DESARROLLO WEB EN ENTORNO CLIENTE

CAPÍTULO 2:

Utilización de los objetos predefinidos del lenguaje JavaScript

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 podemos 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.
- Creado en 1995 por Netscape.
- ECMAScript es su especificación estándar.

“Hola mundo” con JavaScript

- La forma más inmediata de empezar a experimentar con JavaScript es escribir secuencias de comandos simples.
- Es necesario sólo un navegador web y un editor de texto.

“Hola mundo” con JavaScript

- Hay tres formas de embeber el código JavaScript en una página 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>
```

“Hola mundo” con JavaScript

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')
```

“Hola mundo” con JavaScript

- JavaScript en un fichero separado:
  - 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.

“Hola mundo” con JavaScript

“Hola mundo” con JavaScript

3. JavaScript en atributo del elemento HTML (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>
```

“Hola mundo” con JavaScript

- JavaScript como elemento inline (dentro de atributos de etiquetas HTML):
  - 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.

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 www.ecmascript.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>
```

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

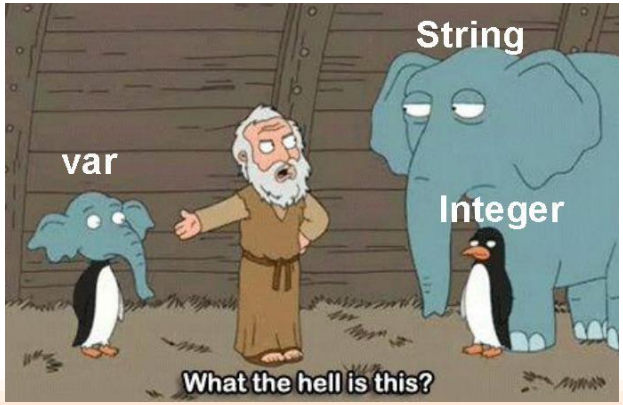
- Inicialización de variables:
  - Se puede asignar un valor a una variable de tres formas:
    - Asignación directa de un valor concreto.
    - Asignación indirecta a través de un cálculo en el que se implican a otras variables o constantes.
    - Asignación a través de la solicitud del valor al usuario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

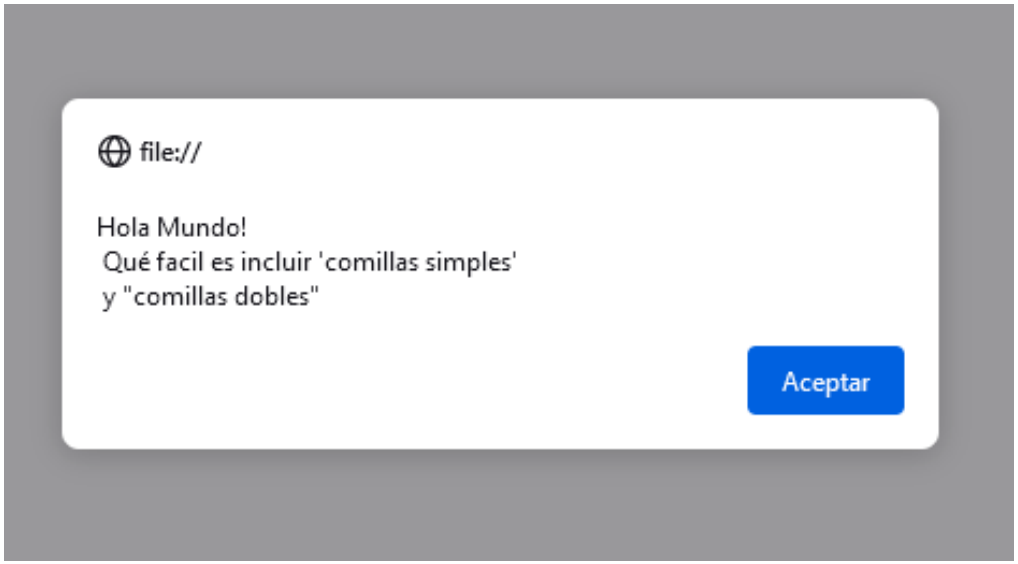
```
• var x = 7;
• x = true;
• x = "hola";
• x = ["lunes", "martes", "miércoles"];
• x = {nombre: "Carlos", apellidos: "Huertas",
  DNI: "123456789-X", edad: 27};
```

  - Esto puede ocasionar problemas.

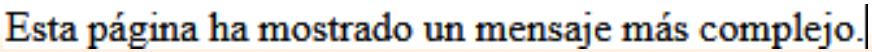


2.1. Ejercicio

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.

2.2 Ejercicio

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

Características de JavaScript

JavaScript es un lenguaje de programación:

- **Ligero:** está diseñado para ocupar poco espacio en la memoria, es fácil de implementar y cuenta con una sintaxis y semántica simples, se puede aprender en poco tiempo.

