MÓDULO 3:
Introducción a JavaScript

Juan Quemada, DIT - UPM

Índice

Introducción a JavaScript (ES6+) de cliente, expresiones y la consola

Juan Quemada, DIT - UPM

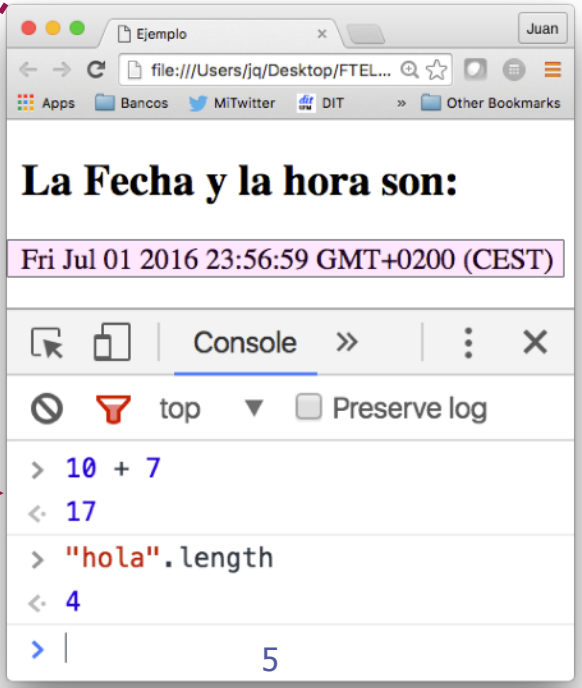
JavaScript

- Lenguaje de programación diseñado en 1995 por Brendan Eich
  - Para animar páginas Web y realizar aplicaciones en el Navegador Netscape
    - Hoy se ha convertido en **el lenguaje de programación** más utilizado en **Internet**
- JavaScript tiene pocos elementos genéricos y potentes
  - Literales, funciones, objetos, tipado débil, prototipos, ...
    - JavaScript tiene además algunas partes mal diseñadas, que se recomienda no utilizar
- Este **curso** se centra en la partes buenas **(Good parts)** de JavaScript
  - Libro**: JavaScript: The Good Parts**, Douglas Crockford, O'Reilly Media, 2008
    - **jslint** (http://www.jslint.com/) o **eslint** (http://eslint.org/) comprueban el **buen uso**
- Llamado también **ECMAScript**: norma de European Comp. Manuf. Asoc.
  - Normas: **ES5** (2011), **ES6** (2015), **ES7** (2016), **ES8** ... (Versión anual desde ES6)
    - Este curso cubre **ES5** y las mejoras más importantes de **ES6**, **ES7**, **ES8**, **ES9**, ...
- Documentación y tutoriales de JavaScript (ES6+)
  - https://javascript.info, https://flaviocopes.com/page/ebooks/
  - https://developer.mozilla.org/es/docs/Web/JavaScript, https://t.co/nLNMwtgHV5

Curso de JavaScript (ES6) en Navegador

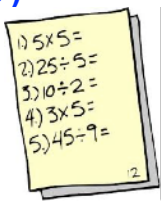
- Este curso se centra en "Vainilla JavaScript"
  - Se denomina "Vainilla JavaScript" a usar ES6 y posteriores
    - Evitando el uso de librerías como jQuery y muchas otras
- La adaptación de los navegadores a ES6 ha sido lenta
  - Las versiones actuales de los navegadores están adaptadas,
    - Ver tabla de soporte: https://kangax.github.io/compat-table/es6/
  - Polyfills: traducen ES6 y posteriores para browsers antiguos
    - Babel: https://babeljs.io
    - Traceur: https://github.com/google/traceur-compiler
    - otros
- Los navegadores ejecutan scripts de 2 formas
  - Ejecutan **scripts** JavaScript en una página HTML
  - Ejecutan sentencias en la **consola**

```
<!DOCTYPE html><html><head>
  <title>Ejemplo</title>
  <meta charset="UTF-8">
</head>

<body>
  <h2> La Fecha y la hora son:</h2>

  <div id="fecha"></div>

  <script type="text/javascript">

    document.getElementById("fecha")
    .innerHTML = new Date();
  </script>
</body>
</html>
```



Tipo number: números y operadores

- JavaScript permite construir expresiones con **operadores** predefinidos
  - +, -, * y / son operadores binarios (2 valores) infijos (operador en el centro)
    - La evaluación (=>) de la expresión genera otro valor como resultado




```
13 + 7        =>   20      //  Suma de números
13 -  1.5     =>   11.5    //  Resta de números
(8*2 - 4)/3   =>    4      // Expresión con paréntesis
 9%6          =>    3      //  Resto de
 2**3         =>    8      //  Potencia (ES6)
```

En JavaScript los decimales se separan con un punto, igual que en inglés.

- Una expresión es una sentencia de JavaScript con una sintaxis estricta
  - Al ejecutarla se comprueba que la sintaxis es correcta y se evalúa el resultado
    - Expresiones mal construidas dan error y pueden bloquear la ejecución

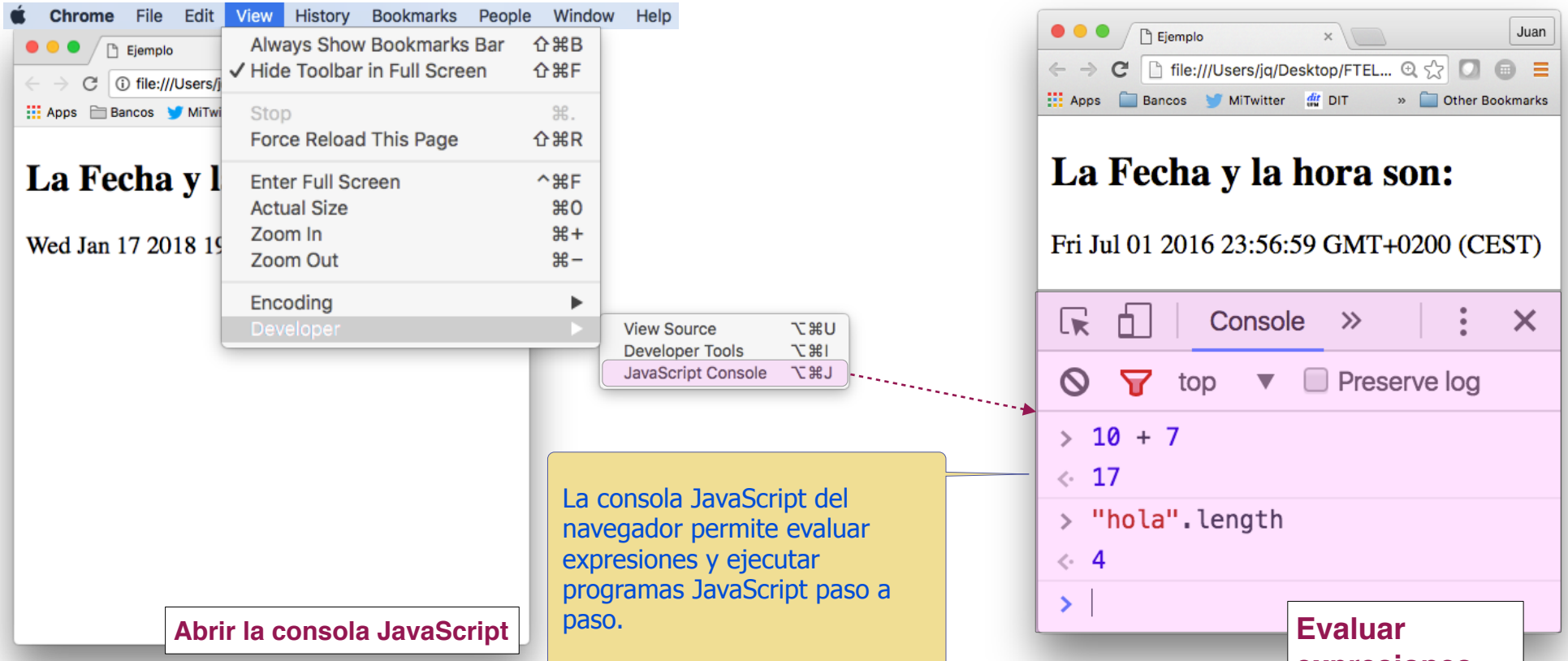
Expresión incorrecta: 2 operadores seguidos

Expresión incorrecta: dos números sin operador entre ellos

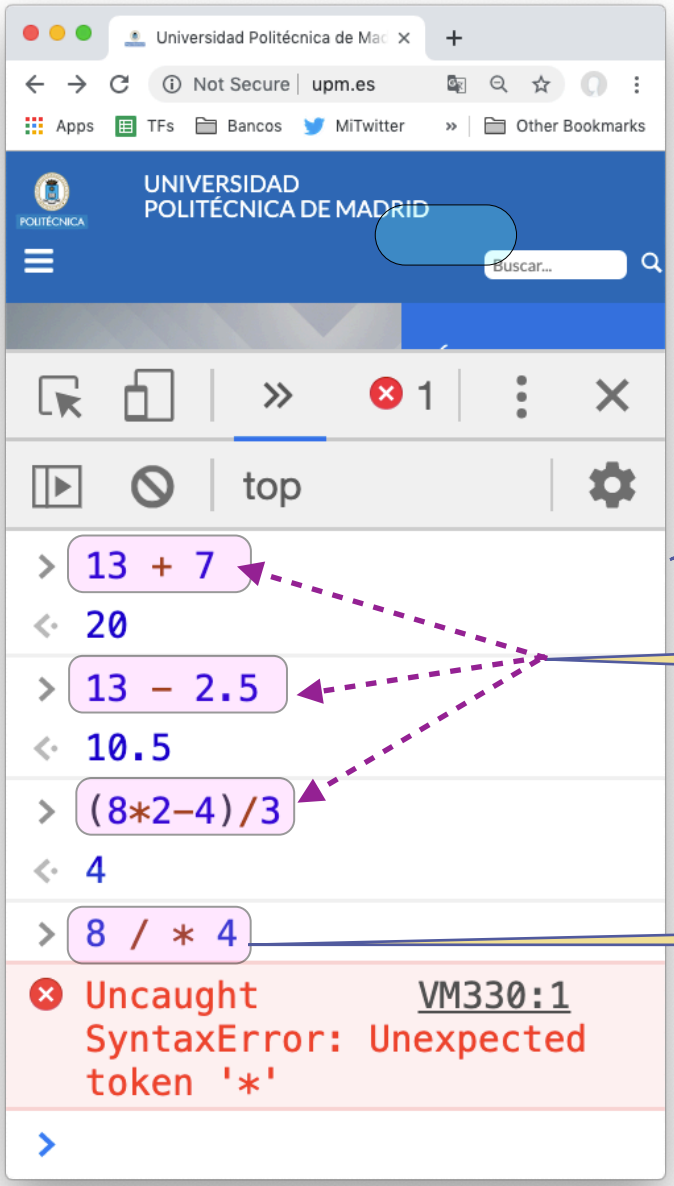
```
8 / * 3      =>   ...?
8   3        =>   ...?
```

- Doc: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Math

La consola JavaScript del navegador (Chrome)



Intérprete JavaScript: expresiones



Consola JavaScript del navegador (chrome). Todos los navegadores permiten desplegarla y ejecutar instrucciones de forma interactiva.

El intérprete analiza y ejecuta el texto introducido al teclear nueva línea (Enter).

Si tecleamos una expresión la evalúa y presenta el resultado.

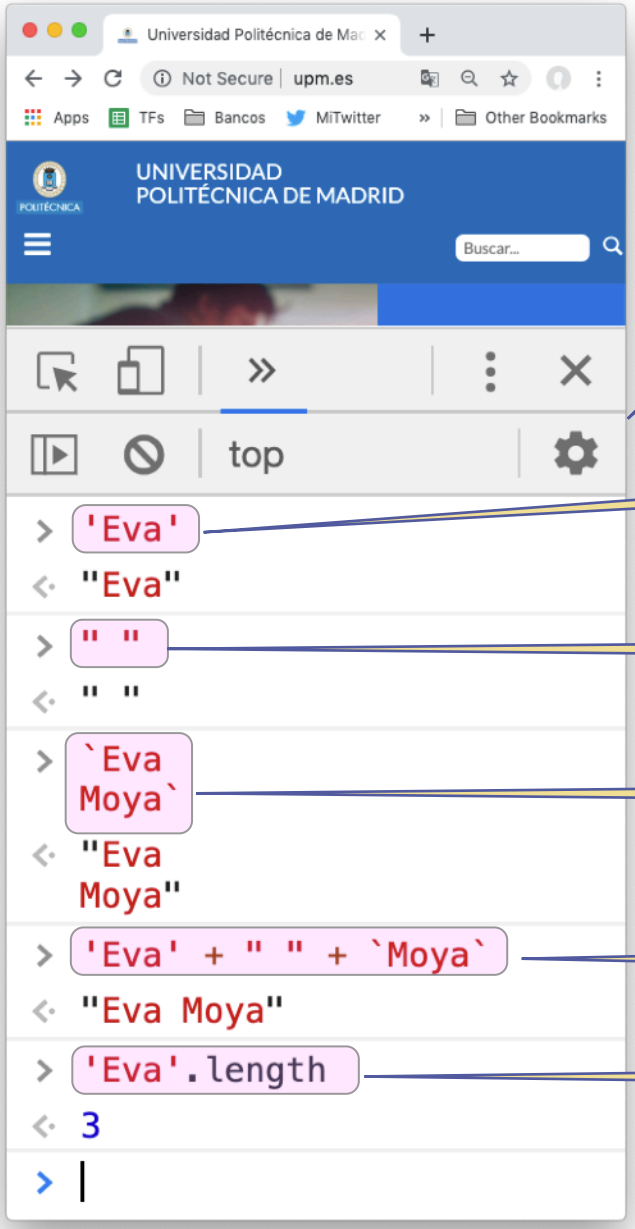
Esta captura muestra cómo evalúa JavaScript las expresiones de la transparencia anterior.

