UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Inicial
Front End Developer

JS - Variables

Universidad Tecnológica Nacional - Facultad Regional Buenos Aires
Medrano 951 (C1179AAQ) C.A.B.A. Tel: (54 11) 4867-7511

Almacenando la información que necesitas - Variables

¿Qué es una variable?

Una variable es un contenedor para un valor, como un número que podríamos usar en una suma, o una cadena que podríamos usar como parte de una oración. Pero una cosa especial acerca de las variables es que los valores que contienen pueden cambiar. Veamos un sencillo ejemplo:

```
<button>Presióname</button>
const button = document.querySelector('button');

button.onclick = function() {
  let name = prompt('¿Cuál es tu nombre?');
  alert('¡Hola ' + name + ', encantado de verte!');
}
```

En este ejemplo, al presionar el botón se ejecutan un par de líneas de código.
La primera línea muestra un cuadro en la pantalla que pide al lector que ingrese su nombre y luego almacena el valor en una variable.
La segunda línea muestra un mensaje de bienvenida que incluye su nombre, tomado del valor de la variable.

Para entender por qué esto es tan útil, pensemos en cómo escribiríamos este ejemplo sin usar una variable

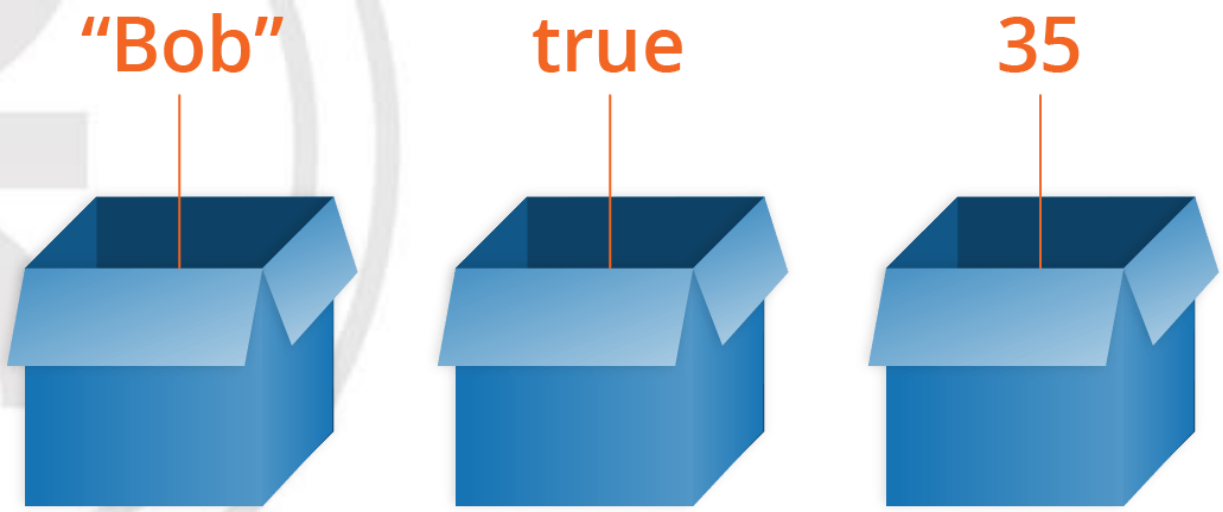
```
let name = prompt('¿Cuál es tu nombre?');

if (name === 'Adam') {
  alert('¡Hola Adam, encantado de verte!');
} else if (name === 'Alan') {
  alert('¡Hola Alan, encantado de verte!');
} else if (name === 'Bella') {
  alert('¡Hola Bella, encantado de verte!');
} else if (name === 'Bianca') {
  alert('¡Hola Bianca, encantado de verte!');
} else if (name === 'Chris') {
  alert('¡Hola Chris, encantado de verte!');
}

// ... y así sucesivamente ...
```

Si no tuviéramos variables disponibles, tendríamos que implementar un bloque de código gigante que verificará cuál era el nombre ingresado, y luego muestra el mensaje apropiado para cualquier nombre.

