



Booleano, igualdad y otros operadores lógicos

Tipo booleano

FALSE
true

◆ El tipo **boolean** (booleano) solo tiene solo 2 valores

- **true**: verdadero
- **false**: falso

!false	=> true
!true	=> false

◆ Operador unitario **negación** (negation): **! <valor booleano>**

- Convierte un valor lógico en el opuesto, tal y como muestra la tabla

◆ Las expresiones booleanas, también se denominan expresiones lógicas

- Se evalúan siempre a verdadero (true) o falso (false)
 - ◆ Se utilizan para **tomar decisiones en sentencias condicionales**: if/else, bucles,
 - ◆ Por ejemplo: **if (expresión booleana) {Acciones_A} else {Acciones_B}**

Conversión a booleano

FALSE
true

◆ La regla de conversión de otros tipos a booleano es

- **false:** 0, -0, NaN, null, undefined, "", "
- **true:** resto de valores

◆ Cuando el operador negación ! se aplica a otro tipo

- convierte el valor a su equivalente booleano
 - ◆ y obtiene el valor booleano opuesto

◆ El operador negación aplicado 2 veces permite obtener

- el booleano equivalente a un valor de otro tipo, por ejemplo **!!1 => true**

!4	=> false
!"4"	=> false
!null	=> true
!0	=> true
!!""	=> false
!!4	=> true

Operadores de identidad e igualdad

- ◆ Identidad o igualdad estricta: `<v1> === <v2>`
 - igualdad de tipo y valor:
 - ◆ aplicable a: **number**, **boolean** y **strings**
 - En objetos es **identidad de referencias**
- ◆ Desigualdad estricta: `<v1> !== <v2>`
 - negación de la igualdad estricta
- ◆ Igualdad y desigualdad débil: `==` y `!=`
 - **No utilizar!** Conversiones impredecibles!

// Identidad de tipos básicos

```

0 === 0           => true
0 === 0.0        => true
0 === 0.00       => true

0 === 1          => false
0 === false      => false

'2' === "2"      => true
'2' === "02"     => false

"" === ""        => true
" " === " "      => false
  
```

◆ Mas info: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Sameness>

Operadores de comparación

◆ JavaScript tiene 4 operadores de comparación

- Menor: `<`
- Menor o igual: `<=`
- Mayor: `>`
- Mayor o igual: `>=`

◆ Utilizar comparaciones solo con números (number)

- poseen una relación de orden bien definida

◆ No se recomienda utilizar con otros tipos: **string**, **boolean**, **object**, ..

- Las relación de orden en estos tipos existe, pero es muy poco intuitiva
 - ◆ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators

`1.2 < 1.3` \Rightarrow true

`1 < 1` \Rightarrow false

`1 <= 1` \Rightarrow true

`1 > 1` \Rightarrow false

`1 >= 1` \Rightarrow true

`false < true` \Rightarrow true

`"a" < "b"` \Rightarrow true

`"a" < "aa"` \Rightarrow true

Operador y de JavaScript: &&

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

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

- ◆ Operador lógico y
 - operador binario: **<valor_1> y <valor_2>**
 - ◆ Verdadero solo si ambos valores son verdaderos
- ◆ **&&** se ha extendido y es más que un operador lógico
 - No convierte el resultado a booleano
 - ◆ Solo **interpreta <valor_1> como booleano** y según sea **false** o **true**
 - ◆ Devuelve como resultado **<valor_1>** o **<valor_2>** sin modificar
- ◆ Semántica del operador lógico y (and) de JavaScript: **<valor_1> && <valor_2>**
 - si **<valor_1>** se evalúa a **false**
 - ◆ devuelve **<valor_1>**, sino devuelve **<valor_2>**

Operador o de JavaScript: ||

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

```
undefined || 0    => 0
13 || 0           => 13
```

```
// Asignar valor por defecto
// si x es undefined o null
```

```
x = x || 0;
```

