SINTAXIS DEL LENGUAJE PARA EXAMEN PRACTICO

1. CONVERSIÓN DE DATOS ENTRE TIPOS DE DATOS

Existen algunas funciones propias del lenguaje que nos van a permitir convertir cadenas a enteros o reales, evaluar una expresión, etc..

- **parseFloat(cadena)** devuelve un número en coma flotante a partir de la cadena especificada. La función finalizará la conversión cuando encuentre un carácter que no sea un dígito (0..9), un punto decimal (.) o una "e". Ejemplos:
  - parseFloat("27.11.2000"); devuelve 27.11
  - parseFloat("20H30M07S"); devuelve 20
  - parseFloat("345.6e5"); devuelve 34560000
  - parseFloat("VUELO506"); devuelve NaN (Not a Number)
- **parseInt(cadena)** devuelve un número entero a partir de la cadena especificada. La función finalizará la conversión cuando encuentre un carácter que no sea un dígito. Ejemplos:
  - parseInt("27.11.2000"); devuelve 27
  - parseInt("20H30M07S"); devuelve 20
  - parseInt("345.6e5"); devuelve 345
  - parseInt("VUELO506"); devuelve NaN (Not a Number)

  En principio se considerará como base la decimal (excepto cuando el número comienza con 0x o con 0); pero este formato se puede añadir un segundo argumento que indicaría la base en la que creemos que está el número que queremos convertir.

  El formato ampliado será **parseInt(cadena, base)**. Ejemplos:
    - parseInt("27", 8); devuelve 23 parseInt("11111111", 2); devuelve 255
    - parseInt("A2", 16); devuelve 162
- **toString(argumento)** convierte a cadena el dato especificado en argumento. Normalmente se utiliza para convertir números a cadena pero también funciona con otros tipos como Booleano, etcétera. Su utilización es restringida ya que JavaScript aplica una conversión automática de tipos a cadena cuando encuentra expresiones de concatenación en las que hay diversos tipos de datos.
- **typeof(argumento)** devuelve una cadena que indica el tipo del argumento (number,string, boolean, undefined).

2. SENTENCIAS DE CONTROL

Al hacer un programa necesitaremos establecer condiciones o decisiones, donde buscamos que se realice una acción A si se cumple una condición o una acción B si no se cumple. Este es el primer tipo de estructuras de control que encontraremos.

Tenemos varias estructuras de control condicionales:

- **Estructura de control if**

  If ***Condición simple: Si ocurre algo, haz lo siguiente...***

If/else ***Condición con alternativa: Si ocurre algo, haz esto, sino, haz lo esto otro...***

?: ***Operador ternario: Equivalente a If/else, forma abreviada.***

Switch ***Estructura para casos específicos: Similar a varios If/else anidados.***

Bucles: Una de las principales ventajas de la programación es la posibilidad de crear bucles y repeticiones para tareas específicas, y que no tengamos que realizar el mismo código varias veces de forma manual**.**

Tipo de bucle	Descripción
while	Bucles simples.
for	Bucles clásicos por excelencia.
do..while	Bucles simples que se realizan siempre como mínimo una vez.
for..in	Bucles sobre posiciones de un array. Los veremos más adelante.
for..of	Bucles sobre elementos de un array. Los veremos más adelante.

3. Arrays: Métodos

- **Método length:** Si queremos conocer el número de elementos del array Podemos modificar el tamaño del array modificando esta propiedad. EJ: ***colores.length = 2;***
- **Método toString():** Convierte todo el array en una cadena, separando cada elemento por comas, pero unidos en un string.
- **Métodos Stack:** Un array puede actuar como una pila, que te permite en un grupo de datos apilar y desapilar estos. A esta pila se conoce como el tipo LIFO(Last in, First out) último en entrar - primero en salir, lo que significa que el elemento más recientemente añadido es el primero en ser eliminado.
  - **Método Push en Array. T**ransforma un array añadiendo los elementos proporcionados y devolviendo la nueva longitud del array**.**
  - **Método Pop en Array.** Este método permite eliminar el último elemento del array, disminuyendo la longitud de dicho array**.**

