

SINTAXIS DEL LENGUAJE

UT2.- INTRODUCCIÓN A JAVASCRIPT



Susana López Luengo

JAVASCRIPT. OBJETIVOS

- En esta unidad se presentan los **conceptos básicos** en la programación del lenguaje JavaScript
 - Aprenderemos la sintaxis básica del lenguaje
 - Utilizaremos los distintos **tipos de datos, variables y operadores** de Javascript
 - Conoceremos las **sentencias condicionales y bucles**
 - Escribiremos nuestros primeros programas que servirán de base para conocer en detalle el lenguaje JavaScript.
-

JavaScript. Características

- Brendan Eich(1961-): Desarrolla LiveScript en solo 10 días en 1995 para Netscape con el objetivo de dinamizar la web.
- Estandarización (1996): ECMAScript o ECMA-26
- Influyó en: JScript, ActionScript, .NET W3C elige JavaScript como estándar para HTML5.
- Es una marca registrada de Oracle Corporation, y es usado con licencia por los productos creados por Netscape Communications y entidades actuales, como la fundación Mozilla



JavaScript. Hitos en el desarrollo de JavaScript

- Inicios (1996-1997): Validación de formularios y rollovers
- DOM (1997-2004): Llega la animación
- Ajax (1999-): La hora de las aplicaciones web
- Bibliotecas (2005-): Hay que mejorar la productividad
- HTML5 (2008-): Pilar esencial de tecnologías revolucionarias:
 - Canvas (*)
 - Audio y vídeo (*)
 - Arrastrar y soltar (*)
 - Aplicaciones offline
 - Almacenamiento web
 - Geolocalización
 - Web Workers (*)
 - Notificaciones (*)

PhoneGap nos permite crear aplicaciones para dispositivos (iPhone, Android, ...) usando HTML5, CSS3 y JavaScript.

JavaScript. Características

- Lenguaje de alto nivel **interpretado** en el lado del cliente
 - Todos los navegadores actuales soportan JavaScript
 - **No es un lenguaje orientado a objetos**
 - Está **basado en objetos** con herencia por prototipos
 - Es un lenguaje **débilmente tipado**, con lo que una variable puede contener valores de distintos tipos en diferentes momentos de la ejecución de un programa
 - Su **sintaxis** es prestada de C y Java, pero también hereda de Awk y Perl
-

JavaScript. Herramientas

- **Editores JavaScript:** Bloc de Notas, NotePad ++, Atom, **Sublime**, Brackets etc.
 - **IDEs:** Eclipse, **Visual Studio Code**, Komodo, Aptana y NetBeans.
 - **Depurador de código:** Herramientas específicas de los navegadores, Chrome y Firefox
 - **Servidor Web:** NodeJS
-

JavaScript en elementos HTML

La integración de JavaScript y HTML/XHTML podemos hacerla de 3 formas:

- **Directamente dentro de las etiquetas** `<script>` y `</script>`
 - Añadir en un **fichero externo con extensión .js** indicado con el atributo `src`
 - Insertar fragmentos de JavaScript directamente **dentro de atributos de etiquetas HTML**
 - Se recomienda añadir justo antes del final del `</body>` para que el navegador cargue primero el resto de objetos DOM y no bloquee el renderizado
-

JavaScript en elementos HTML

Directamente desde las etiquetas

```
<body>  
  <button onclick="alert('Hola')">Haz clic  
  </button>  
</body>
```

Directamente en las etiquetas script

```
<body>  
  <!-- // cosas -->  
  <script>  
    alert ("Hola, muy buenas")  
  </script>  
</body>
```

JavaScript en elementos HTML

Desde un archivo externo

```
<body>
  <!-- // cosas -->
  button onclick="saludo()">
    Haz clic
  </button>
  <script src="js/miCodigo.js"></script>
</body>
```

miCodigo.js

```
function saludo() {
  alert("Hola, muy buenas")
}
```

[Ejemplos HolaMundo](#)

