6. Utilización de arrays mediante estructuras de control.

JavaScript dispone de una estructura denominada **Array**, parecida a las variables, pero en este caso se trata de guardar **varios elementos de información similares** (por ejemplo los nombres de los 20 alumnos de un aula, o sus notas, etcétera).

Podemos definir un Array como una **serie de elementos todos del mismo tipo que ocupan posiciones contiguas en la memoria del ordenador**. A estos elementos se accede mediante un **nombre** o identificador común para todos los elementos que identifica el Array, y un número o **índice** que hace referencia al elemento del Array.

A continuación veremos una representación gráfica de un Array que contiene los nombres de 6 alumnos:

Alumnos

MIGUEL	ELENA	JORGE	CRISTINA	IGNACIO	ALICIA
Alumnos[0]	Alumnos[1]	Alumnos[2]	Alumnos[3]	Alumnos[4]	Alumnos[5]

Observamos que hay seis elementos de información (el Array tiene seis elementos). Todos ellos comparten un mismo identificador (Alumnos) pero cada uno tiene, además, un índice que hace referencia al elemento en particular. Podemos apreciar también que el primer elemento es el 0, el segundo el 1, y así sucesivamente. Así, cuando hagamos referencia a Alumnos[4] estaremos refiriéndonos al elemento que ocupa la posición 5 cuyo contenido en este caso es "IGNACIO".

Esta estructura se utiliza con frecuencia en programación, por ello profundizaremos en sus características y utilización.

6.1 Creación de un Array

Para crear un array utilizaremos el siguiente formato:

var *NombreDelArray* **= new Array (***NumeroDeElementos***)**

Donde :

- *NombreDelArray* es el nombre o identificador del array.
- *NumeroDeElementos* es un número que indica el número de elementos que contendrá.

Para crear el Array Alumnos que contendrá 6 elementos, escribiremos:

```
var Alumnos = new Array (6);
```

Ahora tenemos creada la estructura y las posiciones de memoria están reservadas y disponibles, aunque vacías, porque todavía no hemos introducido en ellas ningún valor.

6.2 Manipulación de los elementos de un Array

Una vez creada la estructura podemos manipular los elementos del array como si se tratase de variables (con la particularidad del índice) para introducir, cambiar o consultar los valores que contienen.

Para **introducir información en un elemento** utilizaremos normalmente un operador de asignación. Por ejemplo, para introducir el valor "MIGUEL" en el elemento Alumnos[0] escribiremos: Alumnos[0] = "MIGUEL"; de manera similar se introducirán todos los elementos del Array:

```
Alumnos[0] = "MIGUEL";
Alumnos[1] = "ELENA";
Alumnos[2] = "JORGE";
Alumnos[3] = "CRISTINA";
Alumnos[4] = "IGNACIO";
Alumnos[5] = "ALICIA";
```

Podemos hacer referencia a cualquiera de los objetos del Array bien para cambiar su valor, o bien para utilizarlo en expresiones como si se tratase de cualquier otra variable, con la particularidad apuntada de la utilización del índice. Así, podemos escribir las siguientes expresiones:

```
alert(Alumnos[4]);
```

Visualiza una ventana de alerta con el contenido de Alumnos[4]

```
Alumnos[4] = prompt("Nombre del alumno", " ")
```

Lee un nuevo nombre de la consola y lo introduce en el elemento Alumnos[4]

```
var DosPrimeros = Alumnos[0] + Alumnos[1];
```

Crea la variable *DosPrimeros* e introduce en ella el resultado de concatenar Alumnos[0] + Alumnos[1];

```
Alumnos[2] > Alumnos[3]
```

Es una expresión que compara el contenido de ambos elementos y determina si el primero es mayor que el segundo según su valor ASCII en cuyo caso el resultado será true, en caso contrario false.

En general, para hacer referencia a un elemento de un Array emplearemos el formato genérico:

***NombreDelArray* [*Indice*]**

El índice no tiene que ser necesariamente una constante numérica (2, 10, 35) se puede referenciar mediante cualquier expresión numérica que devuelva un entero. Así, por ejemplo, las siguientes expresiones serían expresiones válidas (suponiendo que i sea una variable numérica entera):

```
Alumnos[ i ]
```

Hace referencia al elemento i (si i vale 2, al dos, etcétera).

