Tema 2 Introducción al JavaScript.

Practica 02-02b JS var, let y const

Nombre		Curso	
Apellidos		Fecha	

Con ES2015 (ES6) aparecieron muchas características nuevas y llamativas. Ahora, desde el año 2020, se supone que muchos desarrolladores de JavaScript se han familiarizado con estas características y han empezado a utilizarlas.

Una de las características que llegaron con ES6 es la adición de let y const, que se pueden utilizar para la declaración de variables. La pregunta es, ¿qué las hace diferentes del viejo var que hemos venido utilizando? Si todavía no lo tienes claro, este artículo es para ti.

Resumen

- Las declaraciones var tienen un ámbito global o un ámbito función/local, mientras que let y const tienen un ámbito de bloque.

- Las variables var pueden ser modificadas y re-declaradas dentro de su ámbito;
- las variables let pueden ser modificadas, pero no re-declaradas;
- las variables const no pueden ser modificadas ni re-declaradas.

- Todas ellas se elevan a la parte superior de su ámbito. Pero mientras que las variables var se inicializan con undefined, let y const no se inicializan.

- Mientras que var y let pueden ser declaradas sin ser inicializadas, const debe ser inicializada durante la declaración.

Var

Antes de la llegada de ES6, las declaraciones var eran las que mandaban. Sin embargo, hay problemas asociados a las variables declaradas con var. Por eso fue necesario que surgieran nuevas formas de declarar variables. En primer lugar, vamos a entender más sobre var antes de discutir esos problemas.

Ámbito de var

El ámbito, significa esencialmente dónde están disponibles estas variables para su uso. Las declaraciones var tienen un ámbito global o un ámbito de función/local.

El ámbito es global cuando una variable var se declara fuera de una función. Esto significa que cualquier variable que se declare con var fuera de una función está disponible para su uso en toda la pantalla.

var tiene un ámbito local cuando se declara dentro de una función. Esto significa que está disponible y solo se puede acceder a ella dentro de esa función.

Para entenderlo mejor, mira el siguiente ejemplo.

```
var saludar = "hey, hola";

function nuevaFuncion() {
  var hola = "hola";
}
```

Aquí, saludar tiene un ámbito global porque existe fuera de la función mientras que hola tiene un ámbito local. Así que no podemos acceder a la variable hola fuera de la función. Así que si realizamos esto:

```
var tester = "hey, hola";

function nuevaFuncion() {
  var hola = "hola";
}
console.log(hola); // error: hola is not defined
```

Obtendremos un error que se debe a que hola no está disponible fuera de la función.

Las variables con var se pueden volver a declarar y modificar
Esto significa que podemos hacer esto dentro del mismo ámbito y no obtendremos un error.

```
var saludar = "hey, hola";
var saludar = "dice Hola tambien";
```

y esto también

```
var saludar = "hey, hola";
saludar = "dice Hola tambien";
```

Hoisting de var
Hoisting es un mecanismo de JavaScript en el que las variables y declaraciones de funciones se mueven a la parte superior de su ámbito antes de la ejecución del código. Esto significa que si hacemos esto:

```
console.log (saludar);
var saludar = "dice hola"
```

se interpreta así:

```
var saludar;
console.log(saludar); // saludar is undefined
saludar = "dice hola"
```

Entonces las variables con var se elevan a la parte superior de su ámbito y se inicializan con un valor de undefined.

Problema con var

Hay una debilidad que viene con var. Usaré el ejemplo de abajo para explicarlo:

```
var saludar = "hey, hola";
var tiempos = 4;
```


```
if (tiempos > 3) {
    var saludar = "dice Hola tambien";
}

console.log(saludar) // "dice Hola tambien"
```

Por lo tanto, como tiempos > 3 devuelve true, saludar se redefine para "dice Hola tambien". Aunque esto no es un problema si quieres redefinir saludar a conciencia, se convierte en un problema cuando no te das cuenta de que la variable saludar ha sido definida antes.

Si has utilizado saludar en otras partes de tu código, puede que te sorprenda la salida que puedes obtener. Esto probablemente causará muchos errores en tu código. Por eso son necesarios let y const.

Let

let es ahora preferible para la declaración de variables. No es una sorpresa, ya que es una mejora de las declaraciones con var. También resuelve el problema con var que acabamos de cubrir. Consideremos por qué esto es así.

let tiene un ámbito de bloque

Un bloque es un trozo de código delimitado por {}. Un bloque vive entre llaves. Todo lo que está dentro de llaves es un bloque.

