```
/* Declarando variables en distintos SCOPES */

var x = 'Scope global'; //Variable declarada en un scope global,

function ejemplo(){
  var x = 'Scope local'; //Variable con el mismo nombre declarada en un scope local (dentro de la función)
  console.log(x); //imprime en consola 'Scope local'
}

ejemplo()
console.log(x); //imprime en consola 'Scope global'

/*
Una variable declarada exclusivamente dentro de una función, no existe fuera de la misma.
*/

function ejemplo2(){
  var i = 'Scope local';
}
console.log(i); // ReferenceError: i is not defined


/* 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 )
*/

console.log(f); // undefined
var f = 1;

/*
En este caso quiere decir que si bien la variable f fue declarada en el programa, al momento de acceder a ella no tiene un valor asignado. Se podría leer de la siguiente manera:
*/

var j; // Variable declarada pero valor no definido
console.log(j); // undefined
j = 1; // Valor asignado
console.log(j); // 1

/* Si la variable no es declarada en abosluto obtenemos un ReferenceError, que no es lo mismo que obtener un valor indefinido */
```

```
console.log(b); //ReferenceError: b is not defined
var k = 1;

/* Debido a este comportamiento, se suele recomendar mover todas las declaraciones de variables al inicio del scope aunque no se asigne valor alguno, de esta forma se evitan estos posibles errores de ejecución.

El hoisting no es posible en variables declaradas con let o const; estas variables siempre darán un ReferenceError si se intenta acceder a ellas antes de que sean declaradas: */

console.log(z); // ReferenceError: Cannot access 'z' before initialization
let z = 2;

/************************************
DECLARACION 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

/*
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.
*/

let i = 0; // Declaro en un scope global la variable i y le asigno un valor
function ejemplo3(){
  i = 1; // tomo la variable ya creada y le asigno otro valor en un ámbito local
  let j = 2; // declaro otra variable en scope local y le asigno un valor
  if(true){
    console.log(i); // Devuelve 1
    console.log(j); // Devuelve 2
  }
}

/*
Pero si declaramos una variable con let dentro un bloque que a su vez está dentro de una función, la variable pertenece solo a ese bloque:
*/

function ejemplo4(){
```

```
  let i = 0; // Dentro de la función declaro la variable i y le asigno un valor
  if(true){
    let i = 1; // Dentro del condicional declaro una variable i que solo funciona-
ría dentro de éste bloque.
    console.log(i); // Devuelve 1, ya que dentro de éste bloque i tiene asignado es
e valor
  }
  console.log(i); // Devuelve 0, ya que pido el valor de la variable i declarada
dentro del bloque de la función
}

/*
Fuera del bloque donde se delcara con let, la variable no está definida:
*/

function ejemplo5(){
  if(true){
    let i = 1;
  }
  console.log(i); // ReferenceError: i is not defined
}

// 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).

Con let una variable puede ser reasignada:
*/

function pruebaLet(){
  let i = 0;
  if(true){
    i = 1;
  }
  console.log(i); // Devuelve 1
}

/*
Con const no es posible; si se intenta reasignar una variable constante se obtendrá
un error tipo TypeError:
*/

const i = 0;
i = 1; // TypeError: Assignment to constant variable

/*
```

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

//Pero eso lo veremos mas adelante...
```