Programador Web – Nivel 1

Unidad 7: Introducción a Javascript (parte 2)

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

Indice

Unidad 7: Introducción a Javascript

Objetivos

Que el alumno logre:

- Aprender los conceptos básicos del lenguaje.

Introducción a la programación

Antes de comenzar a desarrollar programas y utilidades con JavaScript, es necesario conocer los elementos básicos con los que se construyen las aplicaciones.

Variables

Una variable es un elemento que se emplea para almacenar y hacer referencia a otro valor. Gracias a las variables es posible crear "programas genéricos", es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

De la misma forma que si en Matemáticas no existieran las variables no se podrían definir las ecuaciones y fórmulas, en programación no se podrían hacer programas realmente útiles sin las variables.

Sin el uso de datos variables, un programa que suma dos números podría definirse como:

resultado = 4 + 2

El programa anterior sólo sirve para el caso en el que el primer número de la suma sea el 4 y el segundo número sea el 2. En cualquier otro caso, el programa obtiene un resultado incorrecto.

Sin embargo, el programa se puede rehacer de la siguiente manera utilizando variables para almacenar y referirse a cada número:

primer_numero = 4
segundo_numero = 2
resultado = primer_numero + segundo_numero

Los elementos primer_numero y segundo_numero son variables que almacenan los valores que utiliza el programa.

El resultado se calcula siempre en función del valor almacenado por las variables, por lo que este programa funciona correctamente para cualquier par de números indicado. Si se modifica el valor de las variables primer_numero y segundo_numero, el programa sigue funcionando correctamente.

Las variables en JavaScript se crean mediante la palabra reservada **var**. De esta forma, el ejemplo anterior se puede realizar en JavaScript de la siguiente manera:

```
var primer_numero = 4;
var segundo_numero = 2;
var resultado = primer_numero + segundo_numero;
```

El nombre de una variable también se conoce como **identificador** y debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y el símbolo _ (guión bajo).
- El primer carácter no puede ser un número.

La palabra reservada **var** solamente se debe indicar al definir por primera vez la variable, lo que se denomina **declarar una variable**. Cuando se utilizan las variables en el resto de instrucciones del script, solamente es necesario indicar su nombre. En otras palabras, en el ejemplo anterior sería un error indicar lo siguiente:

```
var primer_numero = 4;
var segundo_numero = 2;
var resultado = var primer_numero + var segundo_numero;
```

Si cuando se declara una variable se le asigna también un valor, se dice que la variable ha sido **inicializada**.

En JavaScript no es obligatorio inicializar las variables, ya que se pueden declarar por una parte y asignarles un valor posteriormente. Por tanto, el ejemplo anterior se puede rehacer de la siguiente manera:

```
var primer_numero;
var segundo_numero;
var resultado;

primer_numero = 4;
segundo_numero = 2;
```

```
resultado = primer_numero + segundo_numero;
```

Tipos de variables

Cuando declaramos una variable en Javascript no es necesario determinar el tipo de dato a almacenar, sin embargo, de acuerdo al contenido, podemos identificar distintos tipos de variables:

Numéricas

Se utilizan para almacenar valores numéricos enteros (llamados integer en inglés) o decimales (llamados float en inglés).

En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter . (punto) en vez de , (coma) para separar la parte entera y la parte decimal:

```
var descuento = 65;      // variable tipo entero
var total = 652.53;      // variable tipo decimal
```

Cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto.

Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
```

En ocasiones, el texto que se almacena en las variables no es tan sencillo. Si por ejemplo el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
/* El contenido de texto1 tiene comillas simples, por lo que
   se encierra con comillas dobles */
var texto1 = "Una frase con 'comillas simples' dentro";

/* El contenido de texto2 tiene comillas dobles, por lo que
   se encierra con comillas simples */
var texto2 = 'Una frase con "comillas dobles" dentro';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.) Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres.

A continuación se muestra la tabla de conversión que se debe utilizar:

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

De esta forma, el ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

```
var texto1 = 'Una frase con \'comillas simples\' dentro';
var texto2 = "Una frase con \"comillas dobles\" dentro";
```

Este mecanismo de JavaScript se denomina "**mecanismo de escape**" de los caracteres problemáticos, y es habitual referirse a que los caracteres han sido "**escapados**".

Booleanos

Las variables de tipo boolean o booleano también se conocen con el nombre de variables de tipo lógico.

Una variable de tipo boolean almacena un tipo especial de valor que solamente puede tomar dos valores: ***true*** (verdadero) o ***false*** (falso). No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

Los únicos valores que pueden almacenar estas variables son *true* y *false*, por lo que no pueden utilizarse los valores verdadero y falso. Ejemplo:

```
var clienteRegistrado = false;
var ivaIncluido = true;
```

Arrays

También podemos trabajar con arrays, también denominados vectores, matrices o arreglos.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:

```
var dia1 = "Lunes";
var dia2 = "Martes";
var dia3 = "Miércoles";
var dia4 = "Jueves";
var dia5 = "Viernes";
var dia6= "Sábado";
var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array. El ejemplo anterior se puede rehacer de la siguiente forma:

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

Ahora, una única variable llamada **dias** almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres [corchetes] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Una vez definido un array, es muy sencillo acceder a cada uno de sus elementos. Cada elemento se accede indicando su posición dentro del array.