- **Interpretado:** significa que se lee y ejecuta cada línea de código de forma secuencial, al contrario que lenguajes compilados cuyas líneas de código son convertidas en su conjunto a lenguaje máquina para se ejecutadas posteriormente.

- **Basado en prototipos:** se trata de una variante de la POO en la que los objetos no se crean instanciando clases sino clonando otros objetos o creándolos directamente. En JS todo es un objeto.

Características de JavaScript

JavaScript es un lenguaje de programación:

- **Case *sensitive*:** es sensible a mayúsculas minúsculas, por lo que rojo, Rojo y ROJO, serán 3 identificadores distintos.

- **Débilmente tipado:** el lenguaje no necesita conocer al detalle con qué tipo de dato está trabajando en cada momento. Es lo suficientemente inteligente para deducirlo o realizar conversiones entre tipos de datos a lo largo de la ejecución del programa.

- **Multiparadigma:** A diferencia de otros lenguajes, permite programar aplicaciones completas desde cero aplicando muy distintos paradigmas de programación.

Características de JavaScript

JavaScript es un lenguaje de programación:

- **Monohilo:** un único hilo de ejecución se encarga de realizar el trabajo interpretado del código, de forma que se dificultan los interbloqueos y se favorece el uso de las llamadas asíncronas.

- **Dinámico:** el lenguaje es capaz de cambiar importantes características del programa como la estructura de un objeto o el tipo de variable, mientras se está ejecutando.

- **Programación orientada a objetos:** un modelo de programación mucho más cercano al mundo real donde se modelan patrones a través de clases y se instancian en forma de objetos. Los objetos se relacionan entre sí para alcanzar los objetivos de las aplicaciones.

Características de JavaScript

JavaScript es un lenguaje de programación:

- **Programación imperativa:** un modelo de programación consistente en una secuencia de instrucciones que determinan qué debe hacer la máquina en cada momento paso a paso. Se centra en el “cómo” de la solución.

- **Programación declarativa:** un modelo de programación que consiste en describir qué se quiere obtener al finalizar la ejecución del programa, en lugar de cómo se quiere obtener. Se centra en el “qué” de la solución.

2.2 Ejercicio - Solución

- 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>
```

2.3 Hoja de ejercicios

- Realiza los ejercicios 1, 2 y 3 del bloque

'UD2_2_3_HojaDeEjercicios_JS_DWEC'

Sentencia de bloque

La **sentencia de bloque** es una estructuras sencilla, que se utiliza simplemente para agrupar declaraciones, y no modifica el flujo de ejecución.

```
{
        var mascota = "Perro";
         console.log("Mi mascota es " + mascota);
}
```

Estructuras de control de flujo

El **control de flujo** (*Control flow*) es, en ciertos lenguajes de programación como en JavaScript, el orden en que son ejecutadas las instrucciones en el código.

El código JavaScript se ejecuta, a grandes rasgos, en un orden líneal, desde la primera línea de código hasta la última. Sin embargo, este orden se altera cuando el intérprete (el navegador) encuentra un bloque de código que cambia el flujo.

A los bloques de código que tienen la capacidad de alterar el flujo de ejecución de nuestro código JavaScript se les conoce como **estructuras de control**.

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`

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 de interación

Bucles o *loops*, se utilizan para ejecutar un bloque de código de forma reiterativa, y basada en alguna condición.

- `while` - sintaxis:

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

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

Sentencias de interación

La **sentencia for** recibe tres argumentos que determinan la forma de iteración del bloque de código.

- Una **expresión inicial** que se ejecuta una sola vez, antes de la primer iteración;
- Una **condición** que debe cumplirse para ejecutar el bloque de código;
- Una **expresión final** que se ejecuta al final de cada iteración;

- `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>")
 }
```

Sentencias de interación

for...of

La **sentencia for...of** ejecuta un bloque de código para cada elemento de un objeto iterable , como lo son: String, Array, objetos similares a array (por ejemplo, arguments or NodeList), TypedArray, Map, Set e iterables definidos por el usuario.

```
let iterable = [10, 20, 30];


for (let value of iterable) {
      value += 1;
       console.log(value);
 }
```

Sentencias de interación

for … in

La **sentencia for…in** itera sobre los elementos de un objeto en un orden arbitrario, ejecutando el bloque de código una vez por cada uno.

```
var automovil = { marca: "Audi", modelo: "A1", año: 2020 }

for (var dato in automovil) { console.log(dato + " es " + automovil[dato]); }
```

Clausura de iteraciones

Se puede causar que la iteración termine de un modo brusco usando: **break, continue, throw** or **return**. En estos casos la iteración se cierra.

2.3 Hoja de ejercicios

- Realiza los ejercicios 4, 5, 6 y 7 del bloque

'UD2_2_3_HojaDeEjercicios_JS_DWEC'

2.4 Hoja de ejercicios

- Realiza todos los ejercicios.

