JAVASCRIPT : VARIABLES

1. 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.

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.

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** 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 ";"
```

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.)

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
```

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	\\
Un Retorno de carro	\r
Un backspace	\b

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".

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";
var dia2 = "Martes";
...
var dia7 = "Domingo";
```

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 dias 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 el mediante dias[0].

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

Array asociativo

Un array asociativo es aquel en el que **cada elemento no está asociado a su posición numérica dentro del array, sino que está asociado a otro valor específico**. • Los **valores** de los **arrays normales se asocian** a **índices** que siempre son **numéricos**.

- Los **valores** de los **arrays asociativos se asocian** a **claves** que siempre son **cadenas de texto**.

La forma tradicional de **definir** los **arrays asociativos** es mediante la clase Array:

```
var elArray = new Array();
elArray['primero'] = 1;
elArray['segundo'] = 2;

alert(elArray['primero']);
alert(elArray[0]);
```

El primer alert() muestra el valor 1 correspondiente al valor asociado con la clave primero. El **segundo alert()** muestra undefined, ya que como **no se trata de un array normal**, sus elementos **no se pueden acceder mediante su posición numérica**.

Afortunadamente, existen **métodos alternativos** abreviados para **crear array asociativos**. El ejemplo anterior se puede rehacer de la siguiente forma:

```
var elArray = new Array();

elArray.primero = 1;
elArray.segundo = 2;

alert(elArray['primero']);
alert(elArray.primero);
alert(elArray[0]);
```

El método seguido en el ejemplo anterior para crear el array asociativo se denomina "**notación de puntos**". Para acceder y/o establecer cada valor, se indica el **nombre del array seguido de un punto** y seguido del **nombre de cada clave**. De forma genérica, la notación de puntos tiene el siguiente formato:

```
nombreArray.nombreClave = valor;
```

Para **acceder** a un **determinado valor**, también se puede utilizar la **notación de puntos** en vez de la tradicional notación de los arrays, de forma que las dos instrucciones siguientes son **equivalentes**:

```
elArray['primero'];
elArray.primero;
```

Booleanos

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

Objetos

Al igual que sucede con otros lenguajes de programación, los objetos se emplean en JavaScript para **organizar** el **código fuente** de una forma más clara y para **encapsular métodos y propiedades comunes**. La forma más sencilla de **crear un objeto** es mediante la **palabra reservada new** seguida del **nombre de la clase** que se quiere instanciar:

```
var elObjeto = new Object();
var laCadena = new String();
```

El objeto laCadena creado mediante el **objeto nativo String** permite almacenar **una cadena de texto** y aprovechar todas las **herramientas y utilidades que proporciona JavaScript** para su manejo. Por otra parte, **la variable elObjeto** almacena un **objeto genérico de JavaScript**, al que se pueden **añadir propiedades y métodos propios** para definir su comportamiento.

Técnicamente, un **objeto** de **JavaScript** es un **array asociativo formado por las propiedades y los métodos del objeto**. Así, la **forma más directa para definir las propiedades y métodos** de un objeto es mediante la ***notación de puntos*** de los arrays asociativos.

Propiedades

Como los objetos son en realidad arrays asociativos que almacenan sus propiedades y métodos, la forma más directa para definir esas propiedades y métodos es la **notación de puntos**:

```
elObjeto.id = "10";
elObjeto.nombre = "Objeto de prueba";
```

Al contrario de lo que sucede en otros lenguajes orientados a objetos, como por ejemplo Java, **para asignar el valor de una propiedad no es necesario que la clase tenga definida previamente esa propiedad**.

También es posible utilizar la **notación tradicional de los arrays** para definir el valor de las propiedades:

```
elObjeto['id'] = "10";
elObjeto['nombre'] = "Objeto de prueba";
```

Métodos

Además de las propiedades, los métodos de los objetos también se pueden definir mediante la **notación de puntos**:

```
elObjeto.muestraId = function() {
 alert("El ID del objeto es " + this.id);
}
```

Los métodos se definen asignando funciones al objeto. Si la función **no está definida previamente, es posible crear una función anónima** para asignarla al nuevo método del objeto, tal y como muestra el ejemplo anterior.

Además de las funciones anónimas, también es posible **asignar a los métodos de un objeto funciones definidas con anterioridad**:

```
function obtieneId() {
 return this.id;
}

elObjeto.obtieneId = obtieneId();
```

Otros tipos

El operador typeof

Se puede utilizar el operador JavaScript typeof para averiguar el tipo de una variable, por ejemplo:

```
typeof "John" // Devuelve string
typeof 3.14 // Devuelve number
typeof false // Devuelve boolean
typeof [1,2,3,4] // Devuelve object
typeof {nombre:'John', edad:34} // Devuelve object
```

Undefined

En JavaScript, una variable sin un valor tiene el valor undefined. El typeof es también undefined. Ejemplo:

```
var persona; // Valor no definido, el tipo es undefined
```

Cualquier variable se puede vaciar, estableciendo su valor como undefined. El tipo será también undefined. Por ejemplo:

```
persona = undefined; // El valor es undefined, el tipo es undefined
```

Valores vacíos (Empty)

Un valor vacío no tiene nada que ver con un valor undefined. Una cadena vacía tiene tanto valor como tipo, por ejemplo:

```
var car = ""; // El valor es "", el typeof es string
```

Null

En JavaScript null es "nada". Se supone que es algo que no existe. Desafortunadamente, en JavaScript, **el tipo de datos de null es un objeto**. Se puede vaciar un objeto estableciéndolo a null.

```
var persona = null; // El valor es null, el tipo todavía es un objeto
```

También se puede vaciar un objeto estableciéndolo como undefined:

```
var persona = undefined; // El valor es undefined, el tipo es

undefined
```

Diferencia entre Undefined y Null

```
typeof undefined // undefined
typeof null // object

null === undefined // false
null == undefined // true
```