



Diferencias var, let y const

Trabajo I: Template HTML y CSS

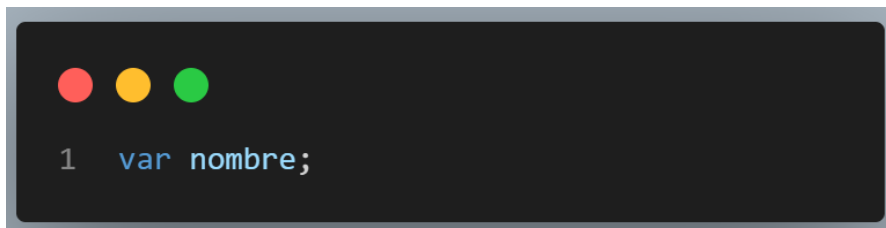
Introducción

JavaScript es un lenguaje de programación muy utilizado en el desarrollo de aplicaciones web. Una de las características más importantes de JavaScript es la capacidad de declarar variables y asignarles valores. En este documento, explicaremos las diferencias entre las palabras clave "var", "let" y "const" en JavaScript, y cómo se pueden utilizar para declarar variables y asignarles valores.

Variables en JavaScript

Antes de profundizar en las diferencias entre "var", "let" y "const", es importante entender cómo se declaran y se asignan valores a las variables en JavaScript.

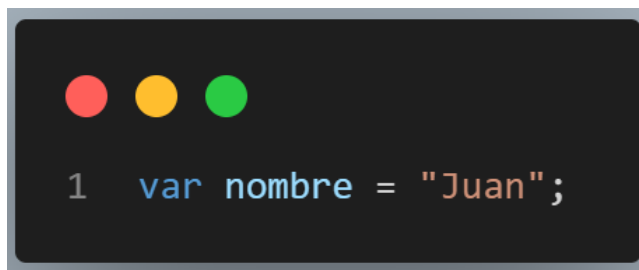
En JavaScript, las variables se declaran utilizando la palabra clave "var", "let" o "const", seguida de un nombre de variable que se utilizará para hacer referencia a ella. Por ejemplo, podemos declarar una variable llamada "nombre" utilizando la palabra clave "var", de la siguiente manera:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains a single line of JavaScript code:

```
1 var nombre;
```

```
1 var nombre;
```

También podemos asignar un valor a una variable utilizando el operador de asignación "=", de la siguiente manera:

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It contains a single line of JavaScript code:

```
1 var nombre = "Juan";
```

```
1 var nombre = "Juan";
```

En este ejemplo, hemos asignado el valor "Juan" a la variable "nombre". Ahora podemos utilizar la variable "nombre" para hacer referencia a este valor en nuestro código.

Diferencias entre var, let y const

Ahora que sabemos cómo declarar y asignar valores a las variables en JavaScript, podemos explorar las diferencias entre "var", "let" y "const".

La palabra clave "var" se utilizaba anteriormente para declarar variables en JavaScript, pero con la introducción de ECMAScript 6 (también conocido como ES6 o ES2015), se introdujeron dos

nuevas palabras clave: "let" y "const". Estas palabras clave tienen algunas diferencias clave en cómo se comportan las variables que declaran.

"var"

La palabra clave "var" se utiliza para declarar variables que tienen un ámbito de función, lo que significa que solo son visibles dentro de la función en la que se declaran. Si se declara una variable con "var" fuera de una función, se convierte en una variable global, lo que significa que es accesible desde cualquier lugar en el código.

Otra característica importante de las variables declaradas con "var" es que pueden ser reasignadas a otro valor en cualquier momento. Por ejemplo, podemos declarar una variable llamada "edad" con "var", asignarle un valor y luego reasignarla a otro valor más adelante en nuestro código, de la siguiente manera:



```
1  var edad = 20;  
2  edad = 30;
```

"let"

La palabra clave "let" se utiliza para declarar variables que tienen un ámbito de bloque, lo que significa que solo son visibles dentro del bloque en el que se declaran. Un bloque puede ser un cuerpo de función, un cuerpo de bucle, un cuerpo de condición, etc.

