VARIABLES Y TIPOS DE DATOS

Valores

JavaScript no tiene un conjunto extenso de tipos de datos, ni hace chequeo de los mismos. Lo anterior significa que para JavaScript no existen diferencias entre un entero y un real. El tipo de dato se define de manera dinámica y su tratamiento depende del contexto en el que se esté trabajando. Para el desarrollo de aplicaciones en JavaScript se dispone de los siguientes tipos de datos o valores:

- Números
- Valores lógicos
- Cadenas de caracteres
- Valores primitivos (null y undefined)

El tratamiento de los datos en JavaScript es dinámico, no requiere de la declaración de variables y la evaluación de las mismas depende del contexto, lo que permite hacer casting libremente. Por ejemplo, se puede definir una variable que contenga una cadena de texto, y, posteriormente asignar a la misma un valor numérico sin que la ejecución del script genere errores. Para comprender lo anterior, escriba lo siguiente:

```
<HTML>
<HEAD>
<TITLE>Casting</TITLE>
      <SCRIPT LANGUAGE="JavaScript1.2">
          var x="Mi nombre es Juan y tengo ";
          document.write(x);
          x=21;
          document.write(x+x);
          x=" años";
          document.write(x);
      </SCRIPT>
</HEAD>

<BODY>

</BODY>
</HTML>
```

El resultado que se obtendrá al abrir la página será:

```
Mi nombre es Juan y tengo 42 años
```

En la primera línea del script se ha definido a x como una variable que contiene una cadena de caracteres. La segunda línea del script imprime el valor de x ("Mi nombre es Juan y tengo "). La misma variable recibe un valor numérico en la tercera línea y posteriormente el script imprimirá el resultado de sumar x+x, en este caso 42. Finalmente, x vuelve a tomar como valor una cadena de caracteres que se imprime en la sexta línea del script.

Lo anterior puede que simplifique y agilice la escritura de código, pero vuelve a JavaScript un lenguaje desordenado, poco elegante y a veces confuso. Considere el siguiente ejemplo:

```
<HTML>
<HEAD>
    <TITLE>Confuso</title>
      <SCRIPT LANGUAGE="JavaScript1.2">
                var x="525";
                document.write(x+5);
      </SCRIPT>
</HEAD>

<BODY>
</BODY>
</HTML>
```

Podría esperarse que el resultado obtenido fuera 530 (x+5), pero en realidad el anterior script mostrará 5255. Por qué? El aparente resultado erróneo que se obtiene se debe a dos razones: la primera, cuando el valor asignado a una variable se encierra entre comillas ("), JavaScript trata a esa variable como una cadena, aunque se trate de un valor numérico y la segunda razón, el operador + en contextos numéricos significa suma, mientras que en operaciones con caracteres es el operador de concatenación.

Si en lugar del signo +, en la segunda línea del script, se utiliza cualquier otro operador aritmético, los resultados que se obtienen serán los esperados desde el punto de vista matemático. Se sugiere al lector utilizar otros operadores aritméticos como resta (-), multiplicación (*), y división (/), en la segunda línea del anterior script, así como, invertir el orden de las variables.

A diferencia de C y C++, JavaScript soporta valores booleanos. Su uso se explicará más adelante en el capítulo de operadores. Por ahora interesa saber que una variable booleana puede tomar cualquiera de los valores true (verdadero) o false (falso).

Variables

Las variables en JavaScript como en otros lenguajes, son contenedores de valores a los cuales se puede acceder a través de sus nombres. Los nombres o identificadores de las variables pueden estar compuestos por una combinación de letras (a-z, A-Z), dígitos (0-9) y el carácter de subrayado (_). El nombre de una variable debe ser diferente a cualquiera de las **palabras reservadas** del lenguaje (**ver Apéndice A**) y finalmente, el nombre de las variables no puede comenzar con un dígito (0-9).

JavaScript es sensible a las mayúsculas (case sensitive), por lo que la variable `Mi_variable` es diferente a `mi_variable`.

