

# BLOQUE DIDÁCTICO D: USO DE LOS OBJETOS PREDEFINIDOS DEL LENGUAJE



Román-Herrera, J.C.

## DESARROLLO WEB EN ENTORNOS CLIENTE

# TEMARIO DEL BLOQUE DIDÁCTICO D

1. Utilización de objetos. Objetos nativos del lenguaje.
2. Interacción con el navegador. Objetos predefinidos asociados.
3. Generación de texto y elementos HTML desde código.
4. Aplicaciones prácticas de los marcos.
5. Gestión de la apariencia de la ventana.
6. Creación de nuevas ventanas y comunicación entre ellas.



## OBJETIVOS DEL BLOQUE D

- Reconocer las ventajas de usar objetos en el desarrollo de aplicaciones web.
- Reconocer las características principales de la POO.
- Conocer y aplicar los objetos y métodos disponibles.
- Establecer un mecanismo de herencia por prototipo de objeto.
- Examinar objetos mediante el recorrido de sus propiedades y métodos.
- Distinguir el funcionamiento básico del JSON
- Conocer y aplicar la potencia del objeto Math.
- Utilizar el objeto Date para manipulación de fechas y horas.
- Distinguir el funcionamiento de expresiones regulares, y validarlas con objetos RegExp.

# JAVASCRIPT Y LA PROGRAMACIÓN ORIENTADA A OBJETOS

¿Qué es la POO? → Es un paradigma de programación ...

¿Qué es eso de un paradigma de programación? → estilo de programación de software o forma de diseñar un programa

y... ¿Cuándo surge? → se inició en los 60, pero mu mayor auge surge en los 80 por influencia del C++

¿Qué características tiene este paradigma? →

- Representación de objetos
  - Uso de unidades mínimas llamadas objetos
- se traduce

A que la POO se encuentra basado en el entendimiento de una serie de conceptos:

- Clases → Es la “plantilla” que permite crear objetos definiendo las características y acciones de cada objeto
- Objetos → Es el “elemento abstracto informático” de un objeto real, que cuenta con características y acciones
- Atributo → Es el valor que almacena internamente el objeto (las características).
- Métodos → Conjunto de instrucciones que se han asociado a un objeto a las que se ha dado un nombre



Pero ... resumiendo, ¿Qué es un objeto?

Una forma de encapsular la información y las operaciones necesarias para operar dicho objeto



# JAVASCRIPT Y LA PROGRAMACIÓN ORIENTADA A OBJETOS

Ejemplo:

Imagina un coche que te guste...

¿Lo visualizas? Seguro que se te vienen muchos coches que te gustan...

Describemelo...

- La carrocería
- El color
- Las ruedas
- Carburante
- ....

Atributos

¿Y que puede hacer?...

- Arrancar
- Parar
- Acelerar
- Repostar
- ....

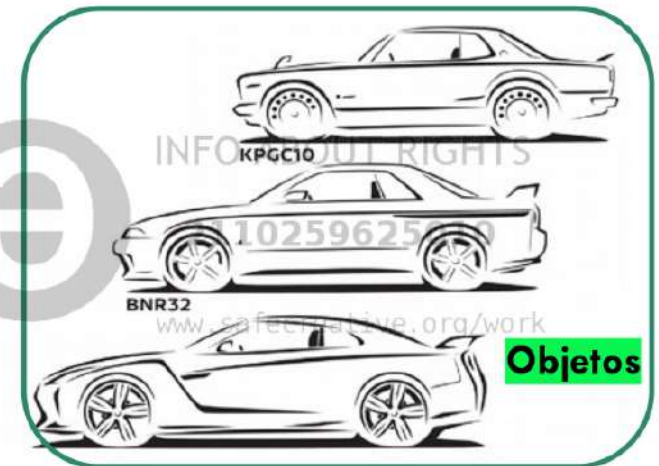
Métodos

¿Y quien me hace el coche?



Clase

¿Y que produce el "molde"?



