Introducción a JavaScript: Expresiones

JavaScript

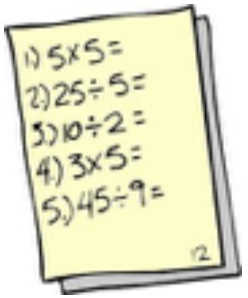
- Lenguaje de programación diseñado en 1995 por Brendan Eich
  - Para animar páginas Web en el Navegador Netscape
    - Hoy se ha convertido en **el lenguaje del Web** y de **Internet**
      - **Guía:** https://developer.mozilla.org/es/docs/Web/JavaScript/Guide

- JavaScript: sigue la norma ECMA-262 (European Computer Manufacturers Assoc.)
  - "Seguida" por **todos los navegadores**: Chrome, Explorer, Firefox, Safari, ..
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript

- ECMA-262 ha evolucionado mucho, siempre con "compatibilidad hacia atrás"
  - **ES3** - ECMAScript 3 (Dec. 1999): navegadores antiguos
    - Primera versión **ampliamente aceptada**, compromiso entre **Netscape** y **Microsoft**
  - **ES5** - ECMAScript 5.1 (Jun. 2011): navegadores actuales
    - Desarrollado junto con **plataforma HTML5** (HTML, CSS y JavaScript) de WHATWG (https://whatwg.org)
  - **ES6** - ECMAScript 6 (Jun. 2015): en vías de introducción en navegadores
    - Introduce **muchas mejoras**, ver http://es6-features.org/

- Este **curso** se centra en la partes buenas **(Good parts)** de **ES3** y **ES5**
  - Siguiendo el libro: **JavaScript: The Good Parts**, Douglas Crockford, 2008

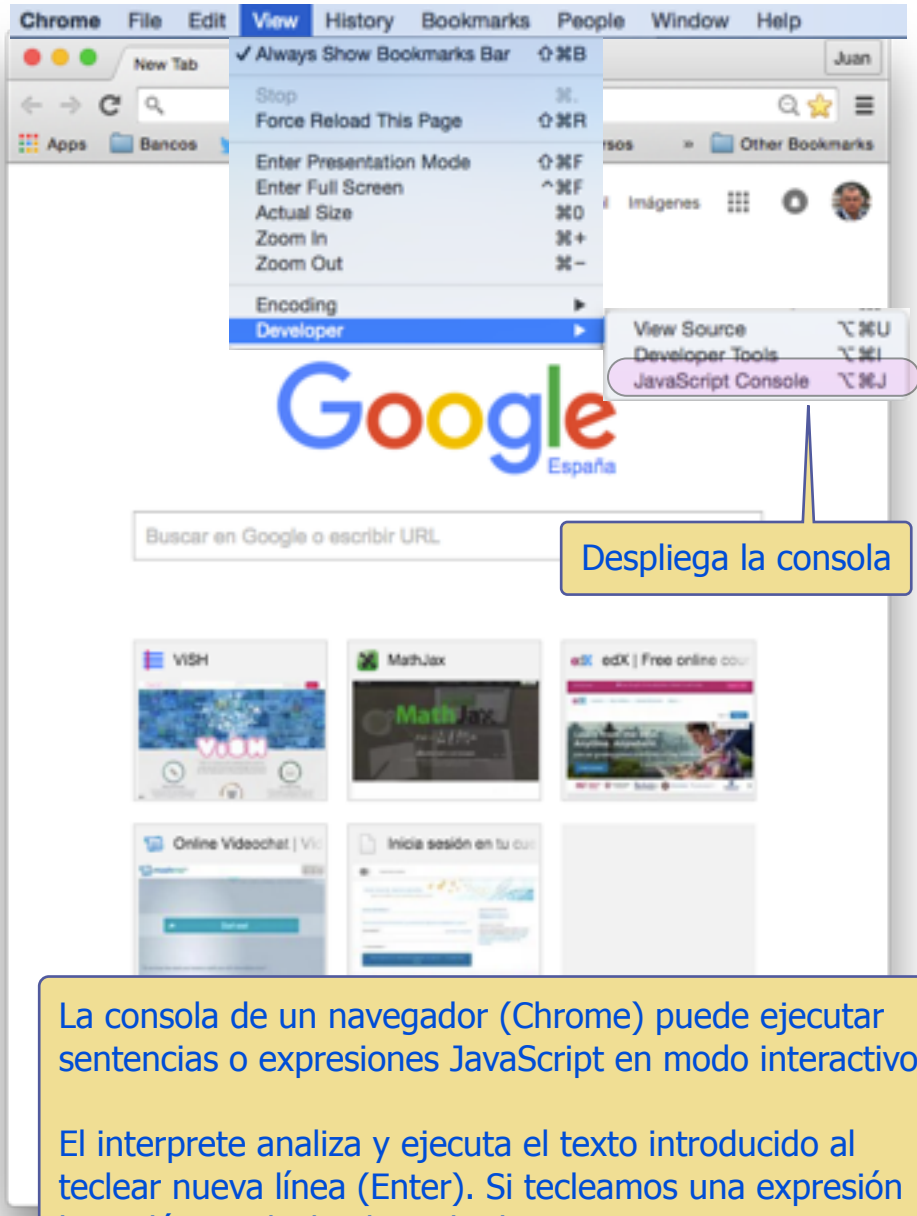
Expresiones numéricas y operadores



- JavaScript incluye **operadores**
  - Los **operadores** permiten formar **expresiones**
    - componiendo **valores** con dichos operadores
  - Una expresión representa un valor, que es el resultado de evaluarla
- Ejemplo: operadores aritméticos +, -, *, / formando expresiones numéricas
  - Las expresiones se evalúan (=>) a los valores resultantes
    - Expresiones mal construidas dan error o llevan el interprete a un estado inconsistente

