Programación con Javascript

Unidad 5: Condicionales y otras estructuras

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

Indice

Unidad 5: Condicionales y otras estructuras

Objetivos

Que el alumno logre:

- Manejar datos a través de estructuras condicionales

Condicionales simples y anidados

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.

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) {
...
}
```

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 && usuario PermiteMensajes 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().

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) {
...
} 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 = "";
if(nombre == "") {
alert("Aún no nos has dicho tu nombre");
} else {
alert("Hemos guardado tu nombre");
}
```

La condición del if() anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores).

En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if(). En otro caso, se muestra el mensaje definido en el bloque else { }.

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {
alert("Todavía eres muy pequeño");
} else if(edad < 19) {
alert("Eres un adolescente");
} else if(edad < 35) {
alert("Aun sigues siendo joven");
} else { alert("Piensa en cuidarte un poco más");
}
```

No es obligatorio que la combinación de estructuras if...else acabe con la instrucción else, ya que puede terminar con una instrucción de tipo else if().

Switch

Una sentencia switch permite a un programa evaluar una expresión e intentar igualar el valor de dicha expresión a una etiqueta de caso (case). Si se encuentra una coincidencia, el programa ejecuta la sentencia asociada. Una sentencia switch se describe como se muestra a continuación:

```
switch (expresión) {
  case etiqueta_1:
    sentencias_1
    [break;]
  case etiqueta_2:
    sentencias_2
    [break;]
    ...
  default:
    sentencias_por_defecto
    [break;]
}
```

El programa primero busca una claúsula case con una etiqueta que coincida con el valor de la expresión y, entonces, transfiere el control a esa cláusula, ejecutando las sentencias asociadas a ella. Si no se encuentran etiquetas coincidentes, el programa busca la cláusula opcional default y, si se encuentra, transfiere el control a esa cláusula, ejecutando las sentencias asociadas. Si no se encuentra la cláusula default, el programa continúa su ejecución por la siguiente sentencia al final del switch. Por convención, la cláusula por defecto es la última cláusula, aunque no es necesario que sea así.

La sentencia opcional break asociada con cada cláusula case asegura que el programa finaliza la sentencia switch una vez que la sentencia asociada a la etiqueta coincidente es ejecutada y continúa la ejecución por las sentencias siguientes a la sentencia switch. Si se omite la sentencia break, el programa continúa su ejecución por la siguiente sentencia que haya en la sentencia switch.

Ejemplo:

En el siguiente ejemplo, si tipoFruta se evalúa como "Plátanos", el programa iguala el valor con el caso "Plátanos" y ejecuta las sentencias asociadas. Cuando se encuentra la sentencia break, el programa termina el switch y ejecuta las sentencias que le siguen. Si la sentencia break fuese omitida, la sentencia para el caso "Cerezas" también sería ejecutada.

```
switch (tipoFruta) {
  case "Naranjas":
    console.log("Naranjas cuestan 0,59€ el kilo.");
    break;
  case "Manzanas":
    console.log("Manzanas cuestan 0,32€ el kilo.");
    break;
  case "Plátanos":
    console.log("Plátanos cuestan 0,48€ el kilo.");
    break;
  case "Cerezas":
    console.log("Cerezas cuestan 3,00€ el kilo.");
    break;
  case "Mangos":
    console.log("Mangos cuestan 0,56€ el kilo.");
    break;
  case "Papayas":
    console.log("Mangos y papayas cuestan 2,79€ el kilo.");
    break;
  default:
   console.log("Disculpa, no tenemos el tipo de fruta " + tipoFruta + ".");
}
console.log("¿Te gustaría tomar algo?");
```

Atributos HTML y propiedades CSS en DOM

Los métodos presentados anteriormente para el acceso a los atributos de los elementos, son genéricos de XML. La versión de DOM específica para HTML incluye algunas propiedades y métodos aún más directos y sencillos para el acceso a los atributos de los elementos HTML y a sus propiedades CSS.

La principal ventaja del DOM para HTML es que todos los atributos de todos los elementos HTML se transforman en propiedades de los nodos. De esta forma, es posible acceder de forma directa a cualquier atributo de HTML. Si se considera el siguiente elemento <img> de HTML con sus tres atributos:

```
<img id="logo" src="logo.gif" border="0" />
Empleando los métodos tradicionales de DOM, se puede acceder y manipular cada atributo:
var laImagen = document.getElementById("logo");

// acceder a los atributos
var archivo = laImagen.getAttribute("src");
var borde = laImagen.getAttribute("border");

// modificar los atributos
laImagen.setAttribute("src", "nuevo_logo.gif");
laImagen.setAttribute("border", "1");
```

La ventaja de la especificación de DOM para HTML es que permite acceder y modificar todos los atributos de los elementos de forma directa:

```
var laImagen = document.getElementById("logo");

// acceder a los atributos
var archivo = laImagen.src;
var borde = laImagen.border;