Alumnos[i + 1]

Hace referencia al elemento siguiente al elemento i (si i vale 2, hará referencia el elemento 3).

Alumnos[i] > Alumnos [i + 1]

Compara el elemento i con el siguiente.

6.3 Recorrido de los elementos de un Array

Las estructuras de control repetitivas se utilizan frecuentemente para recorrer los elementos de un array.

En los arrays está disponible la propiedad **length** que **devuelve el número de elementos que tiene un array** (incluyendo los elementos vacíos, si hubiese). Su formato genérico es:

NombreDelArray.length

Utilizando esta propiedad podemos escribir una estructura que recorrería cualquier array completo:

```
for (i = 0;  i < NombreDelArray.length; i = i + 1)
{
 Tratamiento_del_elemento_NombreDelArray[i]);
}
```

6.4 Introducción de los elementos del array desde la consola del usuario.

Hasta el momento hemos trabajado con un array cuyos datos se introducen directamente desde el código JavaScript. Pero en ocasiones necesitaremos que sea el usuario quien introduzca los elementos del array.

JavaScript también permite crear un array e introducir los elementos simultaneamente desde el código declarando el nombre y enumerando a continuación en una lista los elementos:

```
var Alumnos = new Array ("MIGUEL", "ELENA", "JORGE",
    "CRISTINA","IGNACIO", "ALICIA");
```

El array se dimensionará implícitamente en función del número de elementos. Por su parte, los elementos definidos así se asociarán con el índice según la posición que ocupan el la lista (el primero será el 0, el segundo el 1, y así sucesivamente).

6.5 Búsqueda en un array.

Podemos realizar búsquedas en un array basándonos en los siguientes criterios:

- **Búsqueda a partir del índice**. (Sabiendo cuál es el índice, obtener el elemento).
- **Búsqueda de un elemento para obtener su posición** (o simplemente saber que existe).

La primera no plantea ningún problema ya que si conocemos el índice el acceso al elemento correspondiente es automático. (Por ejemplo, si buscamos el elemento 3 accedemos a él como Alumnos[3]).

La segunda implica un recorrido del array comparando uno a uno cada elemento con el valor que se busca hasta obtener dicho valor o llegar al final sin obtener un resultado satisfactorio.

La salida del bucle se produce por una de las siguientes circunstancias: ha llegado al último elemento; o bien, ha encontrado en valor buscado. Deberemos, por tanto, comprobar si realmente ha encontrado lo que buscaba o no.

6.6. Propiedades de los arrays.

En el lenguaje JavaScript, un array es en realidad un objeto. Este objeto tiene una seria de propiedades y métodos al igual que los otros objetos que hemos estudiado hasta ahora.

El objeto Array tiene dos propiedades: lenght y prototype.

lenght, devuelve el numero de elementos que contiene el array. Esta propiedad es muy útil cuando utilizamos los arrays en los bucles. Su sintaxis es la siguiente:

```
nombre_del_array.lenght
```

La segunda propiedad del Objeto Array, prototype, es mucho mas compleja que la anterior. Con esta propiedad podemos agregar nuevas propiedades y métodos al objeto Array. La sintaxis de prototype es la siguiente:

```
Array.prototype.nueva_propiedad = valor;
Array.prototype.nuevo_metodos = nombre_de_la_funcion;
```

Esta propiedad permite al usuario extender la definición de los arrays que se utilicen en alguna aplicación web en concreto. Si queremos agregar una nueva propiedad, debemos definir un valor concreto de esa propiedad, aunque luego se pueda cambiar. Si además queremos crear un nuevo método, debemos definir previamente una función JavaScript y escribir el nombre de dicha función en la creación del nuevo método. Si, por ejemplo, queremos crear una propiedad llamada dominio y establecer el valor .com, debemos escribir el siguiente código:

```
Array.prototype.dominio = ".com";
```

De este modo, si creamos un array, este tendrá ya definido dicha propiedad. Sin embargo, podemos cambiar el valor de esta propiedad en el momento en que creamos nuevos arrays.

```
var paginas_comerciales = new Array ();
Array.prototype.dominio = ".com";
var paginas_gubernamentales = new Array ();
paginas_gubernamentales.dominio = ".gov";
document.write ("Extensión de las páginas comerciales: " +
    paginas_comerciales.dominio);
document.write ("Extensión de las páginas gubernamentales: " +
    paginas_gubernamentales.dominio);
```