La declaración de variables en JavaScript se puede hacer de dos maneras: utilizando la palabra reservada "`var`" seguida del nombre de la variable (como en los scripts del ejemplo anterior), o mediante la simple asignación de un valor a la variable (`y=21;`). Cuando se declara una variable sin la asignación de un valor inicial esta toma como valor "`undefined`" (indefinida). Si se trata de utilizar en contextos numéricos una variable no definida el resultado puede ser:

1. Un error en tiempo de ejecución, cuando la variable no ha sido declarada utilizando la palabra reservada"`var`"

2. "`NaN`", cuando la variable fue declarada mediante el uso de var.

Se dice que una variable es global, cuando puede ser utilizada o referenciada en cualquier parte del script y una variable es local cuando solo puede ser utilizada en el ámbito de una función. Las variables para que sean globales, deben ser declaradas por fuera de cualquier función. Una variable es local cuando se declara dentro una función, y es obligatorio el uso de var en el proceso de declaración. El alcance de las variables será explicado más profundamente en el capítulo de funciones.

Arreglos

Un arreglo es un conjunto de valores del mismo tipo identificados por el mismo nombre. Para distinguirlos, cada uno de los elementos de un arreglo tiene asociado un índice numérico dependiendo de la posición que ocupan, razón por la que a este tipo de datos se les conoce como variables indexadas.

Arreglo días

Índice	0	1	2	3	4	5	6
Valor	"Lunes"	"Martes"	"Miércoles"	"Jueves"	"Viernes"	"Sábado"	"Domingo"
Posición	1	2	3	4	5	6	7

JavaScript como otros lenguajes, soporta la declaración y el trabajo con variables indexadas que contienen dos o más elementos que constituyen un arreglo. Por ejemplo, el arreglo `dias` contiene los nombres de los siete días de la semana y se declara en JavaScript de la siguiente manera:

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

Aunque la anterior, no es la forma más rigurosa de declarar un arreglo, se utiliza solamente para ilustrar el trabajo con ellos.

Para referirse en cualquier arreglo a un elemento ubicado en la posición x, se debe utilizar el nombre de la variable y el índice del elemento encerrado entre paréntesis cuadrados ([]). El índice corresponde a la posición del elemento disminuida en 1. Esto significa, que para un vector compuesto por n elementos, el primero de ellos siempre tendrá como índice 0, mientras que el índice del último elemento será n-1. Por ejemplo, la siguiente página:

```
<HTML>
    <HEAD><TITLE>Indices</TITLE></HEAD>
    <BODY>

          Hoy es

          <script language="JavaScript1.2">
          dias=[“Lunes”, “Martes”, “Miércoles”, “Jueves”, “Viernes”,
                “Sábado”, “Domingo”];

          document.write(dias[2]);
          </script>

          , el tercer día de la semana.
    </BODY>
</HTML>
```

mostrará como resultado:

```
Hoy es Miércoles, el tercer día de la semana.
```

Se pueden declarar arreglos dejando uno o más de sus elementos sin definir:

```
vocales = [“a”,  , “i”, “o”, “u”];
```

y más adelante darle valor al elemento indefinido:

```
vocales[1] = “e”;
```

O se puede eliminar un elemento cualquiera de un arreglo ya declarado, mediante el uso del operador delete:

```
delete vocales[3];
```

Los arreglos son en realidad objetos definidos en JavaScript, por lo que se tratarán con profundidad en el capítulo correspondiente.

Comentarios

Los comentarios corresponden a texto delimitado por caracteres especiales, que el intérprete del lenguaje ignora y que el programador utiliza para documentar sus aplicaciones haciéndolas más fáciles de entender. JavaScript soporta dos tipos de comentarios: los precedidos por dos barras inclinadas “//”, y los encerrados dentro de los caracteres “/*” y “*/”. El primero tiene como efecto que el intérprete ignore todo lo escrito después de los dos slashs y hasta el final de la línea. En el segundo caso, el intérprete no tendrá en

cuenta lo que se encuentre ubicado dentro de esos caracteres especiales, sin importar si su contenido ocupa más de una línea dentro del código.

```
document.write(dias[2]);//Esta parte será ignorada por el intérprete

/* Todo lo que escribamos aquí
    tampoco será tenido en cuenta.
    Aunque el comentario ocupe más de
    una línea */
```