1 Condiciones y bucles en JavaScript

En los lenguajes de programación, las instrucciones que te permiten controlar las decisiones y bucles de ejecución, se denominan "Estructuras de Control". Una estructura de control, dirige el flujo de ejecución a través de una secuencia de instrucciones, basadas en decisiones simples y en otros factores.

Una parte muy importante de una estructura de control es la ?condición?. Cada condición es una expresión que se evalúa a **true** o **false**.

JavaScript ofrece un total de cuatro instrucciones para procesar código de acuerdo a condiciones determinadas por el programador: **if**, **switch**, **for** y **while**.

1.1 Sumario

- 1 Instrucción if
- 2 Instrucción switch
- 3 Bucles
  - 3.1 while( )
  - 3.2 do...while( )
  - 3.3 for
  - 3.4 for ... of
  - 3.5 for ... in
- 4 Detener la ejecución de bucles y condicionales
  - 4.1 break
  - 4.2 continue

1.2 Instrucción if

La decisión más simple que podemos tomar en un programa, es la de seguir una rama determinada si una condición es **true**.

```
var mivariable = 9;
if(mivariable < 10) {
    alert("El número es menor que 10");
}
```

Como vimos, los operadores de comparación disponibles en JavaScript son: **=== == != > < >= <=**

Después de evaluar una condición, ésta devuelve un valor lógico verdadero o falso.

Y, para combinar condiciones y crear condiciones complejas podemos emplear los siguientes operadores lógicos: **!** (negación), **&&** (y) y **||** (o).

```
//Ejemplo con negación
var mivariable = 9;
if(!(mivariable < 10)) {
alert("El número es mayor que 10");
}

//Ejemplo con "AND"
var carnetConducir = true;
var edad = 19;
if(edad < 21 && carnetConducir == true) {
console.log("Andrea está autorizada");
}
```

JavaScript también es capaz de determinar una condición basándose en los valores de cualquier variable:

- Una variable con un número entero devolverá **false** si el valor es 0.
- Las variables con cadenas de caracteres vacías también devuelven **false**.

Si tenemos que ejecutar instrucciones para cada estado de la condición (verdadero o falso) lo haremos con la instrucción **if ... else if ... else**.

```
var time = 15
var saludo
if (time < 10) {
```

```
    saludo = "Buenos días";
} else if (time < 20) {
    saludo = "Buenas tardes";
} else {
    saludo = "Buenas noches";
}
console.log(saludo);
```

También podemos ver el uso del operador condicional **?:** que se puede utilizar como forma abreviada de una sentencia condicional del tipo **if...else**.

```
var ahora = new Date();
var saludo = "Buenas" + ((ahora.getHours() > 12) ? " tardes." : " días.");
console.log(saludo);
```

1.3 Instrucción switch

Si necesitamos comprobar múltiples condiciones podemos también utilizar la instrucción *switch*.

```
var mivariable = 8;
switch(mivariable) {
case 5:
alert("El número es cinco");
break;
case 8:
alert("El número es ocho");
break;
case 10:
alert("El número es diez");
break;
default:
alert("El numero es " + mivariable);
}
```

Observar la palabra clave **break** utilizada al final de cada cláusula **case** en el código. Esta instrucción hace que la ejecución salte al final de una instrucción **switch** o bucle. Si no existen instrucciones **break**, una instrucción **switch** inicia la ejecución de su bloque de código en la etiqueta case que coincide con el valor de su expresión y continúa la ejecución de instrucciones hasta que llega al final del bloque. Raras veces eso sería útil, por lo que no debemos olvidarnos poner siempre un **break** en cada cláusula **case** (a no ser que se sustituya por una instrucción **return** si se utiliza **switch** dentro de una función).

```
// Función que convierte un valor en una cadena distinta
//dependiendo del tipo del valor.
function convert(x) {
      switch(typeof(x)) {
        case 'number': //Convierte el número en un entero hexadecimal
        return x.toString(16);
        case 'string': //La devolvemos entre comillas
        return '"' + x + '"';
        case 'boolean': //Convertir en TRUE o FALSE en mayúsculas
        return x.toString().toUpperCase();
        default:        //Lo pasa a string en su forma habitual
        return x.toString();
      }
    }
```

1.4 Bucles

Los bucles son estructuras repetitivas, que se ejecutarán un número de veces fijado expresamente, o que dependerá de si se cumple una determinada condición.

1.4.1 while( )

Este tipo de bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle **for** (que luego veremos), ya que no incorpora en la misma línea la inicialización de las variables, su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración o repetición.