// modificar los atributos
laImagen.src = "nuevo_logo.gif";
laImagen.border = "1";
```

Las ventajas de utilizar esta forma de acceder y modificar los atributos de los elementos es que el código resultante es más sencillo y conciso. Por otra parte, algunas versiones de Internet Explorer

no implementan correctamente el método setAttribute(), lo que provoca que, en ocasiones, los cambios realizados no se reflejan en la página HTML.

La única excepción que existe en esta forma de obtener el valor de los atributos HTML es el atributo class. Como la palabra class está reservada por JavaScript para su uso futuro, no es posible utilizarla para acceder al valor del atributo class de HTML. La solución consiste en acceder a ese atributo mediante el nombre alternativo className:

```
<p id="parrafo" class="normal">...</p>
var parrafo = document.getElementById("parrafo");
alert(parrafo.class);     // muestra "undefined"
alert(parrafo.className); // muestra "normal"
```

El acceso a las propiedades CSS no es tan directo y sencillo como el acceso a los atributos HTML. En primer lugar, los estilos CSS se pueden aplicar de varias formas diferentes sobre un mismo elemento HTML. Si se establecen las propiedades CSS mediante el atributo style de HTML:

```
<p id="parrafo" style="color: #C00">...</p>
```

El acceso a las propiedades CSS establecidas mediante el atributo style se realiza a través de la propiedad style del nodo que representa a ese elemento:

```
var parrafo = document.getElementById("parrafo");
var color = parrafo.style.color;
```

Las ventajas de utilizar esta forma de acceder y modificar los atributos de los elementos es que el código resultante es más sencillo y conciso. Por otra parte, algunas versiones de Internet Explorer no implementan correctamente el método setAttribute(), lo que provoca que, en ocasiones, los cambios realizados no se reflejan en la página HTML.

La única excepción que existe en esta forma de obtener el valor de los atributos HTML es el atributo class. Como la palabra class está reservada por JavaScript para su uso futuro, no es posible utilizarla para acceder al valor del atributo class de HTML. La solución consiste en acceder a ese atributo mediante el nombre alternativo className:

```
<p id="parrafo" class="normal">...</p>
var parrafo = document.getElementById("parrafo");
alert(parrafo.class);     // muestra "undefined"
alert(parrafo.className); // muestra "normal"
```

El acceso a las propiedades CSS no es tan directo y sencillo como el acceso a los atributos HTML. En primer lugar, los estilos CSS se pueden aplicar de varias formas diferentes sobre un mismo elemento HTML. Si se establecen las propiedades CSS mediante el atributo style de HTML:

```
<p id="parrafo" style="color: #C00">...</p>
```

El acceso a las propiedades CSS establecidas mediante el atributo style se realiza a través de la propiedad style del nodo que representa a ese elemento:

```
var parrafo = document.getElementById("parrafo");
var color = parrafo.style.color;
```

Para acceder al valor de una propiedad CSS, se obtiene la referencia del nodo, se accede a su propiedad style y a continuación se indica el nombre de la propiedad CSS cuyo valor se quiere obtener.

En el caso de las propiedades CSS con nombre compuesto (font-weight, border-top-style, list-style-type, etc.), para acceder a su valor es necesario modificar su nombre original eliminando los guiones medios y escribiendo en mayúsculas la primera letra de cada palabra que no sea la primera:

```
var parrafo = document.getElementById("parrafo");
var negrita = parrafo.style.fontWeight;
```

El nombre original de la propiedad CSS es font-weight. Para obtener su valor mediante JavaScript, se elimina el guión medio (fontweight) y se pasa a mayúsculas la primera letra de cada palabra que no sea la primera (fontWeight).

Si el nombre de la propiedad CSS está formado por tres palabras, se realiza la misma transformación. De esta forma, la propiedad border-top-style se accede en DOM mediante el nombre borderTopStyle.

Además de obtener el valor de las propiedades CSS, también es posible modificar su valor mediante JavaScript. Una vez obtenida la referencia del nodo, se puede modificar el valor de cualquier propiedad CSS accediendo a ella mediante la propiedad style:

```
<p id="parrafo">...</p>

var parrafo = document.getElementById("parrafo");
```

```
parrafo.style.margin = "10px";   // añade un margen de 10px al párrafo
parrafo.style.color = "#CCC";    // modifica el color de la letra del párrafo
```

Todos los ejemplos anteriores hacen uso de la propiedad style para acceder o establecer el valor de las propiedades CSS de los elementos. Sin embargo, esta propiedad sólo permite acceder al valor de las propiedades CSS establecidas directamente sobre el elemento HTML. En otras palabras, la propiedad styledel nodo sólo contiene el valor de las propiedades CSS establecidas mediante el atributo style de HTML.

Por otra parte, los estilos CSS normalmente se aplican mediante reglas CSS incluidas en archivos externos. Si se utiliza la propiedad style de DOM para acceder al valor de una propiedad CSS establecida mediante una regla externa, el navegador no obtiene el valor correcto:

```
// Código HTML
<p id="parrafo">...</p>

// Regla CSS
#parrafo { color: #008000; }

// Código JavaScript
var parrafo = document.getElementById("parrafo");
var color = parrafo.style.color;  // color no almacena ningún valor
```

Resumen

En esta Unidad…

Trabajamos con Condicionales y otras estructuras

En la próxima Unidad…

Trabajaremos con funciones y eventos