Manejo de la sintaxis del lenguaje

--

Fundamentos básicos

--

Variables

1. Variables

En la mayoría de documentación sobre definición de variables, no se habla de let, una incorporación relativamente reciente al lenguaje. Este artículo explica el uso de esta palabra clave, muy interesante en el uso de las variables: https://lenguajejs.com/javascript/fundamentos/variables/

Aquí tienes una copia del texto de la web con algunas rectificaciones ya que contiene algunos errores de conceptos entre declaración de variable y asignación de valores a una variables.

En javascript es muy sencillo declarar y utilizar variables, pero aunque sea un procedimiento simple, hay que tener una serie de conceptos previos muy claros para evitar futuras confusiones, sobre todo si estamos acostumbrados a otros lenguajes más tradicionales.

a. Declaración

En programación, las variables son espacios donde se puede guardar información y asociarla a un determinado nombre. De esta forma, cada vez que se consulte ese nombre posteriormente, te devolverá la información que contiene. La primera vez que se realiza este paso se suele llamar declarar una variable.

En Javascript, es obligatorio declarar una variable antes de usarla. Una variable definida pero sin valor inicial asignado (sin valor inicial), contendrá un valor especial: undefined, que significa que su valor no está definido aún, o lo que es lo mismo, que no contiene información:

```
var a; // Declaramos una variable "a", pero no le asociamos ningún contenido.
var b = 0; // Declaramos una variable de nombre "b", con valor inicial 0.

console.log(b); // Muestra 0 (el valor guardado en la variable "b")
console.log(a); // Muestra "undefined" (no hay valor guardado en la variable "a")
```

Como se puede observar, hemos utilizado console.log() para consultar la información que contienen las variables indicadas.

OJO: Las mayúsculas y minúsculas en los nombres de las variables de Javascript importan. Es un lenguaje sensible a las mayúsculas. No es lo mismo una variable llamada precio que una variable llamada Precio, pueden contener valores diferentes.

Si tenemos que declarar muchas variables consecutivas, una buena práctica suele ser escribir sólo el primer var y separar por comas las diferentes variables con sus

respectivos contenidos (método 3). Aunque se podría escribir todo en una misma línea (método 2), con el último método el código es mucho más fácil de leer:

```
// Método 1: Declaración de variables de forma independiente
var a = 3;
var c = 1;
var d = 2;

// Método 2: Declaración masiva de variables con el mismo var
var a = 3, c = 1, d = 2;

// Método 3: Igual al anterior, pero mejorando la legibilidad del código
var a = 3,
    c = 1,
    d = 2;
```

Como su propio nombre indica, una variable puede variar su contenido, ya que aunque contenga una cierta información, se puede volver a cambiar. A esta acción ya no se le llama inicialización de una variable, sino asignación. En el código se puede diferenciar porque se omite la palabra reservada var:

```
var a = 40; // Inicializamos la variable "a" al valor 40.
a = 50; // Asignamos el valor numérico 50.
//La variable pasa a contener 50 en lugar de 40.
```

b. Ámbito

Ámbitos de variables: var

Cuando declaramos una variable al principio de nuestro programa y le asignamos un valor, ese valor generalmente está disponible a lo largo de todo el programa. Sin embargo, esto puede variar dependiendo de múltiples factores.

Se conoce como ámbito de una variable a la zona donde esa variable sigue existiendo.