Teclear una expresión erronea genera un mensaje de error. Haciendo clic en enlace da más info sobre el error.

Tipo string: texto

- El texto escrito se representa en JavaScript con strings
  - Un string delimita el texto con **comillas, apóstrofes** o **acento grave**, por ej.
    - **"hola, que tal"** o **'hola, que tal'** // Las comillas y apóstrofes solo permiten strings mono-línea
    - **`hola,**
      que tal`** o **`hola, que tal` // El acento grave permite strings multi-línea y mono-línea
- Algunos ejemplos
  - **"entre 'comillas' "** o **`entre 'comillas' `** // 'comillas' es parte del texto
  - String vacío: "" o '' o ``
- Operador **+** concatena strings, por ejemplo
  - **´Hola´ + " " + 'Pepe'** => **"Hola Pepe"**
- La **propiedad length** de un string indica su longitud (Número de caracteres)
  - **"Pepe".length** => **4**
  - **`Hola Pepe`.length** => **9**
- Doc: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Ejemplos de string



Consola JavaScript del navegador (chrome). Todos los navegadores permiten desplegarla y ejecutar instrucciones de forma interactiva.

Los strings "Eva", 'Eva' o ´Eva´ son literales de string que representan exactamente el mismo string o texto.

El string " ", ' ' o ` ` representa el carácter espacio (space) o blanco (blank), que separa palabras en un texto.

Un string delimitado con acento grave (`..) puede ser multi-línea, como aquí.

El operador + aplicado a strings los concatena o une, generando un nuevo string con su unión. Es asociativo y permite concatenar más de 2 strings.

"Eva".length devuelve el contenido de la propiedad length del string, el número de caracteres del string.

Sobrecarga de operadores

- Los operadores son caracteres especiales
  - Por ejemplo, el operador suma tiene asignado el carácter **+**
    - El operador + está sobrecargado con 3 semánticas (significados) diferentes
      - El intérprete utiliza las reglas sintácticas de JavaScript para determinar la semántica
- Suma de números
  - Operador binario de números (tipo number), infijo (operador en medio)

13 + 7 => 20

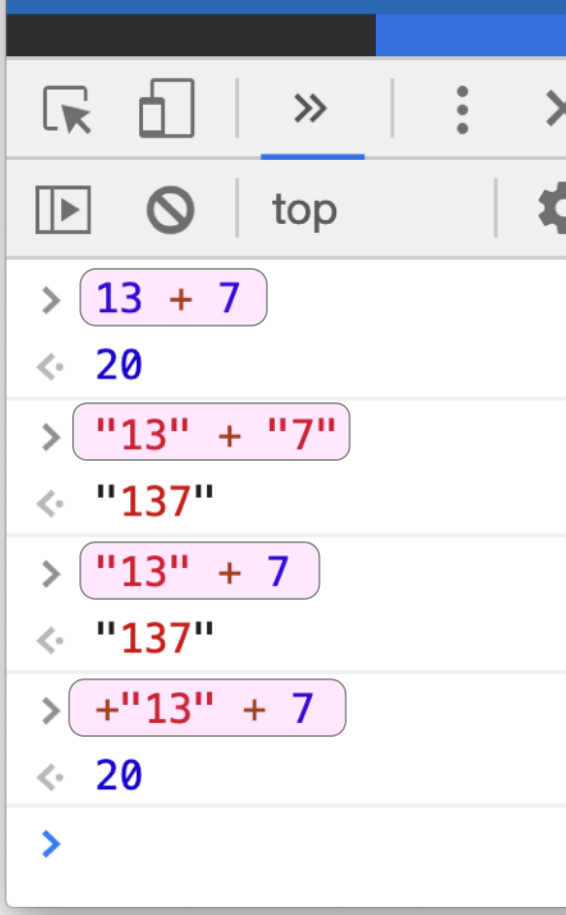
- Signo de un número
  - Operador prefijo unario de número que define el signo positivo de un número

+13 => 13

- Concatenación de strings
  - Operador binario de cadenas de caracteres (tipo string), infijo (op. en medio)

"Hola " + "Pepe" => "Hola Pepe"

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
      - Concatenación (strings) es más prioritaria que la suma
- La expresión "13" + 7 es ambigua
  - porque combina un string con un number
    - Si hay ambigüedad, JavaScript da prioridad al operador + de concatenación de strings, convirtiendo 7 a string y concatenando ambos strings
- La expresión +"13" también necesita conversión automática de tipos
  - El operador + solo está definido para number (no hay ambigüedad)
    - JavaScript debe convertir el string "13" a number antes de aplicar operador +

La prioridad de los operadores es descendente y de izquierda a derecha. (Mayor si más arriba o más a izq.)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

. [] new	**Acceso a propiedad o invocar método; Índice de array; Crear objeto**
()	**Invocación de función/método o evaluar expresión**
++ --	**Pre o post auto-incremento; Pre o post auto-decremento**
! ~	**Negación lógica (NOT); complemento de bits**
+ -	**Operador unitario, números. Signo positivo; Signo negativo**
delete	**Borrar propiedad de un objeto**
typeof void **	**Devolver tipo; Valor indefinido** (operador de escasa utilidad); **Potencia** (ES6)
*** / %**	**Números. Multiplicación; División; Modulo (o resto)**
+ + -	**Concatenación de strings; Números: Suma y Resta**
<< >> >>>	**Desplazamientos de bit**
< <= > >=	**Menor; Menor o igual; Mayor; Mayor o igual**
instanceof in	**¿Objeto pertenece a clase?; ¿Propiedad pertenece a objeto?**
== != === !==	**Igualdad; Desigualdad; Identidad; No identidad**
&	**Operación y (AND) de bits**
^	**Operación ó exclusivo (XOR) de bits**
|	**Operación ó (OR) de bits**
&&	**Operación lógica y (AND)**
||	**Operación lógica o (OR)**
?:	**Operador ternario condicional**
=	**Asignación de valor**
OP=	**Asignación con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=**
yield (ES6)	**Generar nuevo elemento de un "iterable"**
... (ES6)	**Distribuir (Spread) elems (array, obj, ..) o agrupar (Rest) parámetros (función)**
,	**Evaluación múltiple**

8*2 - 4 => 12

* tiene más prioridad que **-**, pero **(..)** obliga a evaluar antes **-** en:

8*(2 - 4) => -16

Operadores ES5 y ES6

La prioridad de los operadores es descendente y de izquierda a derecha. (Mayor si más arriba o más a izq.)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

. [] new	**Acceso a propiedad o invocar método; Índice de array; Crear objeto**
()	**Invocación de función/método o evaluar expresión**
++ --	**Pre o post auto-incremento; Pre o post auto-decremento**
! ~	**Negación lógica (NOT); complemento de bits**
+ -	**Operador unitario, números. Signo positivo; Signo negativo**
delete	**Borrar propiedad de un objeto**
typeof void **	**Devolver tipo; Valor indefinido** (operador de escasa utilidad); **Potencia** (ES6)
*** / %**	**Números. Multiplicación; División; Modulo (o resto)**
+ + -	**Concatenación de strings; Números: Suma y Resta**
<< >> >>>	**Desplazamientos de bit**
< <= > >=	**Menor; Menor o igual; Mayor; Mayor o igual**
instanceof in	**¿Objeto pertenece a clase?; ¿Propiedad pertenece a objeto?**
== != === !==	**Igualdad; Desigualdad; Identidad; No identidad**
&	**Operación y (AND) de bits**
^	**Operación ó exclusivo (XOR) de bits**
|	**Operación ó (OR) de bits**
&&	**Operación lógica y (AND)**
||	**Operación lógica o (OR)**
?:	**Operador ternario condicional**
=	**Asignación de valor**
OP=	**Asignación con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=**
yield (ES6)	**Generar nuevo elemento de un "iterable"**
... (ES6)	**Distribuir (Spread) elems (array, obj, ..) o agrupar (Rest) parámetros (función)**
,	**Evaluación múltiple**

+"3" + 7 => 10

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

Operadores ES5 y ES6

Programas, sentencias var, let, const, y ops ++, --, +=, -=, ..

Juan Quemada, DIT - UPM

Programa, sentencia y código fuente

- **Programa**: secuencia de sentencias
  - Se ejecutan en el orden en que están escritas
    - Salvo tomas de decisiones (if..else, ..) y bucles (while, ..)
- **Sentencia**: orden al procesador
  - Especifica una **tarea** a realizar por el procesador
    - El punto y coma (**;**) indica final de sentencia
      - Aunque es opcional se recomienda incluir siempre!
    - Las sentencias pueden acabar también con nueva línea (\n)
      - Pero, no se recomienda terminarlas así!
- **Comentario**: solo tienen valor informativo
  - Documenta el programa y ayuda a entenderlo mejor
    - Hay dos tipos de comentarios: mono-línea y multi-línea
- **Código fuente**: texto con las sentencias y comentarios de un programa
  - Se edita con **editor de texto plano**: nano, notepad, vi, vim, sublime-text, ….
    - **Fichero fuente**: fichero que contiene un programa JavaScript ejecutable

Sentencia 1: define la **variable x** con **valor 7**.

Comentario multi-línea: delimitado con **/* .... */**

```
/* Ejemplo  de
   programa JavaScript */

let x = 7;  // def. de variable
// visualizar x en el navegador
document.write(x * 1.13);
```

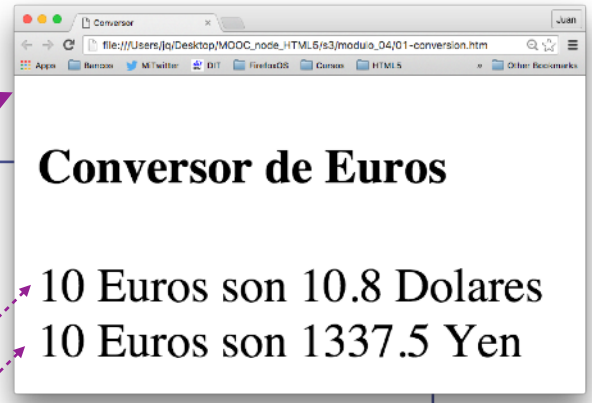
Sentencia 2: Muestra el **valor x * 1,13** en el navegador

Comentario mono-línea: empieza con **//** y acaba al final de la línea.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Values,_variables,_and_literals
https://javascript.info/structure

La etiqueta <script> de HTML

- **Script**: programa **JavaScript** encapsulado entre marcas **<script>**
  - Se ejecuta al cargar en el navegador la página Web que lo contiene
    - JavaScript es un lenguaje interpretado que ejecuta las instrucciones a medida que las va leyendo
  - **document.write(<expresión>)** convierte <expresión> a string y lo visualiza en el navegador
    - El string se interpreta como HTML y se visualiza en el lugar de la página donde esta el script JavaScript
- Una **variable** guarda valores para uso posterior
  - Una variable representa el valor que contiene
    - Puede utilizarse en expresiones como cualquier otro valor




```
<!DOCTYPE html><html>
<head>
  <title>Conversor</title><meta charset="UTF-8">
</head>
<body>
  <h4>Conversor de Euros</h4>

  <script type="text/javascript">

    let euro = 10;
    document.write(euro + " Euros son " + euro*1.08 + " Dolares<br>");
    document.write(euro + " Euros son " + euro*133.75 + " Yen");
  </script>
</body>
</html>
```