'UD2_2_4_HojaDeEjercicios_JS_DWEC'

'UD2_2_4_1 Extra HojaDeEjercicios_JS_DWEC'

DESARROLLO WEB EN ENTORNO CLIENTE

CAPÍTULO 2:

Utilización de los objetos predefinidos del lenguaje 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 y propiedades

Una propiedad de un objeto se puede explicar como una variable adjunta al objeto. Las propiedades de un objeto básicamente son lo mismo que las variables comunes de JavaScript, excepto por el nexo con el objeto.

Las propiedades de un objeto definen las características del objeto. Accedes a las propiedades de un objeto con una simple notación de puntos: objectName.propertyName;

Crear un objeto llamado myCar : **var myCar = new Object();**

```
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
```

```
myCar["make"] = "Ford";
myCar["model"] = "Mustang";
myCar["year"] = 1969;
```

Objetos y propiedades

Se podría escribir utilizando un **iniciador de objeto,** que es una lista delimitada por comas de cero o más pares de nombres de propiedad y valores asociados de un objeto, encerrados entre llaves:

```
var myCar = {
      make: "Ford",
      model: "Mustang",
      year: 1969,
};
```

Las propiedades no asignadas de un objeto son undefined (y no null).

Objetos y propiedades

El nombre de la propiedad de un objeto puede ser **cualquier cadena** válida de JavaScript, o **cualquier cosa que se pueda convertir en una cadena**, incluyendo una cadena vacía.

Sin embargo, cualquier nombre de propiedad que no sea un identificador válido de JavaScript (por ejemplo, el nombre de alguna propiedad que tenga un espacio o un guión, o comience con un número) solo se puede acceder utilizando la notación de corchetes.

Esta notación **es muy útil también cuando los nombres de propiedades son determinados dinámicamente** (cuando el nombre de la propiedad no se determina hasta el tiempo de ejecución). Ejemplos de esto se muestran a continuación:

Objetos y propiedades

```
// Se crean y asignan cuatro variables de una sola vez,
// separadas por comas
var myObj = new Object(),
  str = "myString",
  rand = Math.random(),
  obj = new Object();

myObj.type = "Sintaxis de puntos";
myObj["fecha de creación"] = "Cadena con espacios";
myObj[str] = "Valor de cadena";
myObj[rand] = "Número aleatorio";
myObj[obj] = "Object";
myObj[""] = "Incluso una cadena vacía";

console.log(myObj);
```

Todas las claves con notación en corchetes se convierten a cadenas a menos que estas sean símbolos, ya que los nombres de las propiedades (claves) en Javascript pueden solo pueden ser cadenas o símbolos.

En el código anterior, cuando la clave obj se añadió a myObj, Javascript llamará al método obj.toString(), y usará la cadena resultante de esta llamada como la nueva clave.

Objetos y propiedades

Puedes usar la notación de corchetes con for...in para iterar sobre todas las propiedades enumerables de un objeto.

```
function showProps(obj, objName) {
  var result = ``;
  for (var i in obj) {
    // obj.hasOwnProperty() se usa para filtrar propiedades de la cadena
de prototipos del objeto
    if (obj.hasOwnProperty(i)) {
      result += `${objName}.${i} = ${obj[i]}\n`;
    }
  }
  return result;
}
```

Enumerar las propiedad de un objeto

A partir de ECMAScript 5, hay tres formas nativas para enumerar/recorrer las propiedades de objetos:

•bucles for...in Este método recorre todas las propiedades enumerables de un objeto y su cadena de prototipos

•Object.keys(o) Este método devuelve un arreglo con todos los nombres de propiedades enumerables ("claves") propias (no en la cadena de prototipos) de un objeto o.

•Object.getOwnPropertyNames(o) Este método devuelve un arreglo que contiene todos los nombres (enumerables o no) de las propiedades de un objeto o.

Enumerar las propiedad de un objeto

Para enumerar todas las propiedades de un objeto.

```
function listAllProperties(o) {
  var objectToInspect;
  var result = [];

  for (
    objectToInspect = o;
    objectToInspect !== null;
    objectToInspect = Object.getPrototypeOf(objectToInspect)
  ) {
    result = result.concat(Object.getOwnPropertyNames(objectToInspect));
  }

  return result;
}
```

Esto puede ser útil para revelar propiedades "ocultas" (propiedades de la cadena de prototipos a las que no se puede acceder a través del objeto, porque otra propiedad tiene el mismo nombre en la cadena de prototipos).

Creación de nuevos objetos

Usando un iniciador de objeto

```
var obj = {
  property_1: value_1, // property_# puede ser un identificador...
  2: value_2, // o un número...
  // ...,
  "property n": value_n,
}; // o una cadena
```

Los iniciadores de objetos son expresiones, y cada iniciador de objeto da como resultado un nuevo objeto donde la instrucción de creación sea ejecutada. Los iniciadores de objetos idénticos crean objetos distintos que no se compararán entre sí como iguales.

Los objetos se crean como si se hiciera una llamada a new Object(); es decir, los objetos hechos a partir de expresiones literales de objeto son instancias de Object.

