Var.

Las descripciones de variables, donde sea que ocurran,son procesadas antes que cualquier otro código sea ejecutado. El ámbito de una variable declarada con la palabra reservada **var**  es su contexto de ejecución en curso, que puede ser la función que la contiene o, para las variables declaradas afuera de cualquier función, un ámbito global. Si re-declaras una variable Javascript, esta no perderá su valor.

Asignar un valor a una variable no declarada impica crearla como una variable global( e convierte en una propiedad del objeto global) cuando la asignación es ejecutada. Las diferencias entre una variable declarada y otra sin declarar son:

1. Las variables declaradas se limitan al contexto de ejecución en el cual son declaradas. Las variables no declaradas siempre son globales.

function x() {

y = 1; // Lanza un error de tipo "ReferenceError" en modo estricto ('use strict')

var z = 2;

}

x();

console.log(y); // Imprime "1"

* 1. console.log(z); // Lanza un error de tipo "ReferenceError": z no está definida afuera de x

1. las variables declaradas son creadas antes de ejecutar cualquier otro código. Lasa variables sin declarar no existen hasta que el código que las asigna es ejecutado.

console.log(a); // Lanza un error de tipo "ReferenceError".

console.log('trabajando...'); // Nunca se ejecuta.

var a;

console.log(a); // Imprime "undefined" o "" dependiendo del navegador.

console.log('trabajando...'); // Imprime "trabajando...".

1. las variables declaradas son una propiedad no-configurable de su contexto de ejecución (de función o global). Las variables si declarar son configurables (p. ej. Pueden borrarse).

var a = 1;

b = 2;

delete this.a; // Lanza un error de tipo "ReferenceError" en modo estricto ('use strict'), de lo contrario falla silenciosamente.

delete this.b;

console.log(a, b); // Lanza un error de tipo "ReferenceError".

// La propiedad 'b' se eliminó y ya no existe.

Elevación de elevación

Como la declaración de variables (y todas las declaraciones en general) se procesa antes de ejecutar cualquier código, declarar una variable en cualquier parte del código es equivalente a declararla al inicio del mismo. Esto también significa que una variable puede parecer usarse antes de ser declarada. Este comportamiento es llamado *hoisting*(del inglés "elevación"), ya que la declaración de una variable parecer haber sido movida a la cima de la función o código global.

bla = 2;

var bla;

// ...

// Es entendido implicitamente como:

var bla;

bla = 2;

Por esa razón, se recomienda siempre declarar variables al inicio de su ámbito (la cima del código global y la cima del código de función) para que sea claro cuáles variables pertenecen al ámbito de función (local) y cuáles son resueltas en la cadena de ámbito.

Es importante señalar que **la elevación afectará la declaración** de variables, pero **no su inicialización**. El valor será asignado precisamente cuando la sentencia de asignación sea alcanzada:

function haz\_algo() {

console.log(bar); // undefined (valor indefinido)

var bar = 111;

console.log(bar); // 111

}

// Se entiende implícitamente como:

function haz\_algo() {

var bar;

console.log(bar); // undefined (valor indefinido)

bar = 111;

console.log(bar); // 111

}

Let

**let**permite declarar variables limitando su alcance (scope) al bloque, declaración, o expresión donde se está usando. Lo anterior diferencia  **let** de la palabra reservada [var](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/var" \o "JavaScript/Reference/Statements/var) , la cual define una variable global o local en una función sin importar el ámbito del bloque.

### Alcance (scope) a nivel de bloque con let[Sección](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/let" \l "Alcance_(scope)_a_nivel_de_bloque_con_let)

Usar la palabra reservada let para definir variables dentro de un bloque.

if (x > y) {

let gamma = 12.7 + y;

i = gamma \* x;

}

Es posible usar definiciones let para asociar código en extensiones con un pseudo-espacio-de-nombre (pseudo-namespace). (Ver [Mejores prácticas de seguridad en extensiones](https://developer.mozilla.org/en-US/docs/Security_best_practices_in_extensions" \o "en-US/docs/Security_best_practices_in_extensions).)