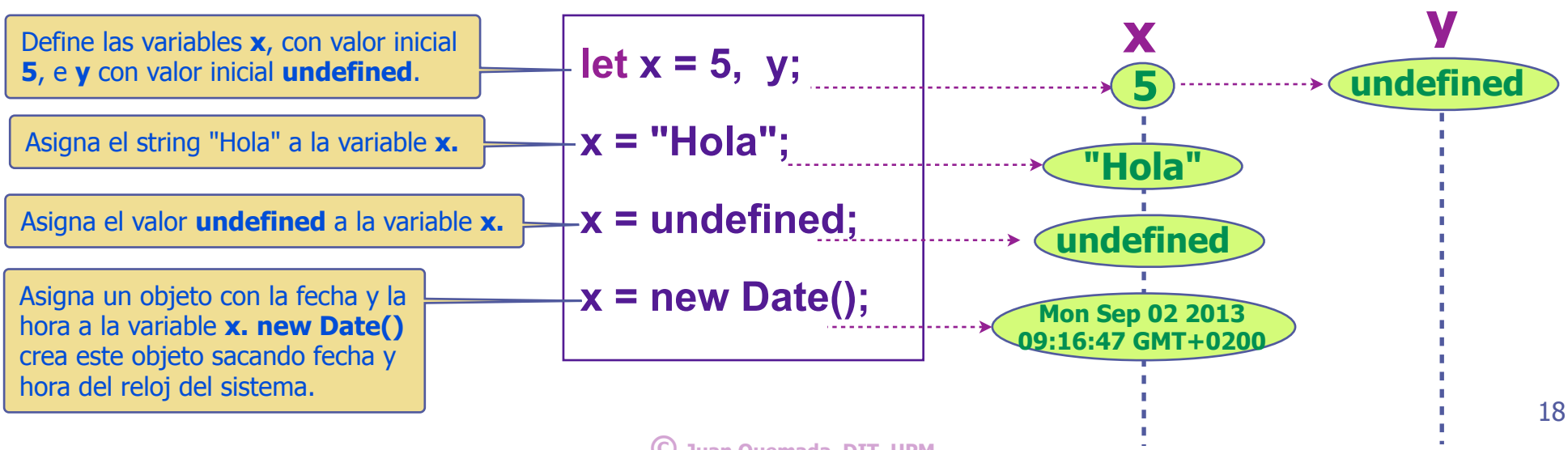
Define la variable euro con valor 10

Visualizan en el navegador el resultado de evaluar las expresiones

Separación de línea HTML

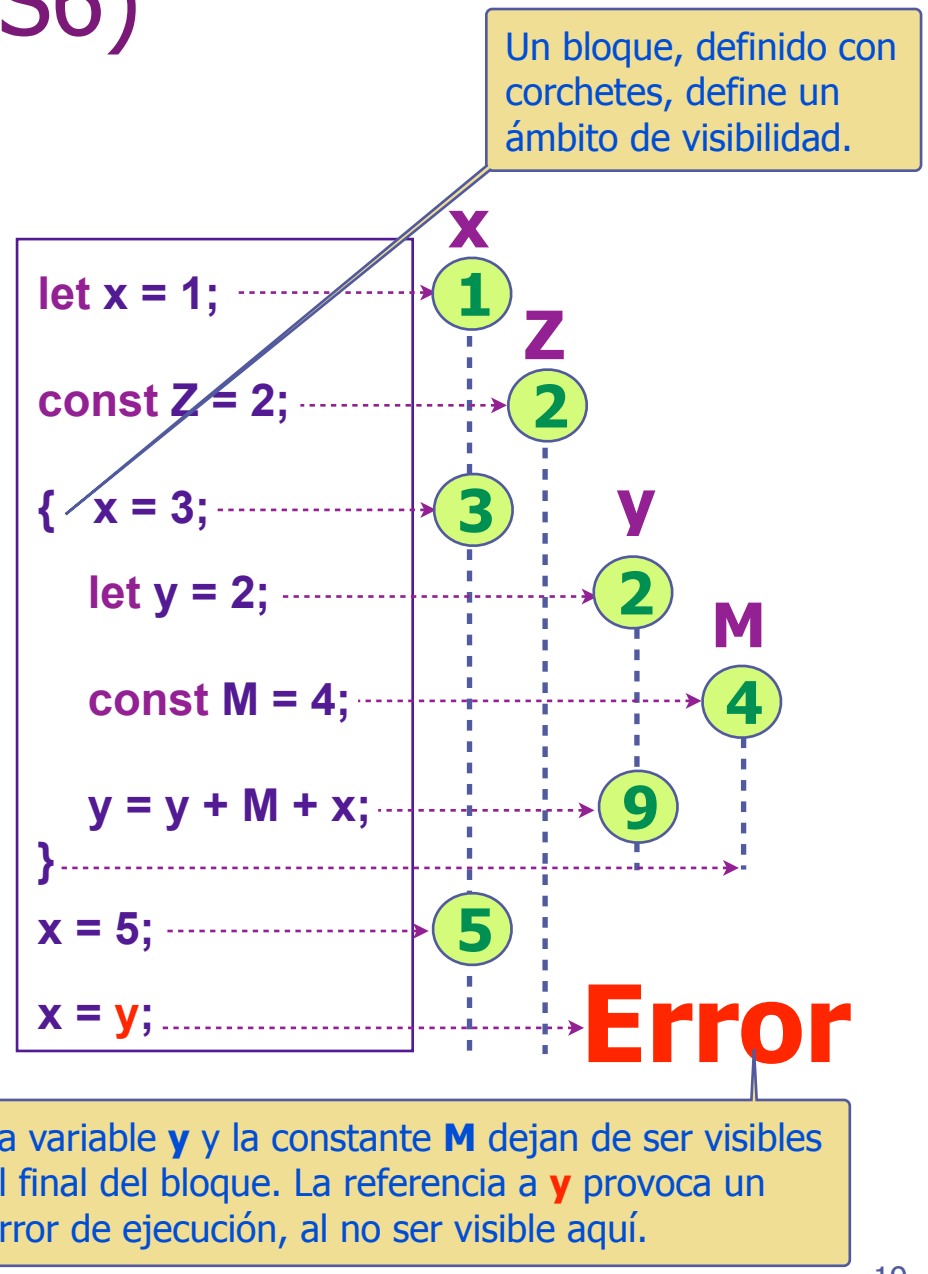
Definición y asignación de variables

- Una variable es un **contenedor de un valor**, cuyo contenido puede variar
- Las variables JavaScript son **no tipadas** y pueden contener cualquier tipo de valor
  - Pueden contener números, strings, undefined, objetos, ..
- La sentencia de **definición de variables** debe empezar por la palabra reservada **let** (ES6+)
  - Seguida de una o más definiciones de variables separadas por comas
    - En ES5 se utilizaba **var** pero ahora no se recomienda utilizarlo
- La sentencia de **asignación** introduce un nuevo valor en la variable, p.e. **x = 8**;
- **undefined**: valor especial que significa **indefinido**
  - Si no se le asigna ningún valor inicial a la variable, contendrá el valor **undefined**



Constantes y variables (ES6)

- **let** (ES6) define **variables**
  - La variables son **no tipadas**
    - Pueden contener cualquier tipo de valor
  - Tienen ámbito de visibilidad de **bloque**
    - Son visibles desde su definición hasta el final del bloque y en bloques de nivel inferior
  - Doc: https://javascript.info/variables
- **const** (ES6) define **constantes**
  - Se debe **asignar un valor** al definirlas
    - El valor **no** puede **modificarse**
  - Suelen estar en **mayúsculas**
- Definiciones del ámbito exterior
  - Se denominan **globales** y son **visibles** en **todo** el programa
- **var** define **variables** ES5
  - Se recomienda **no utilizarlas** salvo cuando se deba utilizar **ES5**
    - Tienen visibilidad en todo el programa y no son seguras (https://javascript.info/var)

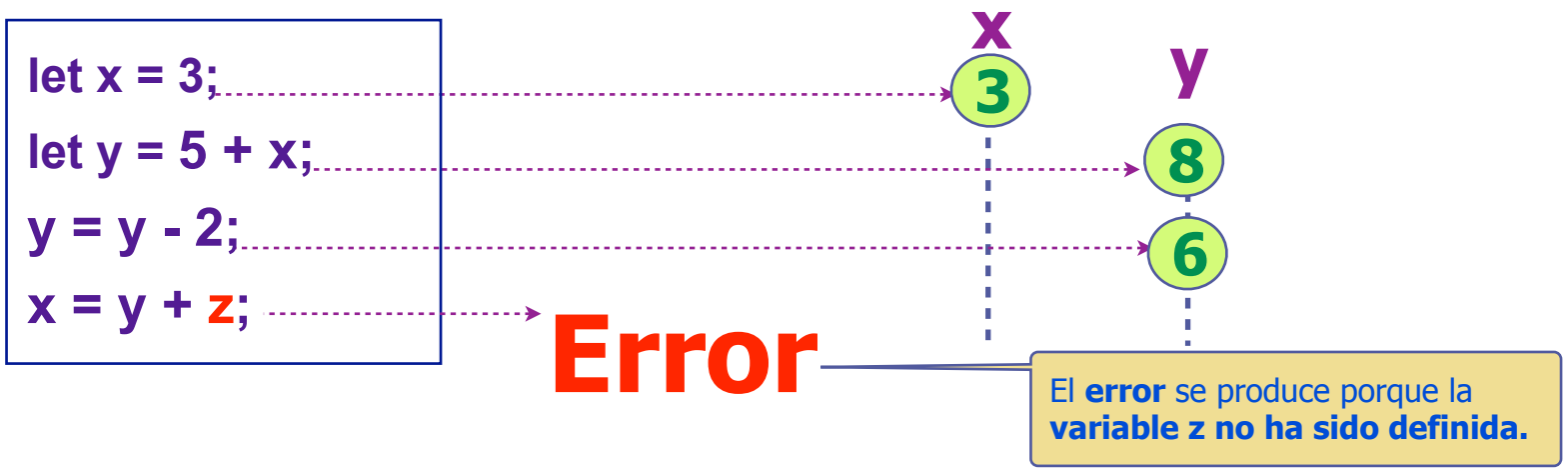


Sintaxis: variables y constantes

- El **nombre** (o identificador) de una variable o constante debe comenzar por:
  - **letra**, **_** o **$**
    - El nombre pueden contener además **números**
  - Nombres **bien construidos:** **x, ya_vás, $A1, $, _43dias**
  - Nombres **mal construidos:** **1A, 123, %3, v=7, a?b, ..**
    - Nombre incorrecto: da error_de_sintaxis e interrumpe el programa
- Un nombre de variable **no** debe ser una **palabra reservada** de JavaScript
  - por ejemplo: **var**, **let**, **const**, **function**, **return**, **if**, **else**, **while**, **for**, **new**, …
- Las variables son sensibles a **mayúsculas**
  - **mi_var** y **Mi_var** son variables distintas

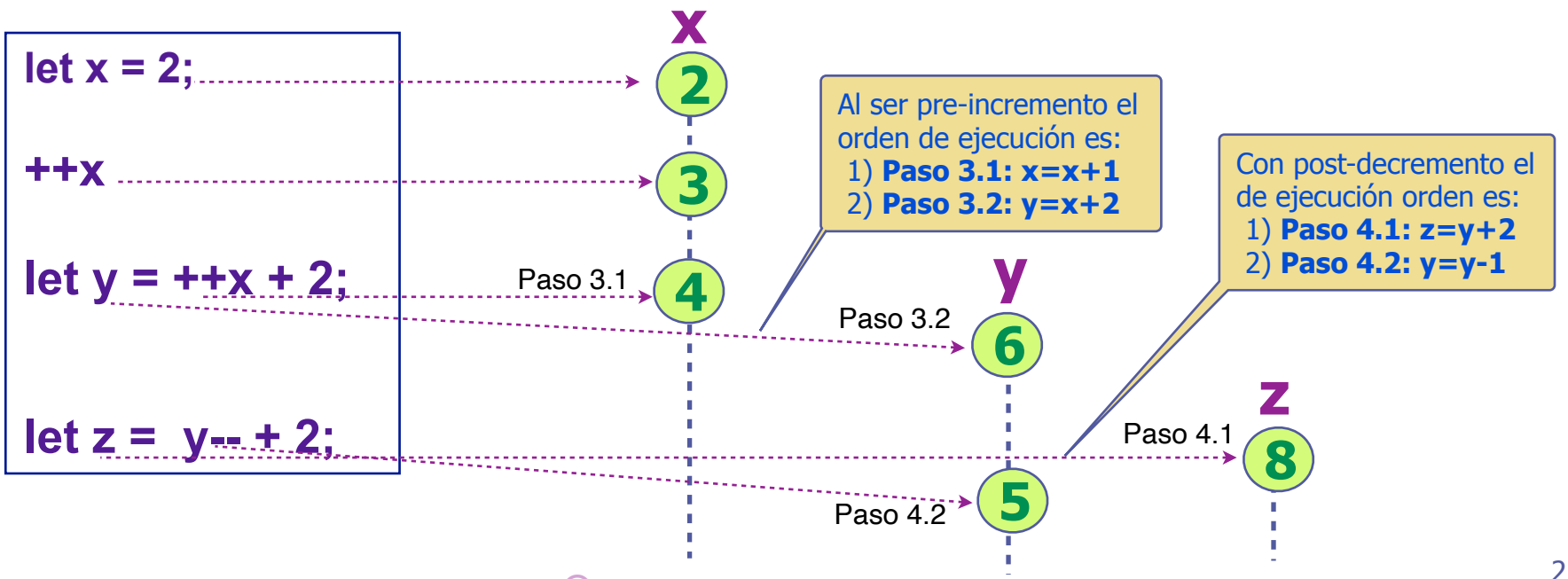
Expresiones con variables

- Una **variable** representa el **valor** que contiene
  - Puede ser usada en expresiones como cualquier otro valor
- Una variable puede utilizarse en una expresión asignada a si misma
  - La parte derecha usa el valor anterior a la ejecución de la sentencia
    - Si **y** contiene **8** antes de ejecutar **y = y - 2**, esta asigna el valor **6 (8-2)** a **y**
- Usar una variable no definida en una expresión
  - provoca un **error** y la ejecución del programa se **interrumpe**



