DWEC – Javascript Web Cliente.

JavaScript 01 A – Sintaxis (I)

Variables

Javascript es un lenguaje débilmente tipado. Esto significa que no se indica de qué tipo es una variable al declararla, incluso puede cambiar su tipo a lo largo de la ejecución del programa. Ejemplo:

```
let miVariable;     // declaro miVariable y como no se asignó un valor valdrá undefined
miVariable='Hola';  // ahora su valor es 'Hola', por tanto contiene una cadena de texto
miVariable=34;      // pero ahora contiene un número
miVariable=[3, 45, 2];  // y ahora un array
miVariable=undefined;   // para volver a valer el valor especial undefined
```

EJERCICIO: Ejecuta en la consola del navegador las instrucciones anteriores y comprueba el valor de ***miVariable*** tras cada instrucción (para ver el valor de una variable simplemente ponemos en la consola su nombre: `miVariable`

Ni siquiera estamos obligados a declarar una variable antes de usarla, aunque es recomendable para evitar errores que nos costará depurar.

Las variables se declaran con **let** (lo recomendado desde ES2015), aunque también pueden declararse con **var**.

La diferencia es que con *let* la variable sólo existe en el bloque en que se declara.

Mientras que con *var* el **ámbito** (***scope***) de la variable es global, se extiende. Es decir, existe en toda la función o ámbito en el que se declara:

```
if (edad > 18) {
  let textoLet = 'Eres mayor de edad';
  var textoVar = 'Eres mayor de edad';
} else {
  let textoLet = 'Eres menor de edad';
  var textoVar = 'Eres menor de edad';
}
console.log(textoLet);  // mostrará undefined porque fuera del if no existe la variable
console.log(textoVar);  // mostrará la cadena
```

Cualquier variable que no se declara dentro de una función (o si se usa sin declarar) es *global*. Debemos siempre intentar NO usar variables globales.

Además, como podemos comprobar con el siguiente ejemplo, las variables declaradas con var pueden duplicarse sin que se produzca error, y que a la larga puede acarrear muchos problemas.

```
// CREACIÓN DE VARIABLES CON LET

/* let declara una variable limitando su ámbito (scope) al bloque,
   declaración o expresión  donde se está usando. */

// SINTAXIS: let nombreVariable [= valor];

var persona = "Santi";
var persona = "Profesor";
console.log (persona);

let persona2 = "Ana";
//let persona2 = "Jefa de Estudios"; //Esta instrucción devuelve un error
console.log (persona2);
```

Se recomienda que Los nombres de las variables sigan la sintaxis *camelCase* (ej.: *miPrimeraVariable*).

Desde ES2015 también podemos declarar constantes con **const**. Se les debe dar un valor al declararlas y si intentamos modificarlo posteriormente se produce un error.

NOTA: en la página de Babel podemos teclear código en ES2015 y ver cómo quedaría una vez ***transpilado*** a ES5.

Cuando veamos "next generation JavaScript" se refiere a la próxima versión de Javascript pendiente de ser aceptada

Use Strict

Para evitar problemas futuros, y que no se produzca algún error si no declaramos una variable, incluiremos al principio de nuestro código la instrucción "use estrict", que nos obliga a declarar las variables.

```
'use strict';
```

Veamos un ejemplo:

```
// USE STRICT O MODO ESTRICTO

/* "use strict" es una línea que indica que el código debe ser usado "en modo estricto",
   es decir, no se pueden utilizar variables no declaradas.
   Fuera de una función tiene ámbito global; dentro de ella, local (el de la función).
*/
```

```
// SINTAXIS: "use strict";

//"use strict"; // si aplicamos este código da error al no declarar persona2

persona2 = "Santi";
let nacimiento;

function informar (){
    "use strict";
    let persona = "Santi";
    nacimiento = "1915";
    console.log(persona + " nacio en "+nacimiento);
}

informar();
```

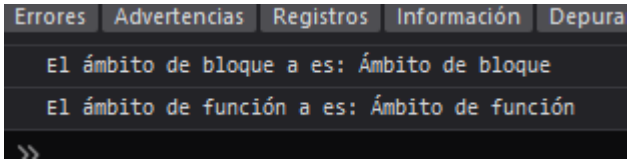
Otro ejemplo de ámbito de variables:

Caso 1º: Usamos dos variables (a) definidas con let, cada una queda circunscrita al ámbito donde ha sido creada.