- ◆ Operador lógico o
 - operador binario: **<valor_1> o <valor_2>**
 - ◆ Verdadero solo si ambos valores son verdaderos
- ◆ || se ha extendido y es más que un operador lógico
 - No convierte el resultado a booleano
 - ◆ Solo **interpreta <valor_1> como booleano** y según sea **false** o **true**
 - ◆ Devuelve como resultado **<valor_1> o <valor_2>** sin modificar
- ◆ Semántica del operador lógico o (or) de JavaScript: **<valor_1> || <valor_2>**
 - si **<valor_1>** se evalúa a **true**
 - ◆ devuelve **<valor_1>**, sino devuelve **<valor_2>**

Operador de asignación condicional: “?:”

- ◆ El operador de asignación condicional
 - devuelve un valor en función de una **condición** lógica
 - ◆ La condición lógica va siempre entre paréntesis
- ◆ Semántica de la asignación condicional: **(condición) ? <valor_1> : <valor_2>**
 - si **condición** se evalúa a **true**
 - ◆ devuelve **<valor_1>**, sino devuelve **<valor_2>**

<code>(true) ? 0 : 7</code>	<code>=> 0</code>
<code>(false) ? 0 : 7</code>	<code>=> 7</code>

<code>(7) ? 0 : 7</code>	<code>=> 0</code>
<code>("") ? 0 : 7</code>	<code>=> 7</code>



Sentencia if/else

Sentencia if/else

- ◆ **if/else** permite ejecución condicional de
 - bloques de instrucciones
- ◆ Comienza por la palabra reservada **if**
 - La condición va después entre paréntesis
- ◆ **Bloque**: sentencias delimitadas por **{..}**
 - Bloque de 1 sentencia puede omitir **{ }**
- ◆ La parte **else** es opcional

```
17-if_bloque.js UNREGISTERED
// Sentencia if/else
//
// -> ejecuta bloque 1
//     si x es true
//
// -> ejecuta bloque 2
//     si x es false

if (x) {
  y = 0;
  z = "hola";
}
else {
  y = 1;
  z = "adios";
}
```

```
18-if_bloque_1_sentencia.js UNREGISTERED
// La parte else es opcional

if (x) {
  y = 0;
}

// Bloque de 1 sentencia
// puede omitir parentesis

if (x) y = 0;

if (x)
  y = 0;
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3> Sentencia if/else </h3>
```

```
<script type="text/javascript">
```

```
// Math.random() devuelve
// número aleatorio entre 0 y 1.
```

```
var numero = Math.random();
```

```
if (numero <= 0.5){
  document.writeln(numero + ' MENOR que 0,5');
}
else {
  document.writeln(numero + ' MAYOR que 0,5');
}
```

```
</script>
```

```
</body>
```

```
</html>
```

