Programación con Javascript

Unidad 3: Vectores y ciclos

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Indice

Unidad 3: Vectores y ciclos

Objetivos

Que el alumno logre:

- Conocer los tipos de variables y los operadores

Vectores simples y complejos

Los vectores o arrays son "objetos tipo lista"; básicamente son objetos individuales que contienen múltiples valores almacenados en una lista.

Los objetos un vector pueden almacenarse en variables y tratarse de la misma manera que cualquier otro tipo de valor, la diferencia es que podemos acceder individualmente a cada valor dentro de la lista y hacer cosas útiles y eficientes con la lista, como recorrerlo con un bucle y hacer una misma cosa a cada valor.

Si no tuviéramos vectores, tendríamos que almacenar cada elemento en una variable separada, luego llamar al código que hace la impresión y agregarlo por separado para cada artículo. Esto sería mucho más largo de escribir, menos eficiente y más propenso a errores. Si tuviéramos 10 elementos para agregar a la factura, ya sería suficientemente malo, pero ¿qué pasa con 100 o 1000 artículos?

Los vectores se construyen con corchetes, que contienen una lista de elementos separados por comas.

Ejemplo:

Queremos almacenar una lista de ingredientes para realizar una torta

```
var ingredientes = ['harina', 'huevos', 'azúcar', 'leche']
```

Podemos acceder a los elementos dentro del array ingredientes por el número de índice automático que asigna Javascript

```
ingredientes[0]
// devuelve: harina
```

También podemos modificar un elemento en un vector simplemente dando a un item un nuevo valor. Por ejemplo:

```
ingredientes[0]= 'manteca'
```

La variable ahora quedará:

```
ingredientes = ['manteca', 'huevos', 'azúcar', 'leche']
```

Un vector dentro de otro vector se llama vector multidimensional. Podemos acceder a los elementos de un vector que está dentro de otro encadenando dos pares de corchetes.

Por ejemplo, para acceder a uno de los elementos dentro del vector que a su vez es el tercer elemento dentro de otro vector, podríamos hacer algo como esto:

```
var compras = ['velas', 'crema', 'chocolate', ['manteca', 'harina', 'huevos', 'azúcar', 'leche']]

compras [3][3]
// devuelve: azúcar
```

Métodos

En este apartado veremos algunos métodos bastante útiles relacionados con vectores que nos permiten dividir cadenas en elementos de matriz y viceversa, y agregar nuevos elementos.

Los vectores tienen distintos métodos, encontrando al método lenght, toString, stack, push en Array, pop en Array y otros métodos adicionales, los cuales detallaremos:

length

Podemos averiguar la longitud de un vector (cuantos elementos contiene) exactamente de la misma manera que determinas la longitud (en caracteres) de una cadena— utilizando la propiedad **length**. Por ejemplo:

```
ingredientes.length
// devuelve 5
```

Si queremos darle otra utilidad a este método, pues podríamos usarlo para agregar elementos al final del array, el cual sería:

```
ingredientes[ingredientes.length]= 'vainilla'
```

Aquí la posición a agregar coincide con el número total de elemento, es decir el string "vainilla", iría a la última posición del array.

toString()

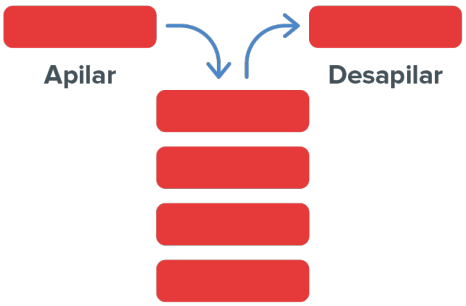
De necesitar que todo el vector se convierta en una cadena, el método toString() nos será de gran utilidad, y su manera de uso, es la siguiente:

```
ingredientes.toString();
```

Nos mostrará todo el array en una cadena, cada elemento separados por comas pero unidos en un string.

stack

Un vector puede actuar como una pila, que nos 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. La inserción (push) y remoción (pop).

Método Push en Array

Transforma un vector añadiendo los elementos proporcionados y devolviendo la nueva longitud del vector.

Veamos de qué manera utilizar el método push, tenemos el array:

```
var ingredientes = ['harina', 'huevos', 'azúcar', 'leche']
```

El método push acepta cualquier número de argumentos y los agrega al final del array y sería:

```
ingredientes.push('manteca', 'vainilla')
['harina', 'huevos', 'azúcar', 'leche', 'manteca', 'vainilla']
```

Entonces tendremos un array con 6 elementos, y con los valores establecidos.

Método Pop en Array

Este método permite eliminar el último elemento del array, disminuyendo la longitud de dicho array, veamos:

```
var ingredientes =['harina', 'huevos', 'azúcar', 'leche', 'manteca', 'vainilla']
ingredientes.pop()
['harina', 'huevos', 'azúcar', 'leche', 'manteca']
```

Si mostramos el array, vemos que el elemento string “vainilla” dejó de pertenecer al array

shift

Este método nos permite obtener el primer elemento de un array, como lo veremos a continuación:

```
var ingredientes =['harina', 'huevos', 'azúcar', 'leche', 'manteca', 'vainilla']
ingredientes.shift()
//devuelve “harina”
```

unshift

Es el método con el cual podemos agregar elementos al inicio del array, podemos contener más de un argumento, esto quiere decir que podemos agregar en la misma sentencia más de un elemento. Veamos su forma de uso:

```
var ingredientes =['azúcar', 'leche', 'manteca', 'vainilla']
ingredientes.unshift('huevos')
['huevos', 'azúcar', 'leche', 'manteca', 'vainilla']
```

reverse

Este método establece que el array invierte sus elementos:

```
var ingredientes =['azúcar', 'leche', 'manteca', 'vainilla']
ingredientes.reverse();
['vainilla', 'manteca', 'leche', 'azúcar']
```

veremos que nos devuelve todos los datos invertidos.

