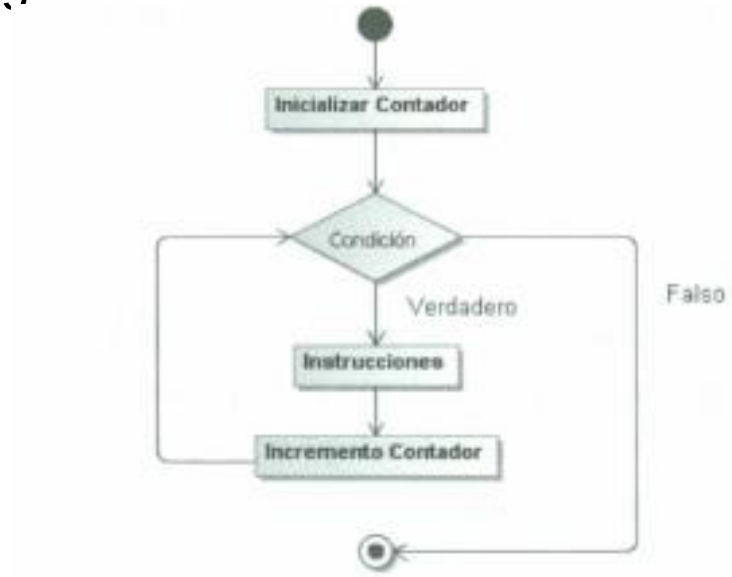


Figura 2.7. *Diagrama de flujo de control bucle* `for`

- Para comprender mejor su funcionamiento, podemos utilizar un ejemplo que muestra en pantalla un valor que se incrementa en cada una de las diez iteraciones del bucle:

```
for(i=0; i<10; i++) {
  document.write("El valor de la variable de control es: “
  + i + "<br>”)
}
```

ACTIVIDADES

Abre el fichero Actividad-2.8-if.html e introduce la información requerida por el navegador. Posteriormente mira el código para que vea un ejemplo del uso de la sentencia condicional `if`.

EJERCICIOS

9. EJERCICIOS PROPUESTOS

1. Cree un fichero HTML vacío y llámelo EjercicioPropuesto-2.1-Numeros.html. Añada el siguiente código JavaScript en el cuerpo de la página (entre las etiquetas <body> y </body»:

```
<script type="text/javascript">
      var maxValue Number.MAX_VALUE;
      var minValue = Number.MIN_VALUE;
      alert(“Max Value: “ + maxValue);
      alert(“Min Value: “ + minValue);
      alert(“Valor especial: “ + maxValue*2);
</script>
```

De este modo podremos comprobar el número más grande representable por JavaScript, el número más cercano a cero y el valor especial que representa el infinito.

2. Utilice el siguiente *script para comprobar el* operador lógico & &:

```
<script type="text/javascript">
   var operando_1 = eval(prompt(“lntroduce el primer valor lógico (true o false) :",
      true));
   var operando_2 = eval(prompt(Ulntroduce el segundo valor lógico (true o false) :",
      true);
   var resultado_logico = operando_1 && operando_2;
   alert(“Resultado: “ + resultado _logico) ;
</script>
```

Con este ejercicio pedimos al usuario que introduzca dos valores lógicos y posteriormente compruebe el resultado del operador &&.

3. Utilice el siguiente *script en un fichero HTML* y compruebe su funcionamiento. Posteriormente, cambie la línea countdown- - por countdown++ y concluya explicando por qué el navegador no deja de solicitarnos valores.

```
<script type="text/javascript">
var countdown = prompt (“lntroduce un número para iniciar la cuenta atrás: “);
while (countdown>0) {
    alert(countdown+ “…”);
    countdown--;
}
</script>
```

4. Realice el mismo ejercicio anterior, pero sustituyendo el bucle *while* por un bucle *for*.