# JAVASCRIPT Y LA PROGRAMACIÓN ORIENTADA A OBJETOS

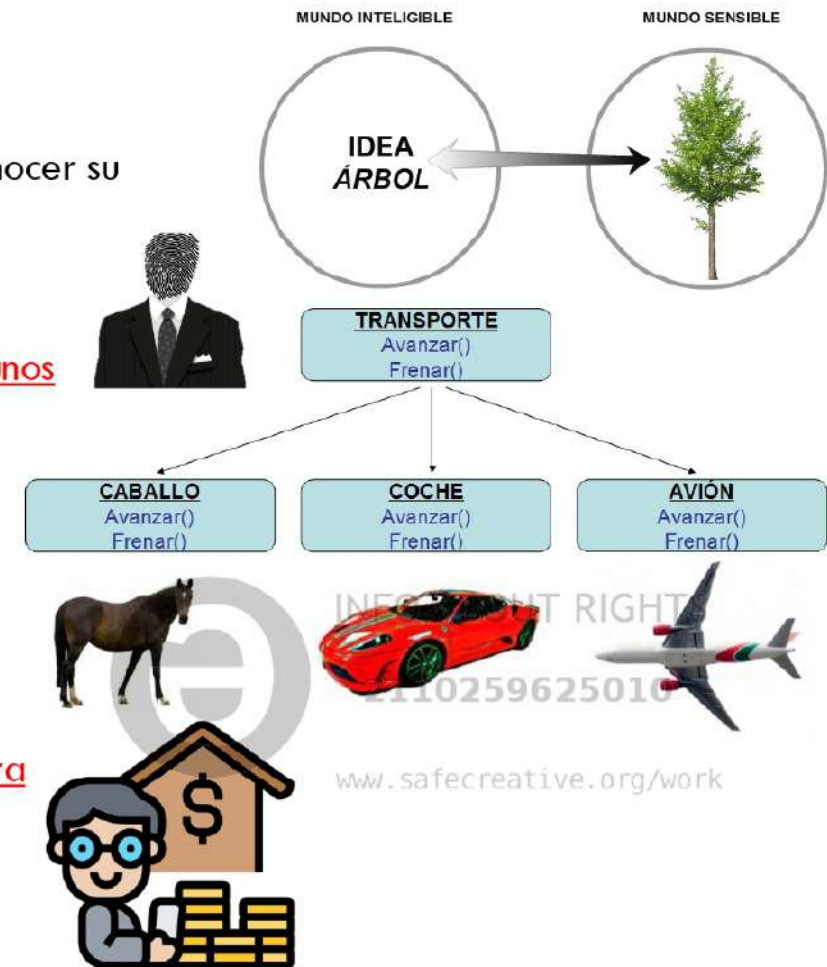
¿Qué características ha de tener la POO?

**ABSTRACCIÓN** → Es la “*propiedad*” que permite reutilizar objetos sin conocer su funcionamiento

**ENCAPSULAMIENTO** → Es la “*capacidad*” de que los puedan ocultar algunos métodos y/o propiedades

**POLIMORFISMO** → Es la “*característica*” que diferentes tipos de objetos puedan tener métodos con el mismo nombre pero actuando de manera distinta