Sintaxis

Mayúsculas y minúsculas

- Al igual que C y Java, JavaScript **distingue entre mayúsculas y minúsculas**
 - Esta regla es muy importante cuando utilizamos objetos, variables, funciones o cualquier otro símbolo del lenguaje
 - No es lo mismo utilizar `alert ()` que `Alert ()`
 - Se recomienda **utilizar minúsculas** para los nombres de las variables, funciones y objetos, recurriendo a la sintaxis de **lowerCamelCase** (sintaxisDeCamello) si están compuestas por varias palabras.
-

Sintaxis

Comentarios

- Los comentarios en el código **no son interpretados por el navegador**
 - Su función principal es facilitar la lectura del código al programador.
 - Una **doble barra (//)** marca el inicio de un comentario, de modo que será ignorado cualquier texto que escribamos a partir de esa posición y hasta el final de línea.
 - Si necesitamos crear un comentario de **varias líneas** lo iniciaremos con **/*** y lo concluiremos con ***/**.
-

Sintaxis

Comentarios

- Ejemplo:

```
/* función que muestra el valor de x */  
function comentario() {  
    alert('Hola' + x + ' !');  
}
```

- En JavaScript puedes incluso incluir un comentario en medio de una sentencia de código, aunque esto hace el código difícil de leer por lo que debería de utilizar con cuidado

```
function comentario() {  
    alert('Hola' + x /* inserta el valor de x */ + ' !');  
}
```

Sintaxis

Espacios en blanco

- Los espacios consecutivos son considerados como un único espacio
 - Se consideran buenas prácticas sangrar el código utilizando como unidad de sangrado 4 espacios, y no escribir líneas de más de 80 caracteres
 - JavaScript ignora los espacios, las tabulaciones y los saltos de línea con algunas excepciones.
 - Emplear la tabulación y los saltos de línea mejora la presentación y la legibilidad del código
-

Sintaxis

Inserciones automáticas de punto y coma

- Se suele insertar un signo de punto y coma (;) **al final de cada instrucción de JavaScript**
- Su utilidad es **separar y diferenciar cada instrucción**.
- Una instrucción puede extenderse a lo largo de varias líneas o podemos escribir varias instrucciones en la misma línea.
- Se puede omitir si cada instrucción se encuentra en una línea independiente (o hay solo una sentencia dentro de llaves { })

La omisión del punto y coma no es una buena práctica de programación

Sintaxis

Inserciones automáticas de punto y coma

```
a = b  
++c  
{ 1 2 } 3  
return  
a + b
```

Es transformado por
la inserción
automática de puntos
y comas a

```
a = b;  
++c;  
{ 1 2 ;} 3;  
return;  
a + b;
```

Sintaxis

Palabras reservadas

- Algunas palabras **no se pueden utilizar** para definir nombres de variables, funciones o etiquetas. (class, const, enum, export...)
- Palabras reservadas para [ECMAScript 2019](#):

break

case

catch

class

const

continue

debugger

default

delete

do

else

export

extends

finally

for

function

if

import

in

instanceof

new

return

super

switch

this

throw

try

typeof

var

void

while

with

yield

TIPOS DE DATOS

- Los tipos de datos especifican qué **tipo de valor** se guardará en una determinada variable.
 - El último estándar ECMAScript define seis tipos primitivos de datos.
 - **Number**
 - **String**
 - **Boolean**
 - **Null**
 - **Undefined**
 - **Symbol**
 - Existe otro tipo de datos compuesto llamado **objeto**, que representa una colección de valores primitivos o compuestos por otros objetos.
-

TIPOS DE DATOS

- JavaScript es un lenguaje de programación de "**tipado débil**":
 - El tipo de datos no es una propiedad de las variables, sino de los datos
 - Una misma variable puede contener datos de tipo diferente en distintos instantes.
 - No es obligatorio declarar las variables, aunque nos permite hacerlo mediante los comandos var, let y const
 - **const** se utiliza para variables inmutables
-

TIPOS DE DATOS. CADENAS DE TEXTO

- El tipo de datos para representar cadenas de texto se llama **string**.
 - Se pueden representar letras, dígitos, signos de puntuación o cualquier otro carácter de Unicode.
 - La cadena de caracteres se debe definir entre **comillas dobles o comillas simples**
 - **Operador de concatenación: +**
 - “hola” + “ “ + “Pepe” → “Hola Pepe”
-