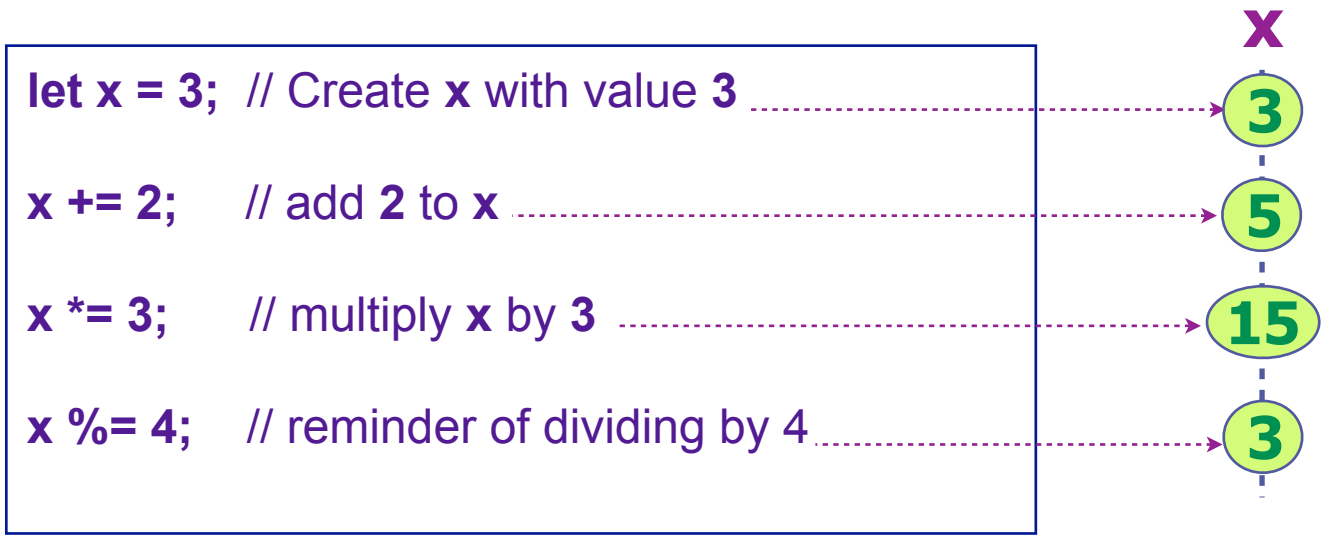
Pre y post auto incremento o decremento

- JavaScript posee los operadores **++** y **--** de **auto-incremento** o **decremento**
  - **++** suma 1 y **--** resta 1 a la variable a la que se aplica
    - **++** y **--** se pueden aplicar por la derecha o por la izquierda a las variables de una expresión
      - 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
  - **Ojo!** Usar con cuidado, sus efectos laterales llevan a programas difíciles de entender.
- Documentación adicional
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators



Operadores de asignación

- JavaScript tiene operadores especiales de asignación
  - +=, -=, *=, /=, %=, …… (para otros operadores del lenguaje)
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators
- Estos componen la variable con el operador y la expresión
  - Por ejemplo: x += 7; es lo mismo que x = x + 7;

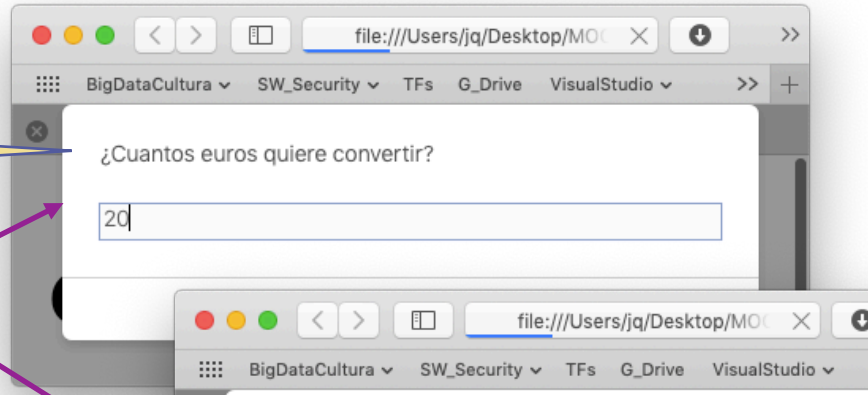


Ventanas popup: prompt, confirm y alert

- **JavaScript** incluye tres funciones globales para interactuar con el usuario
  - **alert(msj)**: Presenta un pop-up con mensaje al usuario y retorna al pulsar OK
  - **confirm(msj)**: Presenta un pop-up con mensaje y pide confirmación/rechazo
    - Retorna al pulsar y devuelve true al pulsar Ok o false al pulsar Cancel
  - **prompt(msj)**: Presenta mensaje y pide un dato de entrada
    - Retorna al pulsar OK (devuelve string introducido) o Cancel (devuelve null)
- Este ejemplo pide un valor con **prompt()** y muestra el resultado con **alert()**

```
<!DOCTYPE html><html>
<head><title>Conversor</title><meta charset="UTF-8">
<body>
  <h1>Conversor de Euros</h1>

  <script type="text/javascript">

    let euro = prompt("¿Cuantos euros quiere convertir?");
    alert(euro + " Euros son " + euro*1.08 + " Dolares\n" +
          euro + " Euros son " + euro*133.75 + " Yen");
  </script>

Recarge la página para volver a convertir!
</body>
</html>
```

Pop-up generado con **prompt(...)** que pide un **valor** para inicializar la variable **euro**.



Pop-up generado con **alert(...)** que muestra el resultado.

Página HTML final.

Las aplicaciones de cliente se relanzan cargando la página de nuevo.

Ejecución de scripts, funciones, objeto función, notación arrow y ámbitos

Juan Quemada, DIT - UPM

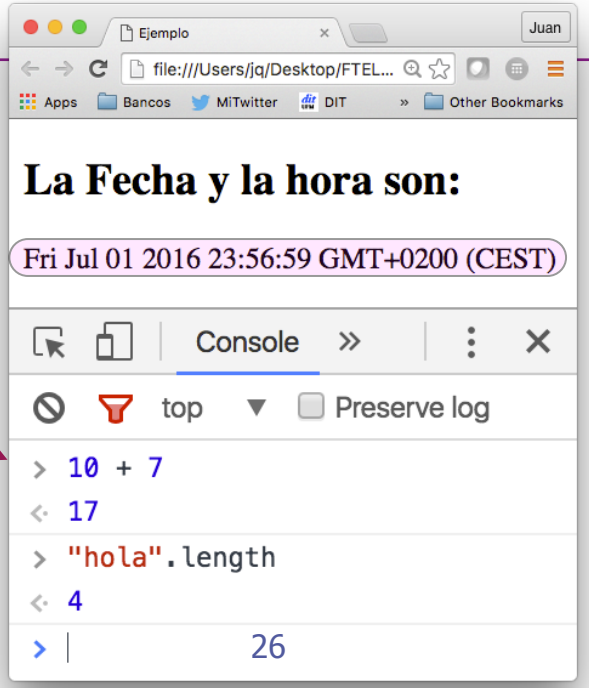
JavaScript en el Navegador

- Navegador Web: ejecuta JavaScript de 2 formas
  1. Como un **script** en una página HTML
  2. En la **consola** JavaScript

- El navegador ejecuta **scripts** JavaScript
  - Al **cargar** una página Web
    - El script permite interacción con ratón, teclado, …
  - Un **script** es un **elemento HTML**
    - **<script type="text/javascript">**
      - ….. programa JavaScript …..
    - **</script>**

- El navegador tiene además una **consola JavaScript**
  - Muy similar a la consola interactiva de node.js
    - Permite ejecutar instrucciones paso a paso una vez cargada la página y ejecutados sus scripts

```
<!DOCTYPE html><html>
<head>
  <title>Date</title>
  <meta charset="UTF-8">
</head>

<body>
  <h2>La fecha y la hora son:</h2>

  <script type="text/javascript">

    document.write(new Date( ));
  </script>

</body>
</html>
```

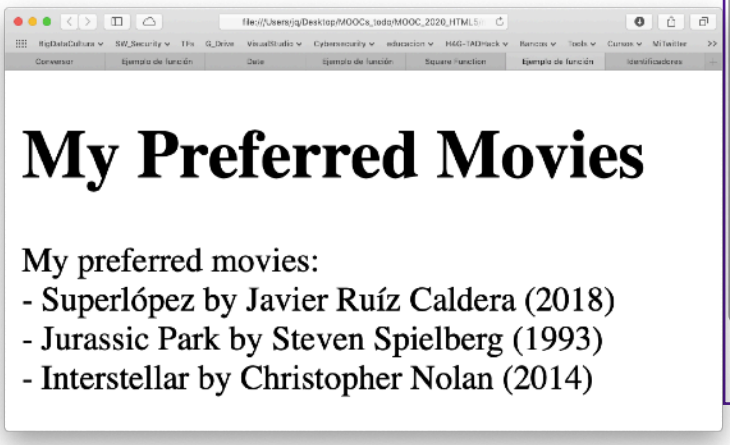


Varios scripts

- **Varios scripts** en una página forman un **único programa JavaScript**
  - Las definiciones (variables, funciones, …) son visibles entre scripts de una misma página Web
- Los **scripts se ejecutan** siguiendo el **orden** en la página
  - **Instrucciones adicionales** ejecutadas en la consola del navegador, se ejecutarán **después del último script**
- **mostrar_fecha()** debe invocarse en el punto donde debe ir el texto
  - Su definición se realiza en la cabecera y es visible en otros scripts


```
<!DOCTYPE html><html><head>
<title>Ejemplo de función</title>
<meta charset="UTF-8">

<script type="text/javascript">

function mostrar_fecha( ) {
  document.write(new Date( ));
}
</script>
</head>
<body>
<h2>La fecha y la hora son:</h2>

<script type="text/javascript">

  mostrar_fecha( );
</script>
</body>
</html>
```

Función



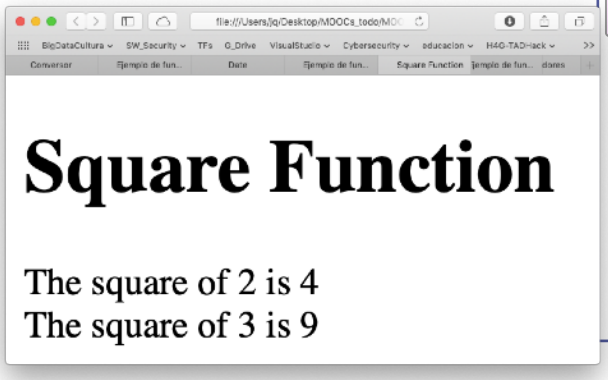

```
<!DOCTYPE html><html>
<head><title>Ejemplo de función</title><meta charset="UTF-8"></head>

<body><h1>My Preferred Movies</h1>

<script type="text/javascript">

function my_preferred_movies () {
  document.write("My preferred movies: <br>");
  document.write(" - Superlópez by Javier Ruíz Caldera (2018) <br>");
  document.write(" - Jurassic Park by Steven Spielberg (1993) <br>");
  document.write(" - Interstellar by Christopher Nolan (2014) <br>");
}

my_preferred_movies();

</script>
</body>
</html>
```

Definición de la función

Invocación (ejecución) de la función

- Una **función** encapsula código, que se invoca (ejecuta) por su **nombre**
  - Una función debe definirse primero, para poder invocarla (ejecutarla) posteriormente
    - Documentación: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Funciones
- La **definición** de la función comienza por la palabra reservada: **function**
  - A continuación viene el **nombre** de la función
    - En tercer lugar vienen los parámetros entre paréntesis: **()** indica sin parámetros en este ejemplo
      - Por último viene el bloque de código, entre corchetes ({...})
- La **invocación** de la función ejecuta el bloque de código de la función
  - Se invoca con el nombre y el operador paréntesis (), por ej. **my_preferred_movies()**

Parámetros de invocación y de retorno




```
<!DOCTYPE html><html>
<head><title>Square Function</title><meta charset="UTF-8"></head>

<body><h1>Square Function</h1>

<script type="text/javascript">

function square (x) {
  return x*x ;
}

document.write("The square of " + 2 + " is " + square(2));
document.write("<br>");
document.write("The square of " + 3 + " is " + square(3));

</script>
</body>
</html>
```

Definición de la función

Invocación (ejecución) de la función

- Una **función** recibe **parámetros** de entrada (parámetro **x** del ejemplo)
  - Cada parámetro define una **variable,** que solo es **visible** dentro del bloque de código
    - Como es una variable, el valor asignado puede cambiarse (aunque esto no se hace en el ejemplo)
  - El valor inicial del parámetro es el **valor pasado al invocar la función (**valores 2 y 3 del ejemplo)
- Una **función** devuelve un **parámetro de retorno** con la sentencia: **return <expr>;**
  - Esta sentencia finaliza la ejecución de la función y devuelve el valor resultante de evaluar **<expr>**
    - Si la **función** llega a final del bloque **sin ejecutar** return, finaliza y devuelve **undefined**
- Una función puede componerse en expresiones como otro valor más
  - La función se ejecutará y se sustituirá por el valor retornado en la expresión

Función con varios parámetros

```
function greet (greeting, person) {
      return  greeting + " " + person + ", how are you?" ;
};

greet ("Good morning", "Peter");     //  =>  "Good morning Peter, how are you?"
greet ("Hi", "Peter");               //  =>  "Hi Peter, how are you?"

greet ("Hi", "Peter", "John");       //  =>  "Hi Peter, how are you?"
greet ("Hi");                        //  =>  "Hi undefined, how are you?"
greet ();                            //  =>  "undefined undefined, how are you?"
```

- La función **greet(..)** genera saludos en inglés utilizando 2 parámetros
  - Los parámetros se concatenan con texto para generar los saludos mostrados
- Una función **se puede invocar** con un **número variable de parámetros**
  - Un parámetro definido, pero **no pasado** en la invocación, toma el valor **undefined**

arguments: el array de parámetros

```
function greet () {
    return  `${arguments[0]} ${arguments[1]}, how are you?`;
};

greet ("Good morning", "Peter");     //  =>  "Good morning Peter, how are you?"

greet ("Hello", "Peter");                    //  =>  "Hello Peter, how are you?"
```

- Una función tiene un array de nombre **arguments**
  - **arguments** contiene los valores de los parámetros en la invocación
    - El array **arguments** permitiría saber su número total y acceder a todos
- Esta función greet(..) es similar a la anterior
  - Pero utiliza **arguments** en vez de **parámetros explícitos**