Creación de nuevos objetos

2 ejemplo más:

- La siguiente declaración crea un objeto y lo asigna a la variable x si y solo si la expresión cond es true.

```
if (cond) var x = { greeting: "¡Hola!" };
```

- El siguiente ejemplo crea myHonda con tres propiedades. Observa que la propiedad engine también es un objeto con sus propias propiedades.

```
var myHonda = { color: "red", wheels: 4, engine: { cylinders: 4, size:
2.2 } };
```

Creación de nuevos objetos

Usando una función constructora

Dos pasos:

1.Definir el tipo de objeto escribiendo una función constructora. Existe una fuerte convención, con buena razón, para utilizar en mayúscula la letra inicial.

2.Crear una instancia del objeto con el operador new.

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
```

```
var mycar = new Car("Eagle", "Talon TSi", 1993);
```

Creación de nuevos objetos

En un ejemplo más complejo. Se crean 2 objetos, y se añade la propiedad a Car de owner, cuyo valor será un objeto Person.

```
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
```

```
var rand = new Person("Rand McKinnon", 33, "M");
var ken = new Person("Ken Jones", 39, "M");
```

Creación de nuevos objetos

Se puede volver a escribir la definición de Car para incluir una nueva propiedad.

```
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
```

```
var car1 = new Car("Eagle", "Talon TSi", 1993, rand);
var car2 = new Car("Nissan", "300ZX", 1992, ken);
```

Y el acceso al valor es:

```
car2.owner.name;
```

Creación de nuevos objetos

Siempre se puede añadir una propiedad a un objeto previamente definido.

```
car1.color = "black";
```

Sin embargo, esto no afecta a ningún otro objeto. Para agregar la nueva propiedad a todos los objetos del mismo tipo, tienes que añadir la propiedad a la definición del tipo de objeto Car.

Creación de nuevos objetos

Usando el método Object.create()

Puede ser muy útil, ya que te permite elegir el prototipo del objeto que deseas crear, sin tener que definir una función constructora.

```
// Propiedades y método de encapsulación para Animal
var Animal = {
  type: "Invertebrates", // Valor predeterminado de las propiedades
  displayType: function () {
    // Método que mostrará el tipo de Animal
    console.log(this.type);
  },
};

// Crea un nuevo tipo de animal llamado animal1
var animal1 = Object.create(Animal);
animal1.displayType(); // Muestra: Invertebrates

// Crea un nuevo tipo de animal llamado Fishes
var fish = Object.create(Animal);
fish.type = "Fishes";
fish.displayType(); // Muestra: Fishes
```

Herencia

Todos los objetos en JavaScript heredan de al menos otro objeto. El objeto del que se hereda se conoce como el prototipo, y las propiedades heredadas se pueden encontrar en el objeto **prototype** del constructor.

Definición de las propiedades de un tipo objeto

Puedes agregar una propiedad a un tipo de objeto definido previamente mediante el uso de la propiedad **prototype**. Esto define una propiedad que es compartida por todos los objetos del tipo especificado, en lugar de por una sola instancia del objeto.

```
Car.prototype.color = null;

car1.color = "black";
```

Definición de métodos

Un *método* es una función asociada a un objeto, o, simplemente, un método es una propiedad de un objeto que es una función. Los métodos se definen normalmente como una función, con excepción de que tienen que ser asignados como la propiedad de un objeto.

```
objectName.methodname = functionName;

var myObj = {
  myMethod: function(params) {
    // ...hacer algo
  }

  // O ESTO TAMBIÉN FUNCIONA

  myOtherMethod(params) {
    // ...hacer algo más
  }
};
```

Usar *this* para referencias a objetos

La palabra clave *this*, se puede usar dentro de un método para referirte al objeto actual.

```
const Intern = {
  name: "Ben",
  age: 21,
  job: "Software Engineer Intern",
};

function sayHi() {
  console.log("Hola, mi nombre es ", this.name);
}

// agrega la función sayHi a ambos objetos
Manager.sayHi = sayHi;
Intern.sayHi = sayHi;

Manager.sayHi(); // Hola, mi nombre es John'
Intern.sayHi(); // Hola, mi nombre es Ben'
```

Definición de captadores (getters) y establecedores (setters)

Un captador (getter) es un método que obtiene el valor de una propiedad específica. Un establecedor (setter) es un método que establece el valor de una propiedad específica.

```
var o = {
  a: 7,
  get b() {
    return this.a + 1;
  },
  set c(x) {
    this.a = x / 2;
  },
};

console.log(o.a); // 7
console.log(o.b); // 8 <-- En este punto se inicia el método get b().
o.c = 50; // <-- En este punto se inicia el método set c(x)
console.log(o.a); // 25
```

Eliminar propiedades

Puedes eliminar una propiedad no heredada mediante el operador delete.

