



UD1: La arquitectura de la web e introducción al lenguaje JavaScript (II)

Parte 2: Aspectos básicos del lenguaje JavaScript

María Rodríguez Fernández mariafer@educastur.org

Al terminar la clase de hoy...

- Habrás recordado elementos básicos de programación comunes a todos los lenguajes
 - Comentarios, variables, entrada/salida
 - Operadores
 - Estructuras de control
- Habrás aprendido cómo programar dichas estructuras en JavaScript



Aspectos básicos

Sintaxis básica

- Case-sensitive
 - Sensible a mayúsculas-minúsculas.
- Formato libre
 - Los saltos de línea y espacios en blanco no aportan significado.
- Fin de instrucción con punto y coma [;]
 - Opcional pero recomendable.
- Tipado blando: No se define el tipo de las variables

Comentarios

- Comentario de una línea:

```
//comentario de una línea  
alert('Hola mundo');
```

- Comentario de varias líneas:

```
/*  
 * Comentarios  
 * de  
 * varias  
 * líneas  
 */  
alert('Hola mundo');
```

El intérprete los ignora, pero se descargan con el resto del script

Declaración de variables

- Antepone la palabra reservada **let** (accesible desde el bloque {} donde se ha declarado) o **var** (accesible desde toda la función donde se ha declarado)
 - No se declara el tipo de las variables
 - El tipo de una variable puede cambiar dinámicamente
 - Puedo asignar un valor al declarar o posteriormente



Ámbito de visibilidad: **let**

- **let** nos permite declarar una variable de alcance limitado al bloque, declaración o expresión donde se está usando (entre llaves)

```
function varTest() {  
  var x = 31;  
  if (true) {  
    var x = 71; // misma variable!  
    console.log(x); // 71  
  }  
  console.log(x); // 71  
}
```

```
function letTest() {  
  let x = 31;  
  if (true) {  
    let x = 71; // Variable distinta  
    console.log(x); // 71  
  }  
  console.log(x); // 31  
}
```



Declaración de constantes [ES6]


- Son variables cuyo valor no puede cambiar
- Se declaran con la palabra reservada **const**



```
const CAPITAL="Oviedo";  
alert(CAPITAL + " es la capital de Asturias");  
  
CAPITAL = "Gijón";  
  
alert(CAPITAL + " es la capital de Asturias");
```


- En React se usan intensivamente (para almacenar objetos, funciones...)

var

var apple = 



a thing in a box named "apple"

apple = 



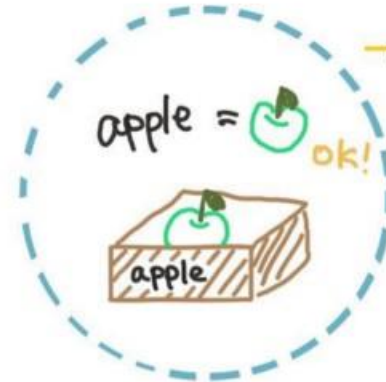
you can swap item later

let

let apple = 



a thing in a box named "apple" w/ protection shield



~~apple =  NG~~

you can swap item only if you ask inside of the shield

const

const apple = 



a thing in LOCKED cage named "apple"



~~apple =  NG~~

you can't swap item later.



apple.multiply(3) ^{OK!}

... but you can ask the item to change itself (if the item has method to do that)

Buenas prácticas


- No es obligatorio declarar las variables
 - Pero sí recomendable, pues las variables no declaradas son implícitamente **globales**
- No es obligatorio inicializar las variables
 - Pero sí recomendable ya que si no tienen el valor **undefined**
 - No es deseable que las variables tengan dicho valor.
 - No confundir con **null**, que es el valor que asignamos a objetos que aún no han sido instanciados.

La sentencia `use strict` al comienzo de un programa fuerza una sintaxis “restrictiva”, que obliga, por ejemplo, a declarar las variables.

Identificadores

- Reglas:

- Formado por letras, números, y los símbolos \$ y _
- El primer carácter NO puede ser un número
- Se recomienda convención **camelCase** para variables y **mayúsculas** para constantes



```
let numero1;  
let $numero1;
```

```
let letra  
let $letra;  
let $_letra;
```



```
let 1Nombre; /* Empieza por número */  
let numero;1; /* Contiene ; */
```

Mostrando y leyendo variables

- La comunicación con el usuario se realiza a través de la página web (formularios, párrafos, etc.), pero de momento...

