

JavaScript

CONTENIDO:

Asignación	2
Incremento y Decremento	3
Lógicos	4
Matemáticos	6
Relacionales	7



Unidad 4 - Estructuras de Control

Los programas que se pueden realizar utilizando solamente variables y operadores son una simple sucesión lineal de instrucciones básicas.

Sin embargo, no se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado y no muestren el mensaje en el resto de casos. Tampoco se puede repetir de forma eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las estructuras de control de flujo, que son instrucciones del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". También existen instrucciones del tipo "repite esto mientras se cumpla esta condición".

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición

2.– Estructura If

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {  
    instrucciones  
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.

Ejemplo:

```
var mostrarMensaje = true;  
  
if(mostrarMensaje) {  
    alert("Hola Mundo");  
}
```

En el ejemplo anterior, el mensaje sí que se muestra al usuario ya que la variable mostrarMensaje tiene un valor de true y por tanto, el programa entra dentro del bloque de instrucciones del if.

El ejemplo se podría reescribir también como:

```
var mostrarMensaje = true;  
  
if(mostrarMensaje == true) {  
    alert("Hola Mundo");  
}
```

En este caso, la condición es una comparación entre el valor de la variable mostrarMensaje y el valor true. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores == y =. Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores:

```
var mostrarMensaje = true;
// Se comparan los dos valores
if(mostrarMensaje == false) {
    ...
}
// Error - Se asigna el valor "false" a la variable
if(mostrarMensaje = false) {
    ...
}
```

La condición que controla el `if()` puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;
if(!mostrado) {
    alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores AND y OR permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
var usuarioPermiteMensajes = true;
if(!mostrado && usuarioPermiteMensajes) {
    alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de `mostrado` es `false`, el valor `!mostrado` sería `true`. Como la variable `usuarioPermiteMensajes` vale `true`, el resultado de `!mostrado && usuarioPermiteMensajes` sería igual a `true && true`, por lo que el resultado final de la condición del `if()` sería `true` y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del `if()`.

```
var numero = 5;
numero++;
alert(numero); // numero = 6
```

El resultado de ejecutar el script anterior es el mismo que cuando se utiliza el operador `++numero`, por lo que puede parecer que es equivalente indicar el operador `++` delante o detrás del identificador de la variable. Sin embargo, el siguiente ejemplo muestra sus diferencias:

```
var numero1 = 5;
```

3.— Estructura if... else

En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

```
if(condicion) {
  Instrucciones;
}
else {
  Instrucciones
}
```

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }.

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. Ejemplo:

```
var edad = 18;
if(edad >= 18) {
  alert("Eres mayor de edad");
}
else {
  alert("Todavía eres menor de edad");
}
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if() se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if() no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque else { }. En este caso, se mostraría el mensaje "Todavía eres menor de edad".

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";
```

3.2. Operador AND

La operación lógica AND obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo && y su resultado solamente es true si los dos operandos son true.

Estos son algunos ejemplos:

```
var valor1 = true;
```

```
var valor2 = false;
```

```
resultado = valor1 && valor2;
```

```
// resultado = false
```

```
valor1 = true;
```

```
valor2 = true;
```

```
resultado = valor1 && valor2;
```

```
// resultado = true
```

3.3. Operador OR

La operación lógica OR también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es true si alguno de los dos operandos es true:

Estos son algunos ejemplos:

```
var valor1 = true;
```

```
var valor2 = false;
```

```
resultado = valor1 || valor2; // resultado = true
```

```
valor1 = false;
```

```
valor2 = false;
```

```
resultado = valor1 || valor2; // resultado = false
```

4.— Estructura for

Las estructuras if y if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción. Por ejemplo, si se quiere mostrar un mensaje cinco veces, se podría pensar en utilizar el siguiente if:

```
var veces = 0;
if(veces < 4) {
  alert("Mensaje");
  veces++;
}
```

Se comprueba si la variable veces es menor que 4. Si se cumple, se entra dentro del if (), se muestra el mensaje y se incrementa el valor de la variable veces. Así se debería seguir ejecutando hasta mostrar el mensaje las cinco veces deseadas.

Sin embargo, el funcionamiento real del script anterior es muy diferente al deseado, ya que solamente se muestra una vez el mensaje por pantalla. La razón es que la ejecución de la estructura if() no se repite y la comprobación de la condición sólo se realiza una vez, independientemente de que dentro del if() se modifique el valor de la variable utilizada en la condición.

La estructura for permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de if():

```
for(inicializacion; condicion; actualizacion) {
  Instrucciones;
}
```

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".

La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.

La "condición" es el único elemento que decide si continua o se detiene la repetición.

La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Las estructuras if y if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción.

4.— Estructura for ... in

Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura for...in adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

```
for(indice in array) {
```

```
  Instrucciones
```

```
}
```

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

```
var días = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

```
for(i in días) {
```

```
  alert(días[i]);
```

```
}
```

La variable que se indica como índice es la que se puede utilizar dentro del bucle for...in para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable i vale 0 y en la última vale 6.

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle for simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle for simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.



Este material forma parte del currículum del módulo **Implantación de Aplicaciones WEB** correspondiente al ciclo formativo de grado superior **“Administración de sistemas informáticos en red”**.

La información contenida en el magazine ha sido extraída de la Web y, en parte, de la experiencia profesional acumulada durante mis años de docencia.

Por tanto, es de uso público para tareas de docencia y aprendizaje.