PRACT : OBJETOS PREDEFINIDOS – MANEJO ESTRUCTURA DOM

Introducción:

1. CLASE MATH:

El objeto Math nos permite realizar una serie de operaciones matemáticas más complicadas que las simples operaciones normales hechas con operadores. No es una clase de objetos, y utiliza los objetos de la clase Number en sus propiedades y métodos.

2. CLASE NUMBER:

Al asignar a una variable un número como valor, ésta pasa a ser automáticamente un objeto de la clase Number().

Pertenecen a esta clase todas las variables que contienen números y además pueden contener algunos valores especiales relacionados con los números (Infinity, -Infinity, NaN);

- **Métodos de la clase Number()**

  - **Método Number():** El método intentará convertir el elemento que se le pase como argumento en un número. Si no lo consigue devuelve el valor NaN. Valor especial que significa "No es un número" ("Not a Number").

  - **Método isNaN() :** Este método comprueba si el elemento que pasamos como argumento es un número. Devuelve siempre un valor booleano: true cuando NO es un número, y false cuando SÍ es un número.

  - **Método parseInt():** Convierte un número que está escrito en una base que no es base 10 a número en base10. El primer argumento recoge el número (en base 2, 8, 16 o otra base); el segundo indica la base en la que está el número.

    num = parseInt("7b",16)

  - **Método toFixed() :** Este método redondea el valor del número al que se le aplica en el número de decimales que se especifica en el argumento: si el número es negativo el redondeo se hace a nivel de decenas (-1), centenas(-2), etc. Si no se especifica el argumento se redondea al número entero más próximo.

3. CLASE STRING:

La clase String incluye las cadenas de texto. Cualquier variable que tenga como valor una cadena de texto es un objeto de la clase String.

Crear un objeto de la clase String es tan simple como asignarle un valor de cadena de texto a una variable; aunque también podemos utilizar el método general par crear objetos:

var texto = new String("mi texto");

La única propiedad de String es la propiedad .length que devuelve el número de caracteres que contiene la cadena de texto.**caracteres = texto.length**

Métodos habituales de String

Nombre	Ejemplo y explicación
toString()	`texto = variable.toString();` Convierte cualquier elemento en una cadena de texto, por lo que en realidad no se aplica a cadenas de texto, sino a cualquier variable.
concat()	`nuevoTexto = texto1.concat(texto2);` Junta dos cadenas. A la cadena `texto1` se le suma la cadena `texto2` pasada como argumento. El resultado es una nueva cadena; igual que cuando las sumamos.
toUpperCase()	`textoMay = texto.toUpperCase();` Crea la misma cadena de texto, pero en la que todos los caracteres se han escrito en letras mayúsculas
toLowerCase()	`textoMin = texto.toLowerCase();` Crea la misma cadena de texto, pero en la que todos los caracteres se han escrito en letras minúsculas.
indexOf()	`posicion = texto.indexOf(caracter);` Pasamos como argumento un carácter. El método lo busca en la cadena, y devuelve un número que indica la posición del carácter encontrado. Si hay más de uno da la posición del primero. Se empieza a contar desde 0, (primer carácter = 0). Si no se encuentra devuelve -1. Podemos pasar como argumento una cadena con más de un carácter . Se busca la cadena y si está está dentro del texto nos dirá en qué posición empieza
lastIndexOf()	`posicion = texto.lastIndexOf(caracter);` Igual que el anterior pero empieza a buscar desde el final de la cadena. El resultado es la posición empezando a contar por el principio a partir del 0.
substring()	`porcionTexto = texto.substring(num1,num2);` Crea una subcadena que va desde el carácter que está en el `num1` al caracter inmediatamente anterior al que está en el `num2`. El segundo número es opcional, y si no se indica la cadena devuelta va hasta el final. La cuenta de caracteres empieza en el 0.
substr()	`porcionTexto = texto.substr(num1,num2);` Crea una subcadena que empieza en el carácter que está en el `num1` y de una longitud indicada por `num2`. El segundo número es opcional, y si no se pone, la cadena va hasta el final. La cuenta de caracteres empieza en el 0.
split()	`nuevoArray = texto.split(separador);` Convierte la cadena de texto en un array. Como argumento se escribe el caracter que separa los elementos del array, por ejemplo un espacio en blanco (" ") los separa en palabras. Si no se pasa ningún argumento el caracter por defecto es la coma.
replace()	`cambiado = texto.replace(buscarTexto,nuevoTexto);` Remplaza una porción de texto por otra. para ello el primer argumento `buscarTexto` es el trozo de texto que hay que reemplazar. El segundo argumento `textoNuevo` es el texto por el cual va a ser reemplazado el primero.