TIPOS DE DATOS. CADENAS DE TEXTO

- ES6 introduce plantillas de string delimitados por **comillas invertidas** ``...``
- Son string multilínea que pueden contener expresiones delimitadas por `${expr}` que evalúan la expresión y la sustituyen por el valor correspondiente

Ej:

``Un día tiene ${24*60} minutos``

TIPOS DE DATOS. CADENAS DE TEXTO

CÓDIGOS ESCAPADOS

- Para incluir comillas en una cadena de caracteres tendremos que escaparla precediéndola con el signo \ en caso de que la cadena esté delimitada por el mismo tipo (simple o doble) de comillas
 - Ejemplo: 'She\'s beautiful'

- Si queremos

Secuencia de escape	Resultado
\\	Barra invertida
\'	Comilla simple
\"	Comillas dobles
\n	Salto de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\f	Salto de página
\r	Retorno de carro
\b	Retroceso

\\

TIPOS DE DATOS. CADENAS DE TEXTO

- También podemos incluir cualquier carácter Unicode de 16 bits utilizando la fórmula `\unnnn`, donde `nnnn` es el valor hexadecimal del carácter.
- En <http://unicode.org/charts/> pueden consultarse todos los caracteres Unicode, como símbolos matemáticos, caracteres de otros idiomas o signos Braille.
- En la siguiente tabla se recogen los códigos Unicode de algunos caracteres especiales del castellano

á	é	í	ó	ú	Á	É	Í
\u00e1	\u00e9	\u00ed	\u00f3	\u00fa	\u00c1	\u00c9	\u00cd
Ó	Ú	ü	Ü	ñ	Ñ	í	¿
\u00d3	\u00da	\u00fc	\u00dc	\u00f1	\u00d1	\u00a1	\u00bf

Tipos de datos. NÚMEROS

- En JavaScript existe sólo un tipo de dato numérico.
 - El valor de doble precisión de 64-bits IEEE 754
 - El número comprende entre Number.MIN_VALUE ($-2^{53} - 1$) y Number.MAX_VALUE ($2^{53} - 1$)
 - No existe un tipo específico para los números enteros
 - También pertenecen a este tipo los valores Infinity, -Infinity y NaN (Not A Number).
-

Tipos de datos. BOOLEANOS

- También conocido como **valor lógico**.
 - Sólo admite dos valores: **true o false** (escritos en minúsculas y sin comillas)
 - Es muy útil a la hora de evaluar expresiones lógicas o verificar condiciones.
-

Tipos de datos undefined

- Este tipo de datos sólo admite el dato **undefined**.
 - Este valor se asigna implícitamente a todas las variables que han sido declaradas pero a las que no se ha asignado un dato aún.
-

Tipos de datos

null

- Este tipo de datos sólo admite el dato null
- Este dato que identifica a **punteros a objetos que aún no tienen un objeto asignado.**
- No se asigna implícitamente, sino que cuando declaremos una variable que posteriormente vamos a referir a un objeto, le asignaremos explícitamente el dato null.

```
var persona=null
```

Tipos de datos

symbol

- Introducido en ECMAScript 6
- Es un valor primitivo único e inmutable
- Puede ser usado como la clave de una propiedad de un Object.

TIPOS DE DATOS. object

- Este tipo de datos puede interpretarse como una **agregación estructurada de propiedades y métodos**, sin ninguna restricción en cuanto a su tipo, en la que cada uno de ellos está identificado por un nombre.
- Se agrupan en **clases**: Object, Date, Array, Function,...
- Para crear un dato del tipo object tenemos que utilizar la expresión `new Object()`

```
var persona = new Object();
```

- Una vez declarada la variable, podremos añadir datos a su colección utilizando la sintaxis de punto



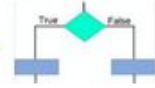



```
persona.nombre = "Pepe";  
persona.dirección = new Object();
```