- `alert(variable)`

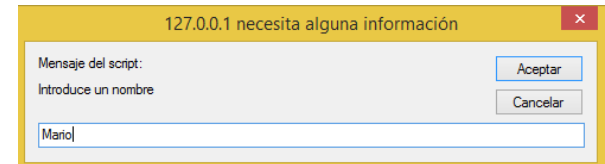
- Muestra el valor de la variable o literal en una ventana emergente

```
var nombre="Mario";  
  
alert(nombre);  
alert ("Mario Bros");
```

- `variable=prompt("Mensaje")`

- Muestra un formulario que permite dar valor a una variable

```
var nombre=prompt("Introduce un nombre");
```



Tipos de variables

- Dependenden del valor que tienen asignado:

- numéricas/**number** (entero o real)

```
var partidas=99;  
var fuerza=77.3;
```

- Cadenas de texto/**string** (entre cadenas dobles o simples)

- Para introducir caracteres especiales es necesario usar caracteres de escape (\n, \t, etc.)

```
var mensajeBienvenida="Bienvenido Mr. Marshall";  
var nombreJugador='El Fary\n';  
var letraPulsada='c';
```

- Booleanos/**boolean** [true/false]

```
var deseaContinuar=true;  
var haPagado=false;
```

Identificador de tipos

- **typeof**

- Devuelve una cadena que representa el tipo de dato contenido en una variable

```
var numero=25;  
alert(typeof(numero));
```

number

```
var numero="25";  
alert(typeof(numero));
```

string

- **isNaN**

- Devuelve verdadero si le pasamos algo que no sea un número

```
var numero=2.5;  
alert(typeof(numero));
```

number

```
isNaN("hola")
```

true

```
var numero=false;  
alert(typeof(numero));
```

boolean

Conversión de tipos: cadena a número

- `parseInt(cadena)` o `parseFloat(cadena)`

```
var numero=parseInt("30");  
alert(numero);
```

30

```
var numero2=parseInt("40px");  
alert(numero);
```

40

```
var numero3=parseInt("5pepe");  
alert(numero);
```

5

```
var numero4=parseInt("8+2");  
alert(numero);
```

8

```
var numero=parseFloat("3,2");  
alert(numero);
```

3

```
var numero2=parseFloat("3.2");  
alert(numero);
```

3.2

Coerción

- Es posible convertir un valor de un tipo a otro

- De forma implícita

```
let numero = 5;  
console.log(numero);
```

- De forma explícita

```
console.log(numero.toString());
```

- Es importante saber cómo funcionan las **coerciones implícitas**:

```
let a = "2", b = 5;  
console.log( typeof a + " " + typeof b); // string number  
console.log( a + b ); // nos muestra 25
```


EJERCICIO PROPUESTO



- Crea un programa en el que crees 4 variables, 2 cadenas y 2 números, con los siguientes valores: tu nombre, tu apellido, tu edad cuando empezaste el ciclo y el número de cursos que te ha llevado terminarlo
 - Muestra en un mensaje los tipos de datos de las cuatro variables
 - Muestra en un mensaje tu nombre y apellidos separados por un salto de línea y a continuación los años que tendrás cuando termines (calculado a partir de la suma de tu edad más el número de cursos)
- Cambia el programa para que los datos sean introducidos en tiempo real por el usuario.
 - ¿Los tipos de datos son iguales?

PARA SABER MÁS

- Lee el artículo [“Cosas extrañas de JavaScript”](#)

```
Programmer:
```

```
"What's 0.1+0.2?"
```

```
"0.30000000000000004"
```

Si no me crees...
¡Pruébalo!