Sort

Este método es muy útil cuando tengamos un array con elementos string, pues estos serán ordenados alfabéticamente. En el caso de nuestro array:

```
var ingredientes =['leche', 'manteca', 'azúcar', 'vainilla']
ingredientes.sort();
['azúcar', 'leche', 'manteca', 'vainilla']
```

Lo ordenará alfabéticamente. De usarse este método en elementos de tipo numéricos nos devolverá datos incorrectos.

¿Qué es un bucle?

Los lenguajes de programación son muy útiles para completar rápidamente tareas repetitivas, desde múltiples cálculos básicos hasta cualquier otra situación en donde tengamos un montón de elementos de trabajo similares que completar. Aquí vamos a ver las estructuras de bucles disponibles en JavaScript que pueden manejar tales necesidades.

Los bucles de programación están relacionados con todo lo referente a hacer una misma cosa una y otra vez — que se denomina como iteración en el idioma de programación.

Un bucle cuenta con una o más de las siguientes características:

- Un **contador**, que se inicia con un determinado valor — este será el valor del punto inicia del bucle
- Una **condición de salida**, que será el criterio bajo el cual, el bucle se romperá — normalmente un contador que alcanza un determinado valor.
- Un **iterador**, que generalmente incrementa el valor del contador en una cantidad pequeña a cada paso del bucle, hasta que alcanza la condición de salida.

for, while y do while

Estructura for

La estructura for permite realizar repeticiones (también llamadas bucles) de una forma muy sencilla.

```
for(contador; condición de salida; iterador) {
...
}
```

La idea del funcionamiento de un bucle for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

```
for(var i = 0; i < 5; i++) {
alert(mensaje);
}
```

La parte de la inicialización del bucle consiste en:

```
var i = 0;
```

Por tanto, en primer lugar se crea la variable i y se le asigna el valor de 0. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es: i < 5

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable i valga menos de 5 el bucle se ejecuta indefinidamente.

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente.

Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle:i++ En este caso, el valor de la variable i se incrementa en una unidad después de cadarepetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el for.

Así, durante la ejecución de la quinta repetición el valor de i será 4. Después de la quint ejecución, se actualiza el valor de i, que ahora valdrá 5. Como la condición es que i sea menor que 5, la condición ya no se cumple y las instrucciones del for no se ejecutan una sexta vez. Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

El ejemplo anterior que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura for:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
for(var i=0; i<7; i++) {
alert(dias[i]);
}
```

while

Mientras que el bucle for se utiliza cuando sabemos cuántas veces queremos repetir un bloque de código, el bucle while nos permite seguir ejecutando un bloque de código mientras la condición a evaluar sea verdadera. Por lo tanto, aunque se puede utilizar con un contador (como un for), se suele utilizar cuando no sabemos cuántas veces tenemoss que repetir el bucle.

```
while (condición) {
  // bloque a ejecutar mientras la condición sea verdadera
}
```

La condición se evalúa antes de cada vuelta del bucle. Si la condición es verdadera, se ejecuta el bloque código del while. Llegado al final del bloque, la ejecución vuelve al while y se vuelve a evaluar la condición.

En caso de que la condición sea falsa, el bloque de código dentro del while no se ejecuta y el flujo del programa continúa en la línea después del while.

Ejemplo:

```
var contador = 0;
while (contador <= 10) {
  document.write('La variable contador es igual a: ' + contador+'<br>');
  contador++;
}
```

En primer lugar declaramos una variable que lleve la cuenta de cuantas veces se ha ejecutado el bucle. Con la expresión **contador++** , a cada "vuelta" que da el bucle, la variable contador se incrementa en 1 .

La condición evaluada por el while es que mientras que contador <= 10 el bucle siga ejecutándose. Debido al incremento del contador, cuando se llega a contador = 11 la expresión evaluada por el while se resuelve con un false (11 <= 10 resulta ser falso). En ese momento, el bucle while deja de ejecutarse y el flujo del programa continúa en la siguiente línea.

Imaginemos que nos olvidamos de añadir la expresión de incrementa el contador contador++ , ¿qué ocurriría? Crearíamos un bucle infinito. La variable contador, tras cada vuelta, permanecería siendo contador = 0 por lo que la condición evaluada por el while contador <= 10 sería siempre verdadera.

do while

Es una variación del bucle while.

En el **do while** la instrucción se ejecuta al menos una vez. Se evalúa la expresión y mientras sea falsa vuelve a empezar.

```
do {
  //Instrucciones
} while (expresion)
```

Ejemplo:

```
var num = 0
do{
  num = num + 1
} while (num < 5)
```

Resumen

En esta Unidad…

Trabajamos con vectores y cicles

En la próxima Unidad…

Trabajaremos con DOM y BOM