Funciones como objetos

- Las **funciones** son objetos de pleno derecho
  - pueden **asignarse a variables**, a **propiedades**, pasarse como **parámetros**, ….
- **Literal de función:** function (<argumentos>){<sentencias>}
  - Construye un objeto de tipo función que no tiene nombre
    - Puede guardarse en variables o parámetros como cualquier otro valor
      - Se invoca aplicando el operador paréntesis: **()**
- El **operador paréntesis ()** ejecuta el código de un objeto function
  - Este operador solo es aplicable a funciones (objetos de la clase Function), sino da error
    - Se pueden incluir parámetros explícitos separados por coma, accesibles en el código de la función

```
const greet = function (greeting, person) {
      return  `${greeting} ${person}, how are you?` ;
};

greet ("Hi", "Peter");                    //   =>  "Hi Peter, how are you?"

const x = greet;

x ("Hi", "Peter");                        //   =>  "Hi Peter, how are you?"
```

Valores por defecto de parámetros (ES6)

```
function greet (greeting = "Hi", person = "my friend") {
      return  `${greeting} ${person}, how are you?` ;
};

greet ("Hello");                    //   =>  "Hello my friend, how are you?"
greet ();                           //   =>  "Hi my friend, how are you?"
```

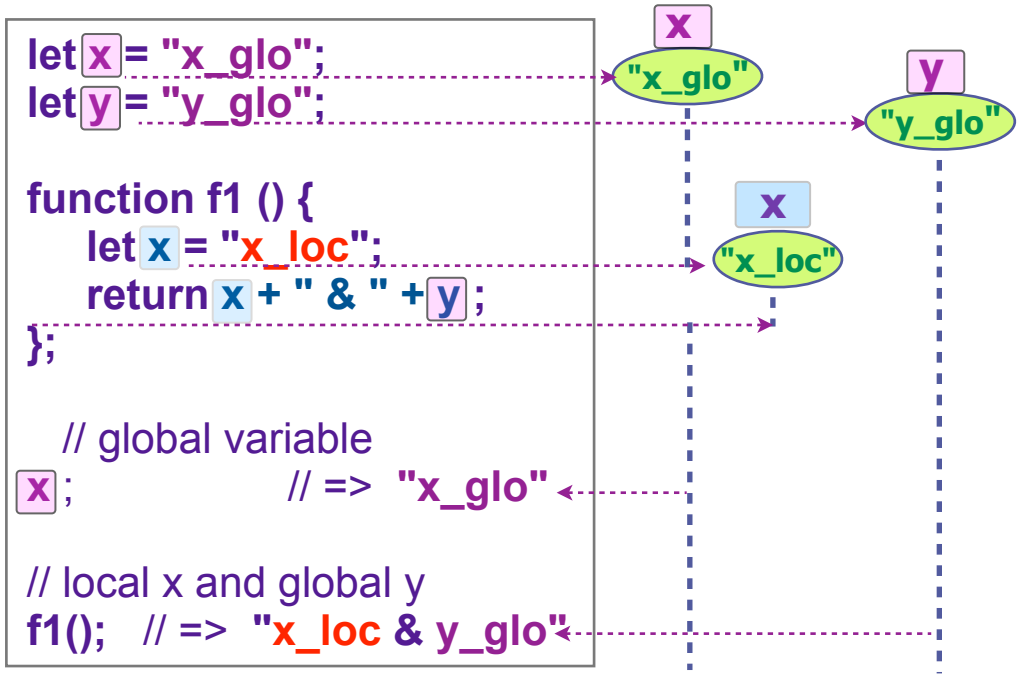
- ES6 permite valores por defecto en parámetros explícitos de funciones
  - Los valores por defecto se asignan al parámetro en la definición
    - utilizando el operador =, como en las definiciones de variables
- El valor por defecto se utiliza en la invocación, cuando ese parámetro está undefined

Notación flecha (arrow) de ES6

- ES6 añade la notación flecha para literales de función (funciones sin nombre)
  - Por ejemplo **(x, y) => {return x+y;)**
    - Es concisa y se recomienda utilizarla en programación con estilo funcional
      - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
- Tiene las siguientes diferencias con funciones tradicionales
  - **No** tienen la variable de argumentos predefinida **arguments**
  - **No modifica** el **contexto**: el objeto **this** tiene **visibilidad léxica**
    - Esto hace que **no** puedan ser **constructores de objetos**

```
const greet = function (greeting, person) {                // defined with function literal
    return `${greeting} ${person}, how are you?` ;
};
      // Similar function to previous one defined with arrow notation
const greet = (greeting, person) => {
    return `${greeting} ${person}, how are you?` ;
};
                                  // Parenthesis may be omitted if only one parameter
const square =   x => x*x;        // One instruction blocks may omit curly brackets and return

const say_hi =   () => "Hi, how are you?";                 // function without parameters
```

Declaraciones locales de una función y ámbito

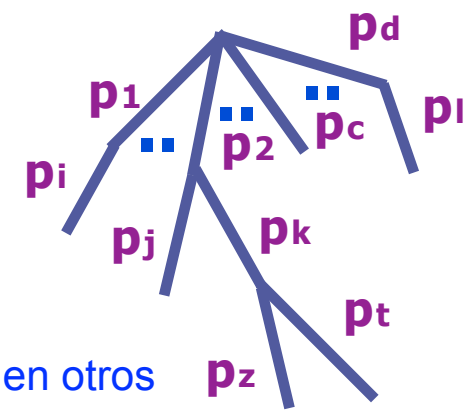


- Las **variables** y **funciones** tienen **visibilidad sintáctica** en JavaScript
  - Son **visibles solo** dentro del **ámbito** donde se declaran
    - En ES5 solo se podían crear **ámbitos de visibilidad** con **funciones**
    - En ES6 el **bloque { ..statements.. }** y el **módulo** crean también nuevos **ámbitos de visibilidad**
  - **OJO!** Las funciones son visibles antes de su declaración (igual que las variables **var**)
- Una **función** puede tener **declaraciones locales** de variables y funciones
  - Las declaraciones son **visibles** solo **dentro de la función**
- **Variables** y **funciones externas** son **visibles** en el bloque de la función
  - Siempre que no sean **tapadas** por otras declaraciones locales del **mismo nombre**
    - Una declaración **local tapa** a una **global** del **mismo nombre**

Objetos, propiedades, métodos, DOM, eventos e interacción

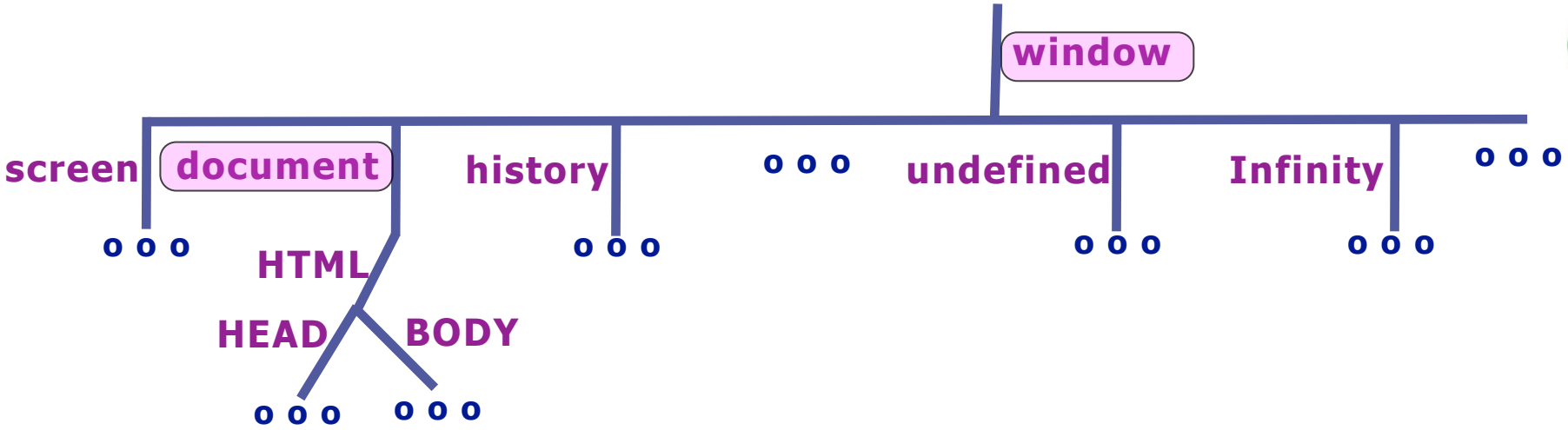
Juan Quemada, DIT - UPM

Objetos JavaScript: métodos y propiedades



- Un **Objeto** es una agrupación de
  - **variables** "especiales" denominadas **propiedades**
  - **funciones** "especiales" denominadas **métodos**
- Las propiedades de los objetos forman un árbol
  - Los objetos pueden crear **árboles** multi-nivel **anidando** unos objetos en otros
- Los **nombres** de **propiedades** y **métodos**
  - Tienen la misma sintaxis que las variables: **a**, **_method**, **$1**, …
  - Conviene que sean diferentes si pertenecen al mismo objeto (aunque no es obligatorio)
- El **operador "."** se utiliza con propiedades y métodos
  - Para acceder a una **propiedad** de un **objeto**, por ejemplo
    - Consulta del valor de la propiedad **y** de **obj**: **obj.y**
    - Asignar el valor de la propiedad **y** de **obj**: **obj.y = 5;**
  - Para invocar un **método** en un **objeto**, por ejemplo **document.write("<h1> Título </h1>")**
    - OJO! Invocar un **método**, que **no pertenece** a un objeto, causa **error**
- Documentación: https://javascript.info/object-basics

Entorno global: window, document y Web APIs

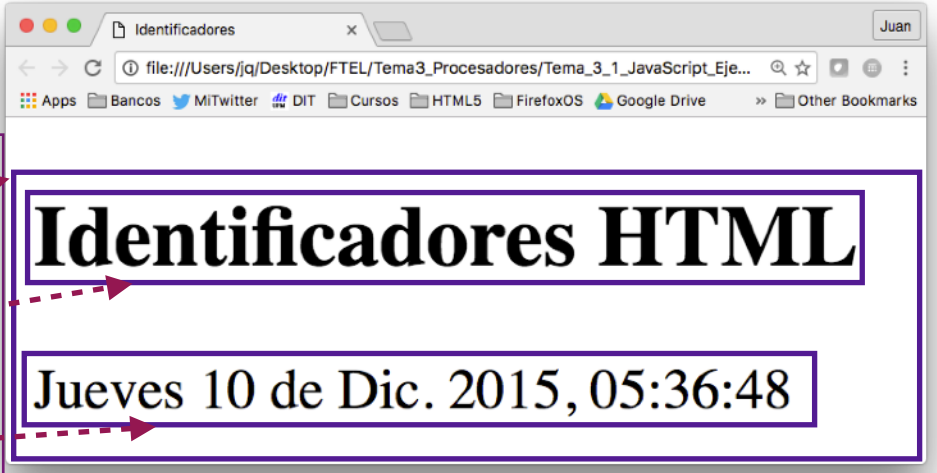


- El objeto **window** es el **entorno global de ejecución** de JavaScript en el navegador
  - Sus propiedades dan acceso a los elementos de pagina Web, del navegador y de JavaScript
    - **this** es una referencia **entorno de ejecución** y referencia **window** cuando el programa esta en el entorno global
      - https://javascript.info/browser-environment
- Este entorno global da también acceso a las numerosas **APIs** del navegador actual
  - DOM, Canvas, Fetch, Storage, Full Screen, Touch Events, Service Workers, WebRTC, WebGL, ..
    - https://developer.mozilla.org/en-US/docs/Web/API
- **document** da acceso a la página HTML con la **API DOM** (Document Object Model)
  - **document** se referencia como: **window.document**, **this.document** o **document**
    - Doc: https://javascript.info/dom-nodes, https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Acceso DOM y cajas visuales

