4.A. Estructuras definidas por el usuario en JavaScript

1.- Estructuras de datos.

1.1.- Objeto Array.

En esta sección trataremos con los **arrays escalares**, (objeto `Array`) es decir aquéllos cuyo índice es numérico. Un array escalar es una de estructuras de datos proporcionadas para almacenar y manipular colecciones de datos. A diferencia de otros lenguajes de programación, los arrays en JavaScript son muy versátiles, en el sentido de los diferentes tipos de datos que podemos almacenar en cada posición del array. Ésto nos permite por ejemplo, tener un array de arrays, proporcionando la equivalencia de arrays multidimensionales, pero adaptado a los tipos de datos que necesite nuestra aplicación.

En programación, **un array se define como una colección ordenada de datos**. Lo mejor es que pienses en un array como si fuera una tabla que contiene datos, o también como si fuera una hoja de cálculo.

JavaScript emplea un montón de arrays internamente para gestionar los objetos HTML en el documento, propiedades del navegador, etc (Etcétera). Por ejemplo, si tu documento contiene 10 enlaces, el navegador mantiene una tabla con todos esos enlaces. Tú podrás acceder a esos enlaces por su número de enlace (comenzando en el 0 como primer enlace). Si empleamos la sentencia de array para acceder a esos enlaces, el nombre del array estará seguido del número índice (número del enlace) entre corchetes, como por ejemplo: `document.links[0]` , representará el primer enlace en el documento.

En la unidad anterior si recuerdas, teníamos colecciones dentro del objeto `document`. Pues bien, cada una de esas colecciones `(anchors[], forms[], links[], e images[])` será un array que contendrá las referencias de todas las anclas, formularios, enlaces e imágenes del documento.

A medida que vayas diseñando tu aplicación, tendrás que identificar las pistas que te permitan utilizar arrays para almacenar datos. Por ejemplo, imagínate que tienes que almacenar un montón de coordenadas geográficas de una ruta a caballo: ésto si que sería un buen candidato a emplear una estructura de datos de tipo array, ya que podríamos asignar nombres a cada posición del array, realizar cálculos, ordenar los puntos, etc. Siempre que veas similitudes con un formato de tabla, será una buena opción para usar un array.

Por ejemplo si tenemos las siguientes variables:

```
var coche1="Seat";

var coche2="BMW";

var coche3="Audi";

var coche4="Toyota";
```

Este ejemplo sería un buen candidato a convertirlo en un array, ya que te permitiría introducir más marcas de coche, sin que tengas que crear nuevas variables para ello. Lo haríamos del siguiente modo:

```
var misCoches=new Array();

misCoches[0]="Seat";

misCoches[1]="BMW";

misCoches[2]="Audi";

misCoches[3]="Toyota";
```

Para crear un objeto array, usaremos el constructor `new Array( )`. Por ejemplo:

```
var miarray= new Array();
```

Creación de un array

Los objetos de tipo array disponen de una propiedad que nos indica su longitud. Esta propiedad es `length` (longitud, que será 0 para un array vacío). Si queremos precisar el tamaño del array durante la inicialización (por ejemplo para que se cargue con valores null), podríamos hacerlo pasándole un parámetro al constructor. Por ejemplo, aquí puedes ver como crear un nuevo array que almacene la información de 40 personas:

```
var personas = new Array(40);
```

A diferencia de otros lenguajes de programación, en los que hacer el dimensionamiento previo del array tiene ventajas, en JavaScript no nos dará ningún tipo de ventaja especial, ya que podremos asignar un valor a cualquier posición del array en cualquier momento, esté o no definida previamente. La propiedad `length` se ajustará automáticamente al nuevo tamaño del array. Por ejemplo podríamos hacer una asignación tal como personas[53] y eso que nuestro array es de 40 posiciones. En este caso no ocurrirá ningún problema, ya que la propiedad `length` del array se ajustará automáticamente para poder almacenar la posición 53, con lo que la nueva longitud del array será 54, ya que la primera posición del array es la [0] y la última es la [53], tendremos por lo tanto 54 elementos en el array.

```
personas[53] =  "Irene Sáinz Veiga";

longitud = personas.length;      // asignará a la variable longitud el valor 54
```

Introducir datos en un array

Introducir datos en un array, es tan simple como crear una serie de sentencias de asignación, una por cada elemento del array. Ejemplo de un array que contiene el nombre de los planetas del sistema solar:

```
sistemaSolar = new  Array( );

sistemaSolar[0] = "Mercurio";

sistemaSolar[1] = "Venus";

sistemaSolar[2] = "Tierra";

sistemaSolar[3] = "Marte";

sistemaSolar[4] = "Jupiter";

sistemaSolar[5] = "Saturno";

sistemaSolar[6] = "Urano";

sistemaSolar[7] = "Neptuno";
```