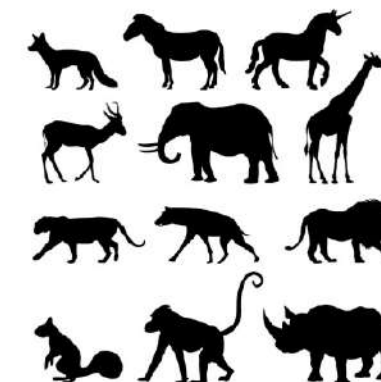
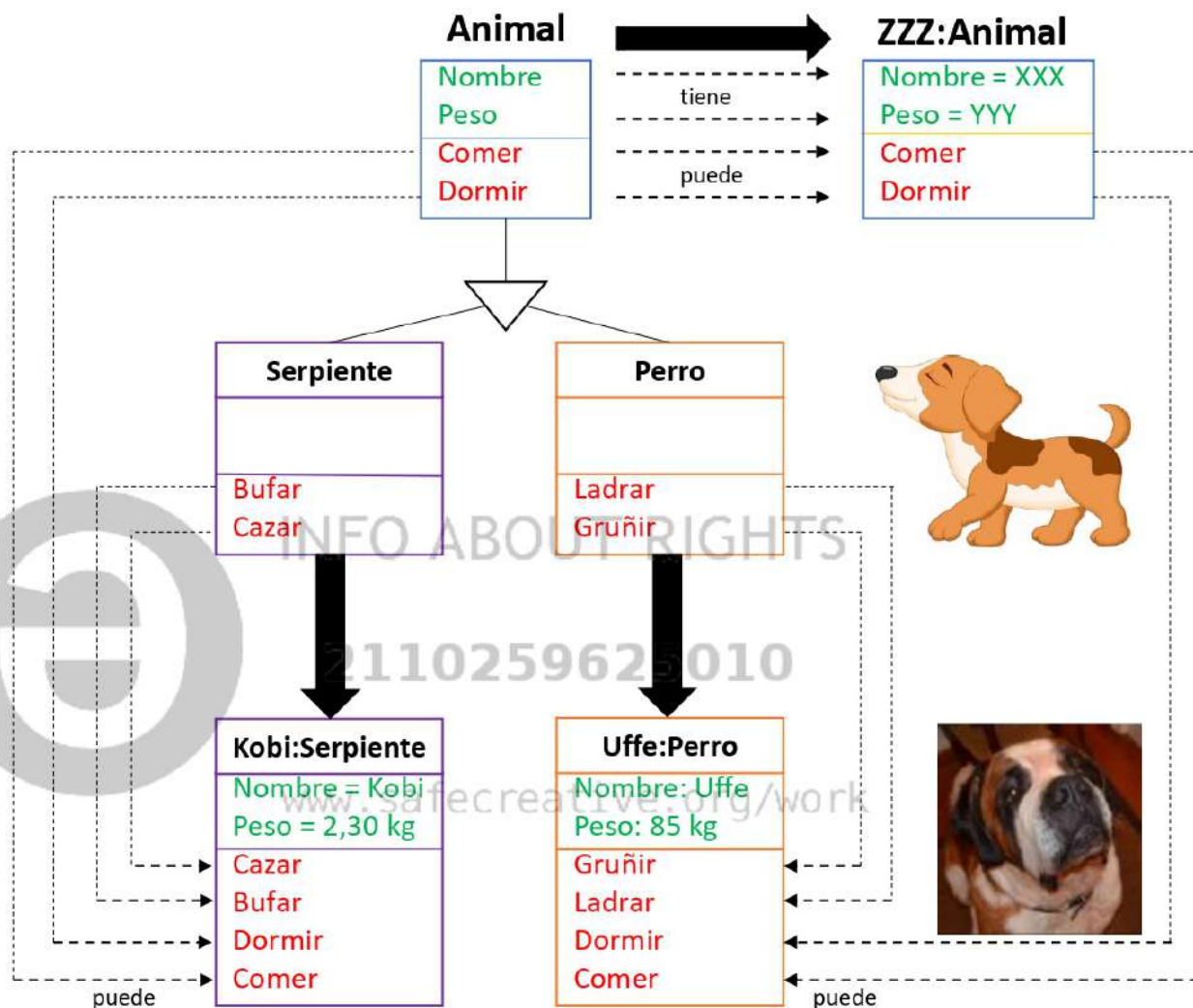
**HERENCIA** → Es la “*cualidad*” por la que una clase que descienda de otra herede (adquieran) las características de la clase superior





# JAVASCRIPT Y POO

## Ejemplo de Herencia



¿Cuáles son **Clases**?

- ✓ Animal
  - Serpiente
  - Perro

¿Cuáles son **Objetos**?

- ✓ Uffe
- ✓ Kobi

¿Cuáles son **Atributos**?

- ✓ Nombre
- ✓ Peso

¿Cuáles son **Métodos**?

- ✓ Comer, Dormir
- ✓ Bufar, Cazar
- ✓ Gruñir, Ladlar

## INFO ABOUT RIGHTS

2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

¿Cómo **creo** un **objeto literal**? →

```
let / var / const nombre_objeto = new Object ( );  
let / var / const nombre_objeto = { };
```

¿Cómo borro una propiedad de un objeto? → `delete objeto.propiedad;`

# JAVASCRIPT Y POO: USO OBJETOS Y MÉTODOS

## Ejemplo A):

“Crear un objeto llamado **punto** el cual se encuentre vacío, →

y posteriormente asignarle las propiedades relativas a sus coordenadas **X** e **Y**, donde **Y** es igual al doble de la coordenada **X**, siendo ésta el valor inicial de 10. →

Crear un método llamado **mostrar** en el objeto que al invocarlo muestre sus coordenadas de la forma ( **X** , **Y** ). →

Invoque dicho método y muestre las coordenadas de dicho objeto.” →

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>POO : Punto</title>
8 </head>
9 <body>
10  <h1>POO : Punto</h1>
11  <script type="text/javascript">
```

```
12    var punto = new Object();
```

```
13    punto.X=10;
14    punto.Y=punto.X*2;
```

```
15    punto.mostrar=function(){
16      return ` ( ${punto.X} , ${punto.Y} ) `;
17    }
```

