javascript const vs var vs let , es una duda muy común entre todos los que desarrollados con JavaScript. Se trata de las tres formas de declarar variables con el lenguaje. Vamos a abordar cada una de ellas . La más común es el uso de “var” .

```
var nombre="pepe";

console.log(nombre);
```

Esta declaración tiene un problema , una vez creada la variable esta variable tendrá un ámbito global es decir podremos acceder a su valor si la solicitamos a través del objeto window.

```
var nombre="pepe";

console.log(nombre);

console.log(window.nombre);
```

Esto puede ser un problema porque no importa a que nivel declaramos la variable que su ámbito siempre será global.

```
var nombre="pepe";

console.log(nombre);
```

```
console.log(window.nombre);

for (var i=0;i<2;i++) {

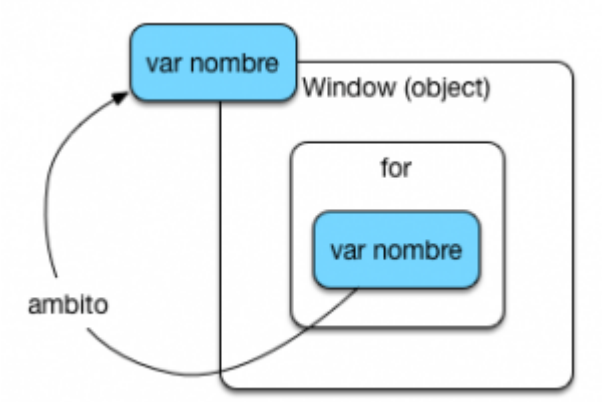
var edad=20;
}

console.log(window.edad);
```

Por ejemplo en este caso tanto el nombre como la edad se imprimen.

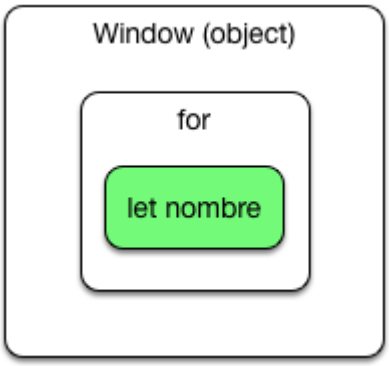
```
pepe
pepe
20
```

Aunque la variable se haya definido dentro de un bucle for su ámbito es global.

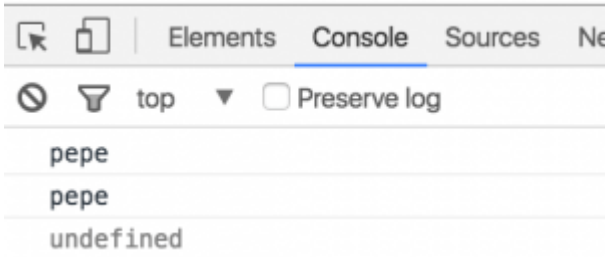


JavaScript const vs var vs let

Este es un problema habitual, para solventarlo deberemos usar la palabra reservada let que declarará la variable dentro de su propio ámbito.



Ahora el resultado cambiará y no podremos acceder desde el objeto windows a esta variable.



El último caso es el uso de const , cuando declaramos una variable con const , la variable no puede apuntar a otro elemento que no sea el inicialmente definido.

```
const otropersona= {nombre:"lucia",apellidos:"sanchez"};
```

```
otropersona.nombre="vanesa";

console.log(otropersona.nombre);

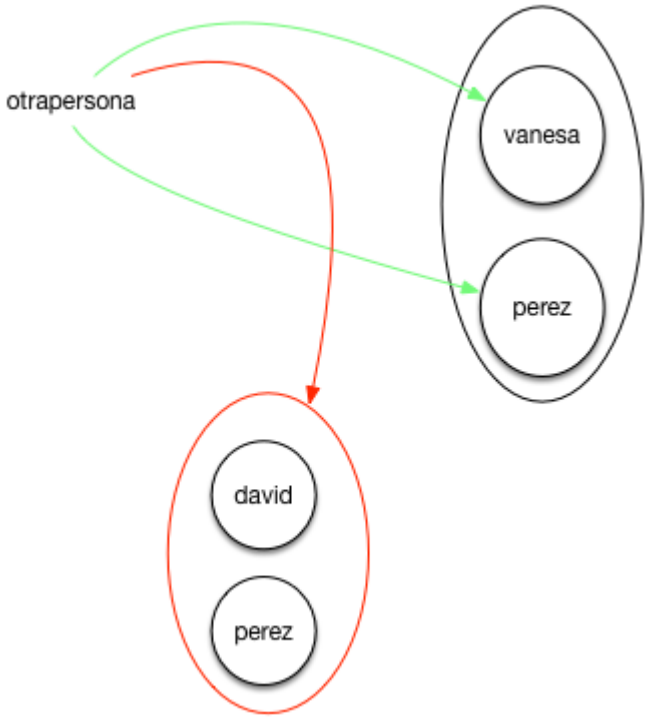
otrapersona={nombre:"david",apellidos:"perez"};

console.log(otropersona.nombre);
```

En este caso si que podemos cambiar el nombre de lucia por vanesa pero no podemos cambiar la variable para que apunte a otro objeto .

```
vanesa
vanesa
```

En ese caso hemos podido cambiar el nombre a vanesa pero no hemos podido variar el puntero al cual la variable apunta para imprimir los datos de david.



Estas son las diferencias entre cada una de las declaraciones de variables.

Otros artículos relacionados: JavaScript pure functions , JavaScript Prototypes