Esta forma es un poco tediosa a la hora de escribir el código, pero una vez que las posiciones del array están cubiertas con los datos, acceder a esa información nos resultará muy fácil:

```
unPlaneta = sistemaSolar[2];     // almacenará en unPlaneta la cadena "Tierra".
```

Otra forma de crear el array puede ser mediante el constructor. En lugar de escribir cada sentencia de asignación para cada elemento, lo podemos hacer creando lo que se denomina un "**array denso**", aportando al **constructor** `array( )`, los datos a cubrir separados por comas:

```
sistemaSolar = new array
("Mercurio","Venus","Tierra","Marte","Jupiter","Saturno","Urano","Neptuno");
```

El término de "**array denso**" quiere decir que los datos están empaquetados dentro del array, sin espacios y comenzando en la posición 0. Otra forma permitida a partir de la versión de JavaScript 1.2+, sería aquella en la que no se emplea el constructor y se definen los arrays de forma literal:

```
sistemaSolar = ["Mercurio","Venus","Tierra","Marte","Jupiter","Saturno","Urano","Neptuno"];
```

Los corchetes sustituyen a la llamada al constructor `new Array()`. Hay que tener cuidado con esta forma de creación que quizás no funcione en navegadores antiguos.

Y para terminar vamos a ver otra forma de creación de un **array mixto** (o también denominado objeto literal), en el que las posiciones son referenciadas con índices de tipo texto o números, mezclándolos de forma aleatoria. Si las posiciones índice están definidas por un texto, para acceder a ellas lo haremos utilizando su nombre y no su número (en este caso si usamos el número nos daría una posición undefined). El formato de creación sería: `nombrearray = { "indice1" : valor , indice2 : "valor" , ...}`

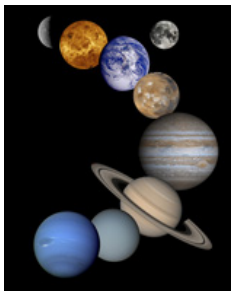
(fíjate que en este caso para definir el array de tipo mixto tendremos que hacerlo comenzando y terminando con **llaves { }**). Por ejemplo:

```
var datos = { "numero": 42, "mes"  : "Junio", "hola" : "mundo", 69 : "96" };

document.write("<br/>"+datos["numero"]+ " -- " + datos["mes"]+ " -- " + datos["hola"]+ " -- " +
datos[69]+ "<br/>");
```

Recorrido de un array.

Existen múltiples formas de recorrer un array para mostrar sus datos. Veamos algunos ejemplos con el array del sistema Solar:

```
var sistemaSolar = new  Array();

sistemaSolar[0] = "Mercurio";

sistemaSolar[1] = "Venus";

sistemaSolar[2] = "Tierra";

sistemaSolar[3] = "Marte";

sistemaSolar[4] = "Jupiter";

sistemaSolar[5] = "Saturno";

sistemaSolar[6] = "Urano";

sistemaSolar[7] = "Neptuno";
```



Empleando un bucle **for**, por ejemplo:

```
for (i=0;i<sistemaSolar.length;i++) {
     document.write(sistemaSolar[i] + "<br/>");
}
```

Empleando un bucle **while**, por ejemplo:

```
var i=0;
while (i < sistemaSolar.length) {
     document.write(sistemaSolar[i] + "<br/>");
     i++;
}
```

Empleando la sentencia **for each in** (disponible en versiones de JavaScript 1.6 o superiores):

```
for each (variable en objeto) {
     sentencia
}
```

Esta sentencia repite la variable indicada, sobre todos los valores de las propiedades del objeto. Para cada propiedad distinta, se ejecutará la sentencia especificada.

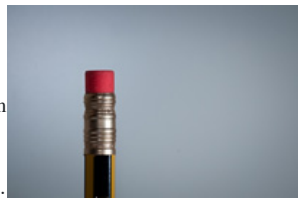
Ejemplo 1:

```
for each (var planeta in sistemaSolar) {
     document.write(planeta+"<br/>");
}
// Imprimirá todos los nombres de planeta con un salto de línea al final de cada nombre.
```

Ejemplo 2:

```
[document.write(planeta +"<br/>") for each (planeta in sistemaSolar)];
```

Borrado de elementos de un array



Para borrar cualquier dato almacenado en un elemento del array, lo podrás hacer ajustando su valor a null o a una **cadena vacía** "".

Hasta que apareció el operador **delete** en las versiones más modernas de navegadores, no se podía eliminar completamente una posición del array.

Al borrar un elemento del array, se eliminará su índice en la lista de índices del array, pero no se reducirá la longitud del array.
Por ejemplo en las siguientes sentencias:

```
elarray.length;          //  resultado: 8

delete elarray[5];

elarray.length;          //  resultado: 8

elarray[5];              //  resultado:  undefined
```

El proceso de borrar una entrada del array no libera la memoria ocupada por esos datos necesariamente. El intérprete de JavaScript, encargado de gestionar la colección de basura en memoria, se encargará de liberar memoria ocupada cuando la necesite.

