UNIDAD 4 Baraja de cartas

Arrays

Arrays

Un *array* es un listado de valores. Estos valores pueden ser de cualquier tipo (literales, numerales, *booleanos*, objetos, otros *arrays*...) y no tienen porqué ser todos del mismo tipo. Un *array* se acota con los signos `[ ]`, y dentro del propio *array*, los valores se separan con comas.

Acceder a un *array*

Podemos acceder a un valor concreto de un *array* indicando el nombre del *array* seguido de los signos `[]` y dentro, la posición del elemento, es decir: **`ejemploArray[3]`**

Conviene recordar que en un *array* (y en general, en JavaScript), las posiciones se empiezan a contar desde el **`0`**, por lo tanto, accederemos al primer valor de un *array* de esta forma: **`ejemploArray[0]`**

En el caso de un *array* dentro de otro, accederemos a un valor concreto del *array* anidado indicando su posición en el primer **`array []`**, y después la posición del segundo **`array []`**, por ejemplo: **`ejemploArray[2][4]`**

Operaciones con arrays

`ejemploArray.concat(…)`

Para combinar un *array* con otro u otros, seleccionaremos el primero y usaremos el método **`concat()`**, con el *array* o *arrays* adicionales como argumentos, separados por comas.

Ejemplo:

```
var totalTarjetas =
 grupoTarjetas1.concat(grupoTarjetas2, grupoTarjetas3);
```

`ejemploArray.length`

Podemos saber el número de valores (longitud) de un *array* concreto accediendo a su propiedad **`length`**.

Conviene recordar que la longitud indicará el número de elementos (por ejemplo, **`3`**), pero que para acceder a ellos, se siguen aplicando las reglas de posición de los *arrays* (en el ejemplo de **`3`** elementos, será `[0]`, `[1]` o `[2]`.

`includes(…)`

Podemos comprobar si un valor existe (nos devolverá **`true`** o **`false`**) en un *array* concreto mediante el método **`includes()`**.

Por ejemplo: **`arrayFrutas.includes("limón")`**

`push(…)`

Podemos añadir un nuevo valor al final de un *array* con el método **`push()`**.

Ejemplo: **`arrayFrutas.push("naranja");`**

`pop()`

Podemos eliminar el valor al final de un *array* con el método **`pop()`**.

Ejemplo: **`arrayFrutas.pop();`**

`shift()`

Podemos eliminar el valor al principio de un *array* con el método **`shift()`**.

Ejemplo: **`arrayFrutas.shift();`**

`unshift(…)`

Podemos añadir un nuevo valor al principio de un *array* con el método **`unshift()`**.

Ejemplo: **`arrayFrutas.unshift(“melón");`**

`ejemploArray.indexOf(…)`

Para conocer la posición que ocupa un valor dentro de un *array*, podemos emplear el método **`indexOf()`**.

Ejemplo: **`var posicionElemento = arrayFrutas.indexOf(“limón");`**

`ejemploArray.splice(…)`

El método **`splice()`** nos permite incluir y modificar valores en posiciones concretas dentro de un *array*. Los argumentos que acepta, separados por comas, son la posición en la que se hace una modificación, el número de valores que serán sustituidos, y los valores nuevos a insertar.

Por ejemplo: **`arrayFrutas.splice(2, 0, "uvas", “kiwi");`**

En este ejemplo accedemos a la posición **`2`** del array, sustituimos **`0`** valores a continuación (por tanto, solo añadimos, no quitamos) y agregamos los valores **`"uvas"`** y **`"kiwi"`**.

`ejemploArray.slice(…)`

El método **`slice()`** nos permite obtener una copia con una porción de un *array*, indicando la posición de inicio y la posición del fin del "corte", separados por comas. Este método nos devolverá la fracción contenida entre estos dos puntos, sin incluir el valor final. Por ejemplo, **`ejemploArray.slice(0, 3)`** nos devolverá los valores en las posiciones **`0`**, **`1`** y **`2`**.

