DESARROLLO WEB EN ENTORNO CLIENTE

U.D. 2 (parte 1/2):
Introducción al lenguaje JavaScript

Versión 23/10/2023

Características de JavaScript

- ¿Qué es JavaScript?
  - Es el lenguaje de scripts en la Web.
  - Lenguaje de programación interpretado (no se compila) utilizado fundamentalmente para dotar de comportamiento dinámico a las páginas Web.
  - De gramática sencilla, soporta los paradigmas de P.O.O. y programación funcional. Tiene todos los operadores y estructuras de control que se puede esperar de un lenguaje de alto nivel.
  - Cualquier navegador Web actual incorpora un intérprete para código JavaScript.
  - Sin coste por licencia.

Características de JavaScript

- Puede cambiar elementos HTML y sus atributos, cambiar estilos CSS y validar entrada de datos.
- Su sintaxis se asemeja a la de C++ y Java (aunque no tiene relación con éste).
- Sus objetos utilizan herencia basada en prototipos.
- Es un lenguaje débilmente tipado.
- Todas sus variables son globales.
- Desarrollado originalmente en 1995 por *Brendan Eich* de Netscape con el nombre de Mocha.
- ECMAScript es su especificación estándar.

Integración del código con las etiquetas HTML

- Hay tres formas de embeber el código JavaScript en una página HTML: Estilo interno, externo y en línea.

- JavaScript en el mismo documento HTML (estilo interno o embebido):

  - Uso de unas etiquetas predefinidas para marcar el texto (`<script>` y `</script>`).

  - Puede incluirse en cualquier parte del documento, aunque se recomienda que se defina dentro de la cabecera del documento HTML.

  - Esta técnica suele utilizarse cuando se definen instrucciones que se referenciarán desde cualquier parte del documento o cuando se definen funciones con fragmentos de código genéricos.

Integración del código con las etiquetas HTML

1. Incluirlo directamente dentro de la página HTML mediante la etiqueta `<script>` (estilo interno o embebido):

```
<!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>
```

Integración del código con las etiquetas HTML

- JavaScript en un archivo externo (estilo externo):
  - Ahorra código JavaScript si se van a ejecutar los mismos scripts en varias páginas HTML y mejora así su mantenimiento.
  - Aprovecha mejor la caché de los navegadores, carga más rápida de la página Web.

Integración del código con las etiquetas HTML

- JavaScript en un archivo externo (estilo externo):
  - Las mismas instrucciones de JavaScript que se incluyen entre un bloque `<script></script>` pueden almacenarse en un fichero externo con extensión “.js”.
  - La forma de acceder y enlazar esos ficheros “.js” con el documento HTML/XHTML es a través de la propia etiqueta `<script>`.
  - No existe un límite en el número de ficheros “.js” que pueden enlazarse en un mismo documento HTML/XHTML.

Integración del código con las etiquetas HTML

2. Utilizar el atributo `src` de la etiqueta `<script>` para especificar el fichero que contiene el código JavaScript (estilo externo):

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/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:

```
alert('Hola Mundo en JavaScript')
```

Integración del código con las etiquetas HTML

- JavaScript en atributos de elementos HTML (estilo en línea o inline):
  - Consiste en insertar fragmentos de JavaScript dentro de atributos de etiquetas HTML de la página.
  - Forma de controlar los eventos que suceden asociados a un elemento HTML concreto.
  - Principal desventaja: el mantenimiento y modificación del código puede resultar más complicado.

Integración del código con las etiquetas HTML

- JavaScript en atributos de elementos HTML (estilo en línea o inline):
  - Suele utilizarse como forma de controlar eventos asociados a elementos HTML.
  - Ensucia el código HTML y complica el mantenimiento del código JavaScript.
  - Solo para casos especiales.

"Hola mundo" con JavaScript

3. JavaScript en atributo del elemento HTML (estilo en línea o inline):

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
      charset=utf-8">
  </head>
  <body>
    <p onclick="alert('funcion');">
      Hola mundo
    </p>
  </body>
</html>
```

Colocación idónea del JavaScript

- El lugar clásico ha sido en la parte `<head>`, pero tiene dos problemas:
  - El script no puede tener todavía acceso al DOM del documento.
  - Retrasa la carga completa de la página.
- Actualmente se hace al final del `<body>`, pero tiene otro problema:
  - La página HTML puede estar disponible para el usuario antes de que el JavaScript se haya cargado: validación formularios, etc.
- También se puede optar por soluciones mixtas.

El lenguaje JavaScript: Sintaxis

- Especifica aspectos como:
  - Definición de comentarios.
  - Nombre de las variables.
  - Separación entre las diferentes instrucciones del código.
  - Etc.

El lenguaje JavaScript: Sintaxis

- Mayúsculas y minúsculas:
  - El lenguaje distingue entre mayúsculas y minúsculas, a diferencia de por ejemplo HTML.
  - No es lo mismo utilizar alert() que Alert().

El lenguaje JavaScript: Sintaxis

- Comentarios en el código:
  - Los comentarios no se interpretan por el navegador.
  - Existen dos formas de insertar comentarios:
    - Doble barra (//) – Se comenta una sola línea de código.
    - Barra y asterisco (/* al inicio y */ al final) – Se comentan varias líneas de código.

El lenguaje JavaScript: Sintaxis

- Comentarios en el código - Ejemplo:

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

El lenguaje JavaScript: Sintaxis

- Tabulación y saltos de línea:
  - JavaScript ignora los espacios, las tabulaciones y los saltos de línea con algunas excepciones.
  - Emplear la tabulación y los saltos de línea mejora la presentación y la legibilidad del código.

El lenguaje JavaScript: Sintaxis

- Tabulación y saltos de línea - Diferencias:

```
<script
type="text/javascript">va
r 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>
```

```
<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>
```

Aunque el código del lado izquierdo funciona, resulta poco legible.

El lenguaje JavaScript: Sintaxis

- El punto y coma:
  - Se suele insertar un signo de punto y coma (;) al final de cada instrucción de JavaScript.
  - Su utilidad es separar y diferenciar cada instrucción.
  - Se puede omitir si cada instrucción se encuentra en una línea independiente (la omisión del punto y coma no es una buena práctica de programación).

El lenguaje JavaScript: Sintaxis

- Palabras reservadas:
  - Algunas palabras no se pueden utilizar para definir nombres de variables, funciones o etiquetas.
  - Es aconsejable no utilizar tampoco las palabras reservadas para futuras versiones de JavaScript.
  - En https://www.ecma-international.org es posible consultar todas las palabras reservadas de JavaScript.

El lenguaje JavaScript: Sintaxis

- Etiqueta <noscript>:
  - JavasScript puede estar deshabilitado por el usuario en alguna ocasión en el navegador (Ej. en la barra de Firefox: about:config y javascript.enabled).
  - Si la página Web requiere JavaScript para su correcto funcionamiento, se puede mostrar un mensaje de aviso al usuario indicándole que lo debería activar para disfrutar completamente de la página:

```
<body>
  <noscript>
      <p>Esta página requiere para su funcionamiento
      el uso de JavaScript. Si lo has deshabilitado,
      por favor vuelve a activarlo.</p>
  </noscript>
</body>
```

EjemploNoScript

Tipos de datos

- Los tipos de datos especifican qué tipo de valor se guardará en una determinada variable.

- Los tres tipos de datos primitivos de JavaScript son (no tienen propiedades ni métodos):
  - Números.
  - Cadenas de texto.
  - Valores booleanos.

- En JavaScript, todo, excepto los tipos de datos primitivos, son objetos.

Tipos de datos

- Números:
  - En JavaScript existe sólo un tipo de dato numérico que se llama `number`.
  - Todos los números se representan a través del formato de punto flotante de 64 bits.
  - Este formato es el llamado `double` en los lenguajes Java o C++.
  - Es posible utilizar valores hexadecimales con `0x`.

Tipos de datos

- Cadenas de texto:
  - El tipo de datos para representar cadenas de texto se llama `string`.
  - Se pueden representar letras, dígitos, signos de puntuación o cualquier otro caracter de Unicode.
  - La cadena de caracteres se debe definir entre comillas dobles o comillas simples.

Tipos de datos

- Cadenas de texto - Secuencias de escape:

Secuencia de escape	Resultado
`\\`	Barra invertida
`\'`	Comilla simple
`\"`	Comillas dobles
`\n`	Salto de línea
`\t`	Tabulación horizontal
`\v`	Tabulación vertical
`\f`	Salto de página
`\r`	Retorno de carro
`\b`	Retroceso

Tipos de datos

- Valores booleanos:
  - También conocido como valor lógico o `boolean`.
  - Sólo admite dos valores: `true` o `false`.
  - Es muy útil a la hora de evaluar expresiones lógicas o verificar condiciones.

Variables

- Se pueden definir como zonas de la memoria de un ordenador que se identifican con un nombre y en las cuales se almacenan ciertos datos.

- El desarrollo de un script conlleva:
  - Declaración de variables.
  - Inicialización de variables.

Variables

- Declaración de variables:
  - Se declaran mediante la palabra clave `var` seguida por el nombre que se quiera dar a la variable.
    - `var mi_variable;`
  - Es posible declarar más de una variable en una sola línea.
    - `var mi_variable1, mi_variable2;`
  - Su ámbito es global (en cambio, `let` y `const` es local).

Variables

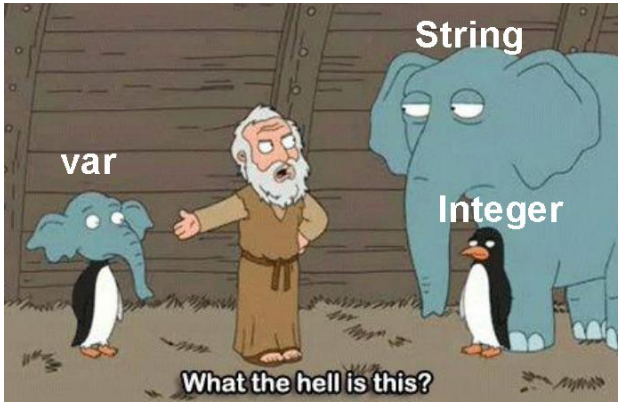
- Inicialización de variables:
  - Se puede asignar un valor a una variable de tres formas:
    - Asignación directa de un valor concreto.
    - Asignación indirecta a través de un cálculo en el que se implican a otras variables o constantes.
    - Asignación a través de la solicitud del valor al usuario del programa.
  - Ejemplos:
    - `var mi_variable_1 = 30;`
    - `var mi_variable_2 = mi_variable_1 + 10;`
    - `var mi_variable_3 = prompt('Introduce un valor:');`

Variables

- Los tipos de datos son dinámicos. Una variable puede cambiar de tipo:

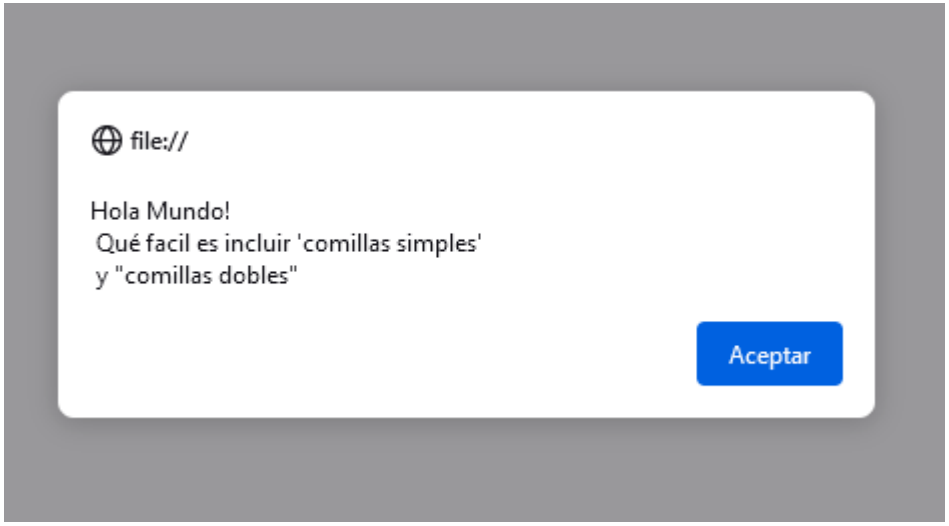
```
• var x = 7;
• x = true;
• x = "hola";
• x = ["lunes", "martes", "miércoles"];
• x = {nombre: "Carlos", apellidos: "Huertas",
  DNI: "123456789-X", edad: 27};
```

  - Esto puede ocasionar problemas.

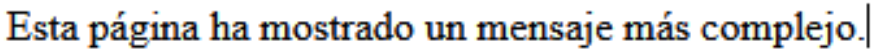


Variables

- Ejercicio: Crea una página que guarde un texto en una variable y luego lo muestre por pantalla así:



Al pulsar en 'Aceptar' se muestre lo siguiente:



Operadores

- Para construir expresiones con las que realizar cálculos más complejos.

- JavaScript utiliza principalmente cinco tipo de operadores:
  - Aritméticos.
  - Lógicos.
  - De asignación.
  - De comparación.
  - Condicionales.

Operadores

- Operadores aritméticos:
  - Permiten realizar cálculos elementales entre variables numéricas.

Operador	Nombre
+	Suma
-	Resta
*	Multiplicación
**	Exponenciación
/	División
%	Módulo (Resto)
++	Incremento
--	Decremento

Operadores

- Operadores lógicos:
  - Combinan diferentes expresiones lógicas con el fin de evaluar si el resultado de dicha combinación es verdadero o falso.

Operador	Nombre
&&	Y
||	O
!	No / Negación

Operadores

- Operadores de asignación:
  - Permiten obtener métodos abreviados para evitar escribir dos veces la variable que se encuentra a la izquierda del operador.

Operador	Nombre
+=	Suma y asigna
-=	Resta y asigna
*=	Multiplica y asigna
/*	Divide y asigna
%=	Módulo y asigna

Operadores

- Operadores de comparación:
  - Permiten comparar todo tipo de variables y devuelve un valor booleano.

Operador	Nombre
<	Menor que
<=	Menor o igual que
==	Igual
>	Mayor que
>=	Mayor o igual que
!=	Diferente
===	Estrictamente igual
!==	Estrictamente diferente

Operadores

- Operadores condicionales:
  - Permite indicar al navegador que ejecute una acción en concreto después de evaluar una expresión.

Operador	Nombre
?:	Condicional

  - Ejemplo:

```
<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>
```

Ejercicios

- Considera lo siguiente para los ejercicios ya que todavía no sabemos acceder al DOM:
  - Para recoger datos de usuario no podremos utilizar formularios, habrá que utilizar la función `prompt`.
  - Para mostrar datos por pantalla tendremos que utilizar el método `write` del objeto `document`, que recibe como parámetro el html a insertar en la página Web.

Ejercicios

- Ejercicio: Desarrolla un programa JavaScript que calcule el área y el perímetro de un rectángulo.

