Universitat Oberta de Catalunya

Aula

M4.250 - Introducción a la programación en JavaScript I aula 1

Control de flujo: Bucles

Inicio:	Fin:	Dedicación:
20/11/19	**30/12/19**	**40 h**

Introducción

En este módulo veremos el capítulo Bucles e iteración .

"Los bucles ofrecen una manera rápida y sencilla de hacer algo repetitivamente".

Es la manera como empieza este capítulo y es realmente la forma más fácil y corta de definir un bucle. Una de las gracias de la programación es la facilidad que nos ofrece para realizar operaciones que requieren muchas repeticiones. Los bucles nos dan la solución perfecta para hacer esas operaciones.

Veremos fundamentalmente tres estructuras de bucle:

- for
- do ... while
- while

Y algunas instrucciones más que las complementan.

Los bucles no son sencillos, así que en este módulo añadimos ejercicios que complementarán a los que os propondremos en el foro o el tablón. No dejéis de hacerlos.

Sentencia for

```
for (let i = 1; i < 5; i++) {
  sentencias;
}
```

Éste es un ejemplo del uso del bucle for. Usamos una variable que nos servirá de contador o índice, para saber cuantas veces se ejecutan las instrucciones que hay dentro del bucle.

En este caso se repetirán 4 veces:

1. Cuando i valga 1
2. Cuando i valga 2
3. Cuando i valga 3
4. Cuando i valga 4

El funcionamiento de la instrucción for vendría a ser éste:

- La primera vez que se llega al for , se crea la variable y se le asigna el valor inicial. En el ejemplo es 1, pero puede ser cualquiera.
- Una vez se ha asignado el valor inicial a la variable, se comprueba la condición. Si el resultado de evaluar la condición es cierto, se ejecutan las instrucciones que hay dentro del bucle.
- Cuando se ejecuta el for por segunda vez (y siguientes) lo que se hace es incrementar la variable según lo indicado para, a continuación comprobar la condición. Si es cierta, se vuelve a entrar en el bucle. Si no, se sale.

Podéis probar la ejecución paso a paso de ese bucle en este enlace: https://goo.gl/11vYo4

Ejercicios

Los ejercicios con bucles no son sencillos, pueden ser difíciles de entender. Si tenéis dudas sobre los enunciados de estos ejercicios, preguntad en el foro. Y si tenéis dudas de como resolverlos, también.

1. Modifica el bucle que hemos presentado de ejemplo, para que escriba los números al revés, de 4 a 1.
2. Modifica el bucle que hemos presentado de ejemplo, para que escriba los números del 50 al 54.
3. Modifica el bucle que hemos presentado de ejemplo, para que escriba los números del 1 al 7 pero de 2 en 2. En la práctica esto se puede hacer de dos maneras:
    1. incrementando la variable del for en dos en vez de uno. ¿Sabrías hacerlo?
    2. poniendo un if dentro del bucle para que solo se escriban los números impares. ¿Sabrías hacerlo?
4. Reflexiona sobre las dos opciones anteriores. ¿Cuál es mejor? ¿Por qué?
5. Hacer una tabla de multiplicar usando los bucles es fácil... Haz la tabla de multiplicar del 4.
6. Un poco más complicado... ¿Cómo harías para escribir las tablas de multiplicar del 1 al 10?. Una pista, necesitarás dos bucles, uno dentro de otro.

Malas prácticas

A la hora de hacer un bucle con for , podemos hacer diversas variaciones:

- No definir una variable índice. Podemos usar una que ya exista.
- No inicializar la variable en el for , usar como valor inicial el valor que ya tuviese.
- No poner condición. Eso hará que el bucle no se acabe nunca.
- No poner incremento. El incremento se podría poner dentro del bucle.

Estas cuatro variaciones son prácticas no recomendables. Una de las características del bucle for es que nos permite rápidamente saber que se está haciendo. Si usamos alguna de estas variaciones se rompe esa característica de simplicidad del for , complicando su comprensión en caso de que estemos revisando el código por un error o por una modificación necesaria.

Como hemos comentado, podemos modificar el valor de la variable índice en cualquier momento dentro del bucle. Tampoco es recomendable hacer esto. Es más, conviene asegurarse de que no se modifica el valor del índice dentro del bucle. De nuevo, el motivo es que complica la comprensión del código.

Y entender el código es muy importante.

Antes de continuar: los Strings

Para poder hacer ejercicios y ejemplos que tengan una cierta gracia, vamos a introducir ahora los strings, o, traducido al castellano, las cadenas de caracteres.

Ya hemos visto que podemos guardar en una variable un texto:

```
MiTexto = "Esto es una frase.";
```

Pero lo más interesante es que podemos acceder a cualquier carácter de este texto de una manera muy fácil.

Por ejemplo, si queremos saber cuál es el primer carácter de este string simplemente hacemos:

```
console.log (MiTexto[0]);
```

Dentro de los corchetes ponemos la posición del carácter que queremos conocer, teniendo en cuenta que la primera posición es la posición 0. Por lo tanto, con esto que acabamos de hacer, el valor que se imprimiría en la consola sería "E".

¡Ojo! Esto nos sirve para consultar el contenido de una posición de la cadena. Pero no lo podemos utilizar para modificar su contenido (no funcionará MiTexto [0] = "H";).