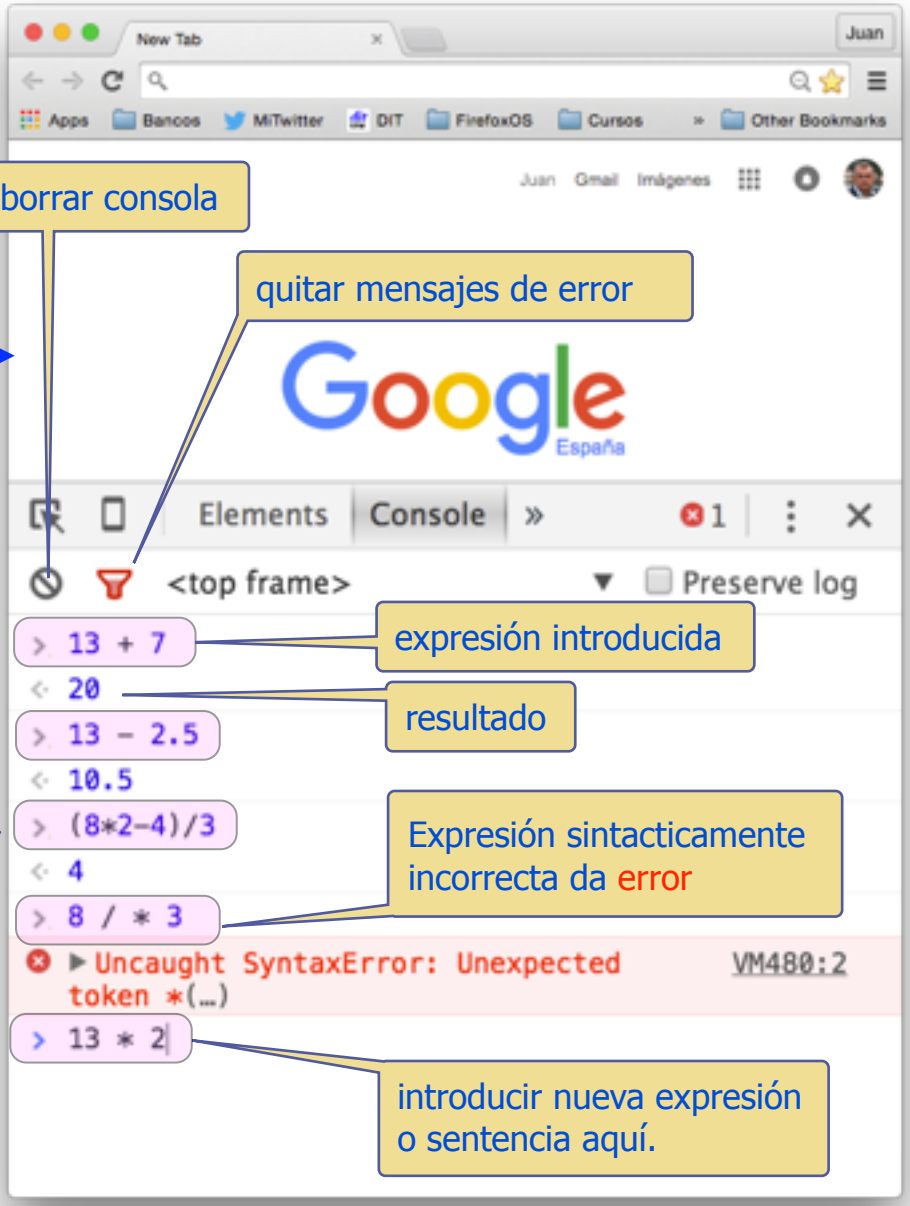
```
13 + 7          =>   20          //  Suma de números
13 -  1.5       =>   11.5        //  Resta de números
     //  OJO! la coma española es un punto en la sintaxis inglesa

(8*2 - 4)/3     =>    4          // Expresión con paréntesis

8 / * 3         =>    ??         // Expresión incorrecta
8  3            =>    ??         // Expresión incorrecta
```

1) Abrir consola JavaScript de chrome



Despliega la consola

2) Ejecutar sentencias en consola de chrome



La consola de un navegador (Chrome) puede ejecutar sentencias o expresiones JavaScript en modo interactivo.

El interprete analiza y ejecuta el texto introducido al teclear nueva línea (Enter). Si tecleamos una expresión la evalúa y calcula el resultado.

Texto: strings

- El texto escrito se representa en JavaScript con strings
  - Un string delimita el texto con **comillas** o **apóstrofes**, por ej.
    - Frases: **"hola, que tal"** o **'hola, que tal'**
    - String vacío: **""** o **''**

- Ejemplo de **"texto 'entrecomillado' "**
  - comillas y apóstrofes se pueden anidar
    - **'entrecomillado'** forma parte del texto

- 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**

La **consola** analiza y ejecuta el texto introducido al teclear nueva línea (Enter).

Si tecleamos una expresión de strings, esta se evalúa, presentando el string resultante.

expresión introducida

resultado

Los strings "**Eva**" y '**Eva**' son literales de string equivalentes, que representan el mismo string o texto.

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

El operador + aplicado a strings los concatena o une, generando un nuevo string con la unión de los dos.

Es asociativo y permite concatenar más de 2 strings.

"Eva".length devuelve la longitud del string (número de caracteres). Length es una "propiedad" del string.

Sobrecarga de operadores

- Los operadores sobrecargados tienen varias semánticas
  - dependiendo del contexto en que se usan en una expresión
- Por ejemplo, el operador **+** tiene 3 semánticas diferentes
  - **Suma de enteros** (operador binario)
  - **Signo de un número** (operador unitario)
  - **Concatenación de strings** (operador binario)

```
13 + 7              =>   20              //  Suma de números
+13                 =>   13              //  Signo de un número
"Hola " + "Pepe"    =>   "Hola Pepe"     //  Concatenación de strings
```

Conversión de tipos en expresiones

- JavaScript realiza conversión automática de tipos
  - cuando hay ambigüedad en una expresión
    - utiliza las reglas de prioridad para resolverla
- La expresión **"13" + 7** es ambigua
  - porque combina un **string** con un **number**
    - Con ambigüedad JavaScript da prioridad al **operador +** de strings, convirtiendo **7** a string
- La expresión **+"13"** también necesita conversión automática de tipos
  - El **operador +** solo esta 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

. []	Acceso a propiedad o invocar método; índice a array
new	Crear objeto con constructor de clase
()	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
* / %	Números. Multiplicación; división; modulo (o resto)
+ + -	Concatenación de strings; Números. Suma; 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)
?:	Asignación condicional
=	Asignación de valor
OP=	Asig. con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=
,	Evaluación múltiple

8*2 - 4 => 12

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

8*(2 - 4) => -16

Operadores ES3, ES5

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

. []	**Acceso a propiedad o invocar método; índice a array**
new	**Crear objeto con constructor de clase**
()	**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**
*** / %**	**Números. Multiplicación; división; modulo (o resto)**
+ + -	**Concatenación de strings; Números. Suma; 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)**
?:	**Asignación condicional**
=	**Asignación de valor**
OP=	**Asig. con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=**
,	**Evaluación múltiple**

+"3" + 7 => 10

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

Operadores ES3, ES5

JavaScript:
Programas, sentencias y variables

Programa, sentencias y comentarios

- Un programa es una secuencia de sentencias
  - que se ejecutan en el orden en que han sido definidas (con excepciones)
- Las sentencias realizan tareas al ejecutarse en un ordenador
  - Cada sentencia debe acabarse con punto y coma: ";"
- Los comentarios solo tienen valor informativo
  - para ayudar a entender como funciona el programa

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

Sentencia 1: define la variable x con valor 7.

Sentencia 2: visualiza x en el navegador

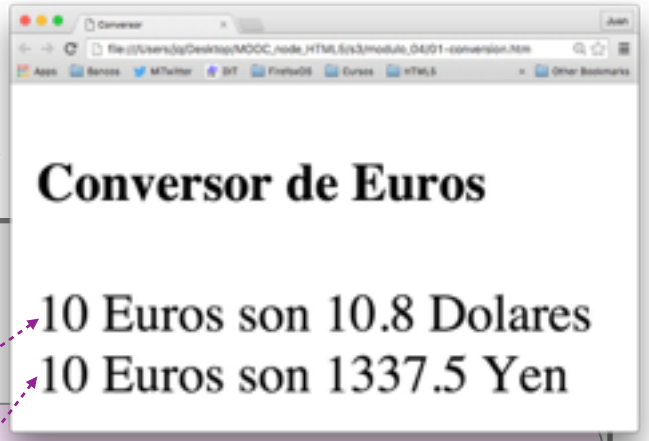
```
/* Ejemplo  de
    programa JavaScript */

var x = 7;  // Definición de variable
            // visualizar x en el navegador
document.write(x);
```