```
<script>
  var lado1 = prompt("Introduce la longitud del lado 1:", "");
  var lado2 = prompt("Introduce la longitud del lado 2:", "");

  area = lado1 * lado2;
  perimetro = parseInt(lado1) + parseInt(lado1) +
       parseInt(lado2) + parseInt(lado2);

  document.write("El área del rectángulo es: " + area);
  document.write("<br>");
  document.write("El perímetro del rectángulo es: " +
       perimetro);
</script>
```

EjercicioRectángulo

Ejercicios

- Realiza los ejercicios 1, 2 y 3 del bloque 'Ejercicios 2.1'.

Sentencias condicionales

- Con las sentencias condicionales se puede gestionar la toma de decisiones y el posterior resultado por parte del navegador.

- Dichas sentencias evalúan condiciones y ejecutan ciertas instrucciones en base al resultado de la condición.

- Las sentencias condicionales en JavaScript son:
  - `if`
  - `switch`
  - `while`
  - `for`

Sentencias condicionales

- `if` - sintaxis (1):

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

Sentencias condicionales

- `if` - sintaxis (2):

```
if (expresión) {
     instrucciones_si_true
} else {
     instrucciones_si_false
}
```

Sentencias condicionales

- `if` - sintaxis (3):

```
if (expresión_1) {
     instrucciones_1
} else if (expresión_2) {
     instrucciones_2
} else {
     instrucciones_3
}
```

Sentencias condicionales

- `switch` - sintaxis:

```
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
}
```

Sentencias condicionales

- `while` - sintaxis:

```
(1)
while (expresión) {
     instrucciones
}

(2)
do {
     instrucciones
} while (expresión)
```

Sentencias condicionales

- `for` - sintaxis:

```
for (valor_inicial_variable;
expresión_condicional;
incremento_o_decremento_de_la_variable) {
    cuerpo_del_bucle
}
```

```
for (i=0; i<10; i++) {
  document.write("Valor variable índice es: " + i + "<br>")
}
```

EjemploFor

Ejercicios

- Realiza los ejercicios 4, 5 y 6 del bloque 'Ejercicios 2.1'.

- Realiza todos los ejercicios del bloque 'Ejercicios 2.2'.

DESARROLLO WEB EN ENTORNO CLIENTE

U.D. 2 (parte 2/2):

Utilización de los objetos predefinidos de JavaScript

Conceptos previos de JavaScript

- **Objetos**: son elementos programables que podemos manipular para cambiar sus propiedades, realizar tareas a través de sus métodos o ejecutar un evento relacionado con el mismo objeto. Ej: `document.`
  - **Propiedades**: son las características de un objeto. Ej: `bgcolor.`
  - **Métodos**: son funciones o tareas específicas que pueden realizar los objetos. Ej: `write().`
  - **Eventos**: son situaciones que pueden llegar a realizarse o no. Ej: `onclick.`

Objetos nativos de JavaScript

- JavaScript proporciona una serie de objetos definidos nativamente que no dependen del navegador.

- Para crear un objeto se utiliza la palabra clave `new`. Ejemplo:

```
var mi_objeto = new Object();
```

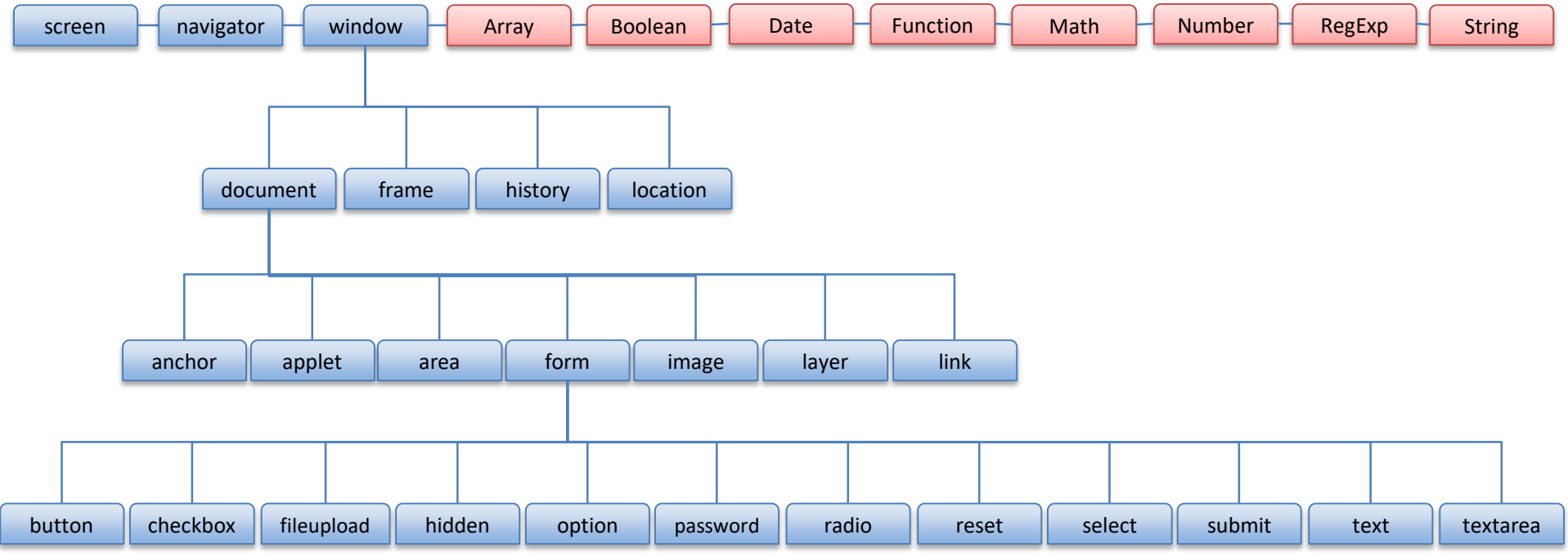
Objetos nativos de JavaScript

- En JavaScript se accede a las propiedades y a los métodos de los objetos mediante el operador punto (".").