Si se quiere tener un mayor control sobre la eliminación de elementos de un array, deberías considerar usar el método **splice(índice, número de elementos a eliminar)**, que está soportado por la mayoría de los navegadores. Este método se puede usar en cualquier array, y te permite eliminar un elemento o una secuencia de elementos de un array, provocando que la longitud del array se ajuste al nuevo número de elementos.

El operador **delete** es compatible a partir de estas versiones : WinIE4 + , MacIE4 + , NN4 + , Moz + , Safari + , Opera + , Chrome +.

Ejemplo de uso del operador delete:

Si consideramos el siguiente **array denso**:

```
var  oceanos = new  array("Atlantico","Pacifico","Artico","Indico");
```

Esta clase de array asigna automáticamente índices numéricos a sus entradas, para poder acceder posteriormente a los datos, como por ejemplo con un bucle:

```
for  (var  i=0; i < oceanos.length;  i++) {
     if  (oceanos[i] == "Atlantico") {
          //  Sentencias a realizar..
     }
}
```

Si ejecutamos la sentencia:

```
delete oceanos[2];
```

Se producirán los siguientes cambios: en primer lugar el tercer elemento del array (“Artico”), será eliminado del mismo, pero la longitud del array seguirá siendo la misma, y el array quedará tal y como:

```
oceanos[0] = "Atlantico";

oceanos[1] = "Pacifico";

oceanos[3] = "Indico";
```

Si intentamos referenciar **oceanos[2]** nos devolverá un resultado indefinido (undefined).

Si queremos eliminar la tercera posición y que el array reduzca su tamaño podríamos hacerlo con la instrucción:

```
oceanos.splice(2,1); // las posiciones del array resultante serán 0, 1 y 2.
```

El operador **delete**, se recomienda para arrays que usen texto como índices del array, ya que de esta forma se producirán menos confusiones a la hora de borrar los elementos.

Vamos a ver a continuación, las propiedades y métodos que podemos usar con cualquier array que utilicemos en JavaScript.

Propiedades del objeto array:

Propiedades del objeto Array

Propiedad	Descripción
constructor	Devuelve la función que creó el prototipo del objeto array.
length	Ajusta o devuelve el número de elementos en un array.
prototype	Te permite añadir propiedades y métodos a un objeto.

Métodos del objeto array:

Métodos del objeto Array

Métodos	Descripción

Métodos	Descripción
concat()	Une dos o más arrays, y devuelve una copia de los arrays unidos.
join()	Une todos los elementos de un array en una cadena de texto.
pop()	Elimina el último elemento de un array y devuelve ese elemento.
push()	Añade nuevos elementos al final de un array, y devuelve la nueva longitud.
reverse()	Invierte el orden de los elementos en un array.
shift()	Elimina el primer elemento de un array, y devuelve ese elemento.
slice()	Selecciona una parte de un array y devuelve el nuevo array.
sort()	Ordena los elementos de un array.
splice()	Añade/elimina elementos a un array.
toString()	Convierte un array a una cadena y devuelve el resultado.
unshift()	Añade nuevos elementos al comienzo de un array, y devuelve la nueva longitud.
valueOf()	Devuelve el valor primitivo de un array.

Ejemplo de algunos métodos del objeto array:

Método reverse():

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

    var frutas = ["Plátano", "Naranja", "Manzana", "Melocotón"];

    document.write(frutas.reverse());          // Imprimirá:  Melocotón,Manzana,Naranja,Plátano

</script>
```

Método slice():

```
<script type="text/javascript">
var frutas = ["Plátano", "Naranja", "Manzana", "Melocotón"];
document.write(frutas .slice(0,1) + "<br />");     // imprimirá: Plátano
document.write(frutas .slice(1) + "<br />");         // imprimirá: Naranja,Manzana,Melocotón
document.write(frutas .slice(-2) + "<br />");         // imprimirá: Manzana, Melocotón
document.write(frutas + "<br />");                    // imprimirá:
Plátano,Naranja,Manzana,Melocotón
</script>
```

Debes conocer

Más información y ejemplos sobre el objeto array.

Citas para pensar

> "Es curioso, que podamos predecir con siglos de antelación el recorrido de las estrellas más lejanas, y en cambio, no seamos capaces de saber cómo soplará mañana el viento en nuestro pequeño planeta." *SPOERL, Heinrich.*

Antena
Autoría: Repoort.
Licencia: CC BY-2.0.
Planetas
Autoría: Kabsik Park.
Licencia: CC BY-2.0.
Lápiz
Autoría: Daniel Novta.
Licencia: CC BY-2.0.
Juguete
Autoría: Elsie esq.
Licencia: CC BY 2.0.

◀ Solución a la tarea para DWEC04

Ir a...

4.B. Objetos JavaScript en W3Schools ▶