



# Introducción a JavaScript 6-7-8-9 (ES6 o ES2015, ES7 o ES2016, ..)

# Índice

- Clases predefinidas
- Arrays



# Clases predefinidas de JavaScript: propiedades y métodos heredados

# Clases predefinidas

## ◆ Object

- Clase raíz que define colecciones de propiedades y métodos.      Literal de objeto: **{a:3, b:"que tal"}**

## ◆ Array

- Define colecciones ordenadas de valores.      Literal de array: **[1, 2, 3]**

## ◆ Date

- Define objetos con hora y fecha del reloj del sistema.      Solo constructor: **new Date(...)**

## ◆ Function

- Define código parametrizado.      Literales de función: **function (x) {....}** o **(x) => {....}** (ES6)

## ◆ RegExp

- Define expresiones regulares para reconocer y procesar patrones de texto.      Literal: **/(hola)+\$/**

## ◆ Error

- Errors de ejecución lanzados por el interprete de JavaScript.      Solo constructor: **new Error(...)**

## ◆ Number, String y Boolean

- Clases que encapsulan valores de los tipos number, string y boolean como objetos
  - ◆ Sus métodos se aplican a los tipos básicos directamente, la conversión a objetos es automática

## ◆ ES6 introduce nuevas clases

- **Promises, Map, Set, Typed Arrays, ....**

## ◆ Más información:

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

# Operador instanceof

- ◆ El operador **instanceof** determina
  - si un objeto o valor pertenece a una clase
- ◆ Los objetos de una clase derivada pertenecen también a la clase padre
  - Un array o una función pertenecen a la clase Object

```
({}) instanceof Object    => true      // {} es un objeto aunque este vacío
({}) instanceof Array     => false     // {} no es un Array, pertenece solo a Object

[] instanceof Array       => true      // [] es un array aunque este vacío
[] instanceof Object      => true      // pertenece a la clase Object,
                                     // porque Array deriva de Object

(function){} instanceof Function => true // function(){ } es una función vacía
(function){} instanceof Object  => true // pertenece a la clase Object,
                                     // porque Function deriva de Object

""                          instanceof String => false // "" es un tipo primitivo
                                     // y los tipos primitivos no son objetos
new String("") instanceof String => true  // new String("") si pertenece a la clase String
```

# Métodos heredados

- ◆ **Método**: función invocable sobre un objeto con el operador punto: "."
  - Por ejemplo, `new Date().toString()`
- ◆ Un objeto **hereda** las propiedades y métodos de su **clase**, por ejemplo
  - los objetos de la **clase Date heredan** métodos como
    - ◆ `toString()`, `getDay()`, `getFullYear()`, `getHours()`, `getMinutes()`, ..... (ver ejemplo)
    - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date)

```
var fecha = new Date(); // The object created contains hour-date of creation
```


```
fecha.toString()      => Fri Aug 08 2014 12:34:36 GMT+0200 (CEST)
```

```
fecha.toTimeString()  => 12:34:36 GMT+0200 (CEST)
```

```
fecha.getHours()      => 12
```

```
fecha.getMinutes()    => 34
```

```
fecha.getSeconds()    => 36
```

A horizontal yellow bar at the top of the slide, consisting of two segments of different lengths joined together.

Clase, prototipo y herencia: métodos de instancia o estáticos y this

# Clases JavaScript

## ◆ JavaScript utiliza "Tipado de Patos"

- "Si anda y grazna como un pato, debe ser un pato"
  - ◆ JavaScript simula clases utilizando **funciones** y **prototipos**

## ◆ Clase

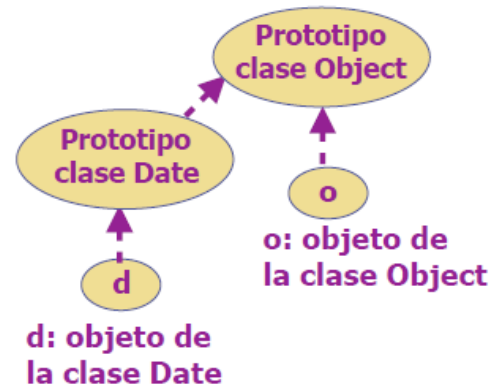
- Conjunto de objetos creados con el mismo **constructor** y que comparten un **prototipo**