Comentario de una línea: empieza con **//** y acaba al final de la línea

Script JavaScript con variables

- **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>
<body>
  <h4>Conversor de Euros</h4>

  <script type="text/javascript">

    var 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

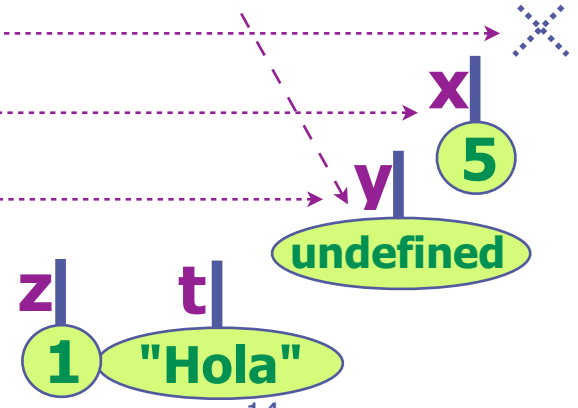
Visualizan en el navegador el resultado de evaluar las expresiones

Separación de línea HTML

Definición de variables y estado del programa

- Las **variables** se crean con la sentencia de definición de variables
  - La sentencia comienza con la palabra reservada **var**
    - A continuación vienen una o más definiciones de variables separadas por comas
  - Cada definición de variable comienza con el nombre de la variable
    - A la variable se le puede asignar un valor usando el operador de asignación: **=**
- **undefined**: valor (y tipo) especial de JavaScript que significa **indefinido**
  - Las variables sin ningún valor asignado contienen el valor **undefined**
- **Estado de un programa**:
  - Conjunto de **todas las variables** creadas por el programa
    - junto con sus correspondientes **valores**

```
var x = 5;    // Define la variable x con el valor 5
var y;        // Define la variable y sin asignar ningún valor
var z = 1, t = "Hola";   // Define las variables z y t con
                         // con los valores 1 y "Hola" respectivamente
```



Sintaxis de los nombres de variables

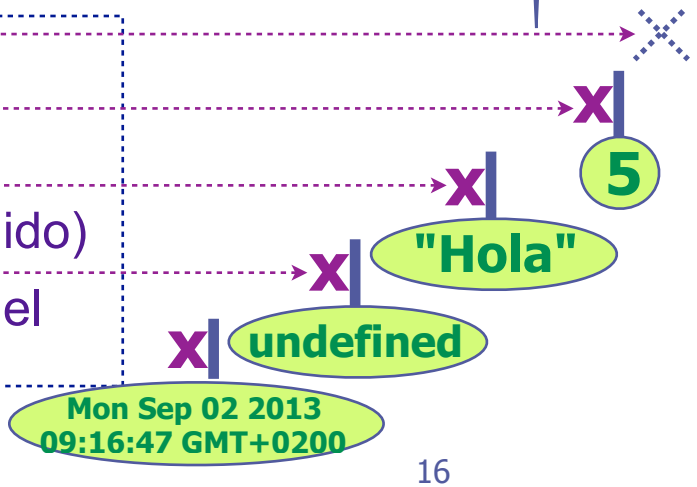
- El **nombre** (o identificador) de una variable debe comenzar por:
  - **letra**, **_** o **$**
    - El nombre pueden contener además **números**
  - Nombres **bien construidos:** **x**, **ya_vás**, **$A1**, **$**, **_43dias**
  - Nombres **mal construidos:** **1A**, **123**, **%3**, **v=7**, **a?b**, **..**
    - 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, function, return, if, else, while, for, in, new, typeof, ….
- Las variables son sensibles a **mayúsculas**
  - **mi_var** y **Mi_var** son variables distintas

Asignación de variables y estado del programa

- Una variable es un **contenedor de valores**, cuyo contenido puede variar
  - La sentencia de asignación de variables asigna un nuevo valor con el operador: **=**
- Las variables de JavaScript son **no tipadas**
  - Esto significa que se puede asignar cualquier tipo de valor
    - Una variable puede contener un número, un string, undefined, ..
- Se denomina **punto de ejecución del programa**
  - al estado en que queda el programa **después de ejecutar una instrucción**
    - el **estado de un programa** varia en función del **punto de ejecución**

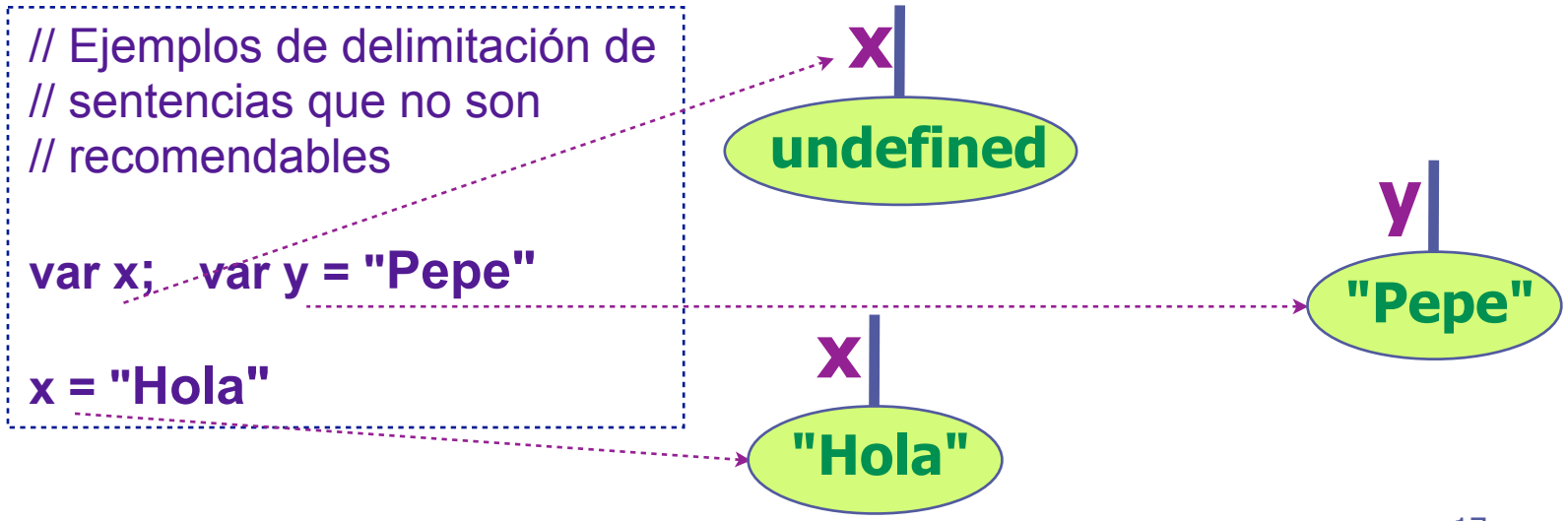
Evolución del **estado** en función del **punto de ejecución** del programa

```
var x = 5;          // Crea la variable x y le asigna 5
x = "Hola";         // Asigna a x el string (texto) "hola"
x = undefined;      // Asigna a x undefined (valor indefinido)
x = new Date();     // objeto Date: fecha/hora del reloj del
```