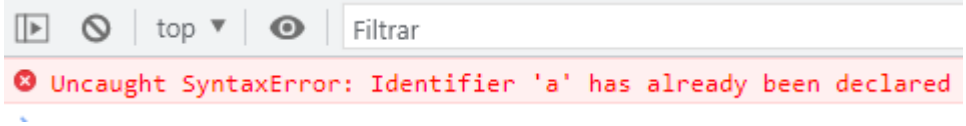
```
/ ÁMBITO DE VARIABLES

/* El ámbito de una variable (scope) es la zona del programa en la que se define.
   Javascript define dos ámbitos para variables: local y global.
   Mediante var podemos definir como ámbito local el ámbito de una función.
   Con let, por el contrario, podemos diferenciar también el ámbito de bloque. */

  function verAmbito(){
    "use strict";
    let a = "Ámbito de función";
    if (true){
      let a = "Ámbito de bloque";
      console.log ("El ámbito de bloque a es: "+a);
    }
    console.log ("El ámbito de función a es: "+a);
  }
  verAmbito();
```

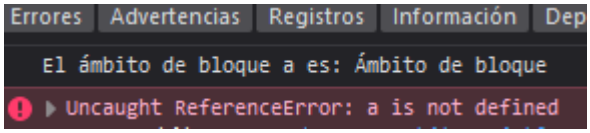


Si en el caso 1º cambiamos el **let** (dentro del if) por **var**, da error de variable ya declarada, puesto que ahora var extiende su scope a toda la función.



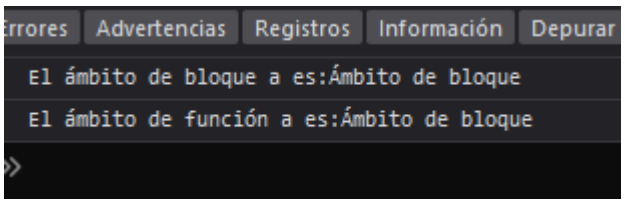
Caso 2º: Usamos una variable (a) definida con **let** dentro de un **if**, su ámbito queda reducido al bloque if, por tanto, fuera del bloque daría error.

```
function verAmbito(){
   "use strict";
   if (true){
     let a = "Ámbito de bloque";
     console.log ("El ámbito de bloque a es: "+a);
   }
   console.log ("El ámbito de función a es: "+a); // produce error
 }
 verAmbito();
```



Caso 3º: Usamos una variable (a) definida con **var** dentro de un **if**, su ámbito se extiende a toda la función.

```
function verAmbito(){
    "use strict";
    //let a = "Ámbito de función"; // SIMILAR AL CASO: var a = "Ámbito de función";
    if (true){
      var a = "Ámbito de bloque";
      console.log ("El ámbito de bloque a es:"+a);
    }
    console.log ("El ámbito de función a es:"+a);
  }
  verAmbito();
// console.log ("El ámbito de función a es: "+a); // ERROR, pues no llega el ámbito
```



Nota: si en el caso 3º cambiamos var por let (estamos igual que el caso

Conclusiones: Se pueden trabajar con **var** y **let** a la vez, pero sería peligroso. En la medida de lo posible trabajaremos con **let**, e incluso con **const** cuando sepamos que esa variable nunca debe cambiar. Además, debemos pensar en que al usar **const** y **let** (su ámbito queda reducido a nuestros bloques) nunca interferirán con otras funciones o código de otro programador en nuestra aplicación.

A modo de repaso recordemos que en este lenguaje de programación no es obligado declarar las variables.

En el caso de ECMAScript 5 usamos la palabra reservada var.

En el caso de ECMAScript 6+ además de var también podemos usar let y const.

Variables locales y variables globales

Una variable global se puede usar en cualquier lugar del script.

Una variable local sólo tiene validez dentro del ámbito de una función.