## ◆ Prototipo

- Objeto del que objetos de una clase **heredan** los métodos y propiedades de clase
  - ◆ Estos se denominan **heredados** y existen en todos los objetos de la clase
- Los prototipos de clases derivadas están enlazados y se hereda de toda la cadena
  - ◆ La clase Object es la clase raíz del árbol de herencia y su prototipo es el único que no está enlazado
    - [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance\\_and\\_the\\_prototype\\_chain](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain)
  - ◆ Si dos prototipos de la cadena tienen una propiedad/método con el mismo nombre, se hereda la más cercana

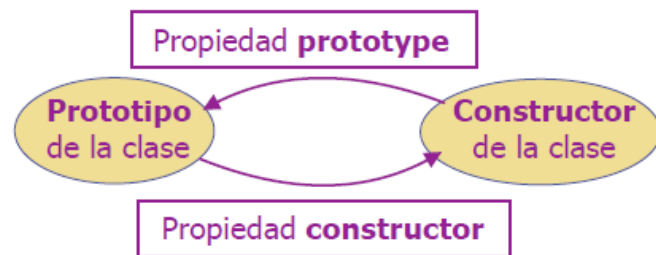
## ◆ Constructor

- **Función** que crea objetos de la clase al invocarse con el operador **new**, por ejemplo
  - ◆ **new Object()** crea un objeto similar a {}
  - ◆ **new Array()** crea un array similar a []
  - ◆ **new Date()** crea un objeto Date con la fecha y la hora de su creación (Date no tiene literal)
- El **nombre** del **constructor** y de la **clase** son los mismos
  - ◆ Existe la convención de utilizar los nombres que comienzan por mayúscula solo para constructores de clase, aunque cualquier función puede utilizarse como constructor de una clase





# Algunas propiedades y métodos



## ◆ Propiedades del **constructor**

### ■ **prototype**

- ◆ Devuelve el prototipo de la clase del objeto
- ◆ Ejemplos de métodos de instancia:
  - **Object.prototype.toString()**
  - **Array.prototype.forEach()**

### ■ **name**

- ◆ Devuelve un string con el nombre del constructor o de la clase asociada

```
Object.prototype; // => {}  
Object.name;      // => "Object"  
  
Array.prototype;  // => []  
Array.name;       // => "Array"  
  
Date.prototype;   // => Date {}  
Date.name;        // => "Date"
```

## ◆ Propiedades del **prototipo**

### ■ **constructor**

- ◆ Devuelve el constructor de la clase

## ◆ **Object.getPrototypeOf(<obj>)**

### ■ **Método estático** de la clase **Object**

- ◆ Da acceso al prototipo de la clase asociada al objeto <obj>
  - Equivale a: **obj.constructor.prototype**

```
({}).constructor; // => [Function: Object]  
({}).constructor.name; // => "Object"  
  
[].constructor; // => [Function: Array]  
[].constructor.name; // => "Array"  
  
new Date().constructor; // => [Function: Date]  
new Date().constructor.name; // => "Date"
```