6.7. Métodos de los arrays.

El objeto Array posee algunos métodos bastantes útiles a la hora de manipular y gestionar todos los elementos presentes en los arrays. Estos métodos permiten unir dos arrays, ordenar, invertir sus valores o eliminar fácilmente algunos de sus elementos. Los principales métodos del objeto Array serian:

Método	Descripción	Sintaxis
`push()`	Añade nuevos elementos al *array* y devuelve la nueva longitud del *array*.	`nombre_array.push(valor1, valor2, ...)`
`concat()`	Selecciona un *array* y lo concatena con otros elementos en un nuevo *array*.	`nombre_array.concat(valor1, valor2, ...)`
`join()`	Concatena los elementos de un *array* en una sola cadena separada por un carácter opcional.	`nombre_array.join([separador])`
`reverse()`	Invierte el orden de los elementos de un *array*.	`nombre_array.reverse()`
`unshift()`	Añade nuevos elementos al inicio de un *array* y devuelve el número de elementos del nuevo *array* modificado.	`nombre_array.unShift(valor1, valor2, ...)`
`shift()`	Elimina el primer elemento de un *array*.	`nombre_array.shift()`
`pop()`	Elimina el último elemento de un *array*.	`nombre_array.pop()`
`slice()`	Devuelve un nuevo *array* con un subconjunto de los elementos del *array* que ha usado el método.	`nombre_array.slice(índice_inicio, [índice_final])`
`sort()`	Ordena alfabéticamente los elementos de un *array*. Podemos definir una nueva función para ordenarlos con otro criterio.	`nombre_array.sort([función])`
`splice()`	Elimina, sustituye o añade elementos del *array* dependiendo de los argumentos del método.	`nombre_array.splice(inicio, [numero_elem_a_borrar], [valor1, valor2, ...])`

A continuación mostramos algunos ejemplos de cada uno de los métodos del objeto `Array`.

- **push()**. En el siguiente ejemplo declaramos un nuevo *array* llamado `pizzas` y se inicializa con tres elementos. Posteriormente, aplicamos el método `push()` para añadir dos nuevas *pizzas*. Este método devuelve el nuevo número de elementos presentes en el *array*:

```
<script type="text/javascript">
  var pizzas = new Array("Carbonara", "Quattro_Stagioni",
    "Diavola");
  var nuevo_numero_de_pizzas = pizzas.push("Margherita",
    "Boscaiola");
  document.write("Número de pizzas disponibles: " +
    nuevo_numero_de_pizzas + "<br />");
  document.write(pizzas);
</script>
```

- **concat()**. El método `concat()` une los elementos de dos o más *arrays* en uno nuevo. En este ejemplo tenemos dos *arrays* que contienen algunos equipos de primera y de segunda división del fútbol español. Posteriormente, concatenamos estos equipos de fútbol en un nuevo *array* llamado `equipos_copa_del_rey` e imprimimos los valores de los elementos.

```
<script type="text/javascript">
  var equipos_a = new Array("Real Madrid", "Barcelona",
    "Valencia");
  var equipos_b = new Array("Hércules", "Elche",
    "Valladolid");
  var equipos_copa_del_rey = equipos_a.concat(equipos_b);
  document.write("Equipos que juegan la copa: " +
    equipos_copa_del_rey);
</script>
```

- **join()**. El método `join()` devuelve una cadena de texto con los elementos del *array*. Los elementos los podemos separar por una cadena que le pasemos como argumento del método. En el siguiente ejemplo utilizamos un guión como separador entre los elementos del *array* `pizzas`:

```
<script type="text/javascript">
  var pizzas = new Array("Carbonara", "Quattro_Stagioni",
    "Diavola");
  document.write(pizzas.join(" - "));
</script>
```

- **reverse()**. Este método invierte el orden de los elementos sin crear un nuevo *array*. En el siguiente ejemplo tenemos un *array* con los números ordenados del 1 al 10. Si se aplica el método `reverse()` y escribimos el resultado en el documento, veremos que ahora van del 10 al 1.

```
<script type="text/javascript">
  var numeros = new Array(1,2,3,4,5,6,7,8,9,10);
  numeros.reverse();
  document.write(numeros);
</script>
```