```
18    document.write(punto.mostrar());
19  </script>
20 </body>
21 </html>
```



# JAVASCRIPT Y POO: USO OBJETOS Y MÉTODOS

## Ejemplo B):

“Crear un objeto llamado **punto** el cual se encuentre vacío, →

y posteriormente asignarle las propiedades relativas a sus coordenadas **X** e **Y**. →

Crear un método llamado **crear\_coordenadas** en el objeto, que al invocarlo necesite pasarle los parámetros **A**, **B** y que lo emplace en las coordenadas ( **A** , **B** ). →

Crear otro método llamado **mostrar\_coordenadas** del objeto, que al invocarlo nos muestre las coordenadas de dicho punto u objeto. →

Invoque al los métodos y emplace el punto en las coordenadas (100,200) y muéstrelo.” →

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>P00 : Punto2</title>
8 </head>
9 <body>
10   <h1>P00 : Punto2</h1>
11   <script type="text/javascript">
```

```
12     var punto = new Object();
```

```
13     punto.X=0;
14     punto.Y=0;
```

```
15     punto.crear_coordenadas=function(a,b){
16         punto.X=a;
17         punto.Y=b;
18     }
```

```
19     punto.mostrar_coordenadas=function(){
20         return ` ${punto.X} , ${punto.Y} `;
21     }
```

```
22     punto.crear_coordenadas(100,200);
23     document.write(punto.mostrar_coordenadas());
24 </script>
25 </body>
26 </html>
```

# JAVASCRIPT Y POO: OBJETO THIS

¿Qué es el **objeto this**? → Es una **palabra reservada** que me **permite referirme al objeto de manera genérica sin mentar su nombre**

Ejemplo

```
var punto={
  x:10,
  y:10,
  mostrar_coord:function(){
    return `( ${punto.x} , ${punto.y} )`;
  }
}
document.write(punto.mostrar_coord());
```

VS

```
var PUNTO={
  X:100,
  Y:100,
  mostrar_coord:function(){
    return `( ${this.X} , ${this.Y} )`;
  }
}
document.write(PUNTO.mostrar_coord());
```

¿Qué  
diferencias  
existen?

Tenemos que llamar a “punto” para mostrar a sus coordenadas, lo que **me limita en la funcionalidad**

Aquí, no nombro al objeto, lo que se traduce en una **mayor versatilidad**

# JAVASCRIPT Y POO: RECORRER PROPIEDADES DEL OBJETO

¿Cómo consulto todas las propiedades **objeto this**? → Con el bucle **for ... in**

## Ejemplo

Dado un array de objetos llamado “puntos” el cual contiene tres objetos que representan puntos espaciales. Se pide recorrer dicho array de objetos mostrando las coordenadas de cada uno de los objetos contenidos.

Nota:

- Punto 1 = (0 , 0 , 0)
- Punto 2 = (10 , 10 , 10)
- Punto 3 = (100 , 100 , 100)

```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Recorrer propiedades</title>
8  </head>
9  <body>
10   <h1>Recorrer propiedades</h1>
11   <script type="text/javascript">
12     var punto_1 = {x:0,y:0,z:0};
13     var punto_2 = {x:10,y:10,z:10};
14     var punto_3 = {x:100,y:100,z:100};
15     var puntos = [punto_1,punto_2,punto_3];
16     var c = 1;
17     for (let k of puntos){
18       console.log(`El punto ${c} tiene: `);
19       for (let m in k){
20         console.log(`Coordenada ${m}: ${k[m]}`);
21       }
22       console.log(`-----`);
23       c+=1;
24     }
25   </script>
26 </body>
27 </html>
```



# JAVASCRIPT Y POO: CONSTRUCTOR DE OBJETOS

¿Qué es un **constructor**? → Es una **función** que me permite **crear un nuevo objeto de un tipo en particular, a partir de parámetros que yo le paso a la función**

## Ejemplo

Crear un constructor llamado “**pointsmaker**” que me permita crear puntos a partir de dos parámetros, que representen sus coordenadas en el espacio.

Existen dos operadores interesantes que devuelven el tipo y/o pertenecen a un objeto:

➤ **typeof** → **typeof.NOMBRE\_OBJETO**

➤ **instanceof** → **OBJETO1 instanceof OBJETO2**

```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Constructor</title>
8  </head>
9  <body>
10   <h1>Constructor</h1>
11   <script type="text/javascript">
12     function pointsmaker(coor_x,coor_y){
13       this.x = coor_x;
14       this.y = coor_y;
15       this.mostrar=()=>` ${this.x} , ${this.y}`
16     }
17
18     var p1 = new pointsmaker(0,0);
19     var p2 = new pointsmaker(10,10);
20
21     document.write(p1.mostrar());
22     document.write('<br\\>')
23     document.write(p2.mostrar());
24   </script>
25 </body>
26 </html>
```

# JAVASCRIPT Y POO: PROTOTIPOS

¿Qué es un **prototipo**? → Es el “**molde**” que crea objetos, lo que en otros lenguajes se llama “**clase**”

La diferencia con el resto de lenguajes, es que **es dinámica**, es decir, **si cambio algo de dicho prototipo afecta a los objetos producidos** con dicho molde, clase o prototipo.