# Ejemplo de definición de clase: Counter

## ◆ Constructor

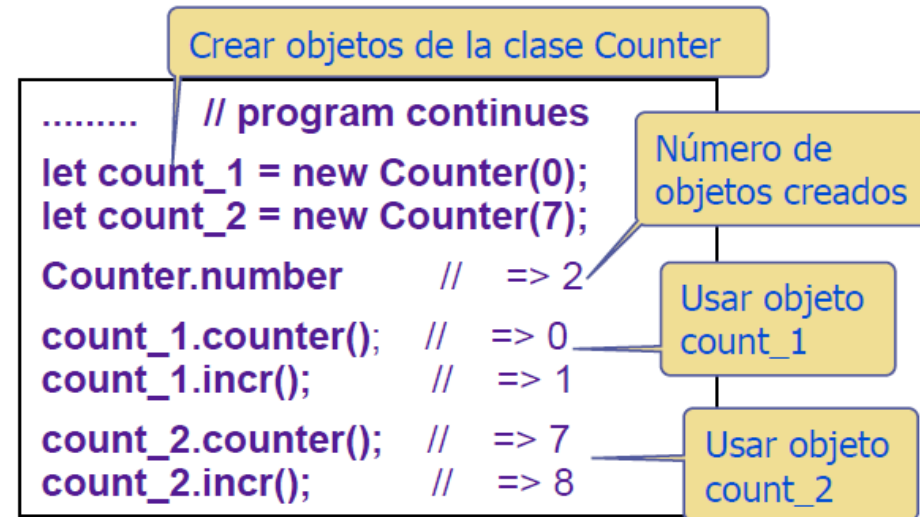
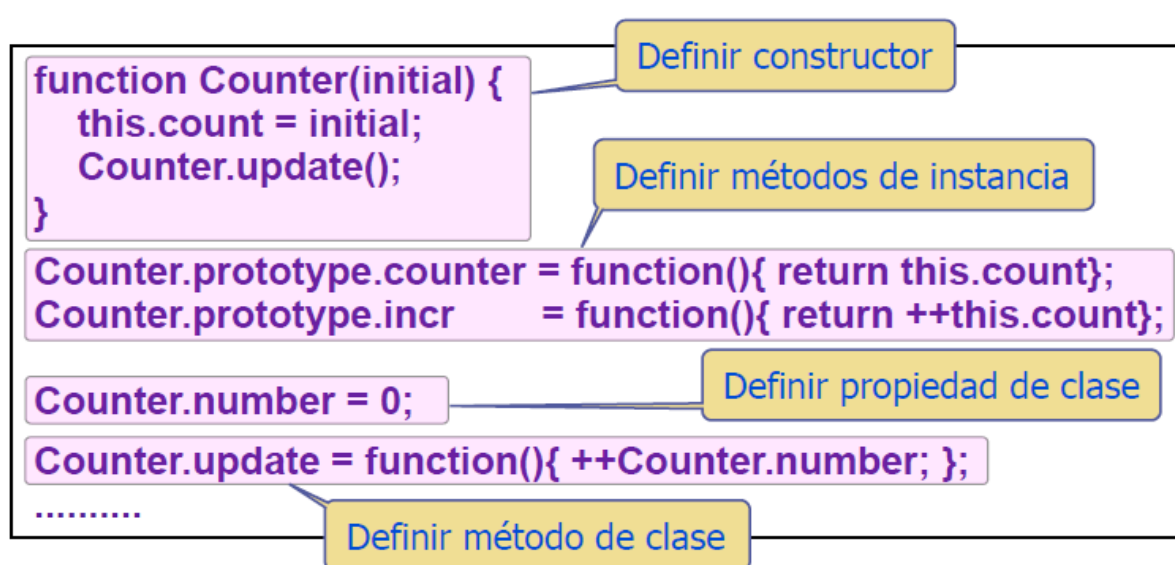
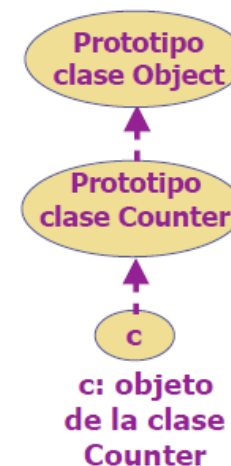
- El primer paso para crear una clase es definir el constructor: **Counter(..){..}**
  - this** referencia el objeto que está creando el constructor, cuando se invoca con **new**
  - this.count** crea dinámicamente la propiedad **count** con el contador del nuevo objeto