- **unshift()**. Con `unshift()` podemos añadir nuevos elementos y obtener la nueva longitud del *array* al igual que con el método `push()`. La diferencia es que `push()` añade los elementos al final del *array*, mientras que `unshift()` los añade al principio. En el siguiente ejemplo tenemos algunas de las últimas sedes de los juegos olímpicos. Si quisiéramos agregar una sede más al inicio del *array*, el método `unshift()` es ideal para esta tarea:

```
<script type="text/javascript">
  var sedes_JJOO = new Array("Atenas", "Sydney",
    "Atlanta");
  var numero_sedes = sedes_JJOO.unshift("Pekín");
  document.write("Últimas " + numero_sedes + " sedes
    olímpicas: " + sedes_JJOO);
</script>
```

- **shift()**. El método `shift()` elimina el primer elemento de un *array* y devuelve dicho elemento. Si, por ejemplo, tenemos el mismo *array* de antes con una lista de pizzas y quisiéramos eliminar la primera de ellas, podemos utilizar el método `shift()` de la siguiente manera:

```
<script type="text/javascript">
  var pizzas = new Array("Carbonara", "Quattro_Stagioni",
    "Diavola");
  var pizza_removida = pizzas.shift();
  document.write("Pizza eliminada de la lista: " +
    pizza_removida + "<br />");
  document.write("Nueva lista de pizzas: " + pizzas);
</script>
```

- **pop()**. El método `pop()` tiene la misma funcionalidad que el método `shift()` con la diferencia de que en vez de eliminar y devolver el primer elemento de un *array*, elimina y devuelve el último. En el siguiente ejemplo tenemos un *array* con tres premios. El método `pop()` permite eliminar y devolver el último premio del *array*, en este caso el tercer premio.

```
<script type="text/javascript">
  var premios = new Array("Coche", "1000 Euros", "Manual de
    JavaScript");
  var tercer_premio = premios.pop();
  document.write("El tercer premio es: " + tercer_premio +
    "<br />");
  document.write("Quedan los siguientes premios: " +
    premios);
</script>
```

- **slice()**. Este método crea un nuevo *array* con un subconjunto de elementos pertenecientes a otro *array*. En el paréntesis especificamos el índice inicial y el final del subconjunto que se almacenará en un nuevo *array*. El índice final es opcional y si no lo especificamos, tomará el subconjunto desde el índice inicial hasta el final del *array*. Además, el índice inicial puede ser un número negativo, con lo cual, la selección comenzaría desde el final del *array*.

```
<script type="text/javascript">
  var numeros = new Array(1,2,3,4,5,6,7,8,9,10);
  var primeros_cinco = numeros.slice(0,5);
  var ultimos_cuatro = numeros.slice(-4);
  document.write(primeros_cinco + "<br>");
  document.write(ultimos_cuatro);
</script>
```

- **sort()**. Este método ordena alfabéticamente un *array*. Sin embargo, podemos crear nuevos criterios de ordenación en una función y pasarle el nombre de la función como parámetro del método. En el siguiente ejemplo tenemos una lista de apellidos, la cual ordenamos alfabéticamente e imprimimos por pantalla:

```
<script type="text/javascript">
  var apellidos = new Array("Pérez", "Guijarro", "Arias",
    "González");
  apellidos.sort();
  document.write(apellidos);
</script>
```

- **splice()**. El método `splice()` es el más complejo del objeto `Array`. Con él es posible añadir o eliminar objetos de un *array*. Los dos primeros parámetros son obligatorios, mientras que el tercero es opcional. El primer parámetro especifica la posición en la cual añadiremos o eliminaremos los elementos. El segundo parámetro especifica cuántos elementos eliminaremos. El tercer y último parámetro son los nuevos elementos que añadiremos en el *array*. El siguiente ejemplo presenta un *array* con una serie de marcas de coches. Con el método `splice()` añadimos en la posición 2 el elemento llamado "`Seat`":

```
<script type="text/javascript">
  var coches = new Array("Ferrari", "BMW", "Fiat");
  coches.splice(2,0,"Seat");
  document.write(coches);
</script>
```

6.8. Arrays multidimensionales.

Gracias a los apartados anteriores hemos comprobado la importancia que tienen los arrays en JavaScript, al igual que en la mayor parte de los lenguajes de programación. Hasta el momento hemos estudiado arrays de una sola dimensión, es decir, una estructura de datos en la que es necesario conocer solo un índice para acceder a cada elemento. Esta estructura es análoga a una lista de elementos. Sin embargo, es posible crear arrays de mas dimensiones. Para obtener esta nueva estructura de datos debemos definir un array que contiene a su vez nuevos arrays en cada una de sus posiciones.