Tipos de datos. object

- Los datos que contienen los objetos se clasifican en dos categorías: propiedades y métodos.
 - Los **métodos** son instancias de un objeto predefinido de JavaScript llamado Function.
 - Todos los datos que no son de este tipo se denominan **propiedades**.
-

Tipos de datos. Operador typeof

- Permite conocer el tipo de un valor. Devuelve un **string** con el **nombre del tipo**:
 - "number", "string", "boolean", "undefined", "object" y "function"
 - Todos los objetos (de cualquier clase) devuelven **"object"**, salvo las funciones

typeof 7	=> "number"	
typeof "hola"	=> "string"	
typeof true	=> "boolean"	
typeof Symbol()	=> "symbol"	
typeof undefined	=> "undefined"	UNDEFINED
typeof null	=> "object"	
typeof new Date()	=> "object"	
typeof new Function()	=> "function"	$f(x)$

VARIABLES

- Se pueden definir como zonas de la memoria que se identifican con un nombre y en las cuales se almacenan ciertos datos.
 - El nombre de una variable también se conoce como identificador y debe cumplir las siguientes **normas**:
 - **Sólo puede estar formado por letras, números y los símbolos \$ (dólar) y _ (guión bajo).**
 - **El primer carácter no puede ser un número**
 - Las siguientes variables tienen nombres correctos:

<code>var \$numero1;</code>	<code>var _\$letra;</code>
<code>var \$\$\$otroNumero;</code>	<code>var \$_a__\$4;</code>
-

VARIABLES. Buenas prácticas de nombrado

- Usar nombres lógicos y descriptivos para variables y funciones
 - No preocuparse por la longitud de los nombres
 - Usar sustantivos para nombrar las variables
 - Comenzar por un verbo los nombres de funciones
 - Comenzar por 'is' o 'has' ('es' o 'tiene') los nombres de los predicados
 - Evitar nombres que no aportan información aux, temp...
 - En los nombres de varias palabras **lowerCamelCase**
 - Escribir en MAYÚSCULAS los nombres de variables constantes (const) y separadas con guión bajo _
 - Las funciones constructoras deben comenzar por mayúsculas
-

Variables.DECLARACIÓN

- Una variable declarada con **var** dentro de una función tiene alcance local dentro de ella
- ES2015 introdujo **let** y **const** que veremos más adelante.

```
const NUM_JUGADORES = 5; // variable inmutable
```

```
function curarTodos() {  
  var curar = 100; // variable local  
  for( let numero = 1; numero < NUM_JUGADORES; numero ++ ) {  
    // numero solo es visible dentro del bloque for  
    jugadores[numero].salud = curar;  
  }  
}
```

Variables. ÁMBITO

- Fuera de un bloque o función, todas las variables son globales
 - Cuando JavaScript intenta resolver una variable, busca en el ámbito de la función local
 - Si no se encuentra este identificador, busca en la función externa en la que se declaró la variable, y así sucesivamente a lo largo de la cadena del ámbito hasta que alcance el ámbito global
 - Si aún así no se encuentra, JavaScript generará una excepción `ReferenceError`.
-

Variables. INICIALIZACIÓN DE VARIABLES

- La asignación de datos a variables se realiza mediante el operador =, y siempre por valor
 - Los datos de tipo object son la excepción, en cuyo caso la asignación se realiza por referencia
 - Se puede asignar un valor a una variable de tres formas:
 - Asignación directa de un valor concreto.
 - Asignación indirecta a través de un cálculo en el que se implican a otras variables o constantes.
 - Asignación a través de la solicitud del valor al usuario del programa
-

INICIALIZACIÓN DE VARIABLES. EJEMPLO

```
var vida = 100;
```

```
var estamina = vida + 20;
```

```
var nombre = prompt('Introduce el nombre del personaje','');
```

Observa sintaxis de `prompt(texto, valor por defecto)`

ESTADO DE LAS VARIABLES. Tipado débil

- El estado varía a medida que se ejecutan las instrucciones

Evolución del estado en función de la instrucción ejecutada

```
var x = 5;           // Creates variable x with initial value 5
x = "Hello";        // Assigns string "Hello" to variable x
x = undefined;      // Assigns undefined to variable x
x = new Date();      // Assigns Date object to variable x
```