Si queremos hacer un duplicado de un *array* (y no reasignarlo a otra variable), podemos emplear el método **`slice()`** sin pasar argumentos.

Ejemplo:
`var duplicado = arrayFrutas.slice();`

`ejemploArray.sort(…)`

Este método permite ordenar los valores de un array de una determinada forma. Acepta como argumento una función. Si ejecutamos este método sin pasarle argumentos, ordenará los valores alfabéticamente o numéricamente, en función del tipo de valor.

Loops

Loops con `forEach()`

Un *array* o una lista de nodos puede ser "recorrida" con el método **`forEach()`**, que realizará una tarea por cada valor que tenga dicho *array* o *nodelist*. El argumento que espera el método **`forEach()`** es una función, por lo que podemos declarar una función anónima y en su interior, desarrollar el código deseado. Si añadimos un primer argumento ("variable local") dentro del paréntesis de la función anónima, podemos referirnos al valor concreto sobre el que el *loop* está ejecutando la función en ese momento. Ejemplo:

```
arrayFrutas.forEach(function (elemento) {
  console.log(elemento);
});
```

Podemos añadir hasta dos argumentos más (si nos interesa) para recuperar la posición que el valor ocupa en el *array*, y el *array* al que pertenece. Ejemplo:

```
arrayFrutas.forEach(function(elemento, posicion, arrayOrigen) {
  console.log(elemento, posicion, arrayOrigen);
});
```

Loops con `for()`

Un *loop* con **`for`** ejecutará un código un número determinado de veces. La construcción más habitual de los *loops* con for es la siguiente:

```
for (var i = 0; i < 9; i++) {
  console.log("posición del loop:", i)
}
```

Donde los argumentos que pasamos, separados por punto y coma, son la variable que contará el número de veces que se ejecuta el *loop*, la condición que se debe cumplir para que el *loop* se siga realizando (en el ejemplo, que el valor de la variable contador sea inferior a **`9`**), y la acción a realizar tras completar una iteración del *loop* (en el ejemplo, sumar **`1`** a la variable contador).

Eventos en *loop*

Podemos usar un *loop* para asignar un *event listener* a cada elemento que cumpla unas características. Por ejemplo:

```
var totalTarjetas = document.querySelectorAll(".tarjeta");
totalTarjetas.forEach(function(elemento) {
  elemento.addEventListener("click", descubrir);
});
```

En este código creamos una lista de nodos a partir e una búsqueda en el DOM de todos los elementos que tengan la clase tarjeta. Después aplicamos un `forEach()` a esta lista de nodos y para cada uno de ellos (que recogemos en el argumento "elemento"), añadimos un *listener* que vincula la función descubrir al evento `click`.

Modificar clases y atributos

elemento.classList.add(...)

Permite añadir una clase a un elemento del DOM.

Ejemplo: `elemento.classList.add("activo");`

elemento.classList.remove(...)

Permite quitar una clase a un elemento del DOM.

Ejemplo: `elemento.classList.remove(“activo");`

elemento.classList.contains(...)

Permite comprobar si un elemento del DOM tiene una clase en particular.

Ejemplo:

```
if (elemento.classList.contains("activo")){
  /* código… */
}
```

elemento.classList.toggle(...)

Permite añadir una clase a un elemento del DOM si no la tiene, o quitársela si la tiene.

Ejemplo: `elemento.classList.toggle(“activo");`

elemento.getAttribute(...)
Permite recuperar el valor de un atributo de un elemento.

Ejemplo: `var valorHref = elemento.getAttribute("href");`

elemento.setAttribute(...)
Asigna el valor de un atributo en un elemento, y el atributo si no existiera.

Ejemplo: `elemento.setAttribute("href", "http://google.com");`

elemento.removeAttribute(...)
Elimina un atributo de un elemento.

Ejemplo: `elemento.removeAttribute("style");`

elemento.hasAttribute(...)
Comprueba si un elemento tiene un atributo o no.