Sentencia if/else

0.5242976508023318 MAYOR que 0,5

Ejemplo con sentencia if/else

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3> Sentencia if </h3>
```

```
<script type="text/javascript">
```

```
    // Math.random() devuelve
    // número aleatorio entre 0 y 1.
```

```
    var numero = Math.random();
```

```
    var str = ' MAYOR que 0,5';
```

```
    if (numero <= 0.5){
        str = ' MENOR que 0,5';
    }
```

```
    document.writeln(numero + str);
```

```
</script>
```

```
</body>
```

```
</html>
```

Sentencia if/else

0.5242976508023318 MAYOR que 0,5

Ejemplo con sentencia if

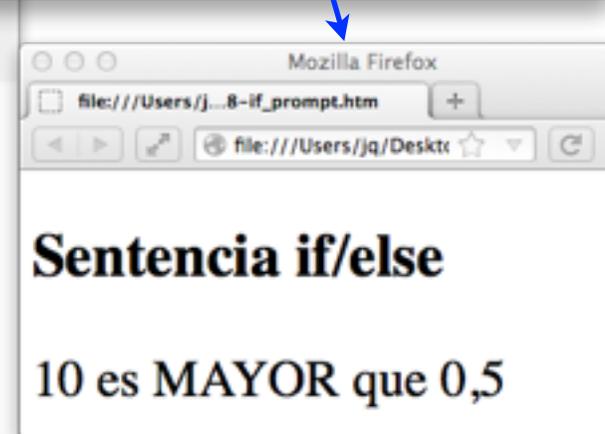
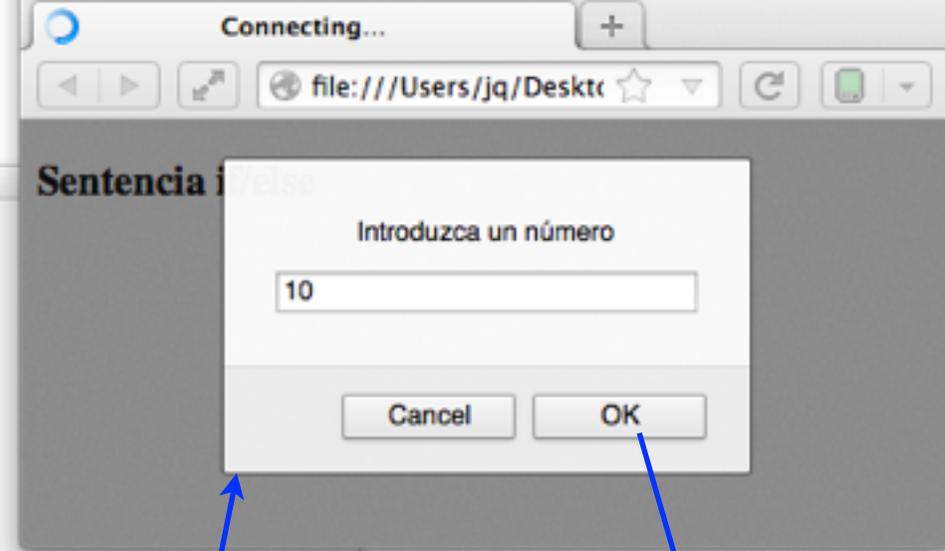
Ejemplo de prompt()

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3> Sentencia if/else </h3>

<script type="text/javascript">

    // Prompt pide un dato con un desplegable
    var numero = prompt("Introduzca un número");

    if (numero <= 0.5){
        document.writeln(numero + ' es MENOR que 0,5');
    }
    else {
        document.writeln(numero + ' es MAYOR que 0,5');
    }
</script>
</body>
</html>
```



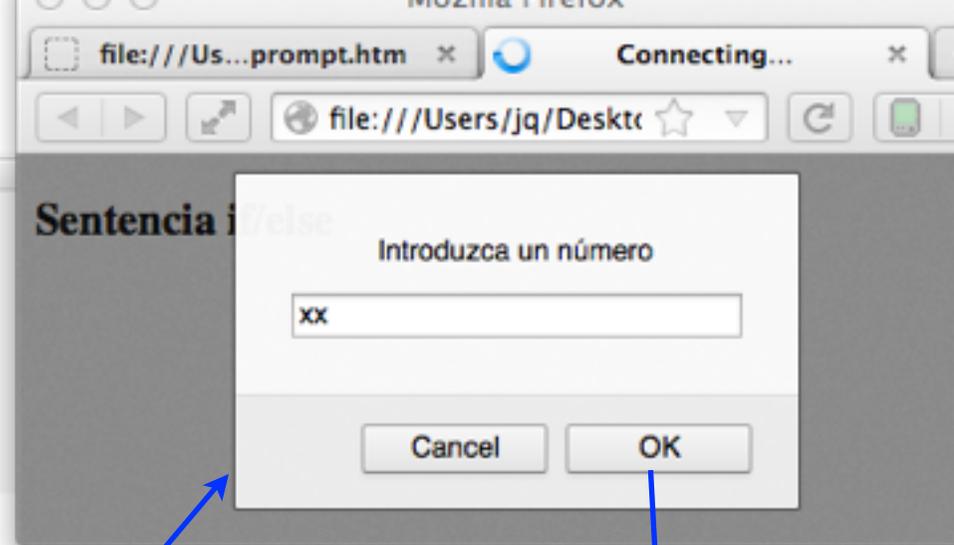
Ejemplo de else-if

```
<!DOCTYPE html>
<html><head>
<meta charset="UTF-8">
</head><body>
<h3> Sentencia if/else </h3>

<script type="text/javascript">

    // Prompt pide un dato con un desplegable
    var numero = prompt("Introduzca un número");

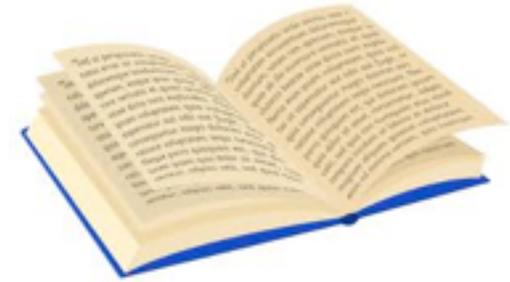
    // isNaN(..) determina si es un número
    if (isNaN(numero)) {
        document.write(numero + ' no es un número');
    }
    else if (numero <= 0.5) {
        document.write(numero + ' es MENOR que 0,5');
    }
    else
        document.write(numero + ' es MAYOR que 0,5');
    }
</script>
</body>
</html>
```





