

CONTENIDO:

Variables	2
Típos de Variables	3
Ámbito de una variable	5

JavaScript



Unidad 2 - Variables en Javascript

Las variables en los lenguajes de programación siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas. 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.

Si no existieran variables, un programa que suma dos números podría escribirse como:

```
resultado = 3 + 1
```

El programa anterior es tan poco útil que sólo sirve para el caso en el que el primer número de la suma sea el 3 y el segundo número sea el 1. 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:

```
numero_1 = 3
```

```
numero_2 = 1
```

```
resultado = numero_1 + numero_2
```

Los elementos `numero_1` y `numero_2` 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 `numero_1` y `numero_2`, el programa sigue funcionando correctamente.

1.— Variables

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 numero_1 = 3;
var numero_2 = 1;
var resultado = numero_1 + numero_2;
```

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 numero_1 = 3;
var numero_2 = 1;
var resultado = var numero_1 + var numero_2;
```

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 numero_1;
var numero_2;
numero_1 = 3;
numero_2 = 1;
var resultado = numero_1 + numero_2;
```

Una de las características más sorprendentes de JavaScript para los programadores habituados a otros lenguajes de programación es que tampoco es necesario declarar las variables. En otras palabras, se pueden

utilizar variables que no se han definido anteriormente mediante la palabra reservada `var`. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

```
var numero_1 = 3;
var numero_2 = 1;
resultado = numero_1 + numero_2;
```

La variable `resultado` no está declarada, por lo que JavaScript crea una variable global (más adelante se verán las diferencias entre variables locales y globales) y le asigna el valor correspondiente. De la misma forma, también sería correcto el siguiente código:

```
numero_1 = 3;
numero_2 = 1;
resultado = numero_1 + numero_2;
```

En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar.

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

Sólo puede estar formado por letras, números y los símbolos `$` (dólar) y `_` (guión bajo).

El primer carácter no puede ser un número. Por tanto, las siguientes variables tienen nombres correctos:

```
var $numero1;
var _$letra;
var $$$otroNumero;
var $_a__$4;
```

Sin embargo, las siguientes variables tienen identificadores incorrectos:

```
var 1numero; // Empieza por un número
var numero;1_123; // Contiene un carácter "1"
```

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

2.— Tipos de Variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada `var`), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

1. 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 iva = 16;    // variable tipo entero
var total = 234.65; // variable tipo decimal
```

2. 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	<code>\n</code>
Un tabulador	<code>\t</code>
Una comilla simple	<code>\'</code>
Una comilla doble	<code>\"</code>
Una barra inclinada	<code>\\</code>

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".

3. Arrays

En ocasiones, a los arrays se les llama vectores, matrices e incluso arreglos. No obstante, el término array es el más utilizado y es una palabra comúnmente aceptada en el entorno de la programación.

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";
```

Aunque el código anterior no es incorrecto, sí que 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 días almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres [y] 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"
```

En el ejemplo anterior, la primera instrucción quiere obtener el primer elemento del array. Para ello, se indica el nombre del

array y entre corchetes la posición del elemento dentro del array. Como se ha comentado, las posiciones se empiezan a contar en el 0, por lo que el primer elemento ocupa la posición 0 y se accede a él mediante dias[0].

El valor dias[5] hace referencia al elemento que ocupa la sexta posición dentro del array días. Como las posiciones empiezan a contarse en 0, la posición 5 hace referencia al sexto elemento, en este caso, el valor Sábado.

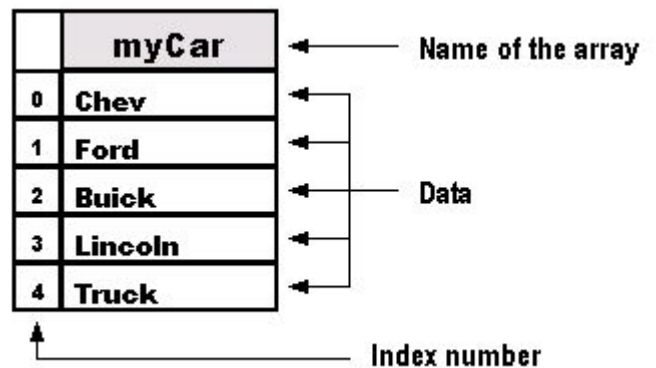
4. Booleanos

Las variables de tipo boolean o booleano también se conocen con el nombre de variables de tipo lógico. Aunque para entender realmente su utilidad se debe estudiar la programación avanzada con JavaScript del siguiente capítulo, su funcionamiento básico es muy sencillo.