La única complicación, que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el 0 y no en el 1:

```
var diaSeleccionado = dias[0];    // diaSeleccionado = "Lunes"
var otroDia = dias[5];            // otroDia = "Sábado"
```

Operadores

Las variables por sí solas son de poca utilidad.

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

Asignación

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable.

El símbolo utilizado es = (no confundir con el operador == que se verá más adelante y sirve para comparar):

```
var numero1 = 3;
```

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc:

```
var numero1 = 3;
var numero2 = 4;
```

Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

Ejemplo:

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

El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad. Por tanto, el anterior ejemplo es equivalente a:

```
var numero = 5;
numero = numero + 1;
alert(numero);  // numero = 6
```

De forma equivalente, el operador decremento (indicado como un prefijo -- en el nombre de la variable) se utiliza para decrementar el valor de la variable:

```
var numero = 5;
--numero;
alert(numero);  // numero = 4
```

El anterior ejemplo es equivalente a:

```
var numero = 5;
numero = numero - 1;
alert(numero);  // numero = 4
```

Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o booleano.

Negación

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable:

```
var visible = true;
alert(!visible);  // Muestra "false" y no "true"
```

La negación lógica se obtiene prefijando el símbolo ! al identificador de la variable.

Si la variable original es de tipo booleano, es muy sencillo obtener su negación. Sin embargo, ¿qué sucede cuando la variable es un número o una cadena de texto? Para obtener la negación en este tipo de variables, se realiza en primer lugar su conversión a un valor booleano:

- ***Si la variable contiene un número***, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).
- ***Si la variable contiene una cadena de texto***, se transforma en false si la cadena es vacía ("") y en true en cualquier otro caso.

```
var cantidad = 0;
vacio = !cantidad;  // vacio = true

cantidad = 2;
vacio = !cantidad;  // vacio = false

var mensaje = "";
mensajeVacio = !mensaje;  // mensajeVacio = true

mensaje = "Bienvenido";
mensajeVacio = !mensaje;  // mensajeVacio = false
```

AND

La operación lógica **AND** (y) 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 (verdaderos):

```
var valor1 = true;
var valor2 = false;
```

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

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

OR

La operación lógica **OR** (o) 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:

Matemáticos

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*) y división (/). Ejemplo:

```
var numero1 = 10;
var numero2 = 5;


resultado = numero1 / numero2;  // resultado = 2
resultado = 3 + numero1;        // resultado = 13
resultado = numero2 – 4;        // resultado = 1
resultado = numero1 * numero 2; // resultado = 50
```

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones.

Se trata del operador "módulo", que calcula el resto de la división entera de dos números. Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2. El resto de esa división es 0, por lo que módulo de 10 y 5 es igual a 0.

Sin embargo, si se divide 9 y 5, la división no es exacta, el resultado es 1 y el resto 4, por lo que módulo de 9 y 5 es igual a 4.

El operador módulo en JavaScript se indica mediante el símbolo %, que no debe confundirse con el cálculo del porcentaje:

```
var numero1 = 10;
var numero2 = 5;
resultado = numero1 % numero2; // resultado = 0

numero1 = 9;
numero2 = 5;
resultado = numero1 % numero2; // resultado = 4
```

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
var numero1 = 5;
numero1 += 3;  // numero1 = numero1 + 3 = 8
numero1 -= 1;  // numero1 = numero1 - 1 = 4
numero1 *= 2;  // numero1 = numero1 * 2 = 10
numero1 /= 5;  // numero1 = numero1 / 5 = 1
numero1 %= 4;  // numero1 = numero1 % 4 = 1
```

Relacionales

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja, como se verá en el siguiente capítulo de programación avanzada. El resultado de todos estos operadores siempre es un valor booleano:

```
var numero1 = 3;
var numero2 = 5;
resultado = numero1 > numero2; // resultado = false
resultado = numero1 < numero2; // resultado = true
```

```
numero1 = 5;
numero2 = 5;
resultado = numero1 >= numero2; // resultado = true
resultado = numero1 <= numero2; // resultado = true
resultado = numero1 == numero2; // resultado = true
resultado = numero1 != numero2; // resultado = false
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// El operador "=" asigna valores

var numero1 = 5;
resultado = numero1 = 3;  // numero1 = 3 y resultado = 3


// El operador "==" compara variables
var numero1 = 5;
resultado = numero1 == 3; // numero1 = 5 y resultado = false
```

Los operadores relacionales también se pueden utilizar con variables de tipo cadena de texto:

```
var texto1 = "hola";
var texto2 = "hola";
var texto3 = "adios";

resultado = texto1 == texto3; // resultado = false
resultado = texto1 != texto2; // resultado = false
resultado = texto3 >= texto2; // resultado = false
```

Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) siguen un razonamiento no intuitivo: se compara letra a letra comenzando desde la izquierda hasta que se

encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (a es menor que b, b es menor que c, A es menor que a, etc.)

Resumen

En esta Unidad…

En la presente unidad desarrollamos los conceptos necesarios comenzar a trabajar con Javascript.

En la próxima Unidad…

En la próxima unidad vamos a trabajar con el modelo de objetos de Javascript