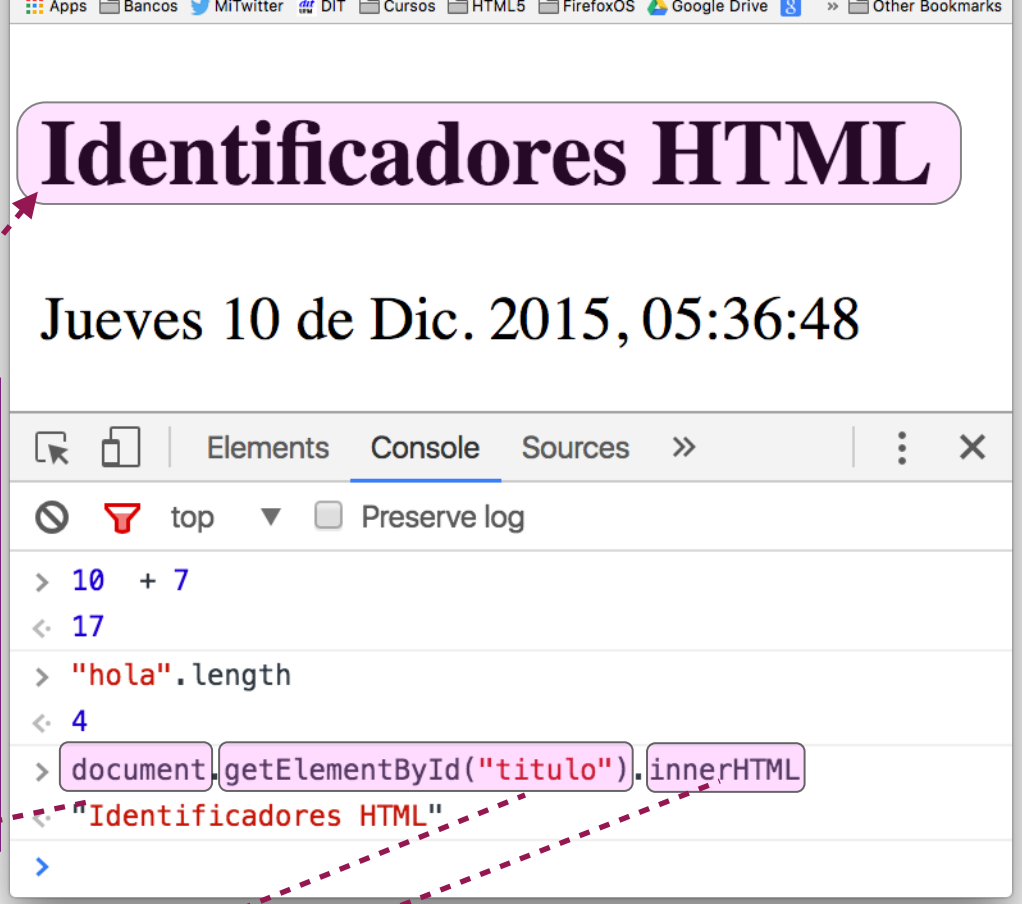
- **Objetos DOM** (Document Object Model)
  - Objetos JavaScript que permiten manipular elementos HTML desde un programa
- **Objeto DOM:** tiene una **caja visual** asociada
  - Las modificaciones afectarán a la caja visual del elemento HTML asociado
    - La caja visual asociada a comandos CSS es la misma que la asociada a DOM
- **document.getElementById("id_x")**
  - Método que obtiene el objeto DOM del elemento HTML con **atributo id="id_x"**

```
<!DOCTYPE html>
<html>
<head>
  <title>Identificadores</title>
  <meta charset="UTF-8">
</head>
<body id="cuerpo">

  <h2 id="titulo">Identificadores HTML</h2>

  <div id="fecha">Jueves 10 de Dic. 2015, 05:36:48</div>
</body>
</html>
```



Acceso a elementos HTML con DOM

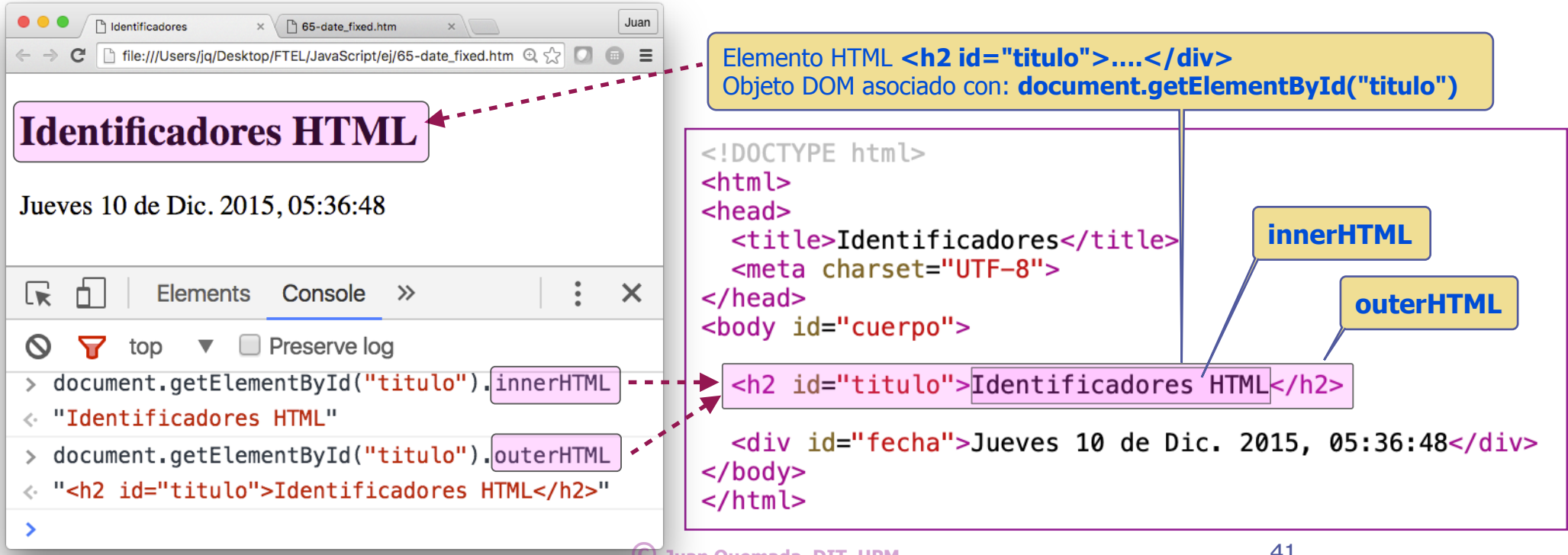
```
<!DOCTYPE html>
<html>
<head>
  <title>Identificadores</title>
  <meta charset="UTF-8">
</head>
<body id="cuerpo">

  <h2 id="titulo">Identificadores HTML</h2>

  <div id="fecha">Jueves 10 de Dic. 2015, 05:36:48</div>
</body>
</html>
```



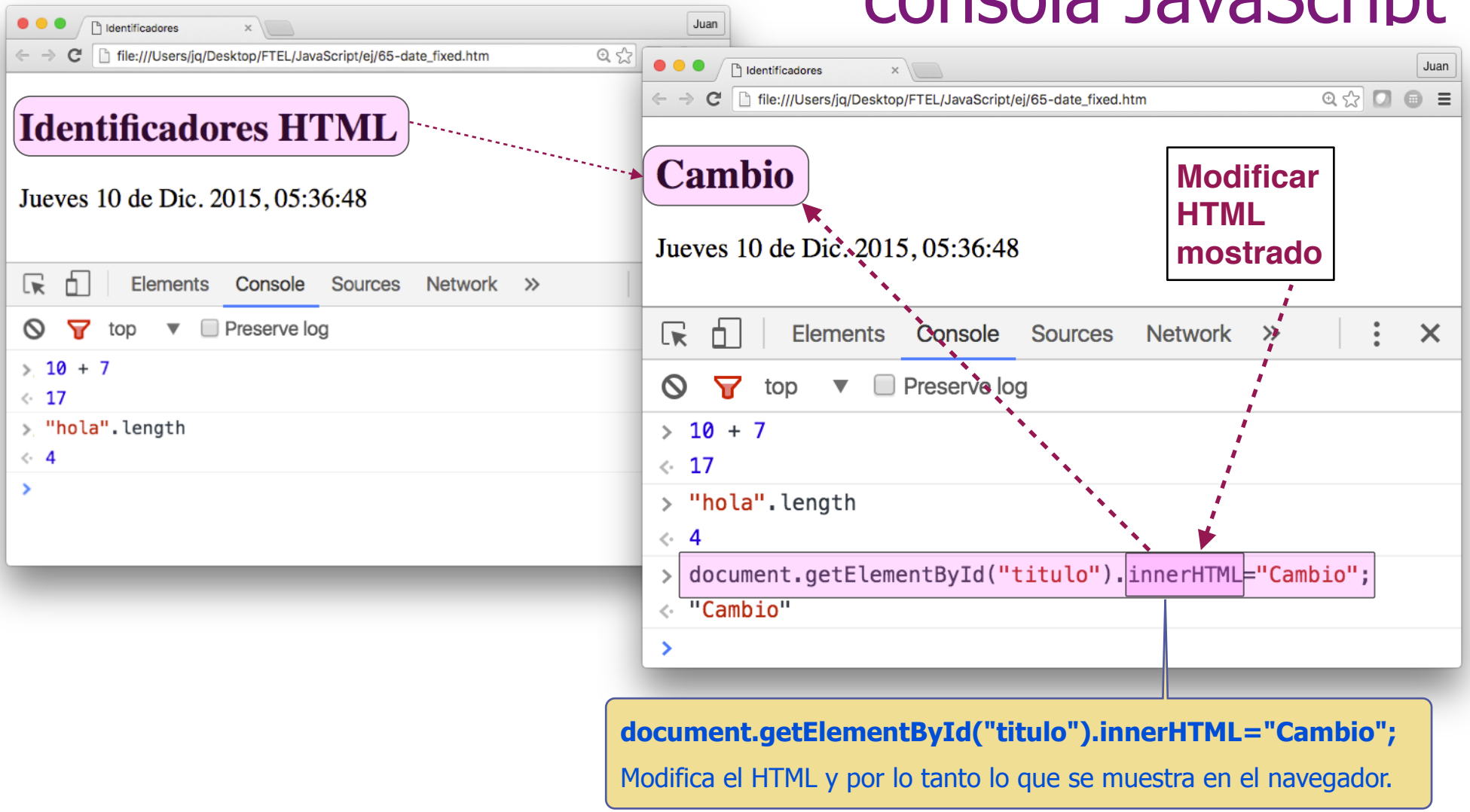
- **document**
  - **Objeto** del entorno que da **acceso al documento HTML**
- **getElementById("titulo")**
  - **Método** que retorna el **objeto DOM** del elemento HTML con **id="titulo"**
- **innerHTML**
  - **Propiedad** con el **HTML interno** asociado a ese objeto DOM

HTML interno y externo

- **HTML interno**: texto HTML contenido entre las marcas del elemento
  - **innerHTML**: propiedad del objeto DOM que da acceso al **HTML interno**
- **HTML externo**: texto HTML completo del elemento (incluyendo sus marcas)
  - **outerHTML**: propiedad del objeto DOM que contiene el **HTML externo**

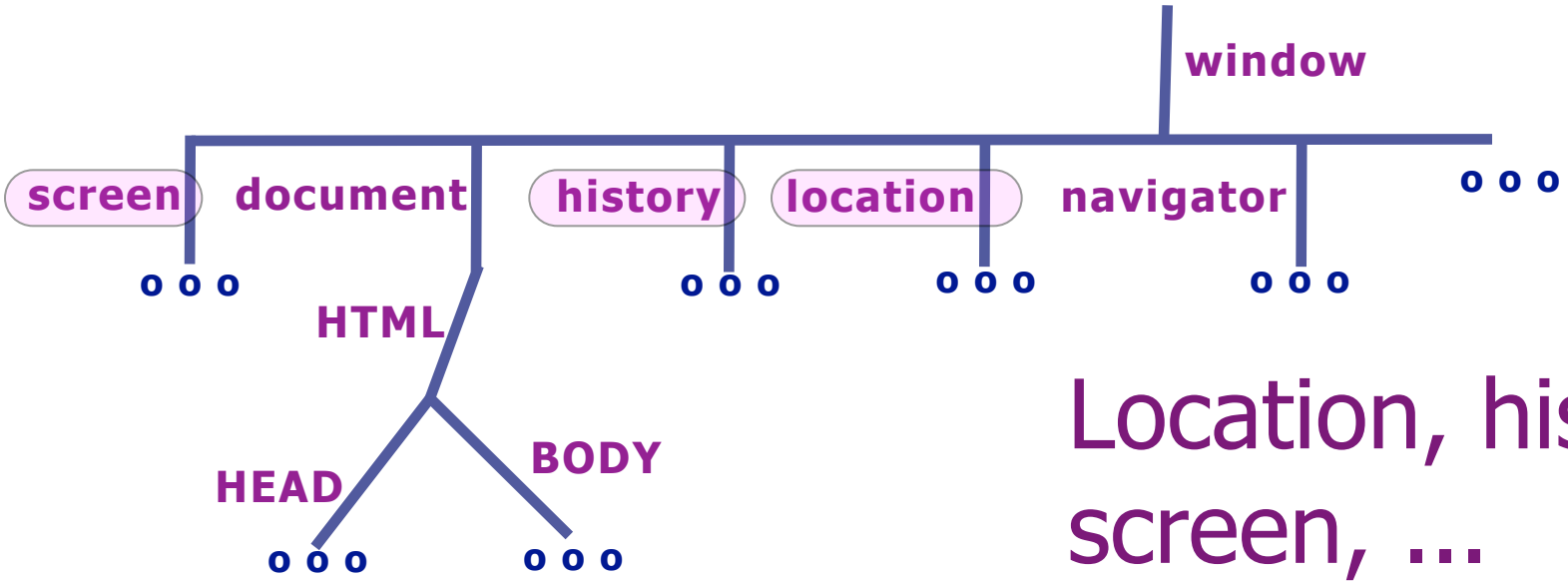


Modificar la página HTML cargada desde la consola JavaScript



document.getElementById("titulo").innerHTML="Cambio";

Modifica el HTML y por lo tanto lo que se muestra en el navegador.



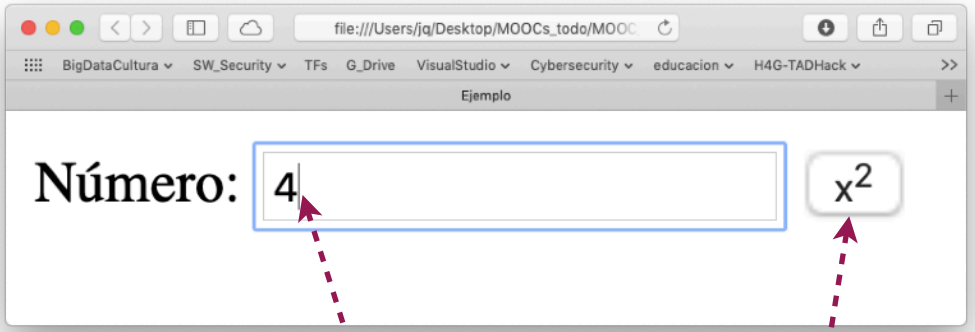
Location, history, screen, ...