```
//Crea un nuevo objeto, myobj, con dos propiedades, a y b.
var myobj = new Object();
myobj.a = 5;
myobj.b = 12;

// Elimina la propiedad a, dejando a myobj solo con la propiedad b.
delete myobj.a;
console.log("a" in myobj); // muestra: "false"
```

Comparar objetos

Como sabemos los objetos son de tipo referencia en JavaScript. Dos distintos objetos nunca son iguales, incluso aunque tengan las mismas propiedades. Solo comparar la misma referencia de objeto consigo misma arroja verdadero.

```
// Dos variables, dos distintos objetos con las mismas propiedades
var fruit = { name: "apple" };
var fruitbear = { name: "apple" };

fruit == fruitbear; // devuelve false
fruit === fruitbear; // devuelve false
```

Comparar objetos

```
// Dos variables, un solo objeto
var fruit = { name: "apple" };
var fruitbear = fruit; // Asigna la referencia del objeto fruit a
fruitbear

// Aquí fruit y fruitbear apuntan al mismo objeto
fruit == fruitbear; // devuelve true
fruit === fruitbear; // devuelve true

fruit.name = "grape";
console.log(fruitbear); // Produce: { name: "grape" }, en lugar de {
name: "apple" }
```

Clases en JS

Las clases de JavaScript, introducidas en ECMAScript 2015, son una mejora sintáctica sobre la herencia basada en prototipos de JavaScript. La sintaxis de las clases **no** introduce un nuevo modelo de herencia orientada a objetos en JavaScript. Las clases de JavaScript proveen una sintaxis mucho más clara y simple para crear objetos y lidiar con la herencia.

Definiendo clases

Las clases son "funciones especiales", como las **expresiones de funciones** y **declaraciones de funciones**, la sintaxis de una clase tiene dos componentes: **expresiones de clases** y **declaraciones de clases**.

Declaración de clases

Una manera de definir una clase es mediante una **declaración de clase**. Para declarar una clase, se utiliza la palabra reservada *class* y un nombre para la clase “Rectangulo”.

```
class Rectangulo {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
}
```

Alojamiento

Una importante diferencia entre las **declaraciones de funciones** y las **declaraciones de clases** es que las **declaraciones de funciones** son alojadas y las **declaraciones de clases** no lo son.

Expresiones de clases

Una **expresión de clase** es otra manera de definir una clase. Las expresiones de clase pueden ser nombradas o anónimas. El nombre dado a la **expresión de clase** nombrada es local dentro del cuerpo de la misma.

```
// Anonima
let Rectangulo = class {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
};

console.log(Rectangulo.name);
// output: "Rectangulo"
```

```
// Nombrada
let Rectangulo = class Rectangulo2 {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
};
console.log(Rectangulo.name);
// output: "Rectangulo2"
```

Constructor

El método *constructor* es un método especial para crear e inicializar un objeto creado con una clase. Solo puede haber un método especial con el nombre "constructor" en una clase. Si esta contiene mas de una ocurrencia del método constructor, se arrojará un *Error SyntaxError*.

Un **constructor** puede usar la palabra reservada *super* para llamar al constructor de una *superclase.*

Métodos prototipo

```
class Rectangulo {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
  // Getter
  get area() {
    return this.calcArea();
  }
  // Método
  calcArea() {
    return this.alto * this.ancho;
  }
}

const cuadrado = new Rectangulo(10, 10);

console.log(cuadrado.area); // 100
```

Métodos estáticos

La palabra clave *static* define un método estático para una clase. Los métodos estáticos son **llamados sin instanciar su clase** y no pueden ser llamados mediante una instancia de clase. Los métodos estáticos son a menudo usados para crear funciones de utilidad para una aplicación.

```
class Punto {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  static distancia(a, b) {
    const dx = a.x - b.x;
    const dy = a.y - b.y;

    return Math.sqrt(dx * dx + dy * dy);
  }
}

const p1 = new Punto(5, 5);
const p2 = new Punto(10, 10);

console.log(Punto.distancia(p1, p2)); // 7.0710
```

Subclases con extends

La palabra clave *extends* es usada en *declaraciones de clase* o *expresiones de clase* para crear una clase hija.

```
class Animal {
  constructor(nombre) {
    this.nombre = nombre;
  }

  hablar() {
    console.log(this.nombre + " hace un ruido.");
  }
}

class Perro extends Animal {
  hablar() {
    console.log(this.nombre + " ladra.");
  }
}
```

Subclases con extends

También se pueden extender las clases tradicionales basadas en funciones:

```
function Animal(nombre) {
  this.nombre = nombre;
}
Animal.prototype.hablar = function () {
  console.log(this.nombre + "hace un ruido.");
};

class Perro extends Animal {
  hablar() {
    super.hablar();
    console.log(this.nombre + " ladra.");
  }
}

var p = new Perro("Mitzie");
p.hablar();
```

¡Cuidado!

Fijarse que las clases no pueden extender **objectos regulares (literales)** Si se quiere heredar de un objecto regular, se debe usar *Object.setPrototypeOf() :*

```
var Animal = {
  hablar() {
    console.log(this.nombre + ' hace ruido.');
  },
  comer() {
    console.log(this.nombre + ' se alimenta.');
  }
};
```

```
class Perro {
  constructor(nombre) {
    this.nombre = nombre;
  }
  hablar() {
    console.log(this.nombre + ' ladra.');
  }
}
```

```
// Solo adjunta los métodos aún no definidos
Object.setPrototypeOf(Perro.prototype, Animal);

var d = new Perro('Mitzie');
d.hablar(); // Mitzie ladra.
d.comer(); // Mitzie se alimenta.
```

Ej 2.5. Buscar info

Object.setPrototypeOf() :

Llamadas a Super

La palabra clave *super* es usada para llamar funciones del objeto padre.