Nombre	Ejemplo y explicación
charAt()	`letra = texto.charAt(num);` Devuelve el carácter que se encuentra en la posición indicada por el número que se le pasa como argumento. Las posiciones se empiezan a contar desde el 0.
charCodeAt()	`codigo = texto.charCodeAt(num);` Devuelve el código Unicode del carácter que se encuentra en la posición indicada por el número que se le pasa como argumento. Las posiciones se empiezan a contar desde el 0.
link()	`enlace = texto.link("http://google.com")` Convierte el texto en un enlace. En el argumento se le pasa la URL del enlace.

4. CLASE DATE:

Para trabajar con fechas necesitamos instanciar un objeto de la clase Date.

Podemos hacerlo de dos maneras diferentes:

- Creando un objeto Date con la fecha y hora actual
  var fecha = new Date();

- Pasándole como parámetros los datos.

```
//Podemos crear el objeto Date con todos los parámetros
var fecha = new Date(año,mes,dia,hora,minutos,segundos);

//O con algunos parámetros
var fecha = new Date(año,mes,dia,hora);
```

En JavaScript los meses van desde el cero (Enero) a once (Diciembre), los días de la semana del cero (Domingo) a seis (Sábado). Es importante tenerlo en cuenta cuando trabajamos con fechas, tanto al crearlas como al utilizar sus métodos.

MÉTODO	QUÉ HACE
getDate()	Devuelve el día del mes. Número entre 1 y 31
getDay()	Devuelve el día de la semana. Entre 0 (domingo) y 6 (sábado)
getFullYear()	Devuelve el año con 4 dígitos
getMilliseconds()	Devuelve los milisegundos entre 0 y 9999
getMinutes()	Devuelve los minutos. Entre 0 y 59
getMonth()	Devuelve el mes. Entre 0 (enero) y 11 (diciembre)
getSeconds()	Devuelve los segundos. Entre 0 y 59
getTime()	Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje
parse()	Analiza una fecha y devuelve el número de milisegundos pasados desde el 1 de enero de 1970 hasta la fecha analizada
setDate()	Actualiza el día del mes
setFullYear()	Cambia el año de la fecha al número que recibe por parámetro
setHours()	Actualiza la hora
setMilliseconds()	Establece el valor de los milisegundos

MÉTODO	QUÉ HACE
setMinutes()	Cambia los minutos
setMonth()	Cambia el mes (atención al mes que empieza por 0)
setSeconds()	Cambia los segundos
setTime()	Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970
toTimeString()	Convierte la parte de tiempo de un objeto Date en una cadena

Para hacer operaciones con fechas, podemos utilizar las fechas en tiempo Unix. Hay que tener en cuenta que el tiempo Unix está en milisegundos, por lo que un día es igual a dia = 24*60*60*1000

También podemos sumar o restar días a una fecha añadiéndolas al método correspondiente. El siguiente ejemplo indica cómo buscar qué fecha será dentro de 120 días:

```
fecha = new Date(); //fecha actual.
diaActual = fecha.getDate(); //dia actual del mes.
diaBuscado = diaActual + 120; //le sumamos 120 días.
fecha.setDate(diaBuscado); //pasamos el resultado a formato de fecha.
```

5. MANEJAR LA ESTRUCTURA DOM

Cómo seleccionar otros nodos según algunas de las relaciones de parentesco establecidas alrededor de tal elemento.

- **parentNode :** Por medio de **parentNode** podemos seleccionar el elemento padre de otro elemento.
- **firstChild:** Con **firstChild** lo que seleccionamos es el primer hijo de un elemento. Por desgracia, hay discrepancias entre los diversos navegadores sobre qué debe considerarse o no hijo de un nodo, por lo que esta propiedad en ocasiones complica demasiado un script.
- **lastChild:** La propiedad **lastChild** funciona exactamente como **firstChild**, pero se refiere el último de los hijos de un elemento. Se aplican, por tanto, las mismas indicaciones anteriores.
- **nextSibling:** Gracias a nextSibling, lo que podemos seleccionar es el siguiente hermano de un elemento. Se aplican las mismas limitaciones que para las dos propiedades anteriores.
- **previousSibling : previousSibling** funciona igual que nextSibling, pero selecciona el hermano anterior de un elemento

Creación de elementos

- **appendChild:** Por medio de appendChild podemos incluir en un nodo un nuevo hijo, de esta manera:

  elemento_padre.appendChild(nuevo_nodo);

  El nuevo nodo se incluye inmediatamente después de los hijos ya existentes —si hay alguno— y el nodo padre cuenta con una nueva rama.