- **location:** propiedad que contiene el URL a la página en curso
  - **location = "http://www.upm.es"** Carga una página en el navegador
  - **location.reload()** re-carga la página en curso
    - Doc: https://developer.mozilla.org/en-US/docs/Web/API/Window/location
- **history**: propiedad con la historia de navegación
  - Métodos para navegar por la historia: **history.back()**, **history.forward()**, ...
    - Doc: https://developer.mozilla.org/en-US/docs/Web/API/Window/history
- **screen**: dimensiones de la pantalla
  - **width, height, availWidth, availHeight**: para adaptar apps a pantallas móviles
    - Doc: https://developer.mozilla.org/en-US/docs/Web/API/Window/screen

Propiedades de window

```
<!DOCTYPE html><html>
<head>
  <title>Ejemplo</title>
  <meta charset="utf-8">
  <style>
    span {font-weight: bold}
  </style>
</head>
<body>
  <h1>Propiedades de window</h1>

  La propiedad <span>location.href</span> contiene el URL:<br>
  <span id="i1"></span> <br><br>
  Pixeles <span>(screen.width x screen.height)</span> de mi pantalla:
  <span id="i2"></span>

  <script type="text/javascript">
  document.getElementById("i1").innerHTML = location.href;

  let p = document.getElementById("i2");
  p.innerHTML = `${screen.width} x ${screen.height}`;
  </script>
</body>
</html>
```

Mini-calculadora: elementos HTML de interacción

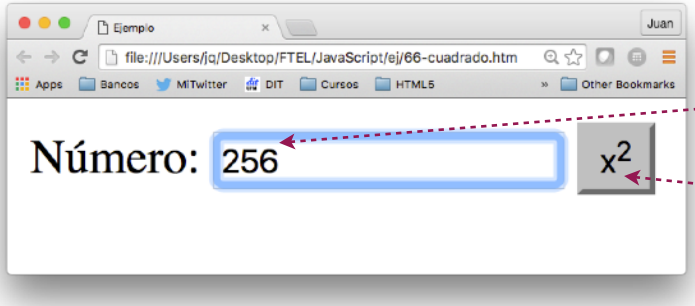


- Interacción con el usuario:
  - **<input type="text" id="n1"….>**
    - **Cajetín**: donde teclear texto
  - **<button ...>nombre</button>**
    - **Botón:** donde hacer clic
- Propiedad **value:** contenido del cajetín
  - Lectura de lo tecleado en el cajetín
    - **document.getElementById("n1").value => "4"**
  - Mostrar resultado en el cajetín
    - **document.getElementById("n1").value = 16**

```
<!DOCTYPE html><html><head>
  <title>Ejemplo</title>
  <meta charset="utf-8">
  <script type="text/javascript">

    function vaciar () {
      document.getElementById("n1").value = "";
    }

    function cuadrado() {
      let num = document.getElementById("n1");
      num.value = num.value * num.value;
    }
  </script>
</head>
<body>
  Número:
  <input type="text" id="n1"
         onclick="vaciar()">

  <button onclick="cuadrado()">
    x<sup>2</sup>
  </button>
</body>
</html>
```

Mini-calculadora: eventos y manejadores

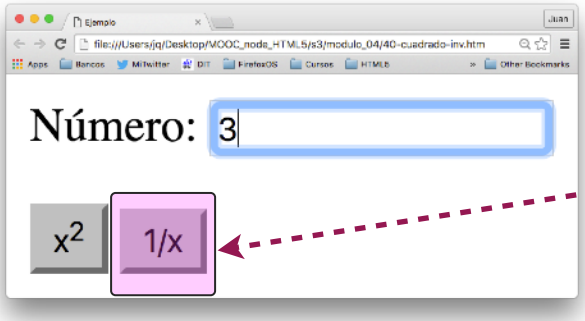
- **Eventos:** se disparan al ocurrir acciones exteriores al programa, p.e. teclear, pulsar botón, ..
  - Tienen funciones asociadas que permiten atender al evento
    - Eventos: http://librosweb.es/libro/javascript/capitulo_6/modelo_basico_de_eventos_2.html
- El atributo **onclick="fx()"**
  - Asocia **fx()** al elemento HTML con el atributo
    - Ejecuta **fx()** al clicar en la caja visual del elemento
- Esta calculadora utiliza 2 eventos
  - Evento 1: **clicar** en el **cajetín**
    - Ejecuta **vaciar():** vacía el cajetín
  - Evento 2: **clicar** en el **botón x²**
    - Ejecuta **cuadrado()**: muestra resultado en cajetín




```
<!DOCTYPE html><html><head>
  <title>Ejemplo</title>
  <meta charset="utf-8">
  <script type="text/javascript">

    function vaciar () {
      document.getElementById("n1").value = "";
    }

    function cuadrado() {
      let num = document.getElementById("n1");
      num.value = num.value * num.value;
    }
  </script>
</head>
<body>
  Número:
  <input type="text" id="n1"
         onclick="vaciar()">

  <button onclick="cuadrado()">
    x<sup>2</sup>
  </button>
 </body>
</html>
```

Mini-calculadora: añadir botón 1/x

- En este ejemplo añadimos
  - Un botón más a la calculadora que calcula el inverso (1/x) del cajetín
- Añadir un nuevo botón es sencillo
  - Se añade la función: **inverso()**
    - Calcula el inverso del número del cajetín
  - Se añaden marcas de nueva línea
    - Para separar el cajetín de los botones
  - Se añade un nuevo **botón** HTML
    - con el texto: **1/x**
    - con atributo : onclick="inverso()"
      - asocia inverso() a clic en él




```
<!DOCTYPE html><html><head>
<title>Ejemplo</title><meta charset="utf-8">
<script type="text/javascript">

function vaciar () {
  document.getElementById("n1").value = "";
}

function cuadrado() {
  let num = document.getElementById("n1");
  num.value = num.value * num.value;
}

function inverso() {
  let num = document.getElementById("n1");
  num.value = 1/num.value;
}
</script>
</head>
<body>

  Número:
  <input type="text" id="n1" onclick="vaciar()">
  <br><br>
  <button onclick="cuadrado()">x<sup>2</sup></button>
  <button onclick="inverso()"> 1/x </button>
</body>
</html>
```

document y sus métodos de acceso

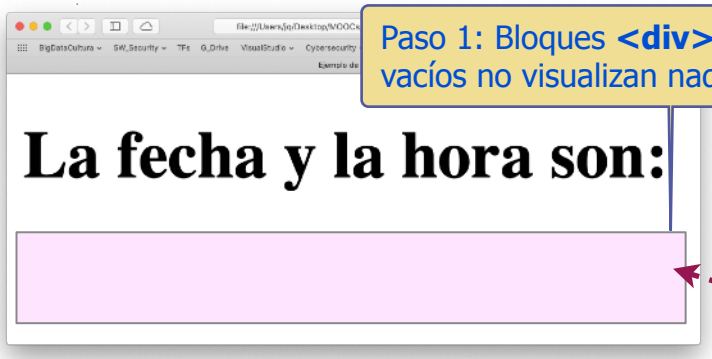
- **document**
  - Objeto DOM de acceso al documento HTML
    - Incluye los métodos y propiedades necesarios para procesar el documento HTML
      - https://developer.mozilla.org/en/docs/Web/API/Document

- **getElementById("<id>")**
  - Devuelve el objeto DOM con el identificador buscado o null si no lo encuentra
    - https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById

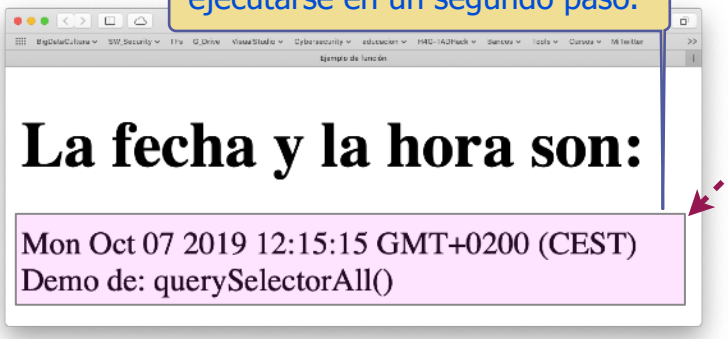
- **querySelectorAll("<CSS_selector>")**
  - Devuelve el array de objetos DOM que casan con <CSS_selector>, por ej.
    - querySelectorAll(**"#id1"**) array con objeto DOM del elemento con **id="id1"**, si existe
    - querySelectorAll(**".cl1"**) array con objetos de todos los elementos con **class="cl1"**
    - querySelectorAll(**"h1.cl1"**) array con objetos de todos los elementos **<h1 class="cl1">**
      - https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll

Ejemplo de querySelectorAll (..)

- **querySelectorAll("<patrón>")** devuelve un array de objetos DOM
  - El array incluye todos los objetos DOM de elementos HTML que cumplen el patrón
    - El array estará vacío si ningún elemento HTML casa con el patrón.
- En el ejemplo los patrones casan de la siguiente forma
  - **"#date"** casa solo con el bloque div con **id="date"** y el array devuelto tiene solo ese objeto DOM.
  - **".msg"** casa solo con el bloque div con **class="msg"** y el array devuelto tiene solo ese objeto DOM.
    - Si hubiese devuelto mas elementos habría que procesarlos co un bucle.



Paso 1: Bloques ** ** vacíos no visualizan nada.



Paso 2: Insertado por el script al ejecutarse en un segundo paso.

```
<!DOCTYPE html><html><head>
<title>Ejemplo de función</title><meta charset="UTF-8">
</head>
<body>
<h1>La fecha y la hora son:</h1>

<div id="date"></div>
<div class="msg"></div>

<script type="text/javascript">

  let date = document.querySelectorAll("#date");
  date[0].innerHTML = new Date();

  let msg = document.querySelectorAll(".msg");
  msg[0].innerHTML = "Demo de: querySelectorAll()";

</script>
</body>
</html>
```

** ** define un bloque vacío donde añadir la fecha por programa.

** ** define un bloque vacío donde añadir un mensaje por progr.

querySelectorAll(..) devuelve un **array** con **un único objeto DOM**. Hay que indexarlo con **[0]**.

Booleanos, sentencias if-else, switch-case y bucles

Juan Quemada, DIT - UPM

Tipo boolean

- El **tipo boolean** tiene 2 valores
  - **true**: verdadero
  - **false**: falso
- Booleanos: permiten crear expresiones lógicas
  - Utilizando los operadores lógicos

((x % 2) === 0) indica si el resto de dividir x por 2 es 0, es decir si **x es par**.

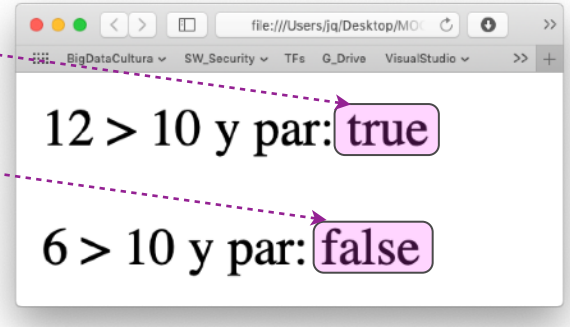
```
<!DOCTYPE html><html><head></head>
<body>
<script type="text/javascript">

function mayorQue_10_y_par (x) {

    return ((x % 2) === 0) && (x > 10);
}
document.write("12 > 10 y par: "
                + mayorQue_10_y_par(12));
document.write("<br><br>");
document.write("6 > 10 y par: "
                + mayorQue_10_y_par(6));
</script>
</body>
</html>
```

Los **operadores lógicos** booleanos son:

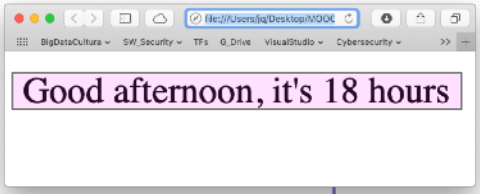
negación:	**!**	!true => false !false => true
operador y:	**&&**	true && true => true true && false => false false && true => false false && false => false
operador o:	**||**	true || true => true true || false => true false || true => true false || false => false

Las comparaciones resultan en un booleano:

- **comparación** de **orden**
  - menor: **<** menor_o_igual: **<=**
  - mayor: **>** mayor_o_igual: **>=**
- **comparación** de **identidad**
  - identidad: **===** no_identidad: **!==**



Ejecución condicional: if-else



Resultado de ejecutar los ejemplos.

- Sintaxis: **if** (<condición>) <sentencia-if> **else** <sentencia-else>
  - Doc: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else
- Si **<condición>** es true, se ejecuta **<sentencia-if>**, sino **<sentencia-else>**
  - El bloque else es opcional
- Las sentencias (if y else) pueden ser una sentencia o varias delimitadas con {..}

Sentencia **if...else**: comienza con la palabra reservada **if.**

```
// new Date().getHours(): hour of the day (0-23h)

let hour = new Date().getHours();

if (hour < 12) {
    document.write(`\n Good morning, it's ${hour} hours`);
} else {
    document.write(`\n Good afternoon, it's ${hour} hours`);
}
```

new Date().getHours() devuelve la hora del día (0-23).

El **primer bloque** de sentencias va después de la condición, delimitado entre llaves: **{}**

El **segundo bloque** de sentencias va precedido por la palabra reservada **else** y delimitado entre llaves: **{}**

La **condición (hora <12)** va entre paréntesis a continuación y según se evalúe a **true** o **false**, decide si se ejecuta el primer o el segundo bloque.

Ejemplo de programa que muestra el mensaje **"Good morning, ..."** o **"Good afternoon, ..."** dependiendo de la **hora**.