```
class Gato {
  constructor(nombre) {
    this.nombre = nombre;
  }

  hablar() {
    console.log(this.nombre + ' hace ruido.');
  }
}

class Leon extends Gato {
  hablar() {
    super.hablar();
    console.log(this.nombre + ' maulla.');
  }
}
```

Ej 2.6 Hamburguesería:

Hamburguesería

Se va a crear un interfaz de ventana (prompt o consola) de pedido de comida de un restaurante.
Un clase contendrá la información de los precios de los tipos de hamburguesa y 2 métodos: uno que diga el precio por selección y otro que diga los ingredientes, por selección.

Al final devolverá el precio de un pedido que puede incluir hamburgues, bebida de distintos tamaños y patatas.

Clases: Bebidas/ Hamburguesas/ Patatas/ Nuggets/ Postre (helado, manzana)
Contempla tamaños, precios y cantidades

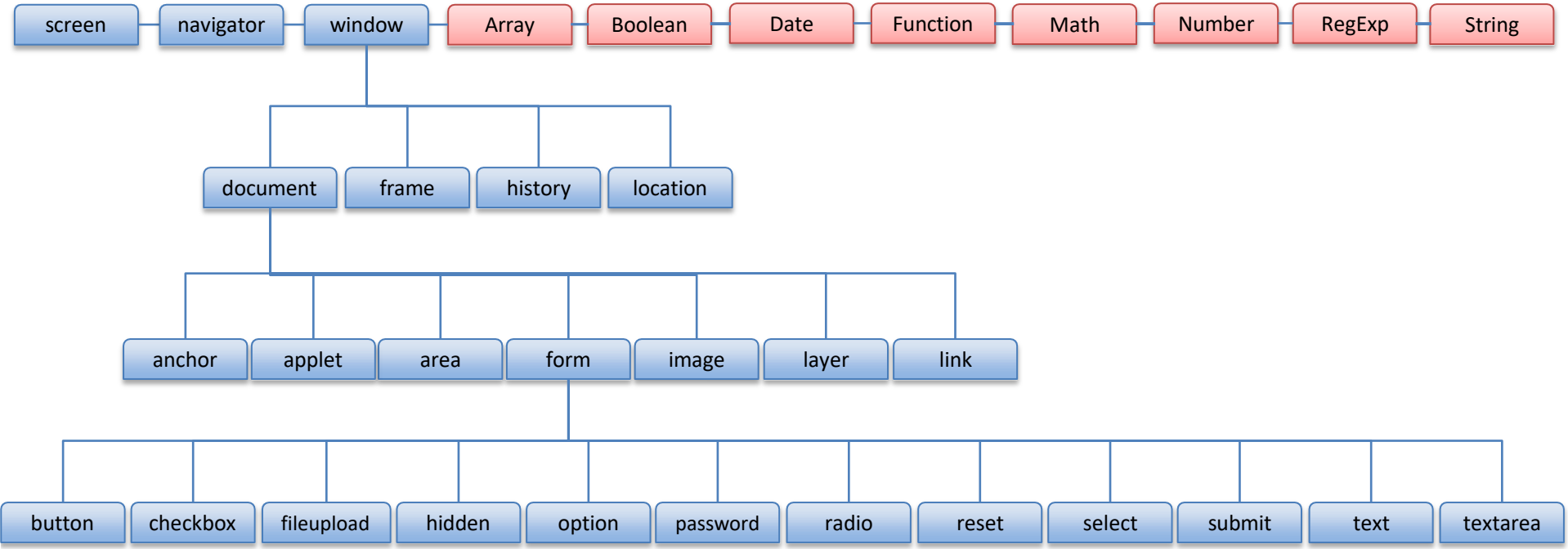
Ej 2.6 Hamburguesería:

RÚBRICA

ID	Criterio	1	2	3	4
1	Aspecto funcional				La interfaz con el usuario es compresible/ guía bien y se entiende bien hasta el final/ tienes control en todo momento
2	Aspecto estético				Utilizas los recursos de los que dispone la consola/html plano/ para comunicar como interfaz.
3	Documentación				Documentación entregada en tiempo y forma. Documento zip con el proyecyo y pdf con diagrama de clases UML, digrama de flujobásico.
4	Estructura de Clases				Implementa un modelo basado en clases y explota su utilización
5	Formato código				Bien estructurado, comentado y tabulado. Es comprensible.
6	Bases de JS				Utiliza bien las variables y las estructuras de control de flujo.