Recomendaciones sobre sintaxis

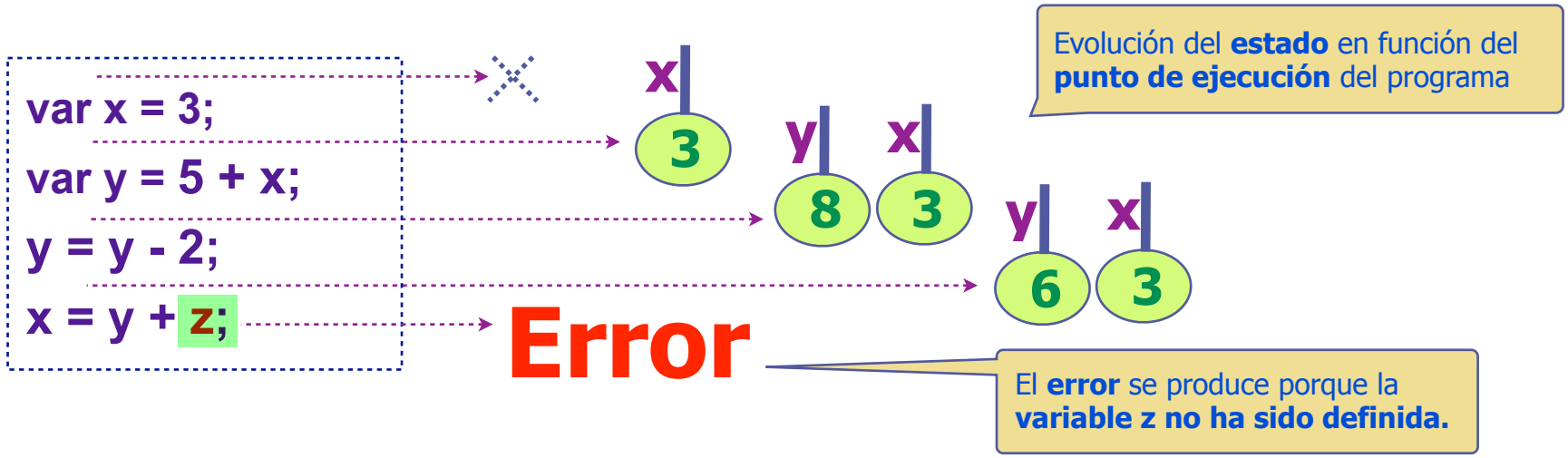
- Se recomienda delimitar las sentencias siempre con: **;**
  - La sintaxis de JS permite introducir caracteres adicionales (blanco, nueva linea, ..) para facilitar la legibilidad del programa
- JavaScript permite omitir **;** si la sentencia acaba con **nueva linea**
  - Esto puede dar problemas y **no debe hacerse nunca**
- Cada sentencia debe **ocupar una línea** por legibilidad, salvo algunas excepciones
  - Las sentencias con **bloques de código**: if/else, while, for, definición de funciones, …..
  - sentencias que contienen **expresiones muy largas**



JavaScript:
Expresiones con variables

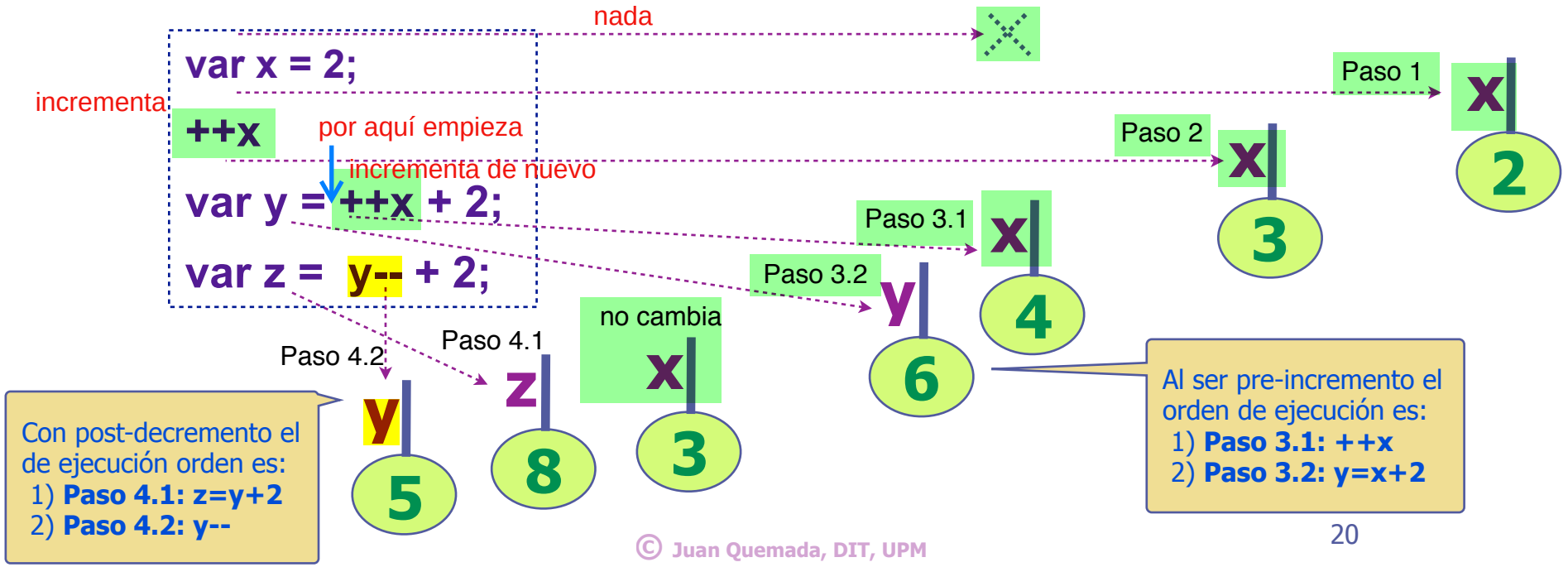
Expresiones con variables

- Una **variable** representa el **valor** que contiene
  - Puede ser usada en expresiones como cualquier otro valor
- Una variable puede utilizarse en la expresión que se asigna a ella misma
  - La parte derecha usa el valor anterior a la ejecución de la sentencia
    - **y = y - 2** asigna a **y** el valor **6 (8-2)**, porque **y** tiene el valor **8** antes de ejecutarse
- 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

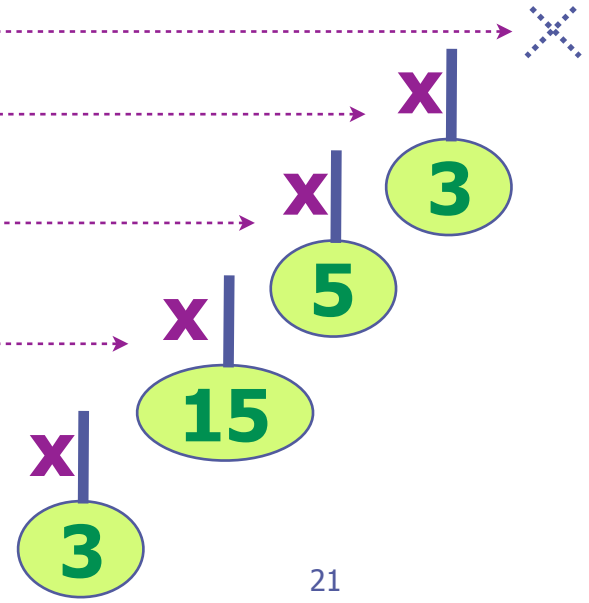
- JavaScript posee los operadores **++** y **--** de **auto-incremento** o **decremento**
  - **++** suma 1 y **--** resta 1 a la variable a la que se aplica
    - **++** y **--** se pueden aplicar por la derecha o por la izquierda a las variables de una expresión
      - Si **++/--** se aplica por la **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 porque tiene efectos laterales y lleva a programas crípticos.
- Documentación adicional
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators



Operadores de asignación

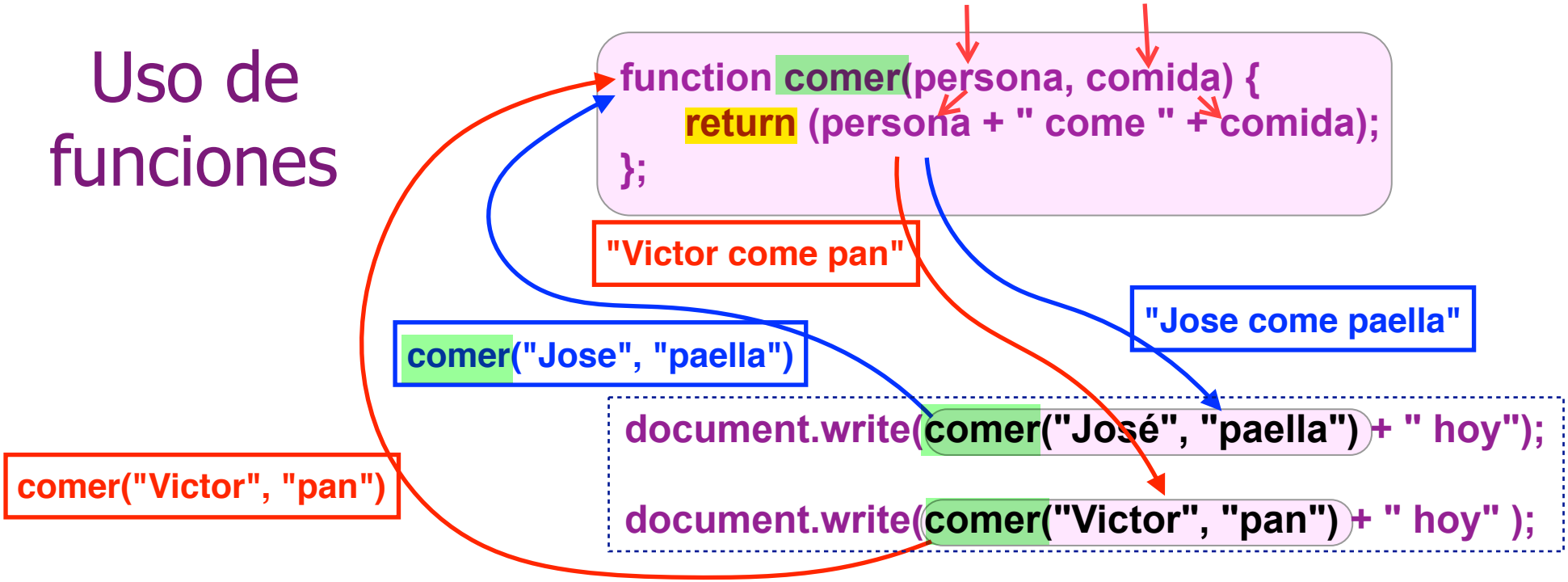
- Es muy común modificar el valor de una variable
  - sumando, restando, ... algún valor
    - Por ejemplo, x = x +7; y = y - 3; z = z * 8; …….
- JavaScript tiene operadores de asignación especiales para estos casos
  - +=, -=, *=, /=, %=, ……(y para otros operadores del lenguaje)
    - x += 7; será lo mismo que x = x + 7;
      - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators

```
var x = 3; // define x inicializada a 3

x += 2;    // suma 2 a x

x *= 3;    // multiplica x por 3

x %= 4;    // % operación módulo:
           // divide x por 4 y devuelve el resto
```



JavaScript:
Funciones

Uso de funciones



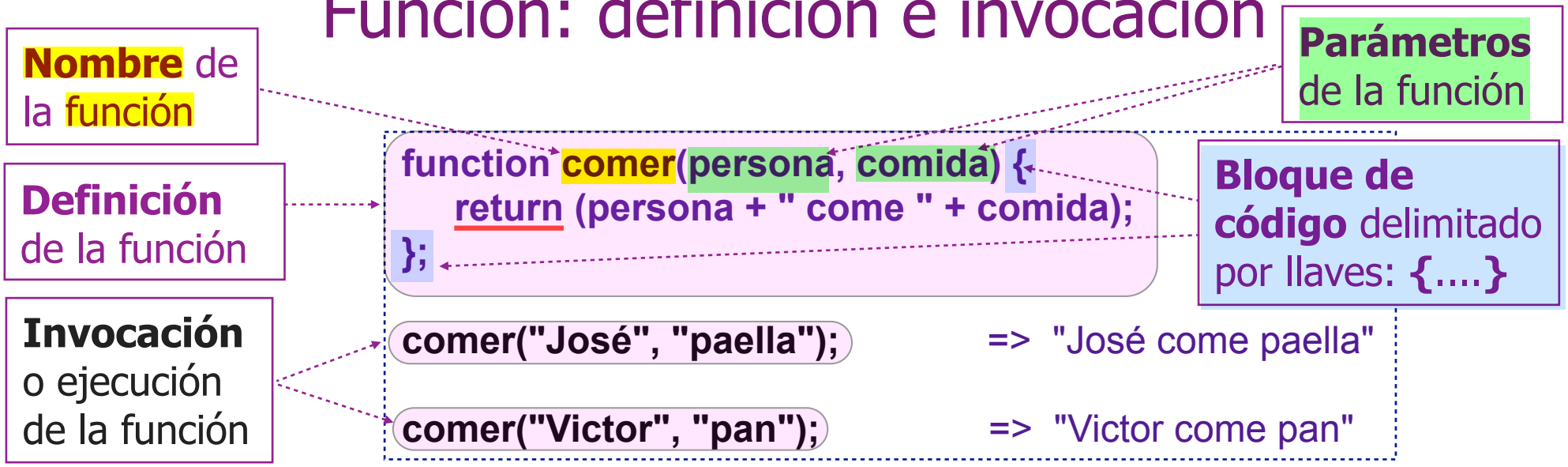
- **Función**: **bloque** de código (con **parámetros**) asociado a un **nombre**
  - La función se **invoca** (o ejecuta) por el nombre y devuelve un valor como **resultado**
    - En la invocación se deben asignar **valores concretos** a los **parámetros**
- Las **funciones** permiten crear **operaciones de alto nivel**
  - Se denominan también **abstracciones** o **encapsulaciones** de código
- La **función** representa el **valor resultante** de su ejecución (evaluación)
  - El resultado de evaluar una función depende del valor de los parámetros
    - Puede utilizarse en **expresiones** como cualquier otro valor

Función: definición e invocación

```
function comer(persona, comida) {
    return (persona + " come " + comida);
};

comer("José", "paella");          => "José come paella"

comer("Victor", "pan");           => "Victor come pan"
```