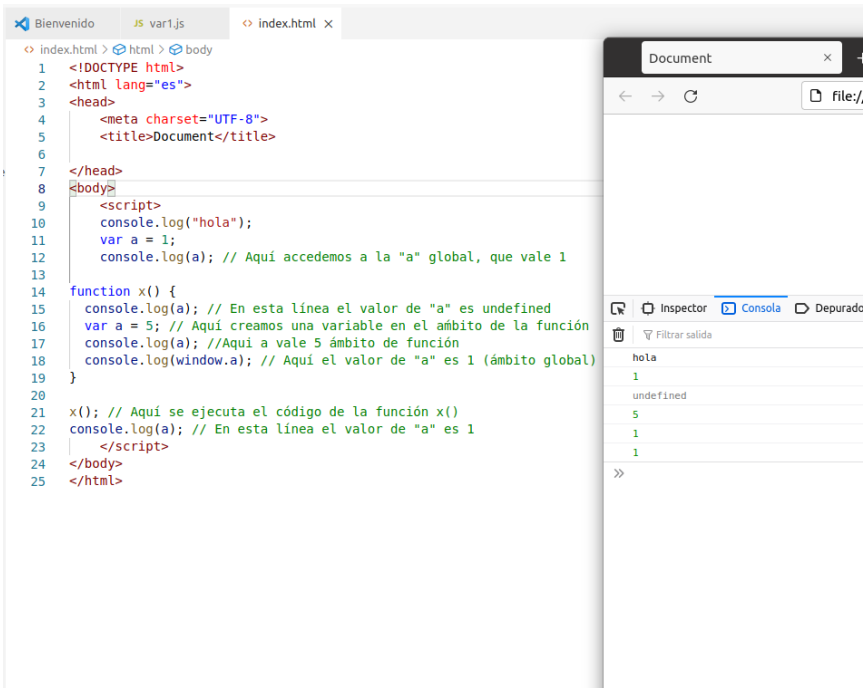
Por ejemplo, si consultamos el valor de una variable antes de inicializarla, tiene tipo undefined:

```
var e;
console.log(e); // Muestra "undefined", en este punto la variable "e"
                //no está inicializada
var e = 40;
console.log(e); // Aquí muestra 40, existe porque ya se ha inicializado anteriormente
```

El ejemplo anterior, además de aclarar el concepto de undefined, es un ejemplo de ámbito global. El ámbito de la variable “e” comienza a partir de su inicialización y "vive" hasta el final del programa.
A esto se le llama ámbito global y es el ejemplo más sencillo. En el enfoque tradicional de Javascript, es decir, cuando se utiliza la palabra clave var para declarar variables, existen dos ámbitos principales: ámbito global y ámbito a nivel de función. Se puede apreciar en el siguiente ejemplo:

```
var x=7;
Function f() {
    var x=9;
}
console.log(x);  //Produce 7
```

Observemos el siguiente ejemplo:

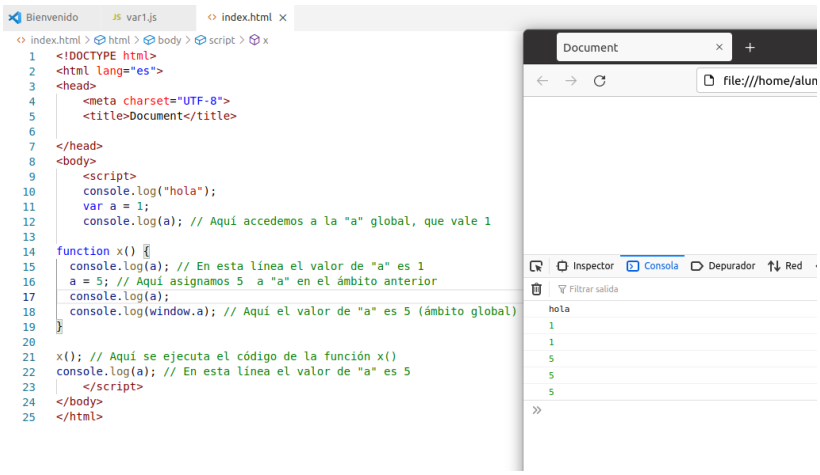


En el ejemplo anterior vemos que el valor de a dentro de una función no es el 1 inicial, sino que estamos en otro ámbito diferente donde la variable a anterior no existe: un ámbito a nivel de función. Mientras estemos dentro de una función, las variables inicializadas en ella estarán en el ámbito de la propia función.

OJO: Podemos utilizar el objeto especial window para acceder directamente al ámbito global independientemente de donde nos encontremos. Esto ocurre así porque las variables globales se almacenan dentro del objeto window (la pestaña actual del navegador web).

