

JS INSTITUTE  
Open Education & Development Group

« 2.1.1.4 Variables - Declarando variables »

Sandbox

Editor, ejecutor en línea y observador de resultados en la consola.

```
var height;
console.log(height); // -> undefined
console.log(weight); // -> Uncaught ReferenceError: weight is not defined
```

La primera línea es la **declaración** de la variable (podemos ver la palabra clave `var`). Esta declaración significa que la palabra `height` será tratada como el nombre del contenedor para ciertos valores.

**let height;**

La declaración, como otras instrucciones de JavaScript, debe terminar con un punto y coma. En la segunda línea, tratamos de escribir el valor de esta variable (es decir, lo que hay en el contenedor) en la consola. Debido a que aún no hemos puesto nada allí, el resultado no está definido (el intérprete conoce esta variable, pero aún no tiene valor, el valor no está definido). En la siguiente línea, tratamos de imprimir el contenido de la variable `weight`, que olvidamos declarar. Esta vez, veremos `ReferenceError`. El intérprete de JavaScript, que ejecuta nuestro programa, nos ha informado que no conoce una variable con este nombre (por lo que la variable en sí no está definida).

```
1 var height;
2 console.log(height); // -> undefined
3 console.log(weight); // -> Uncaught ReferenceError: weight is not defined
```

app.js

Console

```
undefined
Uncaught ReferenceError: weight is not defined
```

JS INSTITUTE  
Open Education & Development Group

« 2.1.1.5 Variables - Declarando variables cont. »

Sandbox

Una de las diferencias básicas en el uso de `var` y `let` es que `let` nos impide declarar otra variable con el mismo nombre (se genera un error). El uso de `var` le permite volver a declarar una variable, lo que puede generar errores en la ejecución del programa.

```
var height;
var height;
console.log(height); // -> undefined
```

El ejemplo anterior demuestra la posibilidad de volver a declarar una variable usando la palabra clave `var`. En esta situación, no causará un error, pero en programas más complejos, una redeclaración, especialmente por accidente, puede tener consecuencias. Al declarar con `let`, el intérprete verifica si dicha variable ya ha sido declarada, sin importar si se usó `let` o `var` en la declaración anterior.

```
let height;
let height; // -> Uncaught SyntaxError: Identifier 'height' has already been declared
console.log(height);
```

Así que usa `let` para declarar variables, aunque solo sea porque no quieres volver a declarar accidentalmente una variable.

```
1 let height;
2 let height; // -> Uncaught SyntaxError: Identifier 'height' has already been declared
3 console.log(height);
```

app.js

Console

```
Uncaught SyntaxError: Identifier 'height' has already been declared
```

JS INSTITUTE  
Open Education & Development Group

« 2.1.1.6 Variables - Inicializando variables »

Sandbox

de intentar leerla, modificarla o mostrarla.

```
let height = 180;
let anotherHeight = height;
let weight;
console.log(height); // -> 180
console.log(anotherHeight); // -> 180
weight = 70;
console.log(weight); // -> 70
```

En el ejemplo anterior (verifícalo en el editor), las declaraciones de las variables `height` y `anotherHeight` se combinan con su inicialización, mientras que la variable `weight` se declara e inicializa por separado. Las variables `height` y `weight` se inicializan proporcionando valores específicos (más precisamente, un número), mientras que la variable `anotherHeight` recibe un valor leído de la variable de `height`. Los valores de todas las variables se muestran en la consola.

Por cierto, presta atención a una cosa. Si especificas un nombre de variable en `console.log`, el intérprete lo reconoce y muestra su valor. Si pones el mismo nombre entre comillas, se tratará como texto sin formato y se mostrará como tal.

```
let height = 180;
console.log(height); // -> 180
console.log("height"); // -> height
```

```
1 let height = 180;
2 let anotherHeight = height;
3 let weight;
4 console.log(height); // -> 180
5 console.log(anotherHeight); // -> 180
6 weight = 70;
7 console.log(weight); // -> 70
```

app.js

Console

```
180
180
70
```

JS INSTITUTE  
Open Education & Development Group

« 2.1.1.6 Variables - Inicializando variables »