Strings e internacionalización (I18N)

El tipo string



- ◆ Texto internacionalizado codificado con el código UNICODE
 - Puede representar muchas lenguas diferentes
- ◆ Literales de string: textos delimitados por **comillas** o **apóstrofes**
 - **"hola, que tal"**, **'hola, que tal'**, **'Γεια σου, ίσως'** o **'嗨, 你好吗'**
 - ◆ string "hola, que tal" en varios idiomas
 - String vacío: **""** o **" "**
 - **"texto 'entrecorillado' "**
 - ◆ comillas y apóstrofes se pueden anidar: **'entrecorillado'** forma parte del texto
- ◆ Operador de concatenación de strings: **+**
 - **"Hola" + " " + "Pepe" => "Hola Pepe"**



Teclado chino

Internacionalización (I18N)

- ◆ UNICODE es un consorcio internacional: <http://www.unicode.org/>
 - Define normas de internacionalización (I18N), como el código UNICODE
 - ◆ UNICODE puede representar muchas lenguas: <http://www.unicode.org/charts/>
- ◆ JavaScript utiliza solo el **Basic Multilingual Plane** de UNICODE
 - Caracteres codificados en 2 octetos (16 bits), similar a BMP
 - ◆ UNICODE tiene otros planos que incluyen lenguas poco frecuentes
- ◆ **Teclado:** suele incluir solo las lenguas de un país
 - Los caracteres de lenguas no incluidas
 - ◆ solo se pueden representar con caracteres escapados
 - ◆ por ejemplo, `'\u55e8'` representa el ideograma chino '嗨'
- ◆ **Pantalla:** es gráfica y puede representar cualquier carácter



Teclado arabe

Caracteres escapados



◆ Los caracteres escapados

- son caracteres no representables dentro de un string
 - ◆ comienzan por la barra inclinada (\) y la tabla incluye algunos de los más habituales

◆ Además podemos representar cualquier carácter UNICODE o ISO-LATIN-1:

- `\uXXXX` carácter UNICODE de código hexadecimal **XXXX** 
- `\xXX` carácter ISO-LATIN-1 de código hexadecimal **XX** 

◆ Algunos ejemplos

- "Comillas dentro de \"comillas\""
 - ◆ " debe ir escapado dentro del string
- "Dos \n líneas"
 - ◆ retorno de línea delimita sentencias
- "Dos \u000A líneas"

CARACTERES ESCAPADOS

NUL (nulo):	<code>\0</code> , <code>\x00</code> , <code>\u0000</code>
Backspace:	<code>\b</code> , <code>\x08</code> , <code>\u0008</code>
Horizontal tab:	<code>\t</code> , <code>\x09</code> , <code>\u0009</code>
Newline:	<code>\n</code> , <code>\x0A</code> , <code>\u000A</code>
Vertical tab:	<code>\v</code> , <code>\x0B</code> , <code>\u000B</code>
Form feed:	<code>\f</code> , <code>\x0C</code> , <code>\u000C</code>
Carriage return:	<code>\r</code> , <code>\x0D</code> , <code>\u000D</code>
Comillas (dobles):	<code>\"</code> , <code>\x22</code> , <code>\u0022</code>
Apóstrofe :	<code>\'</code> , <code>\x27</code> , <code>\u0027</code>
Backslash:	<code>\\</code> , <code>\x5C</code> , <code>\u005C</code>