¿Cómo **accedo** al **prototipo** de un objeto ya creado? → Con la propiedad **\_\_proto\_\_**, por ejemplo:

Con el método **Object.getPrototypeOf(objeto)**, por ejemplo: **Object.getPrototypeOf(p1)**

**p1.\_\_proto\_\_**

INFO ABOUT RIGHTS  
2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

¿Cómo **modifico** el **prototipo**? → Basta con indicar **nombre\_prototipo.prototype** seguido de las propiedad y/o método a cambiar

# JAVASCRIPT Y POO: PROTOTIPOS

## Ejemplo:

Crea una función constructora de objetos que se llame “Punto”, que permita crear objetos que representen puntos en un espacio bidimensional. Además ha de contar con un método llamado “mostrar” que al invocarlo nos de sus cooredenadas (X,Y).

Crea una objeto llamado “a” con la función constructora “Punto”, el cual ha de emplazarse en el (10,70).

Define un nuevo método en la función constructora “Punto” (sin añadir rectificar líneas de código de dicha función constructora) que se llame “sumaXY” y que sume las coordenadas de dicho punto.

Define una nueva propiedad en la función constructora “Punto” que represente la coordenada “z” inicializada en cero.

Invoca al nuevo método llamado “sumaXY” y a la propiedad “z” presentes en el objeto “a”.

```
<script type="text/javascript">
  //Creo Función constructora
  function Punto(coorX,coorY){
    this.x = coorX;
    this.y = coorY;
    this.mostrar={()=>`${this.x} , ${this.y}`};
  }

```

```
//Creo un objeto a que será un pto en 0,0
var a = new Punto(10,70);

```

```
//Defino un nuevo método del prototipo
Punto.prototype.sumaXY=function(){
  return this.x + this.y;
}

```

```
//Defino una nueva propiedad del prototipo
Punto.prototype.z=0;

```

```
//Pruebo el método y la propiedad
document.write(a.sumaXY() + "<br/>" + a.z);
</script>

```



# JAVASCRIPT Y POO: NOTACIÓN JSON

¿Qué son los **JSON**? → Es el acrónimo de “*JavaScript Objects Notation*”, es decir, Notación de Objetos de JavaScript

Creado inicialmente para JS, actualmente **se considera un formato documental independiente** (No-SQL)



INFO ABOUT RIGHTS  
¿Qué **diferencias** existe entre **JS** y **JSON**?  
2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

JSON solo admite propiedades (no se pueden declarar métodos)  
JSON el nombre de las propiedades ha de ir entre comillas “ “

¿Qué **importancia** tiene los **JSON** en **JS**? → Mucha tanto que **se han creado dos métodos específicos**:

- **stringify** → Convierte a string un JSON
- **parse** → Recibe un JSON y lo evalúa sintácticamente para procesarlo

# JAVASCRIPT Y POO: NOTACIÓN JSON

Ejemplo:



```
<script type="text/javascript">
  const musico1 = {
    nombre_artistico:"SHO-HAI",
    nombre:"Sergio",
    apellidos:"Rodriguez Fernández",
    fecha_nacimiento:{
      dia:19,
      mes:02,
      año:1976
    },
    discos:['Doble vida','La última función']
  }
  document.write(JSON.stringify(musico1));
</script>
```

## JSON


```
{"nombre_artistico":"SHO-HAI","nombre":"Sergio","apellidos":"Rodriguez Fernández","fecha_nacimiento":{"dia":19,"mes":2,"año":1976},"discos":["Doble vida","La última función"]}
```

# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué son las **expresiones regulares**? → Son un sistema para **buscar**, **capturar** o **reemplazar texto** utilizando **patrones**

¿Cómo maneja JS las **expresiones regulares**? → Como un **objeto** de tipo **RegExp**

¿Cómo es la **sintaxis**? {  
/ expresión regular /flags  
**`new RegExp('expresión',flag)`**



Por favor,  
¡¡¡Ejemplooooo!!!