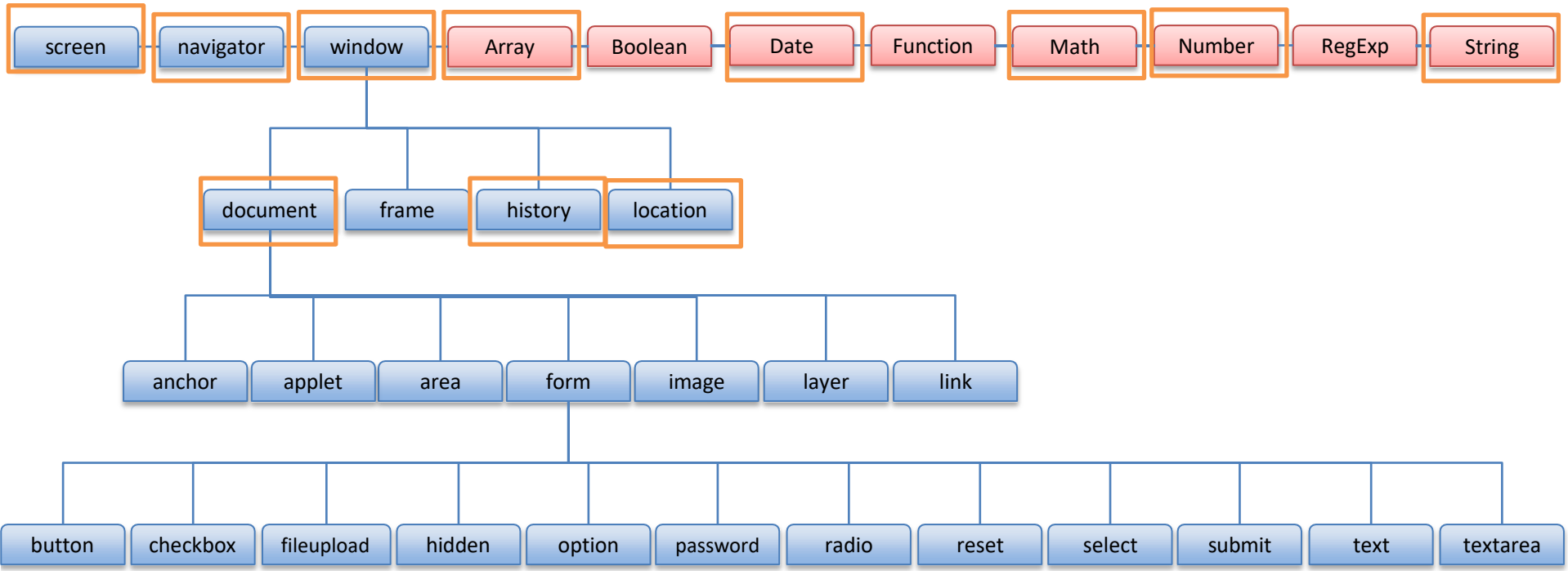
Objetos nativos de JavaScript

- Los objetos de JavaScript se ordenan de modo jerárquico.



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()

2.5 Ejercicio

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

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

2.6 Ejercicio

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

2.7 Ejercicio

Muestra en la pantalla un número entero aleatorio entre 10 y 100 (último no incluido).

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`

2.8 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

2.9 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 UD3): 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

- 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

2.9 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

2.10 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()	scrollBy()
back()	home()	scrollTo()
blur()	moveTo()	setinterval()
close()	open()	setTimeOut()
confirm()	print()	stop()
find()	prompt()	
focus()	resizeTo()	

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

2.10 Ejercicio

1. 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.

2. Después, muévela a 400 a la derecha y 100 abajo respecto de la esquina superior izquierda.

3. Luego, cada 100 milisegundos, muévela 19 veces a 5 píxeles a la derecha y 5 abajo respecto de la posición anterior.

4. 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("http://www.misitioWeb.com/ads",
       "Publicidad", "height=100, width=100");
```