Sintaxis:

```
while (condición) {
    // Instrucciones a ejecutar dentro del bucle.
}
```

Ejemplo:

```
var i=0;
while (i <=10) {
    // Instrucciones a ejecutar dentro del bucle
    //hasta que i sea mayor que 10
    //y ya no se cumpla la condición.

    i++;
}
```

1.4.2 do...while( )

El tipo de bucle **do...while** es muy parecido al bucle **while( )** visto anteriormente. Se utiliza, generalmente, cuando no sabemos el número de veces que se habrá de ejecutar el bucle. La diferencia con el bucle **while( )** es que sabemos, seguro, que el bucle por lo menos se ejecutará una vez.

Sintaxis:

```
do {
    // Instrucciones a ejecutar dentro del bucle.
} while (condición);
```

Ejemplo:

```
//Función que recorre los elementos de un array
  function printArray(a) {
    if (a.length == 0) {
      console.log("Array vacío");
    }
    else {
      let i = 0;
      do {
        console.log(a[i] + "<br>");
      } while (++i < a.length);
    }
  }
  let miArray = [2, 4, 6, 7, 9];
  printArray(miArray);
```

1.4.3 for

La instrucción **for** proporciona una construcción de bucle que normalmente es más conveniente que la instrucción **while**. Como sabemos, la inicialización, prueba y actualización son tres manipulaciones cruciales en la variable de un bucle; esta instrucción **for** hace que estos tres pasos sean una parte explícita de la sintaxis del bucle. Así es más fácil saber lo que está haciendo un bucle **for** y se evitan errores como olvidarse de inicializar o incrementar la variable del bucle.

Sintaxis:

```
for(inicializar; prueba; incremento)
    instrucción;
```

Veamos un ejemplo:

```
//Bucle for que cuenta de 0 a 9.
for(let count = 0; count < 10; count++) {
    console.log(count + "<br>");
}
```

Así y todo, los bucles pueden llegar a ser mucho más complejos, con múltiples variables. Ejemplo:

```
let sum = 0;
```

```
for(let i = 0, j = 10; i < 10; i++, j--) {
    sum += i * j;
    console.log(i + " --> " + j);
    console.log(sum + "<br>");
}
```

1.4.4 for ... of

Recorre una lista de objetos... objeto a objeto. El **for...in** visto antes recorre las propiedades de los objetos, con **for...of** los objetos en sí.

Muy interesante el siguiente modo de iterar en los elementos de un Array, empleando la sentencia **for...of**.

```
const arrayNumeros = [1, 3, 5, 7];
for (let num of arrayNumeros) {
    console.log(`Número : ${num}`);
}
//Número : 1
//Número : 3
//Número : 5
//Número : 7
```

La instrucción **for** es útil cuando podemos determinar ciertos requisitos, como el valor inicial del bucle o el modo en que evolucionarán esos valores en cada ciclo. Cuando esta información es poco clara, podemos utilizar la instrucción **while**.

1.4.5 for ... in

La palabra clave **for** también se utiliza en la instrucción **for** / **in** que se utiliza para recorrer las propiedades de un objeto o, por ejemplo, los elementos de una matriz.

Podemos ver un ejemplo de funcionamiento con un objeto tipo "diccionario", donde recorremos los elementos guardados en él:

```
let dict = {uno:1, dos:2, tres:3};
for(let i in dict) {
    console.log(i + " --> " + dict[i]);
}
```

1.5 Detener la ejecución de bucles y condicionales

JavaScript ofrece varias instrucciones para detener la ejecución de bucles y condicionales.

1.5.1 break

Instrucción que interrumpe el bucle. Todas las instrucciones restantes y los ciclos pendientes se ignoran después de que se ejecuta esta instrucción.

```
//Mostramos números del 10 al 1, pero, se para al llegar a 5
let lista = [];
let numero = 10;
while(numero > 0) {
    lista.push(numero);
    numero -= 1;
    if(numero == 5) {
        break;
    }
}
console.log(lista);
```

1.5.2 continue

Instrucción que interrumpe el ciclo actual y avanza hacia el siguiente. El sistema ignora el resto de instrucciones del bucle después de que se ejecuta esta instrucción.

```
//Mostramos números del 10 al 1,
//menos los comprendidos entre el 4 y el 7
```

```
let lista = [];
let numero = 10;
while(numero > 0) {
    numero -= 1;
    if(numero > 4 && numero < 7) {
        continue;
    }
    lista.push(numero);
}
console.log(lista);
```

Volver