- Una **función** se define con la palabra reservada **function** seguida del nombre
  - A continuación se definen los **parámetros** entre paréntesis
    - Los **parámetros** son variables que se asignan en la invocación
      - Puede asignarse nuevos valores en el bloque igual que a las variables
  - A continuación se define el bloque de código delimitado entre llaves {}
    - El bloque contiene instrucciones
- La sentencia **return <expresion>** finaliza la ejecución
  - Devolviendo el resultado de evaluar **<expresion>** como valor de retorno
    - Si la función llega a final del bloque sin haber ejecutado un **return**, acaba y devuelve **undefined**
- Documentación:
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Funciones

Función: definición e invocación



- Una **función** se define con la palabra reservada **function** seguida del nombre
  - A continuación se definen los **parámetros** entre paréntesis
    - Los **parámetros** son variables que se asignan en la invocación
      - Puede asignarse nuevos valores en el bloque igual que a las variables
  - A continuación se define el bloque de código delimitado entre llaves {}
    - El bloque contiene instrucciones
- La sentencia **return <expresion>** finaliza la ejecución
  - Devolviendo el resultado de evaluar **<expresion>** como valor de retorno
    - Si la función llega a final del bloque sin haber ejecutado un **return**, acaba y devuelve **undefined**
- Documentación:
  - https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Funciones

Parámetros de una función

- La función **se puede invocar** con un **número variable de parámetros**
  - Un parámetro definido, pero **no pasado** en la invocación, toma el valor **undefined**
    - Un parámetro pasado en la invocación, pero **no utilizado**, no tiene utilidad

```
function comer(persona, comida) {
    return (persona + " come " + comida);
};

comer('José', 'paella');                    => 'José come paella'

comer('José', 'paella', 'carne');           => 'José come paella'
comer('José');                              => 'José come undefined'
```

El array de argumentos de funciones

- Los parámetros de la función están accesibles también a través del
  - array de argumentos: **arguments[....]**
    - Cada parámetro es un elemento del array

- En: **comer('José', 'paella')**
  - arguments[0] => 'José'
  - arguments[1] => 'paella'

```
function comer() {
    return (arguments[0] + " come " + arguments[1]);
};

comer('José', 'paella');          => 'José come paella'

comer('José', 'paella', 'carne'); => 'José come paella'
comer('José');                    => 'José come undefined'
```

Funciones como objetos

- Las **funciones** son objetos de pleno derecho
  - pueden **asignarse a variables**, a **propiedades**, pasarse como **parámetros**, ….
- **Literal de función:** function (..){..}
  - **Función sin nombre**, que suele asignarse a una variable, que es la que le da nombre
    - Se puede invocar a través del nombre de la variable
- el **operador (...)** invoca una función ejecutando su código
  - Este operador solo es aplicable a funciones (objetos de la clase Function), sino da error
    - El operador puede incluir parámetros separados por coma, accesibles en el código de la función

```
var comer = function(persona, comida) {
    return (persona + " come " + comida);
};

comer('José','paella');          =>  'José come paella'
```

JavaScript:
Objetos, propiedades, métodos y DOM

Elementos HTML y objetos DOM

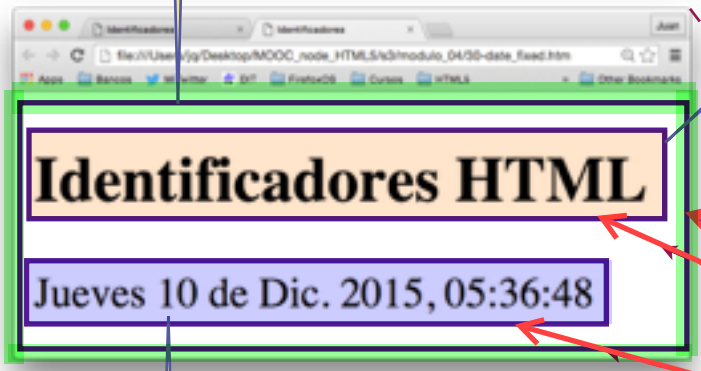
- Los **elementos HTML** se visualizan en el navegador en **cajas asociadas**
  - Los **objetos DOM** de JavaScript permiten **inspeccionar** y **modificar** los **elementos HTML**
    - El **atributo id="…"** es distinto en cada elemento y puede utilizarse para **identificar los elementos HTML** desde JavaScript

objetoDOM . metodoDOM parametros

- **document.getElementById("fecha")**
  - devuelve el **objeto DOM** del elemento HTML con atributo **id="fecha"**
    - El **objeto DOM** obtenido así **permite modificar** el elemento HTML visualizado e **interaccionar** con el usuario

Objeto DOM **<body id="cuerpo">.... **
accesible desde JavaScript con:
document.getElementById("cuerpo")

Objeto DOM **<h2 id="titulo">.... **
accesible desde JavaScript con:
document.getElementById("titulo")



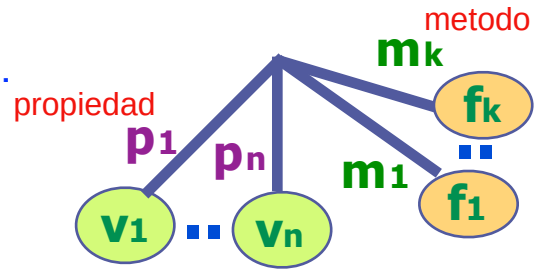

```
<!DOCTYPE html>
<html>
<head>
  <title>Identificadores</title>
  <meta charset="UTF-8">
</head>
<body id="cuerpo">
  <h2 id="titulo">Identificadores HTML</h2>
  <div id="fecha">Jueves 10 de Dic. 2015, 05:36:48</div>
</body>
</html>
```

Objeto DOM ** .... **
accesible desde JavaScript con:
document.getElementById("fecha")

Objetos JavaScript: métodos y propiedades

- Los **objetos** son colecciones de **variables (propiedades)** y **funciones (métodos)**
  - agrupadas en un **elemento estructurado** que llamamos **objecto**
    - Doc: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Trabajando_con_objectos

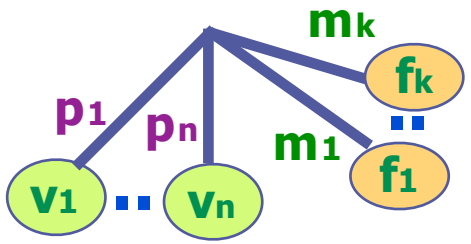
- **Nombres** de propiedades y métodos
  - tienen la misma sintaxis que las variables: **a**, **_method**, **$1**, …



- **Propiedad:** **variable** de un objeto
  - Se acceden con el operador ".": **objeto.propiedad**

- **Método:** **función** "especial" asociada a un objeto
  - invocada con el operador **".":** **objeto.metodo(parametros)**
    - Un **método** devuelve un **valor de retorno** igual que una función
      - El método **tiene acceso al objeto** y puede inspeccionar o modificar sus componentes

Ejemplos de objetos DOM



- Objetos DOM:
  - dan acceso a los **elementos HTML** de una **página Web, por ejemplo**
- **document**
  - **Objeto DOM** que da acceso a la **página Web** cargada en el navegador
    - Es un objeto visible en todo programa JavaScript cuando este se ejecuta en el navegador

metodo

- **getElementById(…….)**
  - Es un **método** que se puede invocar sobre **document** (pertenece a document)

objeto DOM metodo parametro

- **document.getElementById("fecha")**
  - **getElementById("fecha")** devuelve el objeto DOM del elemento HTML con **id="fecha"**
    - cuando se invoca sobre el objeto **document**