Por ejemplo, el siguiente código: Crea un elemento ul y un elemento li, y convierte el segundo en hijo del primero.

```
•
•       var lista = document.createElement('ul');
•       var item = document.createElement('li');
•       lista.appendChild(item);
```

- **insertBefore:** Nos permite elegir un nodo del documento e incluir otro antes que él.

  elemento_padre.insertBefore(nuevo_nodo,nodo_de_referencia);

Por ejemplo, el siguiente código

```
<div id="padre">
        <p>Un párrafo.</p>
        <p>Otro párrafo.</p>
</div>
```

y quisiéramos añadir un nuevo párrafo antes del segundo, lo haríamos así:

```
// Creamos el nuevo párrafo
var nuevo_parrafo
=document.createElement('p').appendChild(document.createTextNode('Nuevo párrafo.'));
// Recojemos en una variable el segundo párrafo
var segundo_p = document.getElementById('padre').getElementsByTagName('p')[1];
// Y ahora lo insertamos
document.getElementById('padre').insertBefore(nuevo_parrafo,segundo_p);
```

- **insertAfter:** No hay un metodo que inserte un nodo detrás de otro
- **replaceChild:** Para reemplazar un nodo por otro contamos con replaceChild, cuya sintaxis es:

  elemento_padre.replaceChild(nuevo_nodo,nodo_a_reemplazar);
- **removeChild:** Dado que podemos incluir nuevos hijos en un nodo, tiene sentido que podamos eliminarlos. Para ello existe el método removeChild.

  elemento_padre.removeChild(nodo_a_eliminar);
- **cloneNode:** Por último, podemos crear un clon de un nodo por medio de cloneNode:

  elemento_a_clonar.cloneNode(booleano);

  El booleano que se pasa como parámetro define si se quiere clonar el elemento —con el valor false—, o bien si se quiere clonar con su contenido —con el valor true—, es decir, el elemento y todos sus descendientes.

Secuencia y desarrollo:

1. **Ejercicio 1: Mostrar la fecha actual de las siguientes formas utilizando la clase Date**
   - **01/10/2019**
   - **01 de octubre de 2019**
   - **Martes,01 de octubre de 2019**

2. **Ejercicio 2: Solicitar al usuario la fecha de su nacimiento y calcular su edad.**
3. **Ejercicio3: Solicitar al usuario la fecha de un evento determinado y calcular los días que quedan hasta el día del evento.**
4. **Ejercicio 4: Prueba el siguiente ejemplo, el cual crea una aplicación web con un cuadro de texto y un botón que, al hacer clic en el botón, añada un nuevo párrafo a la página con el texto que el usuario ha introducido en el campo de texto.**

```
<html lang="en">
<head>
	<meta charset="UTF-8"/>
	<title>Iniciación DOM 1</title>
</head>
<body>
	<form name="formulario">
		Texto: <input type="text" name="texto"/>
		<input type="button" name="crear" value="Crear">
	</form>
</body>
<script type="text/javascript">
	function crear() {
		let texto = document.formulario.texto.value;
		console.log(texto);

		//Creamos la etiqueta
		let parrafo = document.createElement("p");
		let nodoTexto = document.createTextNode(texto);

		let span = document.createElement("span");
		let nodoTexto2 = document.createTextNode("Prueba");
		span.appendChild(nodoTexto2);

		parrafo.appendChild(nodoTexto);
		document.body.appendChild(parrafo);
		let devuelto = document.body.replaceChild(span, parrafo);
		console.log(devuelto);
	}

	window.onload = function () {
		document.formulario.crear.addEventListener("click", crear);
	}
</script>
</html>
```

5. **Ejercicio5: Crea una aplicación web con dos botones: crear y destruir. Cuando el usuario pulse "crear", se añade al body una nueva lista <ul>, cuando pulsa "destruir", se elimina la primera lista que se creó.**