Clase String



◆ La clase String

- incluye métodos y propiedades para procesar strings
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

◆ Un string es un array de caracteres

- un índice entre **0** y **número_de_caracteres-1** referencia cada carácter

◆ Propiedad con tamaño: **'ciudad'.length** ==> **6**

◆ Acceso como array: **'ciudad'[2]** ==> **'u'**

◆ Método: **'ciudad'.charCodeAt(2)** ==> **117**

- devuelve código UNICODE de tercer carácter

◆ Método: **'ciudad'.indexOf('da')** ==> **3**

- devuelve posición de substring

◆ Método: **'ciudad'.substring(2,5)** ==> **'uda'**

- devuelve substring entre ambos índices

Ejemplo I18N

```
<!DOCTYPE html>
<html>
<head>
  <title>I18N</title>
  <meta charset="UTF-8">
</head><body>
<h2>Ejemplo I18N</h2>
Castellano, griego y chino: <p>
<pre>
<script type="text/javascript">
  document.writeln('"hola, que tal": ' + "hola, que tal");
  document.writeln("'hola, que tal': " + 'hola, que tal');
  document.writeln();
  document.writeln("En griego (Γεια σου, ίσως): " + 'Γεια σου, ίσως');
  document.writeln();
  document.writeln("'hola, que tal' en chino (嗨, 你好吗): " + '嗨, 你好吗');
  document.writeln("Caracteres escapados (\\u55e8\\uff0c\\u4f60\\u597d\\u5417): "
    + "\\u55e8\\uff0c\\u4f60\\u597d\\u5417");

  document.writeln();
  var x = '嗨, 你好吗'.charCodeAt(0).toString(16); // conversión char a string hexadec.
  var y = String.fromCharCode(parseInt(x, 16)); // conversión hexadecimal a string
  document.writeln('El caracter escapado \\u' + x + ' representa: ' + y);
</script>
</pre>
</body>
</html>
```

Ejemplo I18N

Castellano, griego y chino:

```
"hola, que tal": hola, que tal
'hola, que tal': hola, que tal
```

En griego (Γεια σου, ίσως): Γεια σου, ίσως

'hola, que tal' en chino (嗨, 你好吗): 嗨, 你好吗

Caracteres escapados (\u55e8\uff0c\u4f60\u597d\u5417): 嗨, 你好吗

El caracter escapado \u55e8 representa: 嗨



Números

Números: tipo number

◆ Los números se representan con literales de

- **Enteros:** 32
 - ◆ Entero máximo: **9007199254740992**
- **Decimales:** 32.23
- **Coma flotante:** 3.2e1 (3,2x10)
 - ◆ Rango real: **1,797x10³⁰⁸ --- 5x10⁻³²⁴**

◆ Todos los números son del tipo **number**

◆ Todos los números se representan igual internamente

- coma flotante de doble precisión (64bits)

◆ El tipo number incluye 2 valores especiales

- **Infinity:** representa desbordamiento
- **NaN:** representa resultado no numérico

```
10 + 4    => 14    // sumar
10 - 4    => 6     // restar
10 * 4    => 40    // multiplicar
10 / 4    => 2.5   // dividir
10 % 4    => 2     // operación resto
```

```
//decimales dan error de redondeo
```

```
0.1 + 0.2 => 0,300000000000004
```

```
3e2       => 300
```

```
3e-2      => 0,03
```

```
+10/0     => Infinity //desborda
```

```
-10/0     => -Infinity //desborda
```

```
5e500    => Infinity //desborda
```

Conversión a enteros

- ◆ Cuando JavaScript calcula expresiones
 - **convirtiendo tipos** según necesita
 - ◆ utiliza las prioridades de operadores

