En **ECMAScript 6** (ES6 o ES2015) fueron publicadas varias características nuevas que dotaron de gran poder al lenguaje, dos de estas son una nueva forma de declaración de variables con let y const, y funciones flechas.

**La nueva forma para declarar variables con *let* y *const***

Hasta ahora aprendiste a declarar variables con var, sin embargo, a partir de la especificación de ES6 se agregaron nuevas formas para la declaración de variables.

Las nuevas palabras reservadas let y const resuelven varios problemas con varcomo el *scope*, *hoisting*, variables globales, re-declaración y re-asignación de variables.

**Variables re-declaradas y re-asignadas**

**La re-declaración es volver a declarar una variable, y la re-asignación es volver a asignar un valor**. Entonces cada palabra reservada tiene una forma diferente de manejar variables:

Una variable declarada con var puede ser re-declarada y re-asignada.

Una variable declarada con let puede ser re-asignada, pero no re-declarada.

Una variable declarada con const no puede ser re-declarada, ni re-asignada. Su declaración y asignación debe ser en una línea, caso contrario habrá un error.

En conclusión, si intentas re-declarar una variable declarada con let y const habrá un error de “variable ya declarada”; por otro lado, si intentas re-asignar una variable declarada con const existirá un “error de tipo”.

En los demás casos, JavaScript lo aceptará como válidos, algo problemático con var, por eso deja de utilizarlo.

**Ejemplo de declaración y asignación en diferentes líneas**

// Declaración de variables

**var** nameVar

**let** nameLet

// Asignación de variables

nameVar= "soy var"

nameLet = "soy let"

Aunque realmente lo que pasa si no asignas un valor en la declaración, JavaScript le asigna un valor undefined.

**Ejemplo de declarar y asignar con *const* en diferentes líneas de código**

**const** pi // SyntaxError: Missing initializer in const declaration.

pi = 3.14

**Ejemplo de re-declaración de variables**

**var** nameVar = "soy var"

**let** nameLet = "soy let"

**const** nameConst = "soy const"

// Re-declaración de variables

**var** nameVar = "var soy"

console.log(nameVar) // 'var soy'

**let** nameLet = "let soy" // SyntaxError: Identifier 'nameLet' has already been declared.

**const** nameConst = "const soy" //SyntaxError: Identifier 'nameConst' has already been declared.

**Ejemplo de re-asignación de variables**

**var** nameVar = "soy var"

**let** nameLet = "soy let"

**const** nameConst = "soy const"

// Re-asignación de variables

nameVar = "otro var"

console.log(nameVar) // 'otro var'

nameLet = "otro let"

console.log(nameVar) // otro let'

nameConst = "otro const" //TypeError: Assignment to constant variable.

Ten en cuenta que los errores pararán la ejecución de tu programa.

***Scope***

En el tema del *scope*, let y const **tienen un *scope* de bloque** y var no.{

**var** nameVar = "soy var"

**let** nameLet = "soy let"

}

console.log(nameVar) // 'soy var'

console.log(nameLet) // ReferenceError: nameLet is not defined

Todo el tema de Scope tiene su propio curso que deberías haber tomado: [Curso de Closures y Scope en JavaScript](https://platzi.com/cursos/javascript-closures-scope/)

**Objeto global**

En variables globales, let y constno guardan sus variables en el objeto global (window, global o globalThis), mientras que var sí los guarda.

**var** nameVar = "soy var"

**let** nameLet = "soy let"

**const** nameConst = "soy const"

globalThis.nameVar // 'soy var'

globalThis.nameLet // undefined

globalThis.nameConst // undefined

Esto es importante para que no exista re-declaración de variables.

**Funciones flecha**

Las funciones flecha *(arrow functions)* consiste en una **función anónima** con la siguiente estructura:

//Función tradicional

**function** **nombre** (parámetros) {

**return** valorRetornado

}

//Función flecha

**const** nombre = (parámetros) => {

**return** valorRetornado

}

Se denominan función flecha por el elemento => en su sintaxis.

**Omitir paréntesis en las funciones flecha**

Si existe un solo parámetro, puedes omitir los paréntesis.

**const** porDos = num => {

**return** num \* 2

}

**Retorno implícito**

Las funciones flecha tienen un retorno implícito, es decir, se puede omitir la palabra reservada return, para que el código sea escrito en una sola línea.

//Función tradicional

**function** **suma** (num1, num2) {

**return** num1 + num2

}

//Función flecha

**const** suma = (num1, num2) => num1 + num2

Si el retorno requiere de más líneas y aún deseas utilizarlo de manera implícita, deberás envolver el cuerpo de la función entre paréntesis.

**const** suma = (num1, num2) => (

num1 + num2

)