Así que una variable declarada en un bloque con let solo está disponible para su uso dentro de ese bloque. Permíteme explicar esto con un ejemplo:

```
let saludar = "dice Hola";
let tiempos = 4;

if (tiempos > 3) {
    let hola = "dice Hola tambien";
    console.log(hola);// "dice Hola tambien"
}
console.log(hola) // hola is not defined
```

Vemos que el uso de hola fuera de su bloque (las llaves donde se definió) devuelve un error. Esto se debe a que las variables let tienen un alcance de bloque.

let puede modificarse pero no volver a declararse.

Al igual que var, una variable declarada con let puede ser actualizada dentro de su ámbito. A diferencia de var, una variable let no puede ser re-declarada dentro de su ámbito. Así que mientras esto funciona:

```
let saludar = "dice Hola";
saludar = "dice Hola tambien";
```

esto devolverá un error:

```
let saludar = "dice Hola";
let saludar = "dice Hola tambien"; // error: Identifier 'saludar' has already been declared
```

Sin embargo, si la misma variable se define en diferentes ámbitos, no habrá ningún error:

```
let saludar = "dice Hola";
if (true) {
  let saludar = "dice Hola tambien";
  console.log(saludar); // "dice Hola tambien"
}
console.log(saludar); // "dice Hola"
```

¿Por qué no hay ningún error? Esto se debe a que ambas instancias son tratadas como variables diferentes, ya que tienen ámbitos diferentes.

Este hecho hace que let sea una mejor opción que var. Cuando se utiliza let, no hay que preocuparse de sí se ha utilizado un nombre para una variable antes, puesto que una variable solo existe dentro de su ámbito.

Además, como una variable no puede ser declarada más de una vez dentro de un ámbito, entonces el problema discutido anteriormente que ocurre con var no sucede.

Hoisting de let

Al igual que var, las declaraciones let se elevan a la parte superior. A diferencia de var que se inicializa como undefined, la palabra clave let no se inicializa. Sí que si intentas usar una variable let antes de declararla, obtendrás un Reference Error.

Const

Las variables declaradas con const mantienen valores constantes. Las declaraciones const similitudes con las declaraciones let.

Las declaraciones const tienen un ámbito de bloque

Al igual que las declaraciones let, solamente se puede acceder a las declaraciones const dentro del bloque en el que fueron declaradas.

const no puede modificarse ni volver a declararse

Esto significa que el valor de una variable declarada con const s el mismo dentro de su ámbito. No se puede actualizar ni volver a declarar. Así que si declaramos una variable con const, no podemos hacer esto:

```
const saludar = "dice Hola";
saludar = "dice Hola tambien";// error: Assignment to constant variable.
```

ni esto:

```
const saludar = "dice Hola";
const saludar = "dice Hola tambien";// error: Identifier 'saludar' has already been declared
```

Por lo tanto, toda declaración const, debe ser inicializada en el momento de la declaración.

Este comportamiento es algo diferente cuando se trata de objetos declarados con const. Mientras que un objeto const no se puede actualizar, las propiedades de este objeto sí se pueden actualizar. Como resultado, si declaramos un objeto const como este:


```
   const saludar = {
      mensaje: "dice Hola",
      tiempos: 4
   }
```

mientras que no podemos hacer esto:

```
   saludar = {
      palabras: "Hola",
      numero: "cinco"
   } // error:  Assignment to constant variable.
```

podemos hacer esto:

```
   saludar.mensaje = "dice Hola tambien";
```

Esto actualizará el valor de saludar.mensaje sin devolver errores.

Hoisting de const

Al igual que let, const las declaraciones const se elevan a la parte superior, pero no se inicializan.

En caso de que no hayas notado las diferencias, aquí están:

1. Crea un archivo "02-02-JSTiposdeDatos.htm" comentalo de las dos maneras posibles
2. Usa la función alert para enseñar el mensaje holamundo
3. en el un archivo y declara varias variables en una sola linea
4. que caracteres y nombres están prohibidos como nombre de variables ?
5. Crea un ejemplo para cada tipos de datos (Crea una tabla en html5 en el documento htm)

Cadena	
Número	
Boolean	
Null	
Object	
arrays	
Function	

6. Usa la función typeof para mostrar el tipo de los códigos
7. Ejecuta estos datos y muestralos en un documento htm
   var a=true;
   var b=hola;
   var c=1;

4+4	
"4"+ 4	
a+c	
b+c	
2.9e3+1	
2.9e3 + b	
2.9e3 + a	

8. Usa la función parseFloat y parseInt para convertir explícitamente los valores 2.9e3, 30.1e-3, 4,6,3000000
9. usa la función length para hallar el tamaño de las cadenas "123456"

10. Escribe el código htm en esta tabla