Arrays

- Es una colección de elementos que pueden ser **del mismo o distinto tipo**

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves",  
"Viernes", "Sábado", "Domingo"];
```

- Podemos acceder a los elementos del array a través de un índice

– Las posiciones de un array comienzan a contarse en 0 y no en 1

```
var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"  
var otroDia = dias[5]; // otroDia = "Sábado"
```

- **Desestructuración:** para inicializar variables de forma rápida

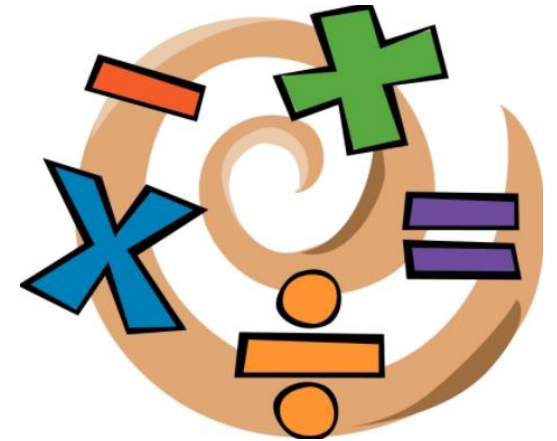
```
const colores = ["#ff0000", "#00ff00", "0000ff"];  
const [rojo, verde, azul] = colores;
```



Operadores

Operadores

- Permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar valores
- Tipos:
 - Asignación
 - Lógicos
 - Matemáticos
 - Relacionales
 - Identificación de tipos



Asignación (=)

- Se utiliza para guardar un valor en una variable
- Dos partes:
 - Izquierda del igual: Nombre de variable.
 - Derecha del igual: Expresión (variables, valores, condiciones lógicas...)

```
var numero1=3;  
var numero2=4;  
  
numero1=5;  
numero1=numero2;
```

Incremento (++) y decremento (--)

- Se aplican a tipos numéricos y permiten sumar 1 a una variable de una forma simplificada
 - ✓ Si va **delante** de la variable su valor se incrementa **antes** de ejecutar la sentencia
 - ✓ Si va **después** de la variable su valor se incrementa **después** de ejecutar la sentencia

```
var numero1=5;  
var numero2=2;  
numero3=numero1++ + numero2;
```

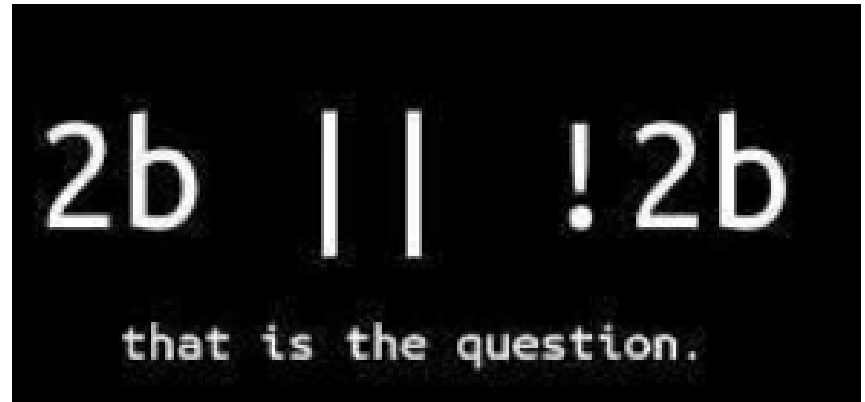
```
numero3=7, numero1=6
```

```
var numero1=5;  
var numero2=2;  
numero3=++numero1 + numero2;
```

```
numero3=8, numero1=6
```

Operadores lógicos

- Devuelven un valor de tipo booleano
- Realizan una operación lógica
 - Negación (!)
 - “Y” Lógico (AND, &&)
 - “O” Lógico (OR, ||)



Operadores lógicos: Ejemplos

```
var valor1=true;
var valor2=false;

resultado=valor1 && valor2; //resultado=false

valor1=true;
valor2=true;

resultado=valor1 && valor2; //resultado=true
```

AND

```
var valor1=true;
var valor2=false;

resultado=valor1 || valor2; //resultado=true

valor1=false;
valor2=false;

resultado=valor1 || valor2; //resultado=false
```

OR

```
var visible;
alert(!visible); // "false"
```

NOT

Operadores matemáticos

- Realizan operaciones matemáticas elementales
 - Suma (+)
 - Resta (-)
 - Multiplicación (*)
 - División (/)
 - Resto de la división entera (%).
- Puedo combinarlos con el operador de asignación para **acumular** sobre una variable (`+=`, `-=`, `*=`, etc.)

Operadores matemáticos: Ejemplos

```
var num1=10;  
var num2=5;
```

Operaciones
elementales

```
resultado=num1/num2; //resultado=2  
resultado=3 + num1; //resultado=13  
resultado=num2-4; //resultado=1  
resultado=num1*num2; //resultado=50
```

