**Para que sirve las keywords: var, let, const. Problema del scope.**

La declaración con var define una variable en el ámbito local actual (lease función) y se hereda a scopes descendientes por referencia. Si la variable es declarada fuera de una función, la variable será una variable global.

Un último apunte sobre el ámbito de var, es que una variable declarada exclusivamente dentro de una función no existe fuera de la misma, pues sería una variable local para esa función

**Hoisting**

Además del ámbito de aplicación visto antes, una variable declarada con var es sometida a hoisting («izamiento» o «levantamiento»): la declaración de la variable es «levantada» hasta el inicio del ámbito de aplicación, pero la asignación al valor permanece en el sitio donde se realice.

Si intentamos acceder a su valor antes de que se asigne el valor, obtendremos un valor indefinido (undefined).

**Declaración de variables con let y const**

let y const son dos formas de declarar variables en JavaScript introducidas en ES6 que reducen el ámbito de la variable a bloques (con var el ámbito era la función actual) y no admiten hoisting. Además, las variables declaradas con const no pueden ser reasignadas (aunque no significa que su valor sea inmutable, como veremos a continuación).

**let**

Un bloque en JavaScript se puede entender como «lo que queda entre dos corchetes», ya sean definiciones de funciones o bloques if, while, for y loops similares. Si una variable es declarada con let en el ámbito global o en el de una función, la variable pertenecerá al ámbito global o al ámbito de la función respectivamente, de forma similar a como ocurría con var.

Debido a este comportamiento, muchos desarrolladores se inclinan hacia let como la forma predeterminada de declarar variables en JavaScript y abandonar var (1, 2, 3), pues el scope más específico previene la sobreescritura de variables de forma accidental al declarar variables sin ensuciar el scope superior

**const**

El ámbito o scope para una variable declarada con const es, al igual que con let, el bloque, pero si la declaración con let previene la sobreescritura de variables, const directamente prohíbe la reasignación de valores (const viene de constant).

Pero que no se puedan reasignar no significa que sean inmutables. Si el valor de una variable constante es «algo» mutable, como un array o un objeto, se pueden cambiar los valores de sus elementos.

Por ejemplo, una variable constante se puede asignar a un objeto con determinadas propiedades. Aunque la variable no se pueda asignar a un nuevo valor, si se puede cambiar el valor de sus propiedades

**Resumen**

**Var** declara una variable de scope global o local para la función sin importar el ámbito de bloque. Permite hoisting.

**let** declara una variable de scope global, local para la función o de bloque. Es reasignable y no permite hoisting.

**const** declara una variable de scope global, local para la función o de bloque. No es reasignable, pero es mutable. No permite hoisting.