Las variables simplemente tienen sentido y, a medida que aprendas más sobre JavaScript, comenzarán a convertirse en una segunda naturaleza.

Otra cosa especial acerca de las variables es que pueden contener casi cualquier cosa, no sólo cadenas y números. **Las variables también pueden contener datos complejos e incluso funciones completas para hacer cosas asombrosas.**

Nota: Decimos que las variables contienen valores. Ésta es una importante distinción que debemos reconocer. Las variables no son los valores en sí mismos; son contenedores de valores. Puedes pensar en ellas como pequeñas cajas de cartón en las que puedes guardar cosas.



Declarar una variable

Para usar una variable, primero debes crearla, a esto lo llamamos declarar la variable. Para hacerlo, escribimos la palabra clave `var` o `let` seguida del nombre con el que deseas llamar a tu variable:

```
let myName;
let myAge;
```

Aquí estamos creando dos variables llamadas `myName` y `myAge`. Intenta escribir estas líneas en la consola de tu navegador web. Después de eso, intenta crear una variable (o dos) eligiendo su nombre.

Nota: En JavaScript, todas las instrucciones en el código deben terminar con un punto y coma `(;)` — tu código puede funcionar correctamente para líneas individuales, pero probablemente no lo hará cuando estés escribiendo varias líneas de código juntas. Trata de adquirir el hábito de incluirlo.

Puedes probar si estos valores existen ahora en el entorno de ejecución escribiendo solo el nombre de la variable, p. ej.

```
myName;
myAge;
```

Actualmente no tienen ningún valor; son contenedores vacíos. Cuando ingreses los nombres de las variables, deberías obtener devuelto un valor `undefined`. Si no existen, recibirás un mensaje de error; intenta escribir:

```
scoobyDoo;
```

Nota: No confundas una variable que existe pero no tiene un valor definido, con una variable que no existe en absoluto — son cosas muy diferentes. En la analogía de cajas que viste arriba, no existir significaría que no hay una caja (variable) para guardar un valor. Ningún valor definido significaría que HAY una caja, pero no tiene ningún valor dentro de ella.

Iniciar una variable

Una vez que hayas declarado una variable, la puedes iniciar con un valor. Para ello, escribe el nombre de la variable, seguido de un signo igual `(=)`, seguido del valor que deseas darle. Por ejemplo:

```
myName = 'Chris';
myAge = 37;
```

Intenta volver a la consola ahora y escribe estas líneas. Deberías ver el valor que le has asignado a la variable devuelto en la consola para confirmarlo, en cada caso. Nuevamente, puedes devolver los valores de tus variables simplemente escribiendo su nombre en la consola; inténtalo nuevamente:

```
myName;
myAge;
```

Puedes declarar e iniciar una variable al mismo tiempo, así:

```
let myDog = 'Rover';
```

Esto probablemente es lo que harás la mayor parte del tiempo, ya que es más rápido que realizar las dos acciones en dos líneas separadas.

Diferencia entre var y let

Cuando se creó JavaScript por primera vez, solo existía `var`. Esto básicamente funciona bien en la mayoría de los casos, pero tiene algunos problemas en la forma en que trabaja — su diseño a veces puede ser confuso. Entonces, se creó `let` en versiones modernas de JavaScript, una nueva palabra clave para crear variables que funciona de manera algo diferente a var, solucionando sus problemas en el proceso.

Para empezar, si escribes un programa JavaScript de varias líneas que declara e inicia una variable, puedes declarar una variable con var después de iniciarla y seguirá funcionando. Por ejemplo:

```
myName = 'Chris';

function logName() {
  console.log(myName);
}

logName();

var myName;
```

Nota: Esto no funcionará al escribir líneas individuales en una consola de JavaScript, solo cuando se ejecutan varias líneas de JavaScript en un documento web.
Esto funciona debido a la **elevación.** La elevación (hoisting) ya no funciona con let. Si cambiamos var a let en el ejemplo anterior, fallaría con un error. Declarar una variable después de iniciarla resulta en un código confuso y más difícil de entender.

