Let, Const

`Let` y **`Const`** son dos constructores de variables que **solucionan la pérdida del *scope* que sufre var al ser utilizada dentro de un bloque de código anónimo o condicional (`simple` o `loop`).**

`Let` introduce al *block scope*. La variable asignada con `let` **solo será accesible dentro del `for loop`**.

```
for(var i=0; i<5; i++) {
    console.log(i)
}
//var existe fuera del bloque de código del
for loop
console.log(i)

//----------------------------------------
//SOLUCIÓN:
for(let i=0; i<5; i++) {
    console.log(i)
}
console.log(i) //causaría error
```

`Const` es igual que `let`, con una gran diferencia: **no podemos re asignar su valor**.

```
var PI = 3.1515926
PI = 2.71     //no da error
console.log(PI)

let PI = 3.1515926
PI = 2.71     //no da error
console.log(PI)

const PI = 3.1515926
PI = 2.71     //da error
console.log(PI)
```

Algunos ejemplos de comportamiento de `var`, `let` y `const` en tres bloques de código conocidos:

```
//--------------------------------
//bloque de código anónimo
{
    //var a = 5
    //let a = 5
    const a = 5
    a = 55          // da error para variable a declarada con const
    console.log(a)
}
//console.log(a)    // da error para variable a declarada con let o const

//--------------------------------
//bloque de código condicional
if(true) {
```

...

```
    //var b = 6
    let b = 6
    console.log(b)
}
//console.log(b)    // da error para variable b declarada con let o const

//--------------------------------
//bloque de código funcional
function foo() {
    //var c = 7
    let c = 7
    console.log(c)
}
foo()
//console.log(c)    // da error para variable b declarada con var, let o const
```

Hoisting

En JavaScript las **var** son *“hoisted”* o “izadas”. Esto, como su nombre lo indica, quiere decir que una variable es *izada* (subida) **hasta el tope de la función o hasta llegar al Objeto global contenedor**. Atención que **solo es izada su declara-
ción** y su inicialización permanece en el mismo lugar.

A las funciones les pasa lo mismo, sus declaraciones son izadas al tope del contenedor donde están declaradas, sin embargo, `let` y `const` **no permiten “hoisted”**, como veremos en la siguiente slide.

```
saludo = 'hola!'
saludar()

function saludar() {
    saludo2 = 'que tal'
    console.log(saludo + ' ' + saludo2) // salida por consola: hola! que tal
    var saludo2
}
var saludo

// ------------------------------------------------
saludar()

function saludar() {
    console.log(saludo + ' ' + saludo2) // salida por consola: undefined undefined
    var saludo2 = 'que tal'
}
var saludo = 'hola!'
```

Si olvidamos poner el constructor de variable `var`, `let` o `const`, el *interpreter* de JavaScript **la va a declarar de forma global**. Este error silencioso se puede exponer **usando el modo estricto del compilador de JavaScript con `use strict` al principio del bloque que se quiera controlar**.

```
function programa() {
    var clave = 1234
}
programa()
//da error, clave está declarada en el scope de la función programa
console.log(clave)

//--------------------------------------
function programa() {
    clave = 1234    //olvidamos var (o let ó const)
}
```

```
programa()
console.log(clave)  //NO da error, clave queda declarada en el scope global

//--------------------------------------
'use strict'     //Habilitación del Modo Estricto del compilador de Javascript
function programa() {
    clave = 1234  //Da error en esta línea, por omisión del constructor de variable
}
programa()
console.log(clave)  //No se llega a ejecutar esta línea de código */
```