Sandbox

de intentar leerla, modificarla o mostrarla.

```
let height = 180;
let anotherHeight = height;
let weight;
console.log(height); // -> 180
console.log(anotherHeight); // -> 180
weight = 70;
console.log(weight); // -> 70
```

En el ejemplo anterior (verifícalo en el editor), las declaraciones de las variables `height` y `anotherHeight` se combinan con su inicialización, mientras que la variable `weight` se declara e inicializa por separado. Las variables `height` y `weight` se inicializan proporcionando valores específicos (más precisamente, un número), mientras que la variable `anotherHeight` recibe un valor leído de la variable de `height`. Los valores de todas las variables se muestran en la consola.

Por cierto, presta atención a una cosa. Si especificas un nombre de variable en `console.log`, el intérprete lo reconoce y muestra su valor. Si pones el mismo nombre entre comillas, se tratará como texto sin formato y se mostrará como tal.

```
let height = 180;
console.log(height); // -> 180
console.log("height"); // -> height
```

```
1 let height = 180;
2 console.log(height); // -> 180
3 console.log("height"); // -> height
```

app.js

Console

180  
height

JS INSTITUTE  
Open Education & Development Group

« 2.1.1.7 Variables - Declaraciones y modo estricto »

Sandbox

### Declaraciones y modo estricto

JavaScript tuvo algunos cambios importantes introducidos en 2009 y 2015. La mayoría de estos cambios ampliaron la sintaxis del lenguaje con nuevos elementos, pero algunos de ellos se referían solo al funcionamiento de los intérpretes de JavaScript. A menudo se trataba de aclarar el comportamiento de los intérpretes en situaciones potencialmente erróneas, como en los casos de inicialización de variables sin ninguna declaración previa.

Veamos un ejemplo:

```
height = 180;
console.log(height); // -> 180
```

A primera vista, puedes ver que nos hemos olvidado de declarar la variable `height`. La sintaxis de JavaScript original permitía tal negligencia, y en el momento de la inicialización hizo esta declaración por nosotros. Parece una solución bastante buena, pero desafortunadamente a veces puede conducir a situaciones ambiguas y potencialmente erróneas (diremos algunas palabras más al respecto mientras discutimos el alcance).

Vamos a modificar nuestro ejemplo:

```
"use strict";
```

```
1 height = 180;
2 console.log(height); // -> 180
```

app.js

Console

180

JS INSTITUTE  
Open Education & Development Group

« 2.1.1.9 Variables - Constantes »

Sandbox

### Constantes

La palabra clave `const` se usa para declarar contenedores similares a variables. Dichos contenedores se denominan **constantas**. Las constantes también se utilizan para almacenar ciertos valores, pero una vez que se han ingresado valores durante la inicialización, ya no se pueden modificar. Esto significa que este tipo de contenedor se declara e inicializa simultáneamente. Por ejemplo, la siguiente declaración de la constante `greeting` es correcta:

```
const greeting = "¡Hola!";
```

Pero esta próxima definitivamente causa un error:

```
const greeting; // -> Uncaught SyntaxError: Missing initializer
greeting = "¡Hola!";
```

Como dijimos, un cambio en la constante es imposible. Esta vez la declaración es correcta, pero tratamos de modificar el valor almacenado en la constante.

```
const greeting = "¡Hola!";
greeting = "Hi!"; // -> Uncaught TypeError: Assignment to constant variable.
```

El propósito principal de una constante es erradicar la posibilidad de cambiar

```
1 const greeting = "¡Hola!";
2 greeting = "Hi!"; // -> Uncaught TypeError: Assignment to constant variable.
3
```

app.js

Console

Uncaught TypeError: Assignment to constant variable.