Podemos también saber qué tamaño (cuántos caracteres) tiene esta cadena, utilizando la instrucción length de este modo:

```
console.log (MiTexto.length);
```

Esto nos escribirá en la consola el número 18, que es el número de caracteres que tiene la frase "Esto es una frase." Contando los espacios en blanco y el punto final.

Hay toda una serie de funciones que nos permiten trabajar con los strings y que puede encontrar en

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Text_formatting#Objetos_cadena

De momento, sin embargo, no avanzaremos más.

Sentencia do...while

Una vez visto, y entendido, el funcionamiento de la instrucción **for** , el resto de instrucciones iterativas son relativamente fáciles.

La instrucción **do ... while** tiene tres características que la diferencian con el **for** . La primera es que la comprobación de la condición se hace al final del bucle. La segunda es que no definimos una variable índice y la tercera es que no indicamos ninguna modificación de ninguna variable dentro de la instrucción. Complicado? No, ¡no lo es!

Esta es la sintaxis de la instrucción **do ... while** :

```
do {
   sentencias;
} while (condición);
```

Como la condición se comprueba al final del bucle, lo que encontramos es que las sentencias siempre se harán al menos una vez. Cuando se comprueba la condición, si es cierta, se vuelve a repetir el bucle.

Antes de ver ejemplos o ejercicios utilizando el **do ... while** , vamos a ver primero el funcionamiento de otra instrucción iterativa: el **while** .

La sentencia while

Esta instrucción, con un nombre bastante parecido a la anterior, hace algo que también se parece mucho. Vamos a ver su sintaxis:

```
while (condición) {
     sentencias;
}
```

Como se puede ver, la principal diferencia que existe entre **do ... while** y **while** es que la primera evalúa la condición al final y la segunda al principio. De nuevo, si la condición es verdad, se ejecutará el bucle. En caso contrario no se ejecutará.

Ejemplos do...while i while

Veremos ahora un par de ejemplos. Vemos **do ... while** y **while** conjuntamente para que quede más clara la diferencia.

Aprovecharemos ahora lo que hemos visto sobre los strings.

Supongamos que tenemos esta cadena de caracteres:

```
cadena = "Una casa muy bonita.";
```

Queremos hacer un programa que, para cualquier cadena de caracteres, nos diga cuantos espacios en blanco hay.

Lo haremos con un **while** :

```
let cadena = "Una casa muy bonita.";
let i = 0;
let contador = 0;
while (i
```

Puedes probar este código en https://goo.gl/nXrZ4p

Cambia la frase y comprueba que, efectivamente calcula el bien número de espacios.

¿Qué pasa si la frase que usamos es ésta: ""?

Pues que como no hay nada el tamaño será 0 y, por tanto, nunca entrará dentro del bucle. Tendria sentido aquí utilizar un **do ... while** ? Pues no.

Pensemos otro posible ejemplo.

Supongamos que tenemos una cadena de caracteres terminada en punto y queremos hacer un programa que nos genere una cadena inversa.

Vemos unos casos posibles:

```
cadena = "Una casa grande.";
// Devolverá ".ednarg asac anU"
cadena = "Coche."; // Devolverá ".ehcoC"
```

Como la cadena siempre estará terminada en punto, siempre tendrá al menos un carácter, por lo que siempre tendremos que entrar, al menos, una vez dentro del bucle. Hagámoslo pues con un **do ... while** .

```
let cadena = "Una casa grande.";
```

```
let y = cadena.length-1;
var cadenaInversa = "";
do {
    cadenaInversa = cadenaInversa + cadena [I--];
} While (i> = 0);
console.log (cadenaInversa);
```

Encontrarás este ejercicio resuelto en https://goo.gl/S6rEaX

Ejercicios

- Hacer el último ejercicio del apartado anterior, pero utilizando, primero la instrucción **while** y después con la instrucción **for** .
- Hacer un programa que cuente el número de "a" que hay en una cadena de caracteres.
- Hacer un programa que cuente el número de palabras que hay en una cadena terminada en punto, teniendo en cuenta que entre palabra y palabra hay siempre un espacio (y sólo un espacio).
- Hacer un programa que cuente el número de palabras que hay en una cadena terminada en punto, teniendo en cuenta que entre dos palabras puede haber uno o más espacios.

Break, continue y label

Estas tres instrucciones permiten saltarse algunas instrucciones dentro de un bucle.

- **break** da por finalizado el bucle, aunque se siga cumpliendo la condición. De hecho, al encontrar la instrucción **break** , automáticamente el programa sigue por la siguiente instrucción que hay después del bucle.
- **continue** hace que el bucle vuelva a su comienzo y vuelva a comprobar la condición. Si hay más instrucciones debajo, no las ejecuta.
- **label** permite marcar una parte del código de forma que cuando usamos alguna de las dos instrucciones anteriores, podamos salir del bucle correcto. Especialmente indicado cuando hay un bucle dentro de otro.

En el material de Mozilla hay algunos ejemplos que pueden resultar de utilidad para comprender estas instrucciones.

Nota importante : Si existen estas instrucciones es, indudablemente, por su utilidad. Pero no es conveniente usarlas cuando se está empezando a programar, pueden dar lugar a errores difíciles de detectar.

Recursos d'aprenentatge

Fuentes de información