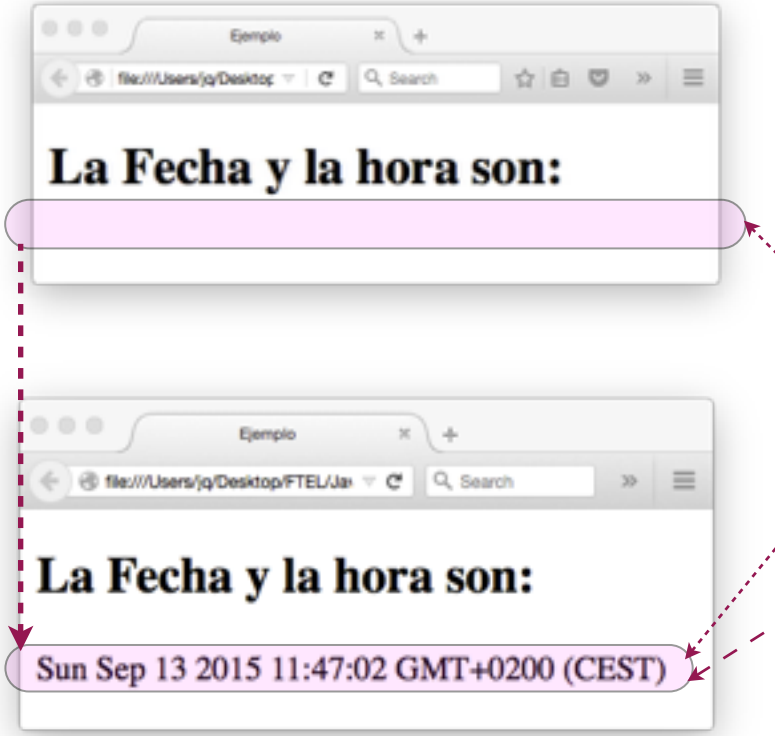
objeto DOM metodo parametro Propiedad

- **document.getElementById("fecha").innerHTML**
  - Es la **propiedad innerHTML** del objeto DOM asociado al elemento HTML con **id="fecha"**

objeto metodo parametro

- **var cl = document.getElementById("fecha")**
  - Asigna a la **variable cl** un objeto DOM
    - **cl.innerHTML** es la propiedad **innerHTML** del objeto contenido en **cl**

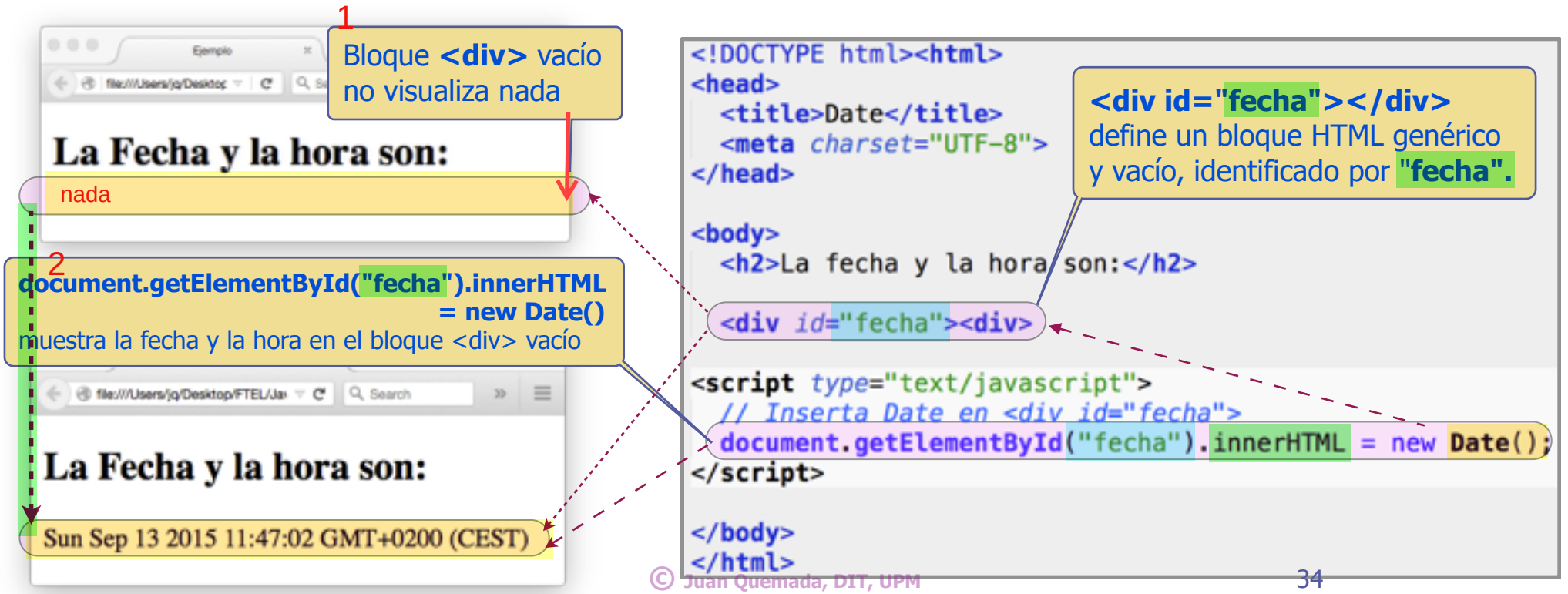
Ejemplo fecha y hora: innerHTML




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

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

  <div id="fecha"><div>

<script type="text/javascript">
  // Inserta Date en <div id="fecha">
  document.getElementById("fecha").innerHTML = new Date();
</script>

</body>
</html>
```

Ejemplo fecha y hora: innerHTML

- La propiedad **innerHTML** de un objeto DOM contiene el HTML interno (delimitado entre marcas)
  - La propiedad **outerHTML** contiene todo el HTML del elemento incluyendo las ~~marcas~~ ETIQUETAS
    - Modificando el contenido de **innerHTML** o **outerHTML** modificamos desde javaScript la página visualizada en el navegador
- La sentencia de asignación **document.getElementById("fecha").innerHTML = new Date();**
  - muestra en el navegador la fecha y la hora en la caja del bloque genérico identificado por "**fecha**"
- ** ** define un bloque HTML sin contenido
  - la propiedad **innerHTML** de su objeto DOM contiene inicialmente: **""** interior de la etiqueta
  - la propiedad **outerHTML** de su objeto DOM contiene inicialmente: **" "** contenido y etiquetas del HTML
- El navegador **no muestra nada** al visualizar la página, pero el script **inserta la hora y la fecha**



Fecha y hora equivalente


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

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

  <div id="fecha"><div>

<script type="text/javascript">
  var cl = document.getElementById("fecha");
  cl.innerHTML = new Date();
</script>

</body>
</html>
```

Fecha y hora equivalente

- El script JavaScript mostrado aquí es totalmente equivalente al anterior
  - pero es la forma habitual de hacerlo, porque es más conciso, más legible e incluso más eficiente
- La sentencia de asignación **document.getElementById("fecha").innerHTML = new Date();**
  - carga el objetoDOM
  - Modifica la propiedad
  - Se descompone aquí en dos sentencias (equivalentes a lo anterior)
    - La primera **carga el objeto DOM** en una variable y la segunda **modifica su propiedad innerHTML**

La referencia **document.getElementById("fecha")** a objetos DOM es larga por lo que es muy habitual almacenar primero el objeto DOM en una variable, para acceder luego a sus propiedades, como hacemos aquí

```
var cl = document.getElementById("fecha");
cl.innerHTML = new date();
```

el programa es totalmente equivalente pero más conciso, más legible e incluso más eficiente.