Gestión de las ventanas

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

```
myWindow1.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=”
  myWindow1=window.open('', 'Nueva Ventana', 'width=300, height=200');
  myWindow1.document.write('<html>');
  myWindow1.document.write('<head>');
  myWindow1.document.write('<title>Ventana Test</title>');
  myWindow1.document.write('</head>');
  myWindow1.document.write('<body>');
  myWindow1.document.writeln('Se usan las propiedades: ');
  myWindow1.document.write('<li>height=200</li> <li>width=300</li>');
  myWindow1.document.write('<input type=button value=Cerrar
        onClick=window.close()>');
  myWindow1.document.write('</body>');
  myWindow1.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>
 <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>
 </center>
</body></html>
```

Aplicaciones prácticas de los marcos (en desuso)

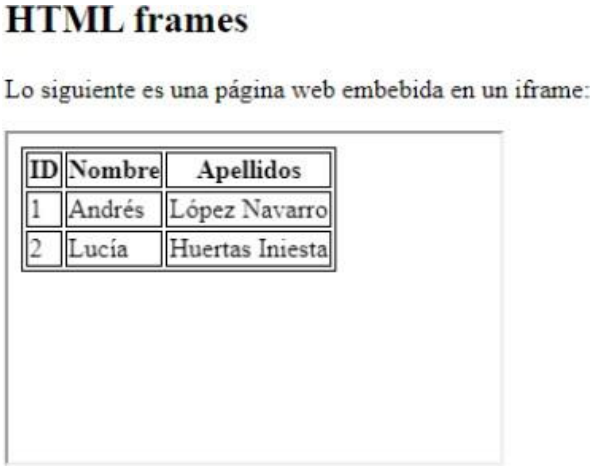
- Es posible dividir la ventana de una aplicación Web en dos o más partes independientes.
- Con JavaScript se puede interactuar entre estos sectores independientes.
- Dichos sectores se denominan marcos.
- HTML5 considera obsoleto el uso de marcos y se desaconseja su uso.

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 definen utilizando HTML mediante estas etiquetas:
  - `<frameset>`
  - `<frame>`
  - `<iframe>` en la actualidad:

https://developer.mozilla.org/es/docs/Web/HTML/Element/iframe
https://developer.mozilla.org/es/docs/Learn/HTML/Multimedia_and_embedding/Other_embedding_technologies

Aplicaciones prácticas de los marcos

- Atributos de la etiqueta `<frame>`:

Atributos
frameborder
marginheight
marginwidth
name
noresize
scrolling
src

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></head>
    <frameset cols="50%,50%">
      <frame src="Marco1.html" name="Marco1" noresize>
      <frame src="Marco2.html" name="Marco2" noresize>
    </frameset>
  <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="marcos">
        <option value="0">Izquierda
        <option value="1">Derecha
      </select>
    </form>
  </body>
</html>
```

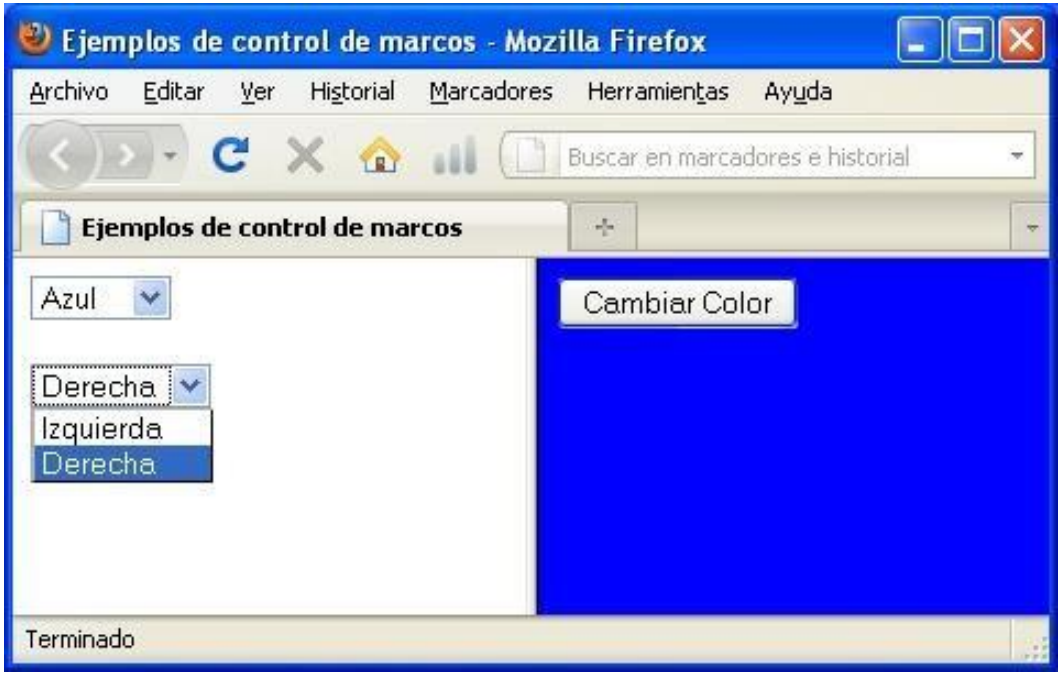
Aplicaciones prácticas de los marcos

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

```
<html><body>
    <form>
      <input type="Button" value="Cambiar Color" onclick="
        campoColor = parent.Marco1.document.form1.color
        if (campoColor.selectedIndex==0){colorin = 'green':}
        else{colorin = 'blue';}
        campoFrame = parent.Marco1.document.form1.marcos
        if (campoFrame.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:



2.11 Hoja de ejercicios

- Realiza todos los ejercicios del bloque:

UD2_2_11_HojaDeEjercicios(entrega)_JS_DWEC_IN2W_2324_0_0