Las normas que aplican para definir el tipo de variable que estamos usando se resumen con 3 normas sencillas:

- Cualquier variable declarada o no declarada en la raíz de un script es siempre de ámbito global.
- Cualquier variable declarada dentro de una función es de ámbito local.
- Una variable NO declarada dentro de una función adquiere ámbito global.

En caso de duda, que es cuando la variable no ha sido declarada en un ámbito determinado, la variable adquiere comportamiento global.

Casos de uso de las propiedades del ámbito de las variables

En javascript se pueden dar algunos casos curiosos:

- Dentro de una función podemos sobrescribir el valor de una variable global. Esto puede suceder por ejemplo por error si nos hemos olvidado de declarar una variable… y cargarnos un dato externo a la función sin darnos cuenta.
- Cuando declaramos una variable local con el mismo nombre que una variable global, temporalmente la variable local tiene prioridad sobre la global. Dentro de la función usaremos la variable local. Deberemos tratarlas como si temporalmente la variable global hubiera dejado de existir para nosotros.
- Hay que recordar que una variable declarada dentro de los paréntesis de declaración de la función se considera declarada de tipo local y que por cuestiones de sintaxis nunca va acompañada de la palabra reservada let.

Variables constantes (const)

Las variables constantes en Javascript (**const**) tienen ámbito de bloque al igual que las variables definidas utilizando **let**. Es importante tener en cuenta que el valor de una constante no puede variar (reasignarse), por tanto, se asignan en el momento en que se declaran. Para diferenciarlo de las variables conviene utilizar ***TODOMAYÚSCULAS***.

Funciones

Se declaran con la palabra reservada **function** y se les pasan los parámetros entre paréntesis. La función puede devolver un valor usando **return** (si no tiene *return* es como si devolviera *undefined*).

Puede usarse una función antes de haberla declarado por el comportamiento de Javascript llamado *hoisting*: el navegador primero carga todas las funciones y mueve las declaraciones de las variables al principio y luego ejecuta el código.

> EJERCICIO **mensaje**: Realiza una función que te pida que escribas algo y muestre un alert diciendo 'Has escrito…' y el valor introducido. Pruébala en la consola (pegas allí la función y luego la llamas desde la consola)

Parámetros

Si se llama a una función con menos parámetros de los declarados el valor de los parámetros no pasados será *undefined*:

```
function potencia(base, exponente) {
    console.log(base);            // muestra 4
    console.log(exponente);       // muestra undefined
    let valor=1;
```

```
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}

let resultado = potencia(4); // devolverá 1 ya que no se ejecuta el for
console.log(resultado);
//console.log(potencia(4));    // devolverá 1 ya que no se ejecuta el for
```

Podemos dar un **valor por defecto** a los parámetros por si no los pasan asignándoles el valor al definirlos:

```
function potencia(base, exponente=2) {
    console.log(base);            // muestra 4
    console.log(exponente);       // muestra 2 la primera vez y 5 la segunda
    let valor=1;
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}

console.log(potencia(4));         // mostrará 16 (4^2)
console.log(potencia(4,5));       // mostrará 1024 (4^5)
```

NOTA: En ES5 para dar un valor por defecto a una variable se hacía:

```
function potencia(base, exponente) {
    exponente = exponente || 2;   // si exponente vale undefined se la asigna el valor 2
...
```

También es posible acceder a los parámetros desde el array **arguments[]** si no sabemos cuántos parámetros recibiremos:

```
function sumar () {
    var result = 0;
    for (var i=0; i<arguments.length; i++)
        result += arguments[i];
    return result;
}

console.log(sumar(4, 2));                    // mostrará 6
console.log(sumar(4, 2, 5, 3, 2, 1, 3));     // mostrará 20
```

En Javascript las funciones son un tipo de datos más, por lo que podemos hacer cosas como pasarlas como argumento o asignarlas a una variable:

```
const cuadrado=function(value){
    return value * value;
}
function aplicarFuncion(dato, funcion_a_aplicar){
```

```
    return funcion_a_aplicar(dato);
}
aplicarFuncion(3,cuadrado); // devolverá 9 (3^2)
```

A este tipo de funciones se llama *funciones de primera clase* y son típicas de lenguajes funcionales.