Crear la expresión regular de los NIF de España, sabiendo que estos números de identificación fiscal están constituidos por 8 números y una letra de la A-Z para los DNI, y en el caso de los NIE posee una letra delante que puede ser K, L, X, Y o Z

INFO ABOUT RIGHTS  
`var document = /[KLXYZ0-9][0-9]{8}[A-Z]{1}/;`

2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

`var document = new RegExp(' [KLXYZ0-9][0-9]{8}[A-Z]{1}');`



# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué es un **flag** de una **RegExp**? → Es un **parámetro** que permite **realizar concreciones** en la expresión regular

Flag	Booleano	Descripción
i	<code>.ignoreCase</code>	Ignora mayúsculas y minúsculas. Se suele denominar <b>insensible a mayús/minús</b> .
g	<code>.global</code>	Búsqueda global. Sigue buscando coincidencias en lugar de pararse al encontrar una.
m	<code>.multiline</code>	Multilínea. Permite a <code>^</code> y <code>\$</code> tratar los finales de línea <code>\r</code> o <code>\n</code> .
u	<code>.unicode</code>	Unicode. Interpreta el patrón como un código de una secuencia Unicode.
y	<code>.sticky</code>	Sticky. Busca sólo desde la posición indicada por <code>lastIndex</code> .

```
const r1 = /reg/;  
const r2 = /reg/i;  
const r3 = /reg/gi;
```

Cada una de estas **flags se pueden comprobar si están activas** desde Javascript **con su booleano asociado**, que es una propiedad de la expresión regular

```
const r = /reg/gi;  
  
r.global; // true  
r.ignoreCase; // true  
r.multiline; // false  
r.sticky; // false  
r.unicode; // false
```

# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué propiedades tiene **RegExp**? → Cada expresión regular creada, tiene unas propiedades definidas, donde podemos consultar ciertas características de la expresión regular en cuestión

También tiene unas propiedades de comprobación para saber si un flag determinado está activo o no

Propiedades	Descripción
<small>STRING</small> <b>.source</b>	Devuelve un string con la expresión regular original al crear el objeto ( <i>sin flags</i> ).
<small>STRING</small> <b>.flags</b>	Devuelve un string con los flags activados en la expresión regular.
<small>NUMBER</small> <b>.lastIndex</b>	Devuelve la posición donde se encontró una ocurrencia en la última búsqueda.
<small>BOOLEAN</small> <b>.global</b>	Comprueba si el flag <b>g</b> está activo en la expresión regular.
<small>BOOLEAN</small> <b>.ignoreCase</b>	Comprueba si el flag <b>i</b> está activo en la expresión regular.
<small>BOOLEAN</small> <b>.multiline</b>	Comprueba si el flag <b>m</b> está activo en la expresión regular.
<small>BOOLEAN</small> <b>.unicode</b>	Comprueba si el flag <b>u</b> está activo en la expresión regular.
<small>BOOLEAN</small> <b>.sticky</b>	Comprueba si el flag <b>y</b> está activo en la expresión regular.

```
const r = /reg/gi;

r.source; // 'reg'
r.flags; // 'ig'

r.flags.includes("g"); // true (equivalente a r.global)
r.flags.includes("u"); // false (equivalente a r.unicode)
```

# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué caracteres especiales tiene las **RegExp**? → Existen ciertos caracteres que tienen un significado especial:

Caracter especial	Descripción
.	Comodín, cualquier caracter.
\	Invierte el significado de un carácter. Si es especial, lo escapa. Si no, lo vuelve especial.
\t	Caracter especial. Tabulador.
\r	Caracter especial. Retorno de carro. A menudo denominado <b>CR</b> .
\n	Caracter especial. Nueva línea. A menudo denominado «line feed» o <b>LF</b> .

Caracter especial	Descripción
[ ]	Rango de caracteres. Cualquiera de los caracteres del interior de los corchetes.
[^ ]	No exista cualquiera de los caracteres del interior de los corchetes.
	Establece una alternativa: lo que está a la izquierda o lo que está a la derecha.

Caracter especial	Alternativa	Descripción
[0-9]	\d	Un dígito del 0 al 9.
[^0-9]	\D	No exista un dígito del 0 al 9.
[A-Z]		Letra mayúscula de la <b>A</b> a la <b>Z</b> . Excluye ñ o letras acentuadas.
[a-z]		Letra minúscula de la <b>a</b> a la <b>z</b> . Excluye ñ o letras acentuadas.
[A-Za-z0-9]	\w	Carácter alfanumérico (letra mayúscula, minúscula o dígito).
[^A-Za-z0-9]	\W	No exista carácter alfanumérico (letra mayúscula, minúscula o dígito).
[ \t\r\n\f]	\s	Carácter de espacio en blanco (espacio, <b>TAB</b> , <b>CR</b> , <b>LF</b> o <b>FF</b> ).
[^ \t\r\n\f]	\S	No exista carácter de espacio en blanco (espacio, <b>TAB</b> , <b>CR</b> , <b>LF</b> o <b>FF</b> ).
	\xN	Carácter hexadecimal número <b>N</b> .
	\uN	Carácter Unicode número <b>N</b> .