## ◆ Métodos de instancia: counter() e incr()

- Los métodos de instancia se añaden al prototipo de la función creado al definirla
  - El prototipo es un objeto JavaScript al que se le puede añadir propiedades y métodos como a los demás
  - obj.counter()** devuelve el estado del contador y **obj.incr()** incrementa el contador y devuelve su contenido

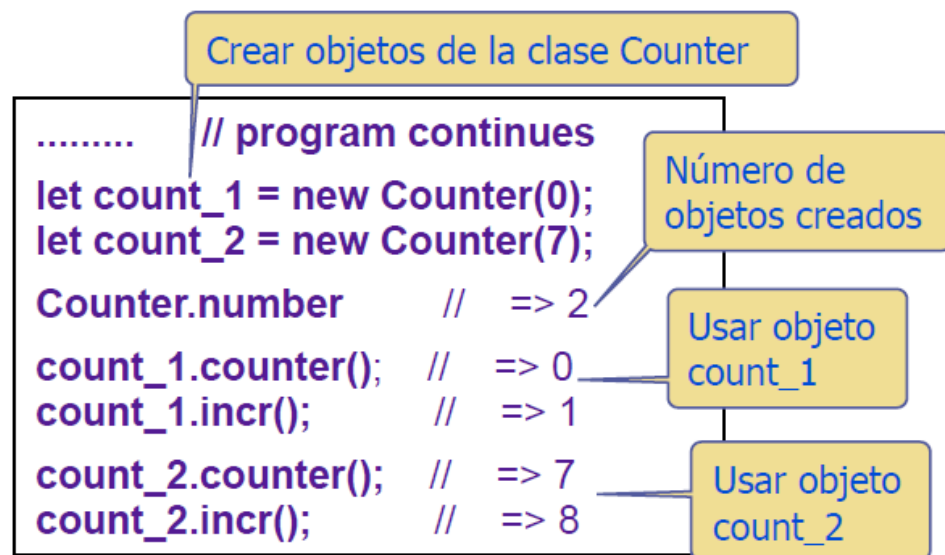
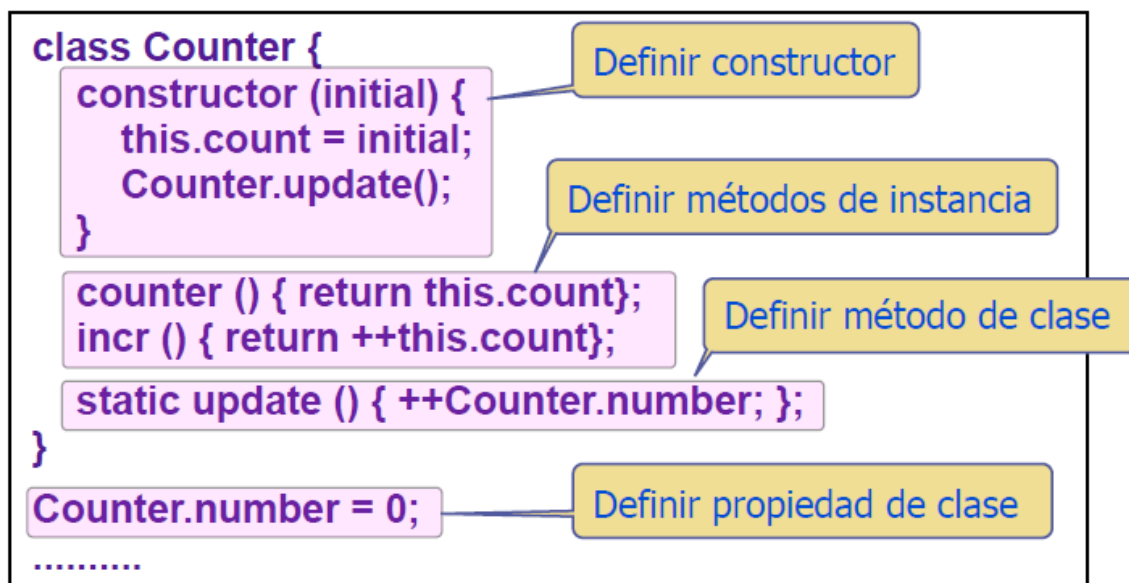
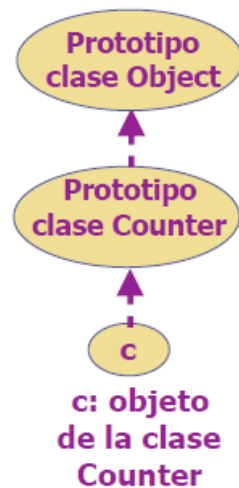
## ◆ Métodos y propiedades de clase: number y update