En segundo lugar, cuando usas var, puedes declarar la misma variable tantas veces como desees, pero con let no puedes. Lo siguiente funcionaría:

```
var myName = 'Chris';
var myName = 'Bob';
```

Pero lo siguiente arrojaría un error en la segunda línea:

```
let myName = 'Chris';
let myName = 'Bob';
```

Tendrías que hacer esto en su lugar:

```
let myName = 'Chris';
myName = 'Bob';
```

Nuevamente, esta es una sensata decisión del lenguaje. No hay razón para volver a declarar las variables — solo hace que las cosas sean más confusas.
Por estas y otras razones, se recomienda utilizar let tanto como sea posible en tu código, en lugar de var. No hay ninguna razón para usar var, a menos que necesites admitir versiones antiguas de Internet Explorer con tu código (no es compatible con let hasta la versión 11; Edge el moderno navegador de Windows admite let perfectamente).

Actualizar una variable

Una vez que una variable se ha iniciado con un valor, puedes cambiar (o actualizar) ese valor simplemente dándole un valor diferente. Intenta ingresar las siguientes líneas en tu consola:

```
myName = 'Bob';
myAge = 40;
```

Un consejo sobre las reglas de nomenclatura de variables

Puedes llamar a una variable prácticamente como quieras, pero existen limitaciones. En general, debes limitarte a usar caracteres latinos (0-9, a-z, A-Z) y el caracter de subrayado.

- No debes usar otros caracteres porque pueden causar errores o ser difíciles de entender para una audiencia internacional.
- No use guiones bajos al comienzo de los nombres de las variables — esto se usa en ciertas construcciones de JavaScript para significar cosas específicas, por lo que puede resultar confuso.
- No uses números al comienzo de las variables. Esto no está permitido y provoca un error.
- Una convención segura a seguir es la llamada **"minúscula mayúsculas intercaladas"**, en la que se juntan varias palabras con minúsculas para la primera

palabra completa y luego en mayúsculas las primeras letras de las siguientes palabras.

- Haz que los nombres de las variables sean intuitivos, para que describan los datos que contienen. No uses solo letras/números o frases grandes y largas.
- Las variables distinguen entre mayúsculas y minúsculas — por lo tanto `myage` es una variable diferente de `myAge`.
- Un último punto: también debes evitar el uso de palabras reservadas de JavaScript como nombres de variables — con esto, nos referimos a las palabras que componen la sintaxis real de JavaScript. Por lo tanto, no puedes usar palabras como `var`, `function`, `let` y `for` como nombres de variables. Los navegadores las reconocen como elementos de código diferentes, por lo que obtendrás errores.

Ejemplos de buenos nombres:

- age
- myAge
- init
- initialColor
- finalOutputValue
- audio1
- audio2

Ejemplos de nombres incorrectos:

- 1
- a
- _12
- myage
- MYAGE
- var
- Document
- skjfndskjfnbdskjfb
- thisisareallylongstupidvariablenameman

Ahora, intenta crear algunas variables más, con la guía anterior en mente.

Tipo de las variables

Hay algunos tipos de datos diferentes que podemos almacenar en variables. Hasta ahora hemos analizado los dos primeros, pero hay otros.

Números

Puedes almacenar números en variables, ya sea números enteros como 30 (también llamados enteros — "integer") o números decimales como 2.456 (también llamados números flotantes o de coma flotante — "number").