¿Qué métodos tiene **RegExp**? → Varios métodos, para saber si la expresión regular “casa” con el texto propuesto

Método	Descripción
<b>BOOLEAN</b> test(str)	Comprueba si la expresión regular «casa» con el texto <b>str</b> pasado por parámetro.
<b>ARRAY</b> exec(str)	Ejecuta una búsqueda de patrón en el texto <b>str</b> . Devuelve un array con las capturas.



# JAVASCRIPT Y POO: EXPRESIONES REGULARES

Ejemplo:

```
const r = /.a.o/i;  
r.test("gato"); // true  
r.test("pato"); // true  
r.test("perro"); // false  
r.test("DATO"); // true (el flag i permite mayús/minús)
```

```
// Buscamos RegExp que encaje con "Manz"  
/M.nz/.test("Manz"); // true  
/M.nz/.test("manz"); // false (La «M» debe ser mayúscula)  
/M.nz/i.test("manz"); // true (Ignoramos mayús/minús con el flag «i»)  
  
// Buscamos RegExp que encaje con "A."  
/A./.test("A."); // true (Ojo, nos da true, pero el punto es comodín)  
/A./.test("Ab"); // true (Nos da true con cualquier cosa)  
/A\./.test("A."); // true (Solución correcta)  
/A\./.test("Ab"); // false (Ahora no deja pasar algo que no sea punto)
```

```
const r = /[aeiou]/i; // RegExp que acepta vocales (mayús/minús)  
r.test("a"); // true (es vocal)  
r.test("E"); // true (es vocal, y tiene flag «i»)  
r.test("t"); // false (no es vocal)  
  
const r = /^[^aeiou]/i; // RegExp que acepta lo que no sea vocal (mayús/minús)  
r.test("a"); // false  
r.test("E"); // false  
r.test("T"); // true  
r.test("m"); // true  
  
const r = /casa|cama/; // RegExp que acepta la primera o la segunda opción  
r.test("casa"); // true  
r.test("cama"); // true  
r.test("capa"); // false
```

# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué son las **anclas** de las **RegExp**? → Recurso que permite **delimitar los patrones de búsqueda** e indicar si empiezan o terminan por caracteres concretos

Caracter especial	Descripción
<b>^</b>	Ancla. Delimita el inicio del patrón. Significa <b>empieza por</b> .
<b>\$</b>	Ancla. Delimita el final del patrón. Significa <b>acaba en</b> .
<b>\b</b>	Posición de una palabra limitada por espacios, puntuación o inicio/final.
<b>\B</b>	Opuesta al anterior. Posición entre 2 caracteres alfanuméricos o no alfanuméricos.

```
const r = /^mas/i;

r.test("Formas"); // false (no empieza por "mas")
r.test("Master"); // true
r.test("Masticar"); // true

const r = /do$/i;

r.test("Vívido"); // true
r.test("Dominó"); // false
```

```
const r = /fo\b/;

r.test("Esto es un párrafo de texto."); // true (tras "fo" hay un límite de palabra)
r.test("Esto es un párrafo."); // true (tras "fo" hay un signo de puntuación)
r.test("Un círculo es una forma."); // false (tras "fo" sigue la palabra)
r.test("Frase que termina en fo"); // true (tras "fo" termina el string)
```

# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué son los **cuantificadores** de las **RegExp**? → Recurso que permite **indicar cuántas veces se puede repetir el carácter inmediatamente anterior**