- Se deben añadir al constructor, por ejemplo **Counter.number** o **Counter.update**
  - Counter.update()** incrementa la cuenta de objetos creados en **Counter.number**



# Definición de clase en ES6: Counter

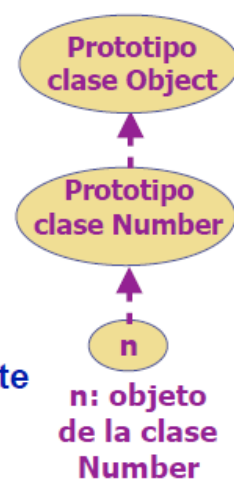
- ◆ **ES6** añade nueva sintaxis para definir clases de forma mas legible y concisa
  - Es azúcar sintáctico: las clases se construyen con funciones y prototipos como en ES5
    - ◆ Mas información en: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>
- ◆ El ejemplo muestra la misma clase Counter de la transparencia anterior
  - La sintaxis incluye la clase, los métodos de instancia y los métodos estáticos o de clase
    - ◆ Las propiedades de clase (estáticas) no están soportadas y se definen como en ES5 (en constructor)
- ◆ La sintaxis de **ES6** permite también extender clases y crear jerarquías
  - Aquí no lo vemos, pero se puede encontrar mas información en:
    - ◆ <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends>
    - ◆ <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super>



# Añadir método integer a la clase Number

## ◆ La clase **Number** encapsula métodos del tipo primitivo **number**

- Esta clase no posee un **método integer()** que calcule la parte entera del número
  - ♦ Para añadir este nuevo método a la clase se debe añadir **al prototipo**
    - Como la **propiedad puede existir**, hay que comprobarlo primero y lanzar un **error si existe**
- Documentación:
  - ♦ [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance\\_and\\_the\\_prototype\\_chain](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain)



## ◆ La parte entera de **n** es **Math.floor(n)** si es positivo o **Math.ceil(n)** si es negativo

- **this** referencia el objeto sobre el que se invoca el método (el número aquí)
  - ♦ Se utiliza notación array para invocar **Math.floor(..)** o **Math.ceil(..)**

Si ya existe una propiedad **integer** se lanza un error.

```
if("integer" in Number.prototype){  
  throw new Error("Number.prototype.integer already exists!");  
}
```

// Añadimos método "integer()" a Number

```
Number.prototype.integer = function () {  
  return Math[this > 0 ? "floor" : "ceil"](this);  
}
```

```
console.log("' 7.3.integer()' se evalúa a: " + 7.3.integer());  
console.log("' -7.3.integer()' se evalúa a: " + -7.3.integer());
```

El nuevo método **integer** se añade al prototipo de la clase **Number**.

**this > 0 ? "floor" : "ceil"** devuelve el nombre de la función que calcula el entero más próximo: **"floor"** o **"ceil"**. Este se aplicaría, por ejemplo si el número es positivo como: **Math["floor"](this)**. **this** referencia el número al que se aplica el método **integer()**.

```
_ $  
_ $ node 50_integer.js  
' 7.3.integer()' se evalúa a: 7  
' -7.3.integer()' se evalúa a: -7  
_ $
```