JS INSTITUTE

« 2.1.1.10 Variables - Alcance y bloques del programas »

Sandbox

Los bloques del programa se pueden anidar, es decir, podemos crear un bloque dentro de otro.

```
let counter;
console.log(counter); // -> undefined
{
  counter = 1;
  {
    console.log(counter); // -> 1
  }
}
counter = counter + 1;
console.log(counter); // -> 2
```

Por cierto, toma en cuenta que el código dentro del bloque se ha movido a la derecha. Esto se denomina **indentación**. Para un intérprete de JavaScript, no importa en absoluto, pero definitivamente aumenta la legibilidad del código, lo que permite a los lectores (incluso a ti) descubrir rápidamente qué partes del código están dentro y cuáles están fuera del bloque. Los editores de código suelen agregar sangrías en los lugares correctos por sí mismos, pero es un buen hábito recordarlo y, si no aparecen automáticamente, da formato al código a mano.

```
1 let counter;
2 console.log(counter); // -> undefined
3 {
4   counter = 1;
5   {
6     console.log(counter); // -> 1
7   }
8 }
9 counter = counter + 1;
10 console.log(counter); // -> 2
```

app.js

Console

```
undefined
1
2
```

Fullscreen

JS INSTITUTE

« 2.1.1.11 Variables - let y const »

Sandbox

bloque. También podemos probar el caso con bloques anidados

```
let height = 200;
{
  let weight = 100;
  {
    let info = "tall";
    console.log(height); // -> 200
    console.log(weight); // -> 100
    console.log(info); // -> tall
  }
  console.log(height); // -> 200
  console.log(weight); // -> 100
  console.log(info); // -> Uncaught ReferenceError: info is not defined
}
```

Como puedes ver, la variable `info` declarada en el bloque más interno solo es visible dentro de él. La variable `weight` es visible tanto dentro del bloque en el que fue declarada como dentro del bloque anidado en ella. Y la variable global `height` es visible en todas partes.

Simple, ¿no?

```
1 let height = 200;
2 {
3   let weight = 100;
4   {
5     let info = "tall";
6     console.log(height); // -> 200
7     console.log(weight); // -> 100
8     console.log(info); // -> tall
9   }
10  console.log(height); // -> 200
11  console.log(weight); // -> 100
12  console.log(info); // -> Uncaught ReferenceError: info is not defined
13 }
```

app.js

Console

```
200
100
tall
200
100
Uncaught ReferenceError: info is not defined
```

Fullscreen

JS INSTITUTE

« 2.1.1.12 Variables - la palabra clave var »

Sandbox

es ligeramente diferente. La variable declarada utilizándola fuera de los bloques será, como en el caso de `let`, global, es decir, será visible en todas partes. Si la declaras dentro de un bloque, entonces... bueno, por lo general volverá a ser global.

Comencemos con un ejemplo simple:

```
var height = 180;
{
  var weight = 70;
  console.log(height); // -> 180
  console.log(weight); // -> 70
}
console.log(height); // -> 180
console.log(weight); // -> 70
```

Como era de esperar, ambas variables, `height` y `weight`, resultan ser globales. ¿Las variables declaradas usando `var` siempre, independientemente del lugar de declaración, serán globales? Definitivamente no. El problema es que `var` ignora los bloques de programa ordinarios, tratándolos como si no existieran. Entonces, ¿en qué situación podemos declarar una variable local usando `var`? Sólo dentro de una función. Dedicaremos mucho espacio a discutir la función y luego volveremos al problema del alcance de la variable. Ahora intentaremos presentar y discutir solo un ejemplo simple, que mostrará que las variables `var` a veces también son locales.

```
1 var height = 180;
2 {
3   var weight = 70;
4   console.log(height); // -> 180
5   console.log(weight); // -> 70
6 }
7 console.log(height); // -> 180
8 console.log(weight); // -> 70
```

app.js

Console

```
180
70
180
70
```

Fullscreen

JS INSTITUTE

« 2.1.1.13 Variables - Unas breves palabras sobre funciones »

Sandbox

paréntesis, que opcionalmente podrían contener parámetros pasados a la función (volveremos a esto cuando analicemos funciones con más precisión). Luego abrimos el bloque del programa, que contiene las instrucciones pertenecientes a la función. Al definir una función, las instrucciones contenidas en la función no se ejecutan. Para ejecutar la función, debes llamarla de forma independiente, usando su nombre.