Métodos adicionales en los Arrays:

- **shift:** Permite obtener el primer elemento de un array
- **unshift:** El método con el cual puedes agregar elementos al inicio del array, puede contener más de un argumento, esto quiere decir que puedes agregar en la misma sentencia más de un elemento. veamos su forma de uso.
- **reverse:** método que establece que el array invierte sus elementos

- **sort:** este método es muy útil cuando tengamos un array con elementos string, pues estos serán ordenados alfabéticamente**.** Lo ordenará alfabéticamente. De usarse este método en elementos de tipo numéricos nos devolverá datos incorrectos.
- **slice:** método que puede contener uno o dos argumentos, que indiquen el inicio y parada de posiciones, pues devuelve los elementos contenidos en el array, de acuerdo a los argumentos indicados.
- **splice:** cambia el contenido de un array removiendo elementos existentes y/o agregando nuevos elementos, hay tres maneras distintas de utilizar este método.
  - **supresión:** Cualquier número de elementos puede ser eliminado de la matriz especificando sólo dos argumentos: la posición del primer elemento que desea eliminar y el número de elementos a eliminar. Por ejemplo,

    colores = ["amarillo", "anaranjado", "azul", "rojo", "verde"]

    colores.splice(0, 2);

    ["azul", "rojo", "verde"]

    Entonces, suprime los dos primeros elementos.
  - **inserción:** Se puede insertar elementos en el array en una posición específica, proporcionando tres argumentos: la posición donde deseas **agregar** elementos,el número de elementos que desea **eliminar** y el elemento a **insertar**. Opcionalmente, se puede especificar una cuarta, quinta, o cualquier número de otros parámetros a insertar. Por ejemplo,

    var colores = ["amarillo", "anaranjado", "azul", "rojo", "verde"];

    colores.splice(1,0,'pepe');

    ["amarillo", "pepe", "anaranjado", "azul", "rojo", "verde"]
  - **reemplazar:** Se comportaría como insertar en una posición específica al eliminar simultáneamente, para esto se tiene que especificar tres argumentos: la posición donde insertar, el número de elementos a borrar, y cualquier número de elementos a insertar. El número de elementos a insertar no tiene que coincidir necesariamente con el número de elementos que desea eliminar. Por ejemplo:

    var colores = ["amarillo", "anaranjado", "azul", "rojo", "verde"];

    colores.splice(2, 1, "morado", " violeta");

    ["amarillo", "anaranjado", "morado", " violeta", "rojo", "verde"]

    Elimina un elemento en la posición 2 y luego inserta las cadenas"rojo" y "verde" en la matriz en la posición 2.

4. El objeto Math:

El Objeto Math javascript nos permite trabajar con funciones matemáticas:

- Math.log(x) = ln(x)
- Math.exp(x) = e^x
- Math.sqrt(x) = raiz cuadrada de "x"
- Math.pow(a, b) = ab
- Math.floor(): número entero más cercano y menor
- Math.ceil(): número entero más cercano y mayor
- Math.round(): redondea al entero más próximo.
- Math.random(): número aleatorio entre 0 y 1
- Math.round(y-x)*Math.random()+x: número aleatorio entre "x" e "y".
- Math.sin(x)= sin(x) x en radianes
- Math.cos(x)= cos(x) x en radianes
- Math.tan(x)= tg(x) x en radianes
- Math.atan(x)= arctg(x) resultado en radianes
- Math.abs(x): valor absoluto de "x"
- Math.max(a,b) : máximo valor de los dos
- Math.min(a,b) : mínimo valor de los dos

5. Objeto Window

Se trata del objeto más alto en la jerarquía del navegador (navigator es un objeto independiente de todos en la jerarquía), pues todos los componentes de una página web están situados dentro de una ventana. El objeto window hace referencia a la ventana actual. Veamos a continuación sus propiedades y sus métodos.