Al igual que las variables declaradas con "var", las variables declaradas con "let" también pueden ser reasignadas a otro valor en cualquier momento. Por ejemplo, podemos declarar una variable llamada "edad" con "let", asignarle un valor y luego reasignarla a otro valor más adelante en nuestro código, de la siguiente manera:

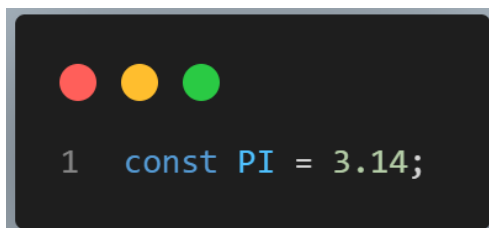


```
1  let edad = 20;  
2  edad = 30;
```

"const"

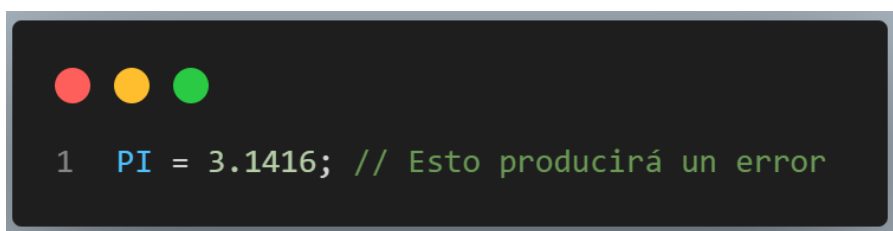
La palabra clave "const" se utiliza para declarar variables que también tienen un ámbito de bloque, pero a diferencia de las variables declaradas con "var" y "let", las variables declaradas con "const" no pueden ser reasignadas a otro valor después de su inicialización. Esto significa que el valor asignado a una variable declarada con "const" no puede cambiar durante la ejecución del programa.

Por ejemplo, podemos declarar una variable llamada "PI" con "const" y asignarle un valor de 3.14, de la siguiente manera:



```
1  const PI = 3.14;
```

Si intentamos reasignar un valor a la variable "PI", obtendremos un error en tiempo de ejecución:



```
1  PI = 3.1416; // Esto producirá un error
```

Scope global y local

El scope, también conocido como ámbito, es uno de los conceptos fundamentales de la programación en JavaScript. Se refiere al alcance o visibilidad de una variable en una parte específica del código. En otras palabras, el scope determina qué variables son accesibles y desde qué parte del código pueden ser accedidas.

En JavaScript, existen dos tipos de scope: scope global y scope local. El scope global se refiere a las variables que son declaradas fuera de cualquier función o bloque, mientras que el scope local se refiere a las variables que son declaradas dentro de una función o bloque.

Cuando se utiliza la palabra clave "var" para declarar una variable, esta variable es visible en todo el ámbito de la función o en todo el ámbito global si se declara fuera de una función. Esto significa que si se declara una variable con "var" dentro de una función, se puede acceder a ella desde cualquier lugar dentro de esa función, incluso desde dentro de los bloques if, while, for, etc.

Sin embargo, si se declara una variable con "var" dentro de un bloque, la variable se comporta como si se hubiera declarado en el ámbito de la función o global. Esto se debe a que JavaScript no tiene un scope de bloque para las variables declaradas con "var". Veamos un ejemplo:

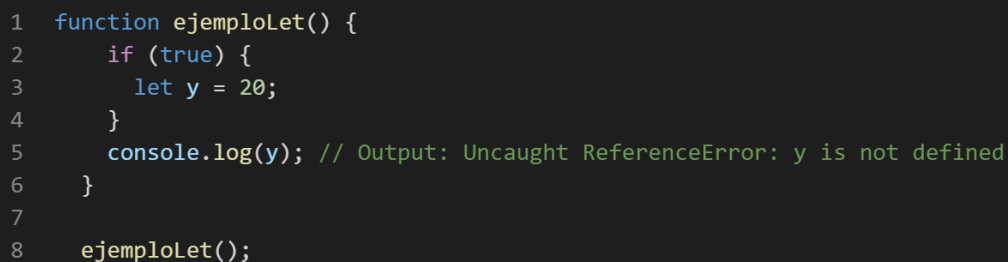


```
1  function ejemploVar() {  
2      if (true) {  
3          var x = 10;  
4      }  
5      console.log(x); // Output: 10  
6  }  
7  
8  ejemploVar();
```

En este ejemplo, hemos declarado una variable "x" con "var" dentro de un bloque if. Sin embargo, cuando intentamos acceder a "x" fuera del bloque if, todavía podemos acceder a ella, lo que demuestra que "x" tiene un alcance más amplio del que se esperaría en otros lenguajes de programación.