VARIABLES.CONVERSIÓN DE TIPOS

- JavaScript realiza conversión automática de tipos
- La ambigüedad de una expresión se resuelve con las reglas sintácticas y la prioridad entre operadores
 - **Concatenar strings tiene más prioridad que suma de números:**
- `4 + 5 // resultado = 9`
`4 + "5" // resultado = "45"` porque combina string y number. + es concatenación
`4 + 5 + "6" // resultado = "96"`
- `+"13" + 2 // resultado= 15` JavaScript convierte "13" a number antes de aplicar operador +

OPERADORES

- JavaScript es un lenguaje rico en operadores
 - Son símbolos y palabras que realizan operaciones sobre uno o varios valores, para obtener un nuevo valor.
 - Cualquier valor sobre el cuál se realiza una acción (indicada por el operador), se denomina un operando.
 - Una expresión puede contener un operando y un operador (denominado operador unario), como por ejemplo en `b++`, o bien dos operandos, separados por un operador (denominado operador binario), como por ejemplo en `a + b`
-

OPERADORES ARITMÉTICOS

Permiten realizar cálculos elementales entre variables numéricas.

Operador	Nombre
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++	Incremento
--	Decremento

El operador % (módulo) devuelve el resto de una división

Ejemplo:

```
var anyo=2019;
```

```
var bisiesto = anyo % 4
```

```
----Si bisiesto es 0, el año es bisiesto
```

Operadores. ASIGNACIÓN

- Asigna el valor a la derecha de la expresión a la variable que está a la izquierda.
- Permiten el uso de métodos abreviados en asignaciones

Operador	Descripción	Ejemplo	Resultado
=	Asignación simple	saldo=1500	saldo=1500
+=	Suma y asigna	saldo +=200	saldo=1700
-=	Resta y asigna	saldo - = 300	saldo=1400
/=	Divide y asigna	saldo /= 7	saldo = 200
*=	Multiplica y asigna	saldo *= 2	saldo = 400
%=	Módulo y asigna	saldo %=10	saldo = 0