Propiedades:

- **closed**: Es un booleano que nos dice si la ventana está cerrada (closed = true) o no (closed = false).
- **defaultStatus**: Cadena que contiene el texto por defecto que aparece en la barra de estado (status bar) del navegador.

- **Frames**: Es un array: cada elemento de este array (frames[0], frames[1], ...) es uno de los frames que contiene la ventana. Su orden se asigna según se definen en el documento HTML.
- **History**: Se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en su historial).
- **Length**: Variable que nos indica cuántos frames tiene la ventana actual.
- **Location**: Cadena con la URL de la barra de dirección.
- **Name**: Contiene el nombre de la ventana, o del frame actual.
- **Opener**: Es una referencia al objeto window que lo abrió, si la ventana fue abierta usando el método open() que veremos cuando estudiemos los métodos.
- **Parent**: Referencia al objeto window que contiene el frameset.
- **Self**: Es un nombre alternativo del window actual.
- **Status**: String con el que tiene la barra de estado.
- **Top**:Nombre alternativo de la ventana del nivel superior.
- **Window**:Igual que self: nombre alternativo del objeto window actual.

Métodos:

- **alert(mensaje):** Muestra el mensaje 'mensaje' en un cuadro de diálogo
- **blur():** Elimina el foco del objeto window actual.
- **clearInterval(id):** Elimina el intervalo referenciado por 'id' (ver el método setInterval(), también del objeto window).
- **clearTimeout(nombre):** Cancela el intervalo referenciado por 'nombre' (ver el método setTimeout(), también del objeto window).
- **close()**: Cierra el objeto window actual.
- **confirm(mensaje):** Muestra un cuadro de diálogo con el mensaje 'mensaje' y dos botones, uno de aceptar y otro de cancelar. Devuelve true si se pulsa aceptar y devuelve false si se pulsa cancelar.
- **focus():** Captura el foco del ratón sobre el objeto window actual.
- **moveBy(x,y):** Mueve el objeto window actual el número de pixels especificados por (x,y).
- **moveTo(x,y):** Mueve el objeto window actual a las coordenadas (x,y).
- **open(URL,nombre,caracteristicas):** Abre la URL que le pasemos como primer pa-

rámetro en una ventana de nombre 'nombre'. Si esta ventana no existe, abrirá una ventana nueva en la que mostrará el contenido con las características especificadas. Las características que podemos elegir para la ventana que queramos abrir son las siguientes:

- **toolbar** = [yes|no|1|0]. Nos dice si la ventana tendrá barra de herramientas (yes,1) o no la tendrá (no,0).
- **location** = [yes|no|1|0]. Nos dice si la ventana tendrá campo de localización o no.
- **directories** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá botones de dirección o no.
- **status** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barra de estado o no.
- **menubar** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barra de menús o no.
- **scrollbars** = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barras de desplazamiento o no.
- **resizable** = [yes|no|1|0]. Nos dice si la nueva ventana podrá ser cambiada de tamaño (con el ratón) o no.
- **width** = px. Nos dice el ancho de la ventana en pixels.
- **height** = px. Nos dice el alto de la ventana en pixels.
- **outerWidth** = px. Nos dice el ancho *total* de la ventana en pixels. A partir de NS 4.
- **outerHeight** = px. Nos dice el alto *total* de la ventana el pixels. A partir de NS 4
- **left** = px. Nos dice la distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana.
- **top** = px. Nos dice la distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana.