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. A continuación se muestra un par de variables de tipo booleano:

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



3.— Ámbito de una variable

El ámbito de una variable (llamado "scope" en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {  
    var mensaje = "Mensaje de prueba";  
}  
  
creaMensaje();  
alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada `creaMensaje` que crea una variable llamada `mensaje`. A continuación, se ejecuta la función mediante la llamada `creaMensaje()`; y seguidamente, se muestra mediante la función `alert()` el valor de una variable llamada `mensaje`.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable `mensaje` se ha definido dentro de la función `creaMensaje()` y por tanto, es una variable local que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable `mensaje`. De esta forma, para mostrar el mensaje en el código anterior, la función `alert()` debe llamarse desde dentro de la función `creaMensaje()`:

```
function creaMensaje() {  
    var mensaje = "Mensaje de prueba";  
    alert(mensaje);  
}  
  
creaMensaje();
```

Además de variables locales, también existe el concepto de variable global, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = "Mensaje de prueba";
```

```
function muestraMensaje() {  
    alert(mensaje);  
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función `muestraMensaje()` se quiere hacer uso de una variable llamada `mensaje` y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable `mensaje`.

El motivo es que en el código JavaScript anterior, la variable `mensaje` se ha definido fuera de cualquier función. Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada `mensaje`, la variable global creada anteriormente permite que la instrucción `alert()` dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada `var` o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante `var` se consideran locales y las variables que no se han declarado mediante `var`, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada `var`, para que se transforme en una variable global:

```
function creaMensaje() {
  mensaje = "Mensaje de prueba";
}
```

```
creaMensaje();
alert(mensaje);
```

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";

function muestraMensaje() {
  var mensaje = "gana la de dentro";
  alert(mensaje);
}
```

```
alert(mensaje);
muestraMensaje();
alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:

```
gana la de fuera
gana la de dentro
gana la de fuera
```

Dentro de la función, la variable local llamada mensaje tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe? En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

```
var mensaje = "gana la de fuera";

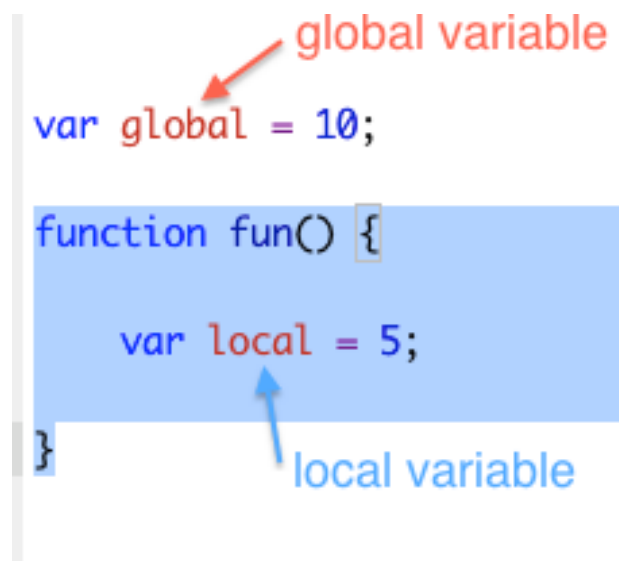
function muestraMensaje() {
  mensaje = "gana la de dentro";
```

```
  alert(mensaje);
}
alert(mensaje);
muestraMensaje();
alert(mensaje);
```

En este caso, los mensajes mostrados son:

```
gana la de fuera
gana la de dentro
gana la de dentro
```

La recomendación general es definir como variables locales todas las variables que sean de uso exclusivo para realizar las tareas encargadas a cada función. Las variables globales se utilizan para compartir variables entre funciones de forma sencilla.





Este material forma parte del currículo 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.