```
mi_objeto.nombre_propiedad;
mi_objeto.nombre_función([parámetros]);
```

Objetos nativos de JavaScript

- Los objetos de JavaScript se ordenan de modo jerárquico.



Objetos nativos de JavaScript

- El objeto Date:
  - Permite realizar controles relacionados con el tiempo en las aplicaciones Web.
  - Cuenta con una serie de métodos divididos en tres subconjuntos:
    - Métodos de lectura.
    - Métodos de escritura.
    - Métodos de conversión.

Objetos nativos de JavaScript

- El objeto Date - Métodos:

Métodos				
getDate()	getTime()	getUTCMonth()	setMonth()	setUTCMonth()
getDay()	getTimezoneOffset()	getUTCSeconds()	setSeconds()	setUTCSeconds()
getFullYear()	getUTCDate()	parse()	setTime()	toDateString()
getHours()	getUTCDay()	setDate()	setUTCDate()	toLocaleDateString()
getMilliseconds()	getUTCFullYear()	setFullYear()	setUTCFullYear()	toLocaleTimeString()
getMinutes()	getUTCHours()	setHours()	setUTCHours()	toLocaleString()
getMonth()	getUTCMilliseconds()	setMilliseconds()	setUTCMilliseconds()	toTimeString()
getSeconds()	getUTCMinutes()	setMinutes()	setUTCMinutes()	toUTCString()

Objetos nativos de JavaScript

- Ejercicio: Muestra la fecha y hora actual mediante una ventana emergente. También se muestre los milisegundos que faltan para que acabe el presente año.

- Ejercicio: Muestra la fecha actual mediante una ventana emergente con el formato “La fecha actual es: *2* de *octubre* de *2023*”.

Objetos nativos de JavaScript

- El objeto Math: Permite realizar operaciones matemáticas complejas.

Métodos		
abs()	exp()	random()
acos()	floor()	round()
asin()	log()	sin()
atan()	max()	sqrt()
ceil()	min()	tan()
cos()	pow()	

Propiedades
E
LN2
LN10
LOG2E
LOG10E
PI
SQRT1_2
SQRT2

Objetos nativos de JavaScript

- Ejercicio: Calcula el área de un círculo en función de su radio (area = π * radio^2).

- Ejercicio: Muestra en la pantalla un número entero aleatorio entre 10 y 100 (primero y último incluidos).

Objetos nativos de JavaScript

- El objeto Number: Permite realizar tareas relacionadas con tipos de datos numéricos.

Métodos
`toExponential()`
`toFixed()`
`toPrecision()`

Propiedades
`MAX_VALUE`
`MIN_VALUE`
`NaN`
`NEGATIVE_INFINITY`
`POSITIVE_INFINITY`

Objetos nativos de JavaScript

- Ejercicio: Muestra en la pantalla el mayor número posible en JavaScript, el menor, el infinito positivo y negativo. También formatea el número π a 3 cifras.

Objetos nativos de JavaScript

- El objeto String: Permite manipular las cadenas de texto.

Métodos			
anchor()	fixed()	link()	strike()
big()	fontcolor()	match()	sub()
blink()	fontsize()	replace()	substr()
bold()	fromCharCode()	search()	substring()
charAt()	indexOf()	slice()	sup()
charCodeAt()	italics()	small()	toLowerCase()
concat()	lastIndexOf()	split()	toUpperCase()

Propiedades
length

Objetos nativos de JavaScript

- Ejercicio: Muestra en la pantalla varias frases, una en cursiva, otra en negrita, tachado, cambia el color de la fuente, su tamaño, etc. También visualiza la longitud de la cadena de caracteres.

Objetos nativos de JavaScript

- El objeto Array (en U.D. 3): Usado en la construcción y manipulación de *arrays*.

Métodos	
`push()`	`shift()`
`concat()`	`pop()`
`join()`	`slice()`
`reverse()`	`sort()`
`unshift()`	`splice()`

Propiedades
`length`
`prototype`

Interacción de los objetos con el navegador

- Además de los objetos presentados anteriormente, existe otro tipo de objetos que permiten manipular diferentes características del navegador en sí mismo. Son los siguientes.

Interacción de los objetos con el navegador

- El objeto Navigator:
  - Permite identificar las características de la plataforma sobre la cual se ejecuta la aplicación Web. Ejemplo:
    - Tipo de navegador.
    - Versión del navegador.
    - Sistema operativo.

Métodos
`javaEnable()`

Propiedades
`appCodeName`
`appName`
`appVersion`
`cookieEnable`
`platform`
`userAgent`
`geolocation`
`plugins`

Interacción de los objetos con el navegador

- Ejercicio: Muestra en la pantalla el navegador y sistema operativo que estás usando.

Interacción de los objetos con el navegador

- El objeto Screen:
  - Corresponde a la pantalla utilizada por el usuario.
  - Todas sus propiedades son solamente de lectura.

Propiedades
availHeight
availWidth
colorDepth
height
pixelDepth
width

Interacción de los objetos con el navegador

- Ejercicio: Muestra la altura y anchura máxima de tu pantalla en píxeles. También la altura y anchura máxima disponible para el uso de ventanas.

Interacción de los objetos con el navegador

- El objeto Window:
  - Se considera el objeto más importante de JavaScript.
  - Permite gestionar las ventanas del navegador.
  - Es un objeto implícito, con lo cual no es necesario nombrarlo para acceder a los objetos que se encuentran debajo de su nivel de jerarquía.

Interacción de los objetos con el navegador

- El objeto Window - Métodos y propiedades:

Métodos		
alert()	forward()	resizeTo()
back()	home()	scrollBy()
blur()	moveBy()	scrollTo()
close()	moveTo()	setInterval()
confirm()	open()	setTimeOut()
find()	print()	stop()
focus()	prompt()	

Propiedades		
closed	location	pageYoffset
defaultStatus	locationbar	parent
document	menubar	personalbar
frames	name	scrollbars
history	opener	self
innerHeight	outerHeight	status
innerWidth	outerWidth	toolbar
length	pageXoffset	top

Interacción de los objetos con el navegador

- Ejercicio: Abre una ventana de altura 150 píxeles y anchura 250 situada a 100 a la derecha y 100 abajo respecto de la esquina superior izquierda de la ventana principal.
- Después, muévela a 400 a la derecha y 100 abajo respecto de la esquina superior izquierda.
- Luego, cada 100 milisegundos, muévela 19 veces a 5 píxeles a la derecha y 5 abajo respecto de la posición anterior.
- Al final, se debe cerrar de forma automática.

Interacción de los objetos con el navegador

- El objeto Document:
  - Se refiere a los documentos que se cargan en la ventana del navegador.
  - Permite manipular las propiedades y el contenido de los principales elementos de las páginas Web.
  - Cuenta con una serie de sub-objetos como los vínculos, puntos de anclaje, imágenes o formularios.

Interacción de los objetos con el navegador

- El objeto Document - Métodos y propiedades:

Métodos	
captureEvents()	open()
close()	releaseEvents()
getSelection()	routeEvents()
handleEvent()	write()
home()	writeln()

Propiedades		
alinkColor	fgColor	plugins
anchors	forms	referrer
applets	images	title
bgColor	lastModified	URL
cookie	layers	vlinkColor
domain	linkColor	
embeds	links	

Interacción de los objetos con el navegador

- El objeto History:
  - Almacena las referencias de las páginas Web visitadas.
  - Las referencias se guardan en una lista utilizada principalmente para desplazarse entre dichas páginas Web.
  - No es posible acceder a los nombres de las URL, ya que es información privada.

Métodos
back()
forward()
go()

Propiedades
current
length
next
previous

Interacción de los objetos con el navegador

- El objeto Location:
  - Corresponde a la URL de la página Web en uso.
  - Su principal función es la de consultar las diferentes partes que forman una URL como por ejemplo:
    - El dominio.
    - El protocolo.
    - El puerto.

Métodos
assign()
reload()
replace()

Propiedades
hash
host
hostname
href
pathname
port
protocol
search

Generación de elementos HTML desde código

- Uno de los principales objetivos de JavaScript es convertir un documento HTML estático en una aplicación Web dinámica.

- Por ejemplo, es posible ejecutar instrucciones que crean nuevas ventanas con contenido propio, en lugar de mostrar dicho contenido en la ventana activa.

Generación de elementos HTML desde código

- Con JavaScript es posible manipular los objetos que representan el contenido de una página Web con el fin de crear documentos dinámicos.

- Por ejemplo, es posible definir el título de una página Web basándose en el S.O. utilizado:

```
<script>
  var SO = navigator.platform;
  document.write("<h1>Documento abierto con: " + SO +
      "</h1>");
</script>
```

Generación de elementos HTML desde código

- Otro ejemplo es crear documentos en ventanas emergentes (secundarias):

```
<script>
  var texto = prompt("Ingresa un título para la nueva
      ventana (secundaria): ");
  var ventanaNueva = window.open();
  ventanaNueva.document.write("<h1>" + texto +
      "</h1>");
</script>
```

Generación de elementos HTML desde código

- La generación de código HTML a partir de JavaScript no se limita sólo a la creación de texto como en los ejemplos anteriores. Es posible crear y manipular todo tipo de objetos:

```
<script>
  document.write("<form name=\"cambiacolor\">");
  document.write("<b>Selecciona un color para el fondo de
       página:</b><br>");
  document.write("<select name=\"color\">");
  document.write("<option value=\"red\">Rojo</option>");
  document.write("<option value=\"blue\">Azul</option>");
  document.write("<option value=\"yellow\">Amarillo</option>");
  document.write("<option value=\"green\">Verde</option>");
  document.write("</select>");
  document.write("<input type=\"button\" value=\"Modifica el color\"
   onclick=\"document.bgColor=document.cambiacolor.color.value\">");
  document.write("</form>");
</script>
```

Generación de elementos HTML desde código

- A partir del script anterior se obtiene la siguiente página Web dinámica:

Gestión de las ventanas

- JavaScript permite gestionar diferentes aspectos relacionados con las ventanas como por ejemplo abrir nuevas ventanas al presionar un botón.

- Cada una de estas ventanas tiene un tamaño, posición y estilo diferente.

- Estas ventanas emergentes suelen tener un contenido dinámico.

Gestión de las ventanas

- Abrir y cerrar nuevas ventanas:
  - Es una operación muy común en las páginas Web.
  - En algunas ocasiones se abren sin que el usuario haga algo.
  - HTML permite abrir nuevas ventanas pero no permite ningún control posterior sobre ellas.

Gestión de las ventanas

- Abrir y cerrar nuevas ventanas:
  - Con JavaScript es posible abrir una ventana vacía mediante el método `open()`:

```
nuevaVentana = window.open();
```

  - De este modo la variable llamada `nuevaVentana` contendrá una referencia a la ventana creada.

Gestión de las ventanas

- Abrir y cerrar nuevas ventanas:
  - El método `open()` cuenta con tres parámetros:
    - URL.
    - Nombre de la ventana.
    - Colección de atributos que definen la apariencia de la ventana.
  - Ejemplo:

```
nuevaVentana = window.open("https://www.misitioWeb.com",
        "Publicidad", "height=500,width=500");
```

EjemploAbrirWeb

Gestión de las ventanas

- Para cerrar una ventana se puede invocar el método `close()`:

```
nuevaVentana.document.write('<input type=button
      value=Cerrar onClick=window.close()>');
```

Gestión de las ventanas

- Apariencia de las ventanas:
  - La ventanas cuentan con propiedades que permiten decidir su tamaño, ubicación o los elementos que contendrá.

Propiedades	
`directories`	`scrollbars`
`height`	`status`
`menubar`	`toolbar`
`resizable`	`width`

Gestión de las ventanas

- Un ejemplo completo de abrir y cerrar una ventana secundaria:

```
<html><head></head><body>
  <h1> Ejemplo de apariencia de una ventana </h1><br>
  <input type="Button" value="Abre una Ventana" onclick="
   myWindow=window.open('', 'Nueva Ventana', 'width=300,height=200');
   myWindow.document.write('<html>');
   myWindow.document.write('<head>');
   myWindow.document.write('<title>Ventana Test</title>');
   myWindow.document.write('</head>');
   myWindow.document.write('<body>');
   myWindow.document.writeln('Se usan las propiedades: ');
   myWindow.document.write('<li>height=200</li> <li>width=300</li>');
   myWindow.document.write('<input type=button value=Cerrar
        onClick=window.close()>');
   myWindow.document.write('</body>');
   myWindow.document.write('</html>');" />
</body></html>
```

Gestión de las ventanas

- Un ejemplo completo de abrir múltiples ventanas secundarias:

```
<html><head></head><body>
  <center><h1>Apertura de múltiples ventanas secundarias (5)</h1>
    <input type="Button" value="Abre múltiples ventanas..."
        onclick="for (i=0;i<5;i++) {
        myWindow=window.open('','','width=200,height=200');
        myWindow.document.write('<html>');
        myWindow.document.write('<head>');
        myWindow.document.write('<title>Ventana '+i+'</title>');
        myWindow.document.write('</head>');
        myWindow.document.write('<body>');
        myWindow.document.write('Ventana ' + i);
        myWindow.document.write('<input type=button value=Cerrar
                onClick=window.close()>');
        myWindow.document.write('</body>');
        myWindow.document.write('</html>'); } />
  </center></body></html>
```

Gestión de las ventanas

- Comunicación entre ventanas:
  - Desde una ventana se pueden abrir o cerrar nuevas ventanas.
  - La primera se denomina ventana principal, mientras que las segundas se denominan ventanas secundarias.
  - Desde la ventana principal se puede acceder a las ventanas secundarias, pero por seguridad, por ejemplo, la secundaria no puede cerrar la principal.

Gestión de las ventanas

- Comunicación entre ventanas:
  - En el siguiente ejemplo se muestra cómo acceder a una ventana secundaria:

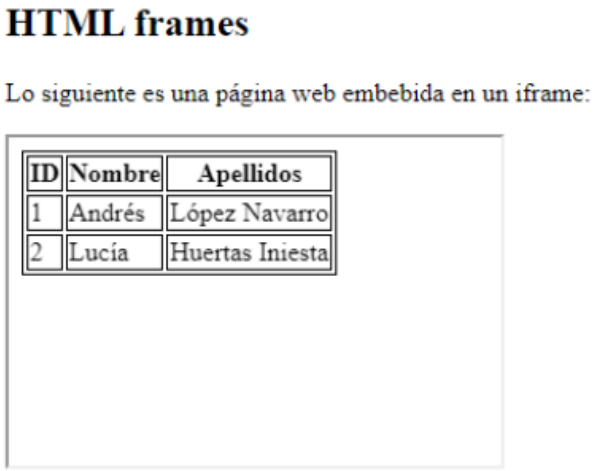
```
<html><head></head><body>
 <script>
  var ventanaSecundaria = window.open("", "ventanaSec", "width=500,
   height=500");
 </script>
 <div align="center"><h1> Comunicación entre ventanas </h1><br>
  <form name=formulario>
   <input type=text name=url size=50 value="https://www.">
   <input type=button value="Mostrar URL en ventana secundaria"
onclick="ventanaSecundaria.location = document.formulario.url.value;">
  </form>
 </div>
</body></html>
```

Aplicaciones prácticas de los marcos

- Es posible dividir la ventana de una aplicación Web en dos o más partes independientes.
- Con JavaScript se puede interactuar entre estos sectores independientes.
- Dichos sectores se denominan marcos.
- HTML5 considera **obsoleto** el uso de marcos y se desaconseja su uso:
  https://www.eniun.com/marcos-frames-html5/
  https://lenguajehtml.com/html/semantica/etiquetas-html-obsoletas/

Aplicaciones prácticas de los marcos

- Algunas páginas Web presentan una estructura en la cual una parte permanece fija mientras que otra va cambiando.



- Por ejemplo, insertar un mapa de Google en una página Web:

https://norfipc.com/mapas/como-insertar-mapa-google-pagina-iframe-codigos-trucos.php

Aplicaciones prácticas de los marcos

- Los marcos se definían antes utilizando HTML mediante estas etiquetas:
  - `<frameset>`
  - `<frame>` →

Atributos
frameborder
marginheight
marginwidth
name
noresize
scrolling
src

  - `<iframe>` en la actualidad: →

Atributos
allow
allowfullscreen
height
name
sandbox
src
width

https://developer.mozilla.org/es/docs/Web/HTML/Element/iframe
https://developer.mozilla.org/es/docs/Learn/HTML/Multimedia_and_embedding/Other_embedding_technologies
https://desarrolloweb.com/articulos/667.php

Aplicaciones prácticas de los marcos

- JavaScript permite manipular los marcos mediante las propiedades `frames`, `parent` y `top` del objeto `window`.

- Por ejemplo, se define un documento HTML con dos marcos:

```
<html><head><title>Ejemplos de control de marcos</title>
    <frameset cols="50%,50%">
      <frame src="Marco1.html" name="Marco1" noresize>
      <frame src="Marco2.html" name="Marco2" noresize>
    </frameset></head>
  <body></body>
</html>
```

Aplicaciones prácticas de los marcos

- El primer marco (`Marco1`) contiene la página `Marco1.html`:

```
<html>
  <body>
    <form name="form1">
      <select name="color">
        <option value="green">Verde
        <option value="blue">Azul
      </select><br><br>
      <select name="marco">
        <option value="0">Izquierda
        <option value="1">Derecha
      </select>
    </form>
  </body>
</html>
```

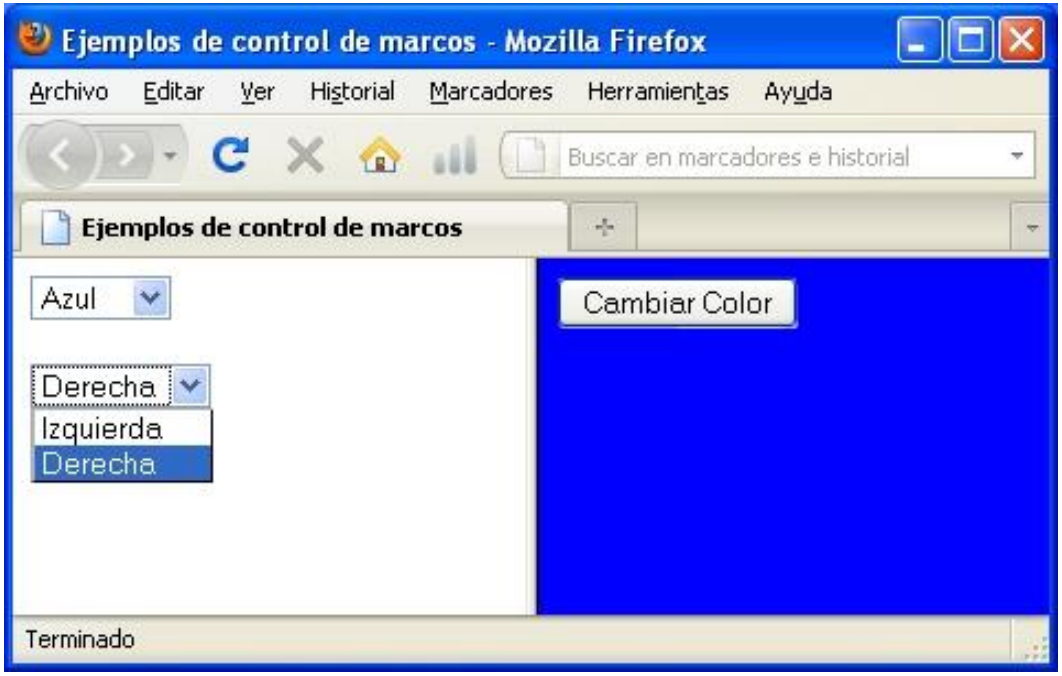
Aplicaciones prácticas de los marcos

- El segundo marco (`Marco2`) contiene la página `Marco2.html`:

```
<html><body>
    <form>
      <input type="button" value="Cambiar Color" onclick="
        var campoColor = parent.Marco1.document.form1.color
        if (campoColor.selectedIndex == 0) {
          var colorin = 'green'
        } else { var colorin = 'blue' }
        var campoMarco = parent.Marco1.document.form1.marco
        if (campoMarco.selectedIndex == 0) {
          window.parent.Marco1.document.bgColor = colorin
        } else {
          window.parent.Marco2.document.bgColor = colorin
        }" />
    </form>
</body></html>
```

Aplicaciones prácticas de los marcos

- El resultado se puede ver en esta imagen:



- Política del “mismo origen”:
  https://es.javascript.info/cross-window-communication

Aplicaciones prácticas de los marcos

- Ejercicio: Inserta solamente dos marcos con `<iframe>` sin comunicación entre ellos.

Ejercicios

- Realiza todos los ejercicios del bloque 'Ejercicios 2.3' (entrega).