Ejemplo:
```
if(elemento.hasAttribute("href")){
  /* código… */
}
```

Objetos

Objetos
Los objetos son modelos de datos que contienen propiedades (relación entre una clave y un valor) y métodos (funciones propias). En JavaScript, prácticamente todos los elementos son concebidos como objetos, y por tanto tienen propiedades y métodos.

Los objetos se pueden crear asignando a una variable directamente valores entre los signos `{}` como en el ejemplo:

```
var animal =  {
  tipo: "gato",
  nombre: "Garfield"
}
```

Y también se pueden declarar como objetos vacíos (a los que no hemos asignado aún propiedades) de esta forma:

```
var animal = new Object();
```

Acceder a las propiedades de los objetos

Podemos acceder a las propiedades (y métodos) de los objetos refiriéndonos al nombre de las mismas y al de dichos objetos, enlazados con un punto.

Ejemplo: `var tipoAnimal = animal.tipo;`

Si queremos ejecutar un método de un objeto, lo haremos usando los paréntesis, como con cualquier función. Es el caso en **`console.log()`**, donde ejecutamos el método **`log()`** del objeto **`console`**.

Modificar propiedades de objetos

Podemos modificar las propiedades de un objeto realizando una asignación de valor a una propiedad concreta, exista esta (su valor se sobreescribirá) o no (se agregará una nueva propiedad).

Ejemplo: `animal.edad = 12;`

Objetos globales

Objetos globales

Los objetos que pueden accederse desde cualquier función son objetos globales. Podemos crear este tipo de objetos si los declaramos en el espacio global (fuera de una función), pero además, JavaScript dispone de un conjunto de objetos ya creados a través de los cuales interpreta una web. Algunos de estos objetos son el objeto **`window`** (la ventana en la que se muestra la web), el objeto **`document`** (la web en sí misma, ya renderizada), el objeto **`Math`** (que permite hacer operaciones matemáticas) o el objeto **`Date`**, que permite conocer la fecha y hora actuales o realizar operaciones con fechas.

Objeto `window`

El objeto **`window`** hace referencia a la ventana que muestra la página web. Tiene muchas propiedades y métodos, como **`window.innerHeight`** (altura de la ventana), **`window.innerWidth`** (anchura de la ventana), **`window.location`** (URL de la página que se está mostrando), etc. Un uso bastante habitual de este objeto aplicarle un event listener para asociar una función al evento **`"resize"`** (redimensiona la ventana) o al evento **`"scroll"`** (cuando el usuario hace scroll).

Objeto `document` (DOM)

El objeto **`document`** hace referencia a la página web renderizada dentro del objeto **`window`**. Dispone también de muchas propiedades y eventos, siendo de las más populares los métodos **`document.querySelector()`** y **`document.querySelectorAll()`**, que permiten realizar búsquedas en el DOM de determinados elementos, mediante selectores de CSS.

Objeto `Math`

El objeto **`Math`** dispone de varias propiedades y métodos relacionados con operaciones matemáticas, como **`Math.PI`**, **`Math.cos(…)`** (devuelve el coseno de un número), **`Math.floor(…)`** (devuelve un número entero, quitando los decimales), **`Math.random()`** (devuelve un número comprendido entre el **`0`** y el **`1`**, normalmente con decimales), etc.

Objeto `Date`

El objeto **`Date`** permite crear una nueva fecha (y hora), bien a partir de la actual, o bien a partir de los datos que le pasemos. Ejemplos:

```
var fechaActual = new Date();
var fechaCreada = new Date('December 17, 1995 03:24:00');
```

A partir de una nueva fecha declarada, podemos emplear sus métodos para acceder a valores concretos, como el año: **`getFullYear()`**, el número de mes del **`0`** al **`11`**: **`getMonth()`**, el día: **`getDay()`**, y horas: **`getHours()`**, minutos: **`getMinutes()`**, segundos: **`getSeconds()`** y milisegundos: **`getMilliseconds()`**

Ejemplo:

```
var fechaActual = new Date();
console.log( fechaActual.getFullYear() );
```