```
var num1=5;
```

Operaciones
acumuladas

```
num1+=3; //num1=num1+3;  
num1-=1; //num1=num1-1;  
num1*=2; //num1=num1*2;  
num1/=5; //num1=num1/5;  
num1%=4; //num1=num1%4;
```

```
var num1=10;  
var num2=5;
```

Resto

```
resultado=num1%num2; //resultado=0
```

```
num1=9;  
num2=5;  
resultado=num1%num2; //resultado=4
```

Operadores relacionales

- Devuelven un valor booleano
 - Mayor que (>)
 - Menor que (<)
 - Mayor o igual (>=)
 - Menor o igual (<=)
 - Igual que (==)
 - Distinto de (!=)

```
var num1=3;
var num2=5;

resultado=num1>num2; //resultado=false;
resultado=num1<num2; //resultado=true;

num1=5;
num2=5;

resultado=num1 >= num2; //resultado=true;
resultado=num1 <= num2; //resultado=true;
resultado=num1 == num2; //resultado=true;
resultado=num1 != num2; //resultado=false;
```

Error común



- El operador *igual* (==) es origen de un gran número de errores de programación
- No confundir con el operador de asignación (=).

```
//El operador "=" asigna valores
var num1=5;
resultado=num1=3;
//num1 vale 3 y resultado vale3

//El operador "==" compara variables
var num1=5;
resultado=num1==3;
//num1 vale 5 luego resultado vale false
```

Comparación estricta

- `===` y `!==` permiten comparación **estricta**
 - Se compara el tipo además de su valor
- `==` y `!=` son de comparación **relajada**
 - Si los tipos son distintos trata de transformarlos para que sean comparables
 - La operación relajada tiene [bastantes reglas](#) que no siempre son fáciles de recordar, por eso por lo general se recomienda el uso de la **comparación estricta**

Comparación estricta vs relajada

```
var numero=10;  
var numero2="10";  
  
if(numero==numero2){  
    /* ..... */  
}
```

El `if` se evalúa como verdadero

```
var numero=10;  
var numero2="10";  
  
if(numero===numero2){  
    /* ..... */  
}
```

El `if` se evalúa como falso

Ayuda para recordarlo

Math Class

$1 = 1$

$1 \neq 2$



Normal Coding Languages

$1 == 1$

$1 != 2$



Javascript

$1 === 1$

$1 !== 2$



Operadores con cadenas

- El operador + también se utiliza para **concatenar** cadenas
- Los operadores lógicos también pueden usarse para comparar cadenas de texto (alfabéticamente)

```
var frase1="Ojos que no ven";  
var frase2="corazón que no siente";  
  
var refran1=frase1+frase2;  
var refran2=frase1+" castañazo que te pegas";  
  
alert(refran1);  
alert(refran2);
```

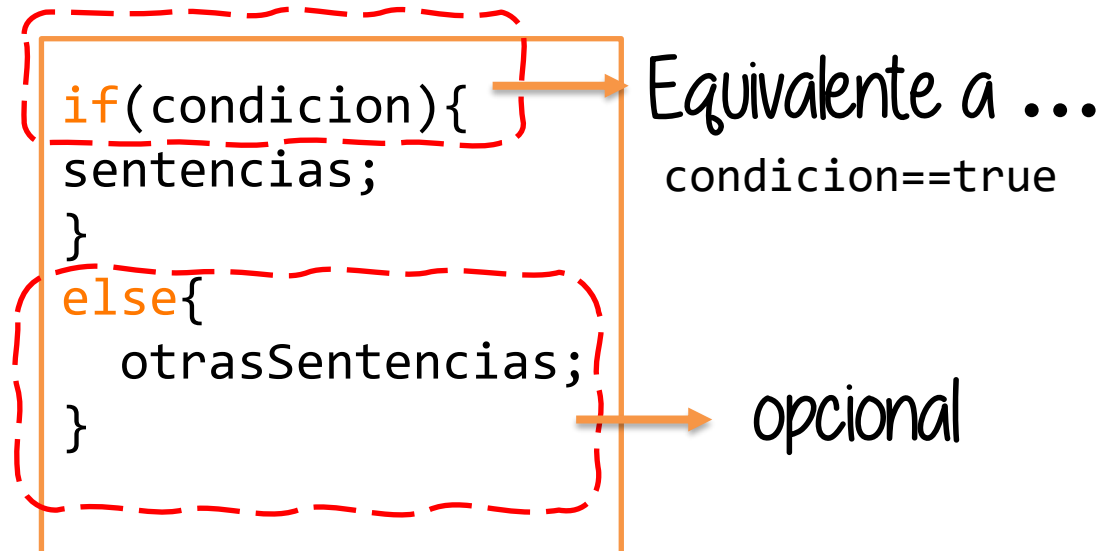
```
var texto1="Hola";  
var texto2="Hola";  
var texto3="Adios";  
  