    - **prompt(mensaje,respuesta_por_defecto):** Muestra un cuadro de diálogo que contiene una caja de texto en la cual podremos escribir una respuesta a lo que nos pregunte en 'mensaje'. El parámetro 'respuesta_por_defecto' es opcional, y mostrará la respuesta por defecto indicada al abrirse el cuadro de diálogo. El método retorna una cadena de caracteres con la respuesta introducida.
    - **scroll(x,y):** Desplaza el objeto window actual a las coordenadas especificadas por (x,y).
    - **scrollBy(x,y):** Desplaza el objeto window actual el número de pixels especificado por (x,y).
    - **scrollTo(x,y):** Desplaza el objeto window actual a las coordenadas especificadas por (x,y).

- **setInterval(expresion,tiempo):** Evalua la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearInterval().
- **setTimeout(expresion,tiempo):** Evalua la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearTimeout().

NOTA: Estas dos funciones permiten ejecutar una función pasado un cierto intervalo de tiempo. La única diferencia entre ellas es que "setInterval" se ejecutará una y otra vez en intervalos de x segundos, en cambio, setTimeout se ejecutará una sola vez pasados x segundos.

Estas funciones se utilizan sobre todo para realizar animaciones o por ejemplo para redireccionar una página transcurridos x segundos.

Ambos se pueden cancelar mediante clearTimeout(temporizador) y clearInterval(intervalo).

Por ejemplo, si quisiéramos programar un reloj en la barra de estado la función más idónea sería setInterval. Teniendo una función llamada "reloj()" que imprime en la barra de estado la hora actual utilizaríamos la función setInterval de esta forma:

```
setInterval("reloj()",1000);
```

El primer parámetro es la función a ejecutar, el segundo es el intervalo en milisegundos, es decir, 1000 milisegundos = 1 segundo. De esta forma, cada segundo se ejecutará la función "reloj()" y por lo tanto se mostrará un reloj con la hora actual que irá cambiando segundo a segundo.

<u>Ejemplo1: Crear una ventana.</u>

```
var ventana=null;
ventana = window.open (“”,””,opciones);
ventana.document.write (“Esta es la ventana que se ha creado”);
```

6. String

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.

7. Date

- 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

MÉTODO	QUÉ HACE
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
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

8. Estructura DOM

- **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.

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

```
elemento_padre.insertBefore(nuevo_nodo,nodo_de_referencia);
```

- **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.

MODIFICAR ATRIBUTOS, CLASES Y ESTILOS EN EL DOM

Los elementos HTML a menudo tienen información adicional asignada en forma de atributos. Los atributos pueden incluir pares de nombre y valor, y algunos de los más comunes son class y style. Pueden ser modificados a través de los métodos de la estructura DOM

MODIFICAR ATRIBUTOS EN EL DOM

Los atributos son valores que contienen información adicional sobre elementos HTML. Normalmente vienen en pares de **nombre y valor**, y pueden ser esenciales según elemento.

En JavaScript, tenemos cuatro métodos para modificar atributos de elementos:

Método	Descripción	Ejemplo
hasAttribute()	Muestra un booleano true o false	element.hasAttribute('href');
getAttribute()	Muestra el valor de un atributo especificado o null	element.getAttribute('href');
setAttribute()	Agrega o actualiza el valor de un atributo especificado.	element.setAttribute('href', 'index.html');
removeAttribute()	Elimina un atributo de un elemento.	element.removeAttribute('href');

METODOS INNERHTML, INNERTEXT, ETC

InnerHTML es una propiedad que nos permite leer un dato o asignarlo al contenido de un div o bien, del mismo control. Nos facilita la asignación de valores a controles.

insertAdjacentHTML() method inserts HTML code into a specified position.

Legal positions:

Value	Description
afterbegin	Despues del comienzo del primer hijo
afterend	Despues del elemento
beforebegin	Antes del elemento
beforeend	Antes del final del elemento (ultimo hijo)

Syntax

element.insertAdjacentHTML(position, html)

Ejemplo1

```
parrafo.insertAdjacentHTML("beforeend", linea + "<br/>");
```