- ◆ Conversión a **entero (o real)**
 - **booleano**: true a 1, false a 0
 - **String**: Convierte número a valor o NaN
 - **null**: a 0, **undefined**: a NaN

- ◆ Convertir un **string** a un **número**
 - se denomina también “parsear” o analizar sintácticamente
 - ◆ es similar al análisis sintáctico realizado a los literales de números

```
'67' + 13    => 6713
+'67'   + 13  => 80
+'6.7e1' + 13 => 80

'xx' + 13    => 'xx13'
+'xx' + 13   => NaN

13 + true    => 14
13 + false   => 13
```

Modulo Math

- ◆ El Modulo Math contiene
 - constantes y funciones matemáticas
- ◆ Constantes
 - Números: E, PI, SQRT2, ...
 - ...
- ◆ Funciones
 - $\sin(x)$, $\cos(x)$, $\tan(x)$, $\text{asin}(x)$,
 - $\log(x)$, $\text{exp}(x)$, $\text{pow}(x, y)$, $\text{sqrt}(x)$,
 - $\text{abs}(x)$, $\text{ceil}(x)$, $\text{floor}(x)$, $\text{round}(x)$,
 - $\text{min}(x,y,z,...)$, $\text{max}(x,y,z,...)$, ...
 - $\text{random}()$

```
Math.PI => 3.141592653589793
```

```
Math.E => 2.718281828459045
```

```
// numero aleatorio entre 0 y 1
```

```
Math.random() => 0.7890234
```

```
Math.pow(3,2) => 9 // 3 al cuadrado
```

```
Math.sqrt(9) => 3 // raíz cuadrada de 3
```

```
Math.min(2,1,9,3) => 1 // número mínimo
```

```
Math.max(2,1,9,3) => 9 // número máximo
```

```
Math.floor(3.2) => 3
```

```
Math.ceil(3.2) => 4
```

```
Math.round(3.2) => 3
```

```
Math.sin(1) => 0.8414709848078965
```

```
Math.asin(0.8414709848078965) => 1
```

Mas info:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Clase Number

- ◆ La clase Number encapsula números
 - como objetos equivalentes
- ◆ Number define algunos métodos útiles
 - **toFixed(n)** devuelve string
 - ◆ redondeando a n decimales
 - **toExponential(n)** devuelve string
 - ◆ redondeando mantisa a n decima.
 - **toPrecision(n)** devuelve string
 - ◆ redondeando a n dígitos
- ◆ JS convierte una expresión a objeto al
 - aplicar el método a una expresión
 - ◆ **Ojo!** literales dan error sintáctico