Los arrays bidimensionales son los arrays multidimensionales mas comunes en los lenguajes de programación. Podemos pensar en ellos como si fuesen una tabla con filas y columnas, con lo cual, necesitamos conocer dos índices para acceder a cada uno de sus elementos, tal y como podemos ver en la siguiente tabla:

[Filas, Columnas]	0	1	2
0	elemento[0,0]	elemento[0,1]	elemento[0,2]
1	elemento[1,0]	elemento[1,1]	elemento[1,2]
2	elemento[2,0]	elemento[2,1]	elemento[2,2]

En JavaScript no existe un objeto llamado array multidimensional. Para poder utilizar este tipo de estructuras de datos debemos definir un array, en el que en cada una de sus posiciones debemos crear a su vez otro array.

En el siguiente ejemplo definimos un array bidimensional en el que por un lado tenemos

el nombre de algunos países (España, Suiza y Portugal) y, por el otro, las cinco palabras mas buscada en Internet de cada país en el año 2011, según los datos ofrecidos por Google.

```
var palabras_espana = new Array(5);
palabras_espana[0] = 'facebook';
palabras_espana[1] = 'tuenti';
palabras_espana[2] = 'youtube';
palabras_espana[3] = 'hotmail';
palabras_espana[4] = 'marca';

var palabras_suiza = new Array(5);
palabras_suiza[0] = 'facebook';
palabras_suiza[1] = 'youtube';
palabras_suiza[2] = 'hotmail';
palabras_suiza[3] = 'google';
palabras_suiza[4] = 'blick';

var palabras_portugal = new Array(5);
palabras_portugal[0] = 'facebook';
palabras_portugal[1] = 'youtube';
palabras_portugal[2] = 'hotmail';
palabras_portugal[3] = 'jogos';
palabras_portugal[4] = 'download';

var palabras_mas_buscadas = new Array(3)
palabras_mas_buscadas[0] = palabras_espana;
palabras_mas_buscadas[1] = palabras_suiza;
palabras_mas_buscadas[2] = palabras_portugal;
```

Inicialmente hemos definido tres arrays de una dimensión. Cada uno de ellos contiene las cinco palabras mas buscada en Internet en España, Suiza y Portugal. Posteriormente hemos creado un array de tres elementos. Estos tres elementos corresponden a los arrays que contienen las palabras mas buscadas en Internet en cada uno de los tres países.

En este caso, hemos utilizado una línea de código para definir e inicializar cada elemento, pero es posible usar los métodos abreviados que hemos visto en apartados anteriores. Tal y como describe el siguiente código:

```
var palabras_espana = new Array('facebook','tuenti',
  'youtube', 'hotmail', 'marca');

var palabras_suiza = new Array('facebook', 'youtube',
  'hotmail', 'google', 'blick');

var palabras_portugal = new Array('facebook', 'youtube',
  'hotmail', 'jogos', 'download');

var palabras_mas_buscadas = new Array(palabras_espana, palabras_suiza, palabras_
portugal);
```

De este modo obtendríamos una estructura de datos similar a una tabla en la que las filas corresponden a los países y las columnas a las palabras mas buscadas en Internet en cada uno de ellos.

[País,Ranking]	0	1	2	3	4
0 (España)	facebook	tuenti	youtube	hotmail	marca
1 (Suiza)	facebook	youtube	hotmail	google	blick
2 (Portugal)	facebook	youtube	hotmail	jogos	download

Al igual que con los arrays de una sola dimensión, podemos utilizar bucles para definir e inicializar arrays multidimensional. Para ello, debemos utilizar lo que se denomina un bucle anidado.