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

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

  <div id="fecha"><div>

<script type="text/javascript">
  var cl = document.getElementById("fecha");
  cl.innerHTML = new Date();
</script>

</body>
</html>
```

Varios scripts


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

<script type="text/javascript">

function mostrar_fecha( ) {
  var cl = document.getElemtById("fecha");
  cl.innerHTML = new Date( );
}
</script>

</head>

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

<div id="fecha"><div>

<script type="text/javascript">
  mostrar_fecha( ); // Invocar función
</script>

</body>
</html>
```

Varios scripts

- **Varios scripts** en una página forman un **único programa JavaScript**
  - Las definiciones (variables, funciones, …) son visibles entre scripts
- Los **scripts se ejecutan** siguiendo el **orden de definición** en la página
  - **Instrucciones adicionales** ejecutadas en la consola del navegador, **se ejecutan después del último script**
- Este ejemplo también es equivalente a los anteriores
  - Define la **función que inserta fecha** y hora en un script en la cabecera y **la invoca en el script del final**
    - La invocación debe realizarse al final, para que el árbol DOM esté ya construido y **el elemento DOM se haya construido ya**

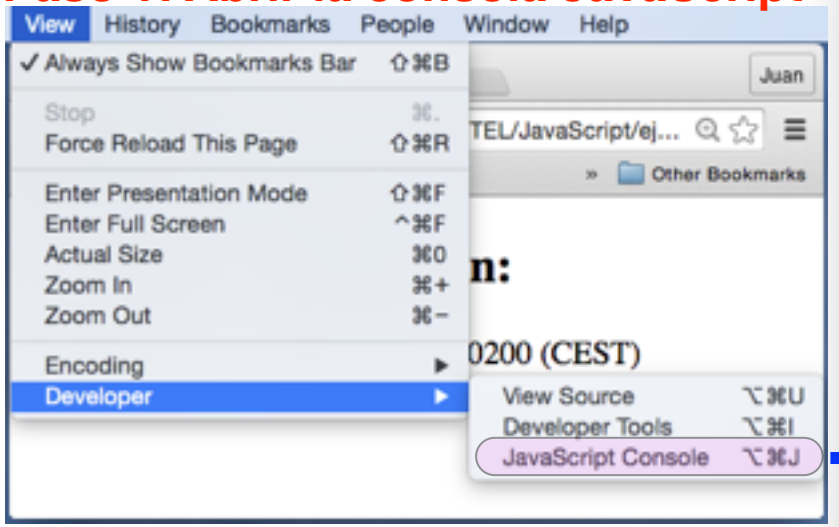

```
<!DOCTYPE html><html>
<head>
<title>Ejemplo de función</title>
<meta charset="UTF-8">

<script type="text/javascript">

function mostrar_fecha( ) {
  var cl = document.getElemtById("fecha");
  cl.innerHTML = new Date( );
}
</script>

</head>

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

<div id="fecha"><div>

<script type="text/javascript">
  mostrar_fecha( ); // Invocar función
</script>

</body>
</html>
```

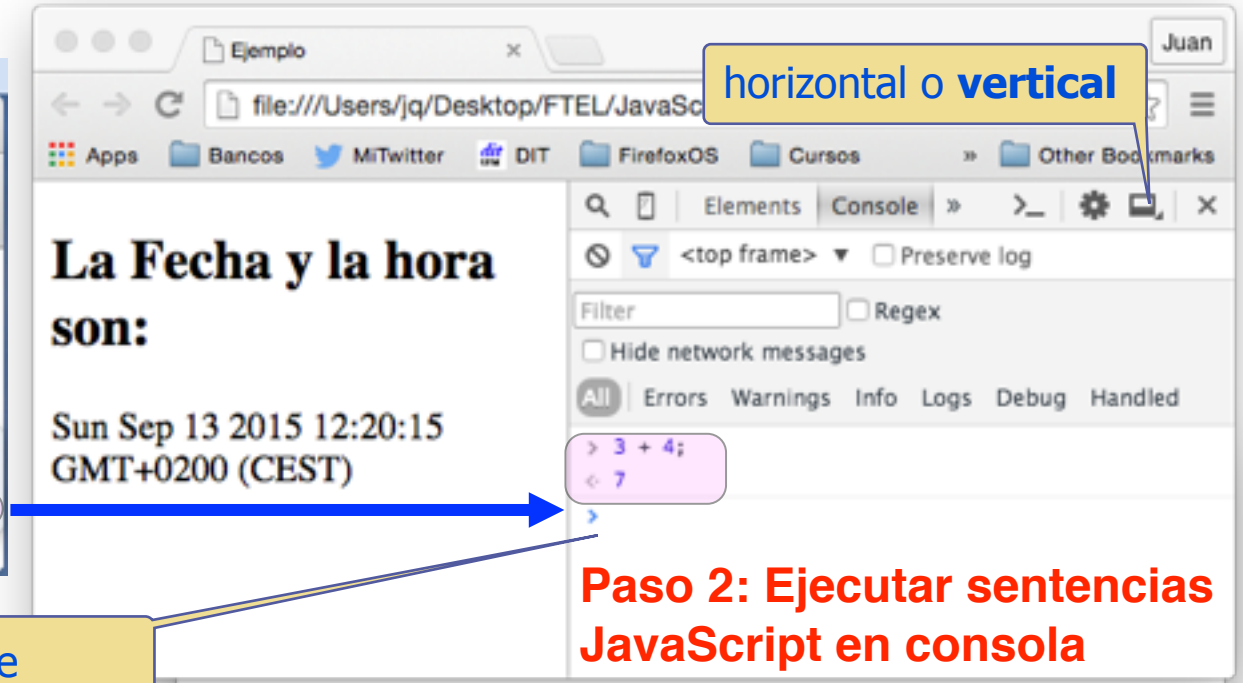
Funciones de selección de elementos DOM

- **getElementById("my_id")**
  - Es el mas sencillo de utilizar porque devuelve
    - El objeto DOM con el identificador buscado o null si no lo encuentra
      - ¡Un identificador solo puede estar en un objeto de una página HTML!

- **getElementsByName("my_name"),**
  getElementsByTagName("my_tag"),
  getElementsByClassName("my_class"),
  querySelectorAll("CSS selector"),...
  - Devuelven una matriz de objetos
    - Por ejemplo: getElementsByName("my_name")[0]
      - referencia el primer elemento con atributo **name="my_name"**
  - **Doc:** https://developer.mozilla.org/en/docs/Web/API/Document

Paso 1: Abrir la consola JavaScript





horizontal o **vertical**

Paso 2: Ejecutar sentencias JavaScript en consola

La consola del navegador (Chrome) puede ejecutar sentencias JavaScript en modo interactivo, igual que node en modo interactivo.

Aquí visualizamos el nuevo HTML modificado con JavaScript.

La ejecución de la sentencia JavaScript

```
document.getElementById("fecha").innerHTML="Hola pepsicola"
```

se realiza después de ejecutar el último script. Es parte del mismo programa y puede acceder a sus, variables, funciones y elementos HTML.

Final del tema