let Cc = Components.classes, Ci = Components.interfaces;

let puede ser útil para escribir código más limpio cuando usamos funciones internas.

var list = document.getElementById("list");

for (var i = 1; i <= 5; i++) {

var item = document.createElement("LI");

item.appendChild(document.createTextNode("Item " + i));

let j = i;

item.onclick = function (ev) {

console.log("Item " + j + " is clicked.");

};

list.appendChild(item);

}

El ejemplo anterior trabaja como se espera porque las cinco instancias de la función (anónima) interna hacen referencia a cinco diferentes instancias de la variable j. Nótese que esto no funcionaría como se espera si reemplazamos let con var o si removemos la variable j y simplemente usamos la variable i dentro de la función interna.

#### Reglas de alcance

Variables declaradas por let tienen por alcance el bloque en el que se han definido, así mismo, como en cualquier bloque interno. De esta manera, let trabaja muy parecido a var. La más notable diferencia es que el alcance de una variable var es la función contenedora:

function varTest() {

var x = 31;

if (true) {

var x = 71; // ¡misma variable!

console.log(x); // 71

}

console.log(x); // 71

}

function letTest() {

let x = 31;

if (true) {

let x = 71; // variable diferente

console.log(x); // 71

}

console.log(x); // 31

}

// llamamos a las funciones

varTest();

letTest();

En el nivel superior de un programa y funciones, let , a diferencia de var, **no crea** una propiedad en el objeto global, por ejemplo:

var x = 'global';

let y = 'global';

console.log(this.x); // "global"

console.log(this.y); // undefined

La salida de este código desplegaría "global" una vez.

### Zona muerta temporal y errores con let[Sección](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/let" \l "Zona_muerta_temporal_y_errores_con_let)

La **redeclaración** de la misma variable bajo un mismo [ámbito léxico](https://www.ecma-international.org/ecma-262/6.0/" \l "sec-lexical-environments) terminaría en un error de tipo [SyntaxError](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/SyntaxError" \o "SyntaxError). Esto también es **extensible** si usamos var dentro del ámbito léxico. Esto nos salvaguarda de redeclarar una variable accidentalmente y que no era posible  solo con var.

if (x) {

let foo;

let foo; // Terminamos con un SyntaxError.

}

if (x) {

let foo;

var foo; // Terminamos con un SyntaxError.

}

En ECMAScript 2015, let [no eleva](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/var" \l "Description) la variable a la parte superior del bloque. Si se hace una referencia a la variable declarada con let (let foo) antes de su declaración, terminaríamos con un error de tipo [ReferenceError](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/ReferenceError" \o "TypeError) (al contrario de la variable declarada con var, que tendrá el valor undefined), esto porque la variables vive en una "zona muerta temporal" desde el inicio del bloque hasta que la declaración ha sido procesada.

function do\_something() {

console.log(bar); // undefined

console.log(foo); // ReferenceError: foo no está definido

var bar = 1;

let foo = 2;

}

Es posible encontrar errores en bloques de control [switch](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/switch" \o "switch) debido a que solamente existe un block subyacente.

switch (x) {

case 0:

let foo;

break;

case 1:

let foo; // Terminamos con un error de tipo SyntaxError.

// esto debido a la redeclaracion

break;

}

### Otro ejemplo de zona muerta temporal combinada con [ámbito léxico](https://www.ecma-international.org/ecma-262/6.0/" \l "sec-lexical-environments)[Sección](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/let" \l "Otro_ejemplo_de_zona_muerta_temporal_combinada_con_%C3%A1mbito_l%C3%A9xico)

Debido al alcance léxico, el identificador num dentro de la expresión (num + 55) se evalúa como num del bloque if, y no como la variable num con el valor 33 que esta por encima

En esa misma línea, el num del bloque if ya se ha creado en el ámbito léxico, pero aún no ha alcanzado (y **terminado**) su inicialización (que es parte de la propia declaración): todavía está en la zona muerta temporal.

function prueba(){

var num = 33;

if (true) {

let num = (num + 55);//ReferenceError: no se puede acceder a la declaración léxica `num'antes de la inicialización

}

}

prueba();