Ejemplo **equivalente** al primero pero con sentencia if **sin bloque else**.

```
// new Date().getHours(): hour of the day (0-23h)

let hour = new Date().getHours();
let greeting = "\n Good afternoon";

if (hour < 12) {
    greeting = "\n Good morning";
}

document.write(`${greeting}, its ${hour} hours`);
```

Sentencia switch-case

- La sentencia **switch-case** está controlada por el <valor> de una variable
  - Comienza la ejecución en **"case <valor>:"** de la variable
    - Documentación: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch
- La sentencia **break** finaliza la ejecución (de la sentencia switch-case)
- Una sentencia **switch-case** puede sustituirse por un **if-else** encadenado

La sentencia **switch-case** evalúa la expresión asociada (result) y pasa a ejecutar la sentencia justo después del **"case"** que coincide con el valor resultante de evaluar la expresión.

La sentencia **break** finaliza la ejecución de la sentencia **switch-case**. Es necesario ponerla para finalizar cada **case** (o conjunto de **case**), porque sino continuará ejecutando las sentencias del siguiente **case**.

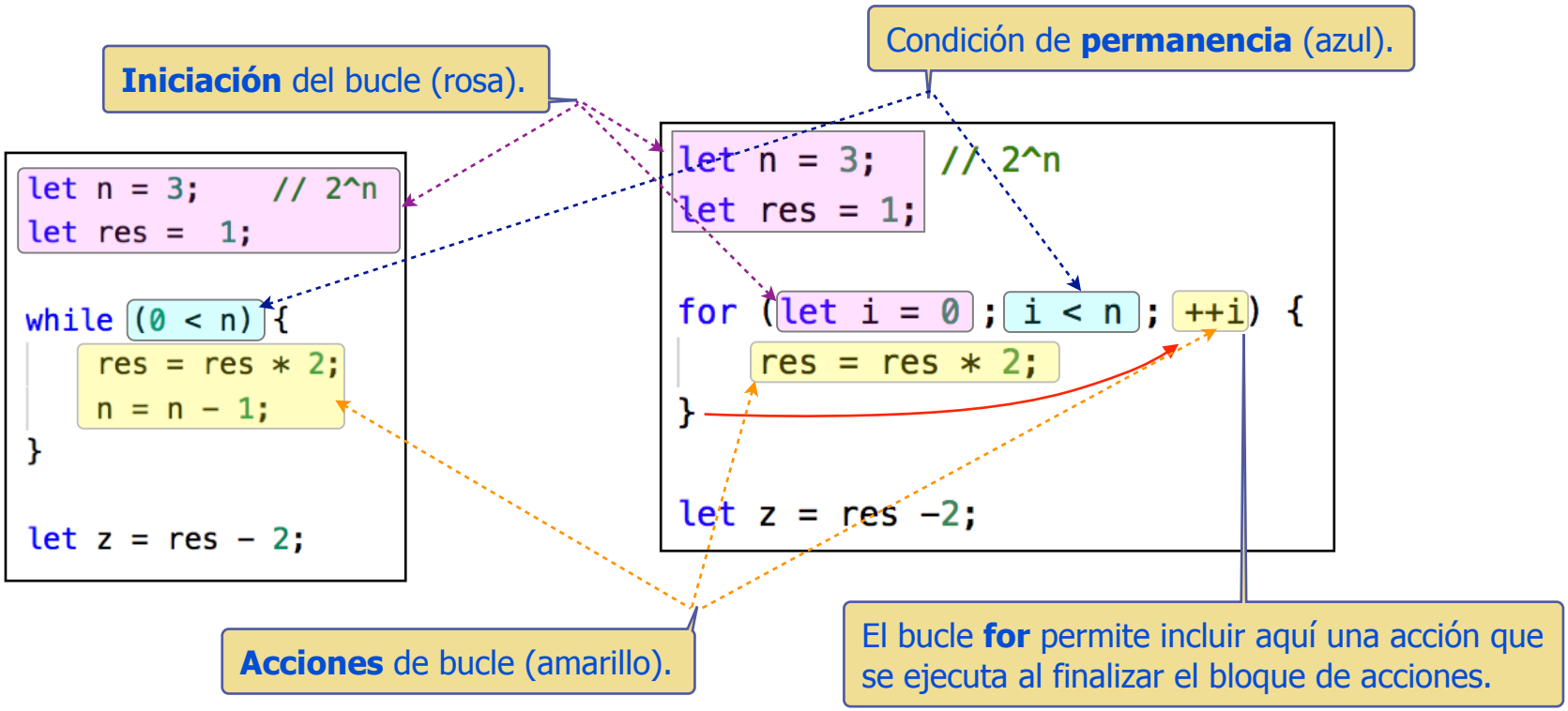
Cuando el valor resultante de evaluar la expresión no coincide con ninguno asociado a un **case**, se pasa a ejecutar las sentencias asociadas a **default:** (es opcional incluirlo).

```
// Math.round(Math.random()*10):
// entero aleatorio entre 0 y 9

let result = Math.round(Math.random()*10);

switch (result) {
   case 9:
      document.write("\n You win the first prize!");
      break;
   case 8:
   case 7:
      document.write("\n You win the second prize!");
      break;
   default:
      document.write("\n Sorry, no prize!");
}
```

Math.round(Math.random()*10) genera un número aleatorio de 0 a 9

Se pueden agrupar varios **case** si comparten el mismo código. **case** es solo un punto de comienzo de ejecución.

Bucles while y for



Estos dos bucles son equivalentes. Calculan 2n (2*2*…*2, una multiplicación en cada iteración). Tienen 3 partes:

1) **Iniciación del bucle**: define e inicia las variables **n** y **res** utilizadas para el cálculo y para control del bucle.

2) **Condición de permanencia**: indica si el bucle se debe seguir ejecutando o finalizar.

3) **Acciones del bucle**: instrucciones ejecutadas en cada vuelta al bucle. Se delimitan con llaves **{}**, salvo en bloques de una sentencia, en que las llaves pueden omitirse.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

STATEMENT	SINTAX	DESCRIPCIÓN DE LA SENTENCIA JAVASCRIPT
block	{ statements } | statement;	Agrupar un bloque de sentencias (como 1 sentencia)
break	break [label];	Salir del bucle o switch o sentencia etiquetada
case	case expression: [statements]	Etiquetar sentencia dentro de sentencia switch
const	const name [= expr] [,...];	Declarar e initializar una o mas constantes (ES6)
continue	continue [label];	Salto a sig. iteración de bucle actual/etiquetado
debugger	debugger:	Punto de parada (breakpoint) del depurador
default	default:	Etiquetar sentencia default de sentencia switch
do-while	do block while(expression);	Alternativa al bucle while con condición al final
empty	;	Sentencia vacía, no hace nada
expression	expression;	Evaluar expresión (incluyendo asignación a variables)
for	for(init; test; incr) block	Bucle: init: iniciación; test: condición; incr: acciones final bucle
for-in	for (var in obj) block	Bucle for-in: Itera en los nombres de propiedades de obj
for-of	for (var in obj) block	Bucle for-of: Itera en los elementos del objeto iterable obj (ES6)
function	function name([param[,-]]) block	Declarar una función de nombre "name"
func_arrow	param | ([param[,-]]) => block | expr	Definir un literal de función con visibilidad léxica (ES6)
if-else	if (expr) block1 [else block2]	Ejecutar block1 o block2 en función de expr
label	label: statement	Etiquetar sentencia con nombre label
let	let name [= expr] [,-];	Declarar e inicializar una o mas variables (ES6)
return	return [expression];	Devolver un valor desde una función
switch	switch (expression) { case }	Multi-opción con etiquetas "case" o "default"
throw	throw expression;	Lanzar una excepción o error
try	try block [catch block] [finally block]	Define un manejador de excepciones con el bloque catch que procesa las exceptions lanzadas (throw) en el bloque try, el bloque finally, si existe, se ejecuta siempre
strict	"use strict";	Activar restricciones strict a scripts o funciones
var	var name [= expr] [,...];	Declarar e initializar una o mas variables
while	while (expression) block	Bucle básico con condición al principio
with	with (object) block	Extender cadena de ámbito (no recomendado)

Arrays

Juan Quemada, DIT - UPM

Arrays

```
// Otros ejemplos de array
let nombres = ["Olga", "Eva", "Juan"];
let mezcla = [1, "uno", "1", true];
```

- **Array**
  - Colección indexada de elementos
    - Pueden ser **heterogéneos**
- **literal de array: [e1, e2, ...]**
  - Define un array con los valores de una lista
- **length**: propiedad con el tamaño del array
- **array[i]** da acceso al **elemento i** del array
  - El 1er elemento se accede con índice **0**
  - El último elemento se accede con **array.length-1**
  - Indexar elementos fuera de rango devuelve undefined
  - **[i]** puede aplicarse varias veces para acceder a arrays de arrays
- Doc: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array

```
// Ejemplo de array
let a = [7, 4, [1, 2]];
a.length    => 3
a[0]        => 7
a[1]        => 4
a[2]        => [1, 2]
a[2][0]     => 1
a[2][1]     => 2
a[8]        => undefined
```

Sumatorio de números

- Ilustra el uso de bucles para procesar arrays
  - Con la función sumatorio de n números: **Σ x**
- El sumatorio de los números 1, 2, 3, 4 es:
  - **Σ (1, 2, 3, 4) = 1 + 2 + 3 + 4 = 10**
- Formato **CSV** (Coma Separated Values)
  - String con **valores separados por comas**
    - por ejemplo: **"1, 2, 3, 4"** o **"1,2,3,4"**
  - Muy utilizado en BBDD, hojas de calculo, …
- Los números se introducen en el cajetín
  - En formato CSV (Coma Separated Values)

```
<!DOCTYPE html><html>
<head><title>Sumatorio</title><meta charset="utf-8">

<script type="text/javascript">

function vaciar () {
  document.getElementById("n1").value = "";
}

function sumatorio() {
  let num = document.getElementById("n1");

  let list = num.value.split(",");

  let i = 0, acc = 0;
  while (i < list.length) acc += +list[i++];

  num.value = acc;
}
</script>
</head><body>

  Número:
  <input type="text" id="n1" onclick="vaciar()">
  <br>
  <button onclick="sumatorio()">Σ x</button>
  <body>
</html>
```

En este bucle se **suman** los números del **array** obtenido con **split()**.

Los **bloques** de una sentencia pueden **omitir** las llaves **{…}**

+ convierte **string** a **number** (suma aritmética).

El método **split(",")** transforma un string en un array con los substrings separados **","** (coma).
Por ejemplo, **"1, 2, 3, 4". split(",") => ["1", " 2", " 3", " 4"]**
Mas info: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/split

Clase Array: ordenar, invertir, concatenar o buscar

¡Los métodos de esta transparencia **no modifican** el array sobre el que se aplican!

- **sort()**
  - devuelve el array ordenado

```
[1, 5, 3].sort()     // => [1, 3, 5]
```

- **reverse()**
  - devuelve el array invertido

```
[1, 5, 3].reverse()  // => [3, 5, 1]
```

- **concat(e1, .., en)**
  - devuelve un nuevo array con **e1, .., en** añadidos al final
    - si ex es un array, añade sus elementos esparcidos

```
[1, 5, 3].concat(9, 3)    // => [1, 5, 3, 9, 3]
[1, 5, 3].concat([9, 3])  // => [1, 5, 3, 9, 3]
```

```
[1, 5, 3].concat([2, 4]).sort().reverse()    // => [5, 4, 3, 2, 1]
```

Los **métodos** que retornan un **array** (3 anteriores) pueden **encadenarse** tal y como se ilustra aquí.

- **join(<separador>)**
  - concatena elementos en un string
    - introduce <separador> entre elementos

```
[1, 5, 3, 7].join(';')    // => '1;5;3;7'
[1, 5, 3, 7].join('')     // => '1537'
```

- **indexOf(elem, offset)**
  - devuelve índice de primer **elem** o -1
    - **offset:** comienza búsqueda (por defecto 0)

```
[1, 5, 3, 5, 7].indexOf(5)      // => 1
[1, 5, 3, 5, 7].indexOf(5, 2)   // => 3
```

Más métodos: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Clase Array: extraer, modificar o añadir elementos

- **slice(i,j):** devuelve la rodaja entre **i** y **j**
  - Indice negativo (j) es relativo al final
    - índice "-1" es igual a a.length-2
      - No modifica el array original

```
[1, 5, 3, 7].slice(1, 3)   =>  [5, 3]
[1, 5, 3, 7].slice(1, -1)  =>  [5, 3]
```

- **splice(i, n, e1, e2, .., en)**
  - sustituye **n** elementos en array, desde **i**
    - por e1, e2, ..,en
      - Modifica el array original
  - Devuelve rodaja eliminada

```
let a = [1, 3, 7];
a.splice(1, 0, 5)   =>  []   // añade 1 elem
a                   =>  [1, 5, 3, 7]
a.splice(2, 1)      =>  [5]  // quita 1 elem
a                   =>  [1,5, 7]
a.splice(1, 1, 4, 2)  =>  [5]
a                     =>  [1, 4, 2, 7]
```

- **push(e1, .., en)**
  - añade **e1, .., en** al final del array
    - devuelve el tamaño del array (a.length)
      - Modifica el array original
- **pop()**
  - elimina último elemento y lo devuelve
    - Modifica el array original

```
let b = [1, 5, 3];
b.push(6, 7)    =>  5
b               =>  [1, 5, 3, 6, 7]
b.pop()         =>  7
b               =>  [1, 5, 3, 6]
```

Más métodos: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Final del tema