Funciones anónimas

Como acabamos de ver podemos definir una función sin darle un nombre. Dicha función puede asignarse a una variable, autoejecutarse o asignasrse a un manejador de eventos. Ejemplo:

```
let holaMundo = function() {
    alert('Hola mundo!');
}

holaMundo();        // se ejecuta la función
```

Como vemos, asignamos una función a una variable de forma que podamos "ejecutar" dicha variable.

Arrow functions (funciones flecha)

ES2015 permite declarar una función anónima de forma más corta. Ejemplo sin *arrow function*:

```
let potencia = function(base, exponente) {
    let valor=1;
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}
```

Al escribirla con la sintaxis de una *arrow function* lo que hacemos es:

- Eliminamos la palabra *function*
- Si sólo tiene 1 parámetro podemos eliminar los paréntesis de los parámetros
- Ponemos el símbolo =>
- Si la función sólo tiene 1 línea podemos eliminamr las { } y la palabra *return*

El ejemplo con *arrow function*:

```
let potencia = (base,exponente) => {
    let valor=1;
    for (let i=1; i<= exponente; i++){
        valor= valor * base
    }
    return valor
}
```

Otro ejemplo, sin *arrow function*:

```
let cuadrado= function(base) {
    return base * base;
}
```

conn *arrow function*:

```
let cuadrado = (base) => base * base;
```

EJERCICIO: Haz una *arrow function* que devuelva el cubo del número pasado como parámetro y pruébala desde la consola. Escríbela primero en la forma habitual y luego la "traduces" a *arrow function*.

Estructuras y bucles

Estructura condicional: if

El **if** es como en la mayoría de los lenguajes. Puede tener asociado un **else** y pueden anidarse varios con **else if**.

```
if (condicion) {
    ...
} else if (condicion2) {
    ...
} else if (condicion3) {
    ...
} else {
    ...
}
```

Ejemplo:

```
if (edad < 18) {
    console.log('Es menor de edad');
} else if (edad > 65) {
    console.log('Está jubilado');
} else {
    console.log('Edad correcta');
}
```

Se puede usar el operador **? :** que es como un *if* que devuelve un valor:

```
let esMayorDeEdad = edad > 18 ? true : false;
```

Estructura condicional: switch

El **switch** también es como en la mayoría de lenguajes. Hay que poner *break* al final de cada bloque para que no continúe evaluando:

```
switch(color) {
    case 'blanco':
    case 'amarillo':    // Ambos colores entran aquí
        colorFondo='azul';
        break;
    case 'azul':
        colorFondo='amarillo';
        break;
    default:            // Para cualquier otro valor
        colorFondo='negro';
```

```
}
```

Javascript permite que el *switch* en vez de evaluar valores pueda evaluar expresiones. En este caso se pone como condición *true*:

```
switch(true) {
    case age < 18:
        console.log('Eres muy joven para entrar');
        break;
    case age < 65:
        console.log('Puedes entrar');
        break;
    default:
        console.log('Eres muy mayor para entrar');
}
```

Bucle while

Podemos usar el bucle *while…do*

```
while (condicion) {
    // sentencias
}
```

que se ejecutará 0 o más veces. Ejemplo:

```
let nota = prompt("Introduce una nota (o cancela para finalizar)");
while (nota) {
    console.log("La nota introducida es: " + nota);
    nota = prompt("Introduce una nota (o cancela para finalizar)");
}
```

O el bucle *do…while*:

```
do {
    // sentencias
} while (condicion)
```

que al menos se ejecutará 1 vez. Ejemplo:

```
let nota;
do {
    nota=prompt('Introduce una nota (o cancela para finalizar)');
    console.log('La nota introducida es: '+nota);
} while (nota)
```

EJERCICIO **adivina**: Haz un programa para que el usuario juegue a adivinar un número. Obtén un número al azar (busca por internet cómo se hace o simplemente guarda el número que quieras en una variable) y ve pidiendo al usuario que introduzca un número. Si es el que busca, le dices que lo ha encontrado y si no le mostrarás si el número que busca es mayor o menor que el introducido. El juego acaba cuando el usuario encuentra el número o cuando pulsa en 'Cancelar' (en ese caso le mostraremos un mensaje de que ha cancelado el juego).