No es necesario declarar el tipo de las variables en JavaScript, a diferencia de otros lenguajes de programación. Cuando le das a una variable un valor numérico, no incluye comillas:

```
let myAge = 17;
```

Cadenas de caracteres (Strings)

Las strings (cadenas) son piezas de texto. Cuando le das a una variable un valor de cadena, debes encerrarlo entre comillas simples o dobles; de lo contrario, JavaScript intenta interpretarlo como otro nombre de variable.

```
let dolphinGoodbye = 'Hasta luego y gracias por todos los peces';
```

Booleanos

Los booleanos son valores verdadero/falso — pueden tener dos valores, `true` o `false`. Estos, generalmente se utilizan para probar una condición, después de lo cual se ejecuta el código según corresponda. Así, por ejemplo, un caso simple sería:

```
let iAmAlive = true;
```

Mientras que en realidad se usaría más así:

```
let test = 6 < 3;
```

Acá se está usando el operador "menor que" `(<)` para probar si 6 es menor que 3. Como era de esperar, devuelve false, ¡porque 6 no es menor que 3!

Arreglos

Un arreglo es un objeto único que contiene múltiples valores encerrados entre corchetes y separados por comas. Intenta ingresar las siguientes líneas en tu consola:

```
let myNameArray = ['Chris', 'Bob', 'Jim'];
let myNumberArray = [10, 15, 40];
```

Una vez que se definen estos arreglos, puedes acceder a cada valor por su ubicación dentro del arreglo. Prueba estas líneas:

```
myNameArray[0]; // debería devolver 'Chris'
myNumberArray[2]; // debe devolver 40
```

Los corchetes especifican un valor de índice correspondiente a la posición del valor que deseas devolver.
Posiblemente hayas notado que los arreglos en JavaScript tienen índice cero: **el primer elemento está en el índice 0**.

Objetos

En programación, un objeto es una estructura de código que modela un objeto de la vida real.
Puedes tener un objeto simple que represente una caja y contenga información sobre su ancho, largo y alto, o podrías tener un objeto que represente a una persona y contenga datos sobre su nombre, estatura, peso, qué idioma habla, cómo saludarlo, y más

Intenta ingresar la siguiente línea en tu consola:

```
let dog = { name : 'Spot', breed : 'Dalmatian' };
```

Para recuperar la información almacenada en el objeto, puedes utilizar la siguiente sintaxis:

```
dog.name
```

Tipado dinámico

JavaScript es un "lenguaje tipado dinámicamente", lo cual significa que, a diferencia de otros lenguajes, no es necesario especificar qué tipo de datos contendrá una variable (números, cadenas, arreglos, etc.).

Por ejemplo, si declaras una variable y le das un valor entre comillas, el navegador trata a la variable como una cadena (string):

```
let myString = 'Hello';
```

Incluso si el valor contiene números, sigue siendo una cadena, así que ten cuidado:

```
let myNumber = '500'; // Esto sigue siendo una cadena
typeof myNumber;
myNumber = 500; // mucho mejor -- ahora este es un número
typeof myNumber;
```

Intenta ingresar las cuatro líneas anteriores en tu consola una por una y ve cuáles son los resultados.

Notarás que estamos usando un operador especial llamado `typeof` — esto devuelve el tipo de datos de la variable que escribes después.

La primera vez que se llama, debe devolver string, ya que en ese punto la variable myNumber contiene una cadena, '500'.

Constantes en JavaScript

Muchos lenguajes de programación tienen el concepto de una *constante* — un valor que, una vez declarado, no se puede cambiar. Hay muchas razones por las que querrías hacer esto, desde la seguridad (si un script de un tercero cambia dichos valores, podría causar problemas) hasta la depuración y la comprensión del código (es más difícil cambiar accidentalmente valores que no se deben cambiar y estropear cosas claras).

En los primeros días de JavaScript, las constantes no existían. En JavaScript moderno, tenemos la palabra clave const, que nos permite almacenar valores que nunca se pueden cambiar:

```
const daysInWeek = 7;
const hoursInDay = 24;
```

`const` funciona exactamente de la misma manera que `let`, excepto que a const no le puedes dar un nuevo valor. En el siguiente ejemplo, la segunda línea arrojará un error:

```
const daysInWeek = 7;
daysInWeek = 8;
```