resultado=texto1==texto3; //false  
resultado=texto1!=texto2; //false  
resultado=texto1>=texto2; //false
```



Estructuras de control

Estructura condicional (if ... [else ...])

- Las sentencias se ejecutan si se cumple la condición
 - Las llaves no son obligatorias si hay una única sentencia
 - La parte del else se ejecuta si no se cumple la condición
- Puedo encadenar varias condiciones simples o complejas



```
var mostrado=false;
var usuarioPermiteMsj=true;

if(!mostrado && usuarioPermiteMsj)
  alert("PUM!");
```

Estructura condicional

There are two types of people.

```
if (Condition)
{
    Statements
    /*
    ...
    */
}
```

```
if (Condition) {
    Statements
    /*
    ...
    */
}
```

Programmers will know.

Operador ternario

- Permite evaluar una condición y ejecutar una de dos expresiones en función de la misma

```
condición ? expresión1 : expresión2;
```

- **Condición:** Expresión que podemos evaluar como verdadero o falso.
- **Expresión 1:** Se ejecuta si condición es verdadero
- **Expresión 2:** Se ejecuta si condición es falso.

```
var miEdad = 24;  
var mayorEdad = (miEdad > 18) ? "Sí, eres mayor de edad" : "No, sigue intentando";
```

Estructura múltiple (switch)

- Se basa en evaluar una expresión con resultado escalar, para decidir el punto de entrada en la estructura

```
switch(expresion){  
  case valor1:  
    sentencia1;  
    break;  
  
  case valor2:  
    sentencia2;  
    break;  
  
  case valor3:  
    sentencia3;  
    break;  
}
```

Bucle for

- Permite repetir una o varias sentencias un número determinado de veces
 - **inicialización**: Valor inicial de la variable que controla la repetición.
 - **condición**: Que debe cumplirse para que las sentencias se ejecuten.
 - **actualización**: Se ejecutan después de cada repetición. Normalmente actualiza la variable que controla la repetición.

```
var mensaje="Yuhuuu! Estoy dentro de un bucle!!";
```

Inicialización

Condición

Actualización

```
for (let i=0;      i<5;      i++)  
  alert(mensaje);
```

Bucle while



- Repite una serie de sentencias mientras se cumpla una condición
 - Habitualmente en las sentencias controlo la condición de salida del bucle
- ¿Qué hace este ejemplo? →

```
while(condición){  
    sentencias;  
}
```

```
var resultado=0;  
var numero=100;  
var i=0;  
  
while(i<=numero){  
    resultado+=i;  
    i++;  
}  
  
alert(resultado);
```


Bucle do..while

- Repite una serie de sentencias mientras se cumpla una condición
 - La diferencia con el **while** es que las sentencias se ejecutan **al menos una vez**, mientras que con el while podrían no ejecutarse nunca
- ¿Qué hace este ejemplo? →

```
do{  
  sentencias;  
} while(condición);
```

```
var resultado=1;  
var numero=5;  
do{  
  resultado*=numero;  
  numero--;  
} while(numero>0);  
alert(resultado);
```

EJERCICIO PROPUESTO

- Define un array numérico formado por números del 1 al 9.
- El script debería comprobar si hay algún número fuera de su posición, informando SOLO del primer caso que se encuentre.
 - Por ejemplo, en el array [1,2,3,9,5,6,7,8,9] se debería obtener el mensaje "Error en la posición 3, hay un 9 cuando se esperaba un 4".
- En caso de que todos los elementos estén en la posición correcta, se mostrará un mensaje indicativo.
- Los mensajes se mostrarán por consola.