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators>

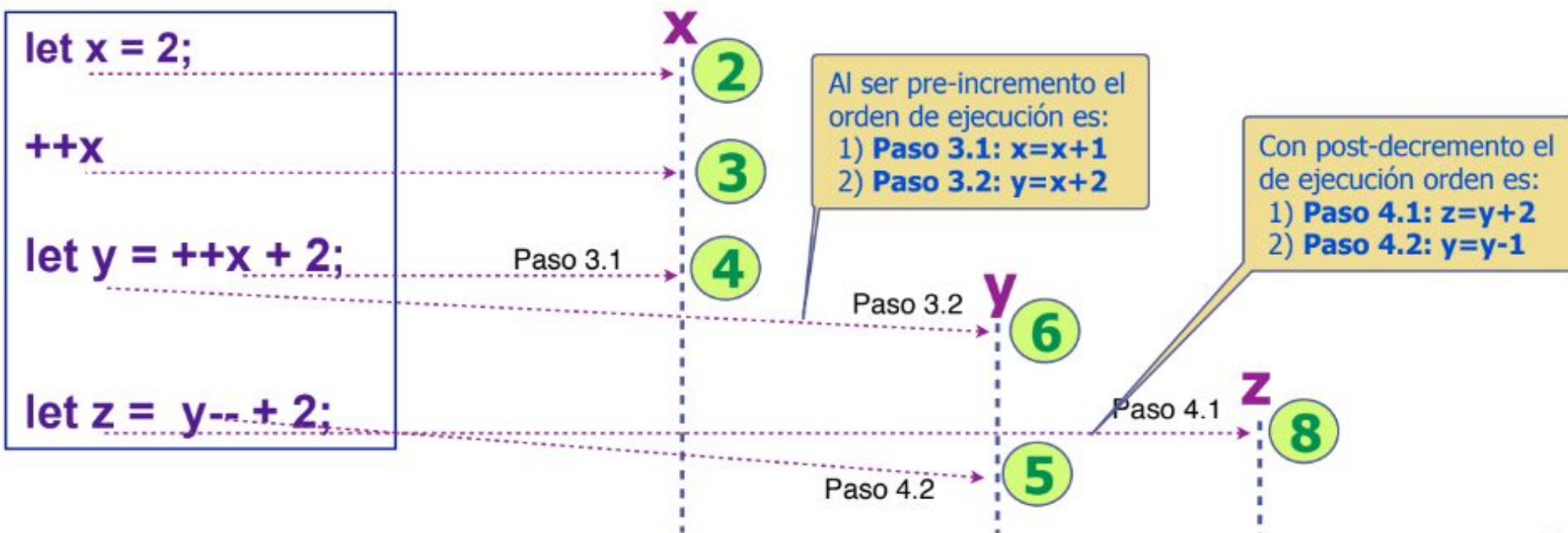
Operadores. COMPARACIÓN

Comparan los valores de 2 operandos, devolviendo un resultado de true o false https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness

Operador	Nombre	Ejemplo
<	Menor que	Por ejemplo, <code>5 < 3; //false</code> Y <code>"casa" < "tos"; //true</code> Pero <code>"Tos" < "casa"; //true</code>
<=	Menor o igual que	Por ejemplo, <code>5 <= 3; //false</code>
==	Igual	Por ejemplo, <code>5 == 3 + 2; //true</code> Pero <code>0.3 == 0.1 + 0.2; //false</code> Y <code>false == "false"; //false</code> Pero <code>false == "0"; //true</code> Y <code>"2" == 2; //true</code>
>	Mayor que	<code>5 > 3; //true</code>
>=	Mayor o igual que	<code>5 >= 3; //true</code>
!=	Diferente	<code>5 != 3 + 2 // false</code>
===	Estrictamente igual(incluso en tipo)	<code>"2" === 2; //false</code>
!==	Estrictamente diferente (en dato y tipo)	<code>"2" !== 2; //true</code>

Operadores. PRE Y POST AUTOINCREMENTO ++ --

- ++ (autoincremento) y -- (autodecremento)
- Si ++/-- se aplica por la izquierda a la variable (pre), el incremento/decremento se realiza **antes** de evaluar la expresión
- Si ++/-- se aplica por la derecha (post) se incrementa/decrementa **después** de evaluarla



Operadores

Bit a bit

- Realizan operaciones aritméticas o de desplazamiento de columna en las representaciones binarias de dos operandos.

&

|

^

~

<<

>>

>>>

OPERADORES LÓGICOS && (and) || (or)

◆ Operador lógico y (and): **<v1> && <v2>**

- devuelve <v1> o <v2> **sin modificarlos**
 - ◆ **<v1> && <v2>**
 - ◆ devuelve <v1> -> si <v1> es equivalente a **false**
 - ◆ devuelve <v2> -> en caso contrario

```

true && true   => true
false && true  => false
true && false  => false
false && false => false
  
```

```

0 && true      => 0
1 && "5"       => "5"
  
```

◆ Operador lógico o (or): **<v1> || <v2>**

- devuelve <v1> o <v2> **sin modificarlos**
 - ◆ **<v1> || <v2>**
 - ◆ devuelve <v1> -> si <v1> es equivalente a **true**
 - ◆ devuelve <v2> -> en caso contrario

```

true || true   => true
false || true  => true
true || false  => true
false || false => false
  
```

```

undefined || 1   => 1
13 || 1          => 13
  
```

INTERACCIÓN SENCILLA BASADA EN POP-UPS

Funciones alert(), confirm() y prompt()

◆ alert(msj):