Echa un vistazo al siguiente programa.

```
console.log("Comencemos:"); // -> Comencemos:
console.log("Hola"); // -> Hola
console.log("Mundo"); // -> Mundo
console.log("y otra vez:"); // -> y otra vez:
console.log("Hola"); // -> Hola
console.log("Mundo"); // -> Mundo
console.log("y una vez más:"); // -> y una vez más:
console.log("Hola"); // -> Hola
console.log("Mundo"); // -> Mundo
```

Imprimirá una secuencia de texto en la consola:

```
Comencemos:
Hola
Mundo
y otra vez:
Hola
```

```
1 console.log("Comencemos:"); // -> Comencemos:
2 console.log("Hola"); // -> Hola
3 console.log("Mundo"); // -> Mundo
4 console.log("y otra vez:"); // -> y otra vez:
5 console.log("Hola"); // -> Hola
6 console.log("Mundo"); // -> Mundo
7 console.log("y una vez más:"); // -> y una vez más:
8 console.log("Hola"); // -> Hola
9 console.log("Mundo"); // -> Mundo
```

app.js

Console

```
Comencemos:
Hola
Mundo
y otra vez:
Hola
Mundo
y una vez más:
Hola
Mundo
```

JS INSTITUTE

« 2.1.1.14 Variables - la palabra clave var cont. »

Sandbox

```
var localGreeting = "Días";
console.log("función:");
console.log(globalGreeting);
console.log(localGreeting);
}

testFunction();

console.log("programa principal:");
console.log(globalGreeting);
console.log(localGreeting); // -> Uncaught ReferenceError: localGreeting is not defined
```

En primer lugar, ejecuta este programa y observa los resultados en la consola. ¿Qué pasó y, sobre todo, por qué pasó?


Echemos un vistazo más de cerca al código. En el ejemplo, declaramos la variable global `globalGreeting`. Luego definimos la función `testFunction`, dentro de la cual declaramos la variable local `localGreeting`. Luego llamamos a la función `testFunction`, que resultó en escribir los valores de ambas variables (dentro de la función, tenemos acceso tanto a la variable global como a las locales). Intentar acceder a la variable local `localGreeting` fuera de la función fallará. Así que finalmente logramos demostrar que las declaraciones de variables que usan la palabra `var` también pueden ser locales.

```
1 var globalGreeting = "Buenos";
2
3 function testFunction() {
4   var localGreeting = "Días";
5   console.log("función:");
6   console.log(globalGreeting);
7   console.log(localGreeting);
8 }
9
10 testFunction();
11
12 console.log("programa principal:");
13 console.log(globalGreeting);
14 console.log(localGreeting); // -> Uncaught ReferenceError: localGreeting is not defined
```

app.js

Console

```
función:
Buenos
Días
programa principal:
Buenos
Uncaught ReferenceError: localGreeting is not defined
```

Expert Education & Championship Growth

« 2.1.1.15 Variabes - Sombreado »

Sandbox

La variable `counter`, declarada al principio del programa, es una variable global. A lo largo del programa, también dentro del bloque, operamos sobre esta misma variable. Un pequeño cambio en el código es suficiente para que el programa se comporte de manera completamente diferente.

```
let counter = 100;
console.log(counter); // -> 100
{
  let counter = 200;
  console.log(counter); // -> 200
}
console.log(counter); // -> 100
```

¿Ves la diferencia? Esta vez en el bloque, en lugar de `counter = 200;` (es decir, cambios en el contenido de la variable `counter` global), aparece la instrucción `let counter = 200;` (es decir, declaraciones de la variable local combinada con su inicialización). El intérprete consideraría que tal situación es incorrecta si la redeclaración apareciera en el mismo alcance.

Sin embargo, la declaración es local (tiene un alcance diferente al global) y todas las referencias a la variable con este nombre dentro del bloque se referirán a esta variable local. Fuera del bloque, la variable global se seguirá viendo con el mismo nombre. Presta atención a los valores que muestra la consola.

app.js

```
1 let counter = 100;
2 console.log(counter); // -> 100
3 {
4   let counter = 200;
5   console.log(counter); // -> 200
6 }
7 console.log(counter); // -> 100
```

Console

100  
200  
100

Fullscreen