En el ejemplo siguiente se omite el var dentro de la función, y vemos que en lugar de crear una variable en el ámbito de la función, se modifica el valor de la variable a nivel

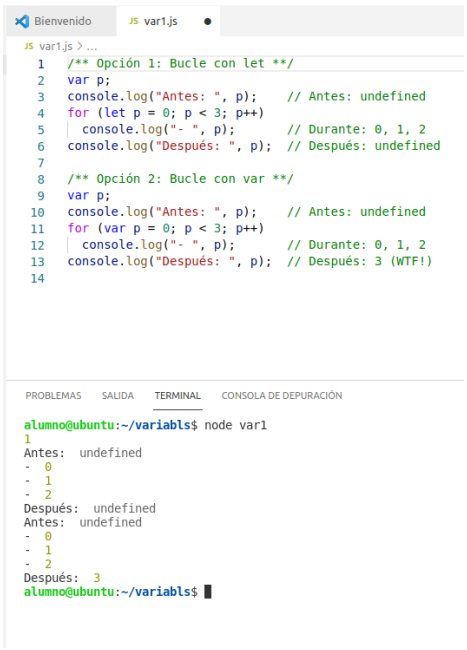
global. Dependiendo de dónde y cómo accedemos a la variable a, obtenemos un valor u otro.



Siempre que sea posible se debería utilizar let y const (ver a continuación), en lugar de var. Declarar variables mediante var se recomienda en fases de aprendizaje o en el caso de que se quiera mantener compatibilidad con navegadores muy antiguos utilizando ECMAScript 5, sin embargo, hay estrategias mejores a seguir que utilizar var en la actualidad.

Ámbitos de variables: let

En las versiones modernas de Javascript (ES6 o ECMAScript 2015) o posteriores, se introduce la palabra clave let en sustitución de var. Con ella, en lugar de utilizar los ámbitos globales y a nivel de función (var), utilizamos los ámbitos clásicos de programación: ámbito global y ámbito local. La diferencia se puede ver claramente en el uso de un bucle for con var y con let:



Vemos que utilizando let la variable p sólo existe dentro del bucle, ámbito local, mientras que utilizando var, la variable p sigue existiendo fuera del bucle, ámbito global.

c. Constantes

De forma tradicional, Javascript no incorporaba constantes. Sin embargo, en ECMAScript 2015 (ES6) se añade la palabra clave const, que inicializada con un valor concreto, permite crear “variables” con valores que no pueden ser cambiados.

```
const NAME = "Marta";
console.log(NAME);
NAME=”Alicia”; //produce error
```

En el ejemplo anterior vemos un ejemplo de const, que funciona de forma parecida a let. Una buena práctica es escribir el nombre de la constante en mayúsculas, para identificar rápidamente de que se trata de una constante y no de una variable, cuando leemos código ajeno.

Realmente, las constantes de Javascript son variables inicializadas a un valor específico que no se puede modificar posteriormente. No confundir con valores inmutables, ya que como veremos posteriormente, los objetos sí pueden ser modificados aún siendo constantes.

d. Identificadores y palabras reservadas

Una palabra reservada es un término especial del lenguaje con significado especial, como var o let. Estas palabras no se pueden utilizar para dar nombres a elementos que el programador necesita en su código, los identificadores.

Además los identificadores deben cumplir las siguientes reglas:

i. Un identificador de JavaScript debe comenzar con una letra, un guión bajo (_) o un signo de dólar ($). Los siguientes caracteres también pueden ser dígitos (0-9).
ii. Dado que JavaScript distingue entre mayúsculas y minúsculas, las letras incluyen los caracteres "A" a "Z" (mayúsculas), así como "a" a "z" (minúsculas).
iii. Puedes utilizar la mayoría de las letras ISO 8859-1 o Unicode como å y ü en los identificadores.

e. Resumen

➢ Una variable no declarada produce error.

- Declarar una variable sin asignarle valor genera una variable de tipo undefined.
- Las variables definidas con var tienen ámbito de ejecución:
  - programa->global
  - función->local
  - bloque->el valor sobrevive al ámbito del bloque
- Las variables definidas con let o const tienen ámbito de bloque.
- JavaScript es un lenguaje case-sensitive y tiene reglas para formar los identificadores.