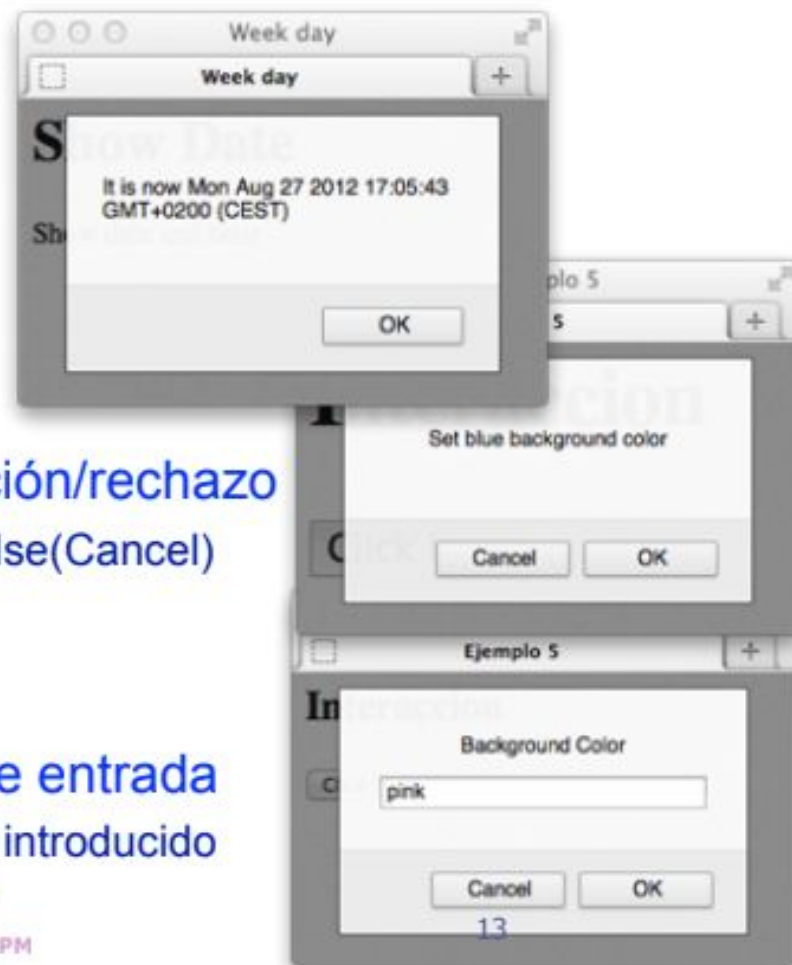
- presenta un mensaje al usuario
 - ◆ Retorna al pulsar OK

◆ confirm(msj):

- Presenta mensaje y pide confirmación/rechazo
 - ◆ Retorna al pulsar, devuelve true(Ok)/false(Cancel)

◆ prompt(msj):

- Presenta mensaje y pide un dato de entrada
 - ◆ Al pulsar OK, retorna y devuelve string introducido
 - Si se pulsa Cancel devuelve "null"



PRECEDENCIA DE OPERADORES ES5 Y ES6

En la siguiente [tabla](#) se recoge el orden de precedencia (mayor cuanto más arriba en la tabla) de los operadores. Observe que los paréntesis ocupan el nivel de mayor precedencia, de modo que es aconsejable utilizarlos para agrupar las operaciones siempre que existan dudas respecto al orden de precedencia de otros operadores. Un resumen:

```
( )
++ -- !
* / %
+ -
< > <= >=
== !=
&&
||
= += -= *= /= %=
```

$8 * 2 - 4 = > 12$

$8 * (2 - 4) = > -16$





$+ "3" + 7 = > 10$



+unitario (signo) tiene más prioridad que
+binario (suma) y se evalúa antes

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence#Table

Hola !!
Mi primer programa

[Práctica UT2Pr1Introducción](#)

  Inspector  Depurador  **Consola**

  Filtrar salida

Errores **Advertencias** **Registros** Información Depurar

```
>> var var1=0
< undefined

>> var2=1
< 1

>> let var3=2
< undefined

>> const saludo="hola"
< undefined

>> var1+var2
< 1

>> saludo
< "hola"

>> console.log(saludo)
hola

< undefined
```

WEBGRAFÍA Y ENLACES DE INTERÉS

- <https://www.w3schools.com/js/>
 - <https://developer.mozilla.org/es/docs/Web/JavaScript>
 - Curso Desarrollo de aplicaciones con HTML,node.js y Javascript. UPM. Miriadax
-