En cambio, cuando se utiliza la palabra clave "let" o "const" para declarar una variable, la variable solo es visible dentro del bloque en el que se declaró, incluyendo cualquier bloque interno. Esto significa que si se declara una variable con "let" o "const" dentro de una función o bloque, solo se puede acceder a ella dentro de ese bloque.

Veamos un ejemplo:



```
1 function ejemploLet() {
2   if (true) {
3     let y = 20;
4   }
5   console.log(y); // Output: Uncaught ReferenceError: y is not defined
6 }
7
8 ejemploLet();
```

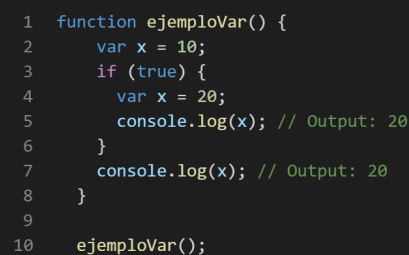
En este ejemplo, hemos declarado una variable "y" con "let" dentro de un bloque if. Cuando intentamos acceder a "y" fuera del bloque if, obtenemos un error, lo que demuestra que "y" no es visible fuera de ese bloque.

En resumen, el scope en JavaScript es importante para determinar la visibilidad de una variable en una parte específica del código. Cuando se utiliza la palabra clave "var" para declarar una variable, esta variable es visible en todo el ámbito de la función o global. Cuando se utiliza la palabra clave "let" o "const" para declarar una variable, la variable solo es visible dentro del bloque en el que se declaró. Es importante tener en cuenta estas diferencias al escribir código en JavaScript para evitar errores y comportamientos inesperados.

Ejemplos de código

Veamos algunos ejemplos de cómo se pueden utilizar "var", "let" y "const" en JavaScript.

Ejemplo 1: "var"



```
1 function ejemploVar() {
2   var x = 10;
3   if (true) {
4     var x = 20;
5     console.log(x); // Output: 20
6   }
7   console.log(x); // Output: 20
8 }
9
10 ejemploVar();
```

En este ejemplo, hemos declarado una variable "x" con "var" dentro de una función. Luego, dentro de un bloque de condición "if", hemos declarado otra variable "x" con "var" y le hemos asignado un valor de 20. Al imprimir el valor de "x" dentro del bloque de condición "if", obtenemos el valor 20. Luego, al imprimir el valor de "x" fuera del bloque de condición "if", obtenemos el mismo valor de 20, lo que demuestra que la variable "x" declarada dentro del bloque de condición "if" también es visible fuera de ese bloque.

Ejemplo 2: "let"



```
1 function ejemploLet() {
2   let x = 10;
3   if (true) {
4     let x = 20;
5     console.log(x); // Output: 20
6   }
7   console.log(x); // Output: 10
8 }
9
10 ejemploLet();
```

Ejemplo 3: "const"



```
1 function ejemploConst() {
2   const PI = 3.14;
3   PI = 3.1416; // Esto producirá un error
4   console.log(PI);
5 }
6
7 ejemploConst();
```

En este ejemplo, hemos declarado una variable "PI" con "const" y le hemos asignado un valor de 3.14. Luego, hemos intentado reasignar un valor a la variable "PI", lo que produce un error en el tiempo de ejecución. Al imprimir el valor de "PI", obtenemos el valor original de 3.14, lo que demuestra que la variable "PI" no cambió.

Conclusiones

En resumen, las palabras clave "var", "let" y "const" se utilizan para declarar variables en JavaScript, pero tienen diferencias importantes en cómo se comportan las variables que declaran. "var" se utiliza para declarar variables con ámbito de función o global y pueden ser reasignadas a otro valor en cualquier momento. "let" se utiliza para declarar variables con ámbito de bloque y también pueden ser reasignadas a otro valor en cualquier momento. "const" se utiliza para declarar variables con ámbito de bloque y no pueden ser reasignadas a otro valor después de su inicialización.

Es importante entender estas diferencias para poder utilizar correctamente estas palabras clave en nuestro código. Las variables declaradas con "var" pueden ser útiles en algunas situaciones, pero a menudo pueden causar problemas debido a su ámbito de función o global. Las variables declaradas con "let" y "const" son más seguras y fáciles de entender debido a su ámbito de bloque y la imposibilidad de reasignar una variable declarada con "const".

Espero que este documento haya sido útil para comprender las diferencias entre "var", "let" y "const" en JavaScript y cómo se pueden utilizar en nuestro código. Recuerda que es importante comprender los conceptos básicos de JavaScript para poder escribir código efectivo y eficiente.