Mas info:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

```
var x = 1.1;

x.toFixed(0)      => "1"
x.toFixed(2)     => "1.10"

(1).toFixed(2)   => "1.00"

1.toFixed(2)     => Error sintáctico

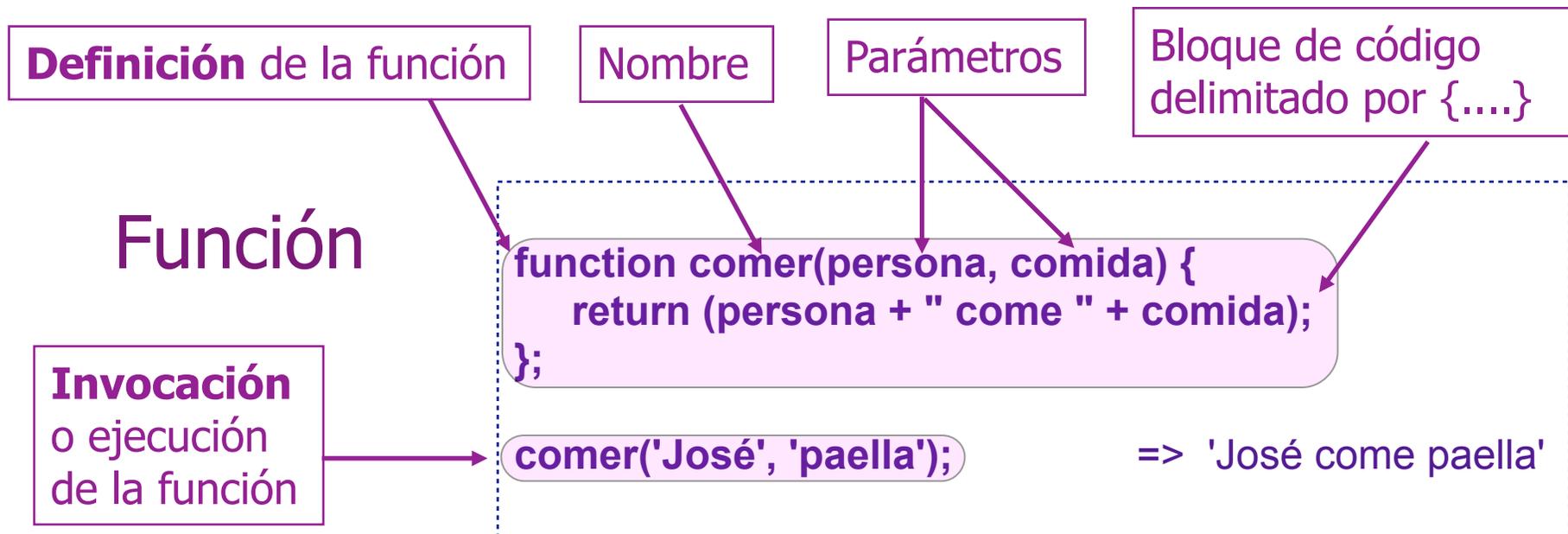
Math.PI.toFixed(4) => "3.1416"

(0.1).toExponential(2) => "1.00e-1"
x.toExponential(2)    => "1.10e+0"

(0.1).toPrecision(2)  => "0.10"
x.toPrecision(2)     => "1.1"
```



Funciones



◆ **Función:**

- bloque de código con parámetros, invocable (ejecutable) a través del nombre
 - ◆ La ejecución finaliza con la sentencia **“return expr”** o al **final** del bloque
- Al acabar la ejecución, devuelve un resultado: **valor de retorno**

◆ **Valor de retorno**

- resultado de evaluar **expr**, si se ejecuta la sentencia **“return expr”**
- **undefined**, si se alcanza final del bloque sin haber ejecutado ningún **return**

Parámetros de una función

- ◆ Los parámetros de la función son variables utilizables en el cuerpo de la función
 - Al invocarlas se asignan los valores de la invocación
- ◆ La función se puede invocar con un **número variable de parámetros**
 - Un **parámetro inexistente** está **undefined**

```
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

- ◆ 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'
```

Parámetros por defecto

- ◆ Funciones invocadas con un número variable de parámetros
 - Suelen definir parámetros por defecto con el operador ||
 - ◆ "x || <parámetro_por_defecto>"
- ◆ Si x es "undefined", será false y devolverá **parámetro por defecto**
- ◆ Los parámetros son variables y se les puede asignar un valor

```
function comer (persona, comida) {  
  persona = (persona || 'Alguién');  
  comida = (comida || 'algo');  
  return (persona + " come " + comida);  
};
```

```
comer('José');    => 'José come algo'  
comer();          => 'Alguien come algo'
```

Funciones como objetos

- ◆ Las funciones son **objetos** de pleno derecho
 - pueden asignarse a **variables, propiedades, parámetros,**
- ◆ “**function literal**”: es una función que se define sin nombre
 - Se suele asignar a una variable, que le da su nombre
 - ◆ Se puede invocar a través del nombre de la variable

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

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

Operador de invocación de una función

- ◆ El objeto función puede asignarse o utilizarse como un valor
 - el objeto función contiene el código de la función
- ◆ el operador (...) invoca una función ejecutando su código
 - Solo es aplicable a funciones (objetos de la clase Function)
 - Puede incluir una lista de parámetros separados por coma

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

```
var x = comer; // asigna a x el código de la función  
x('José','paella'); => 'José come paella'  
x() => 'undefined come undefined'
```

```
var y = comer(); // asigna a y el resultado de invocar la función  
y => 'undefined come undefined'
```