Carácter especial	Descripción
*	El carácter anterior puede aparecer <b>0</b> o más veces.
+	El carácter anterior puede aparecer <b>1</b> o más veces.
?	El carácter anterior puede aparecer o no aparecer.
{n}	El carácter anterior aparece <b>n</b> veces.
{n,}	El carácter anterior aparece <b>n</b> o más veces.
{n,m}	El carácter anterior aparece de <b>n</b> a <b>m</b> veces.

```
const r = /disparos?/i;

r.test("Escuché disparos en la habitación."); // true
r.test("Efectuó un disparo al sujeto."); // true
r.test("La pistola era de agua."); // false
```

```
// Un número formado de 2 dígitos (del 0 al 9)
const r = /[0-9]{2}/;

r.test(42); // true
r.test(88); // true
r.test(1); // false
r.test(100); // true
```

```
const r = /^[0-9]{2}$/;

r.test(4); // false
r.test(55); // true
r.test(100); // false
```

```
const r = /^[0-9]{3,}$/;

r.test(33); // false
r.test(4923); // true
```

```
const r = /^[0-9]{2,5}$/;

r.test(2); // false
r.test(444); // true
r.test(543213); // false
```

```
// 'a' aparece 0 o más veces en el string
const r = /a*/;

r.test(""); // true ('a' aparece 0 veces)
r.test("a"); // true ('a' aparece 1 vez)
r.test("aa"); // true ('a' aparece 2 veces)
r.test("aba"); // true ('a' aparece 2 veces)
r.test("bbb"); // true ('a' aparece 0 veces)
```

```
// 'a' aparece 1 o más veces (equivalente a /aa*/)
const r = /a+/;

r.test(""); // false ('a' aparece 0 veces)
r.test("a"); // true ('a' aparece 1 vez)
r.test("aa"); // true ('a' aparece 2 veces)
r.test("aba"); // true ('a' aparece 2 veces)
r.test("bbb"); // false ('a' aparece 0 veces)
```



# JAVASCRIPT Y POO: EXPRESIONES REGULARES

¿Qué son las **captura de patrones**? → Es la **captura de texto** mediante la parentización, **por medio** del método **exec(str)** el cual **devuelve un array con las capturas realizadas**.

Carácter especial	Descripción
(x)	El patrón incluido dentro de paréntesis se captura y se guarda en \$1 o sucesivos.
(?:x)	Si incluimos <b>?:</b> al inicio del contenido de los paréntesis, evitamos capturar ese patrón.
x(?:y)	Busca sólo si <b>x</b> está seguido de <b>y</b> .
x(?:!y)	Busca sólo si <b>x</b> no está seguido de <b>y</b> .

www.safecreative.org/work

```
// RegExp que captura palabras de 3 letras.
const r = /\b([a-z]{3})\b/gi;
const str = "Hola a todos, amigos míos. Esto es una prueba que permitirá ver que ocurre."

r.global; // true (el flag global está activado)

r.exec(str); // ['una', 'una']      index: 35
r.exec(str); // ['que', 'que']      index: 46
r.exec(str); // ['ver', 'ver']      index: 60
r.exec(str); // ['que', 'que']      index: 64
r.exec(str); // null
```

```
const r = /\b([a-z]{3})\b/gi;
const str = "Hola a todos, amigos míos. Esto es una prueba que permitirá ver que ocurre."

str.match(r); // Devuelve ['una', 'que', 'ver', 'que']

const r = /\b([0-9]+\.[0-9]+\.[0-9]+\b)/g;
const str = "v1.0.21";

str.match(r); // Devuelve ['v1.0.21', '1', '0', '21']

const daenerys = "Javascript es un gran lenguaje";
daenerys.replace(/[aeou]/g, "i"); // 'Jiviscript is in grin lingiiji'
```

# EJERCICIOS



INFO ABOUT RIGHTS

2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

# PROPUESTA EJERCICIO DE CLASE

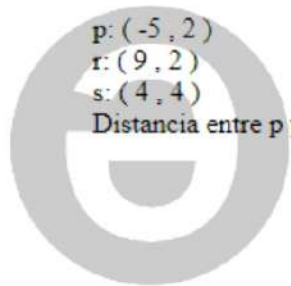


Realiza un aplicativo web en HTML5 y JS que permita crear objetos que representen puntos. De ellos necesitamos:

- Que tengan exclusivamente dos propiedades: “x” e “y”, las cuales servirán para representar las coordenadas del punto.
- Que los puntos sean creados mediante una función constructora que emplee dos parámetros que representan las coordenadas de dicho punto, y en caso de pasarle cualquier variable para las coordenadas no fuera un número lo sitúe en el cero.
- Disponer de un método llamado “cambiar” que se le pase dos coordenadas y cambie la situación de dicho objeto punto.
- Disponer de un método llamado “copia” que retorne una copia del objeto.
- Disponer de un método llamado “suma” que reciba un segundo objeto (un punto) y devuelva un tercer punto que sea la suma de las coordenadas.
- Disponer de un método llamado “obtenerDistancia” que reciba un objeto punto y nos devuelva la distancia en línea recta de ambos punto.
- Disponer de un método llamado “toString” que retorne un texto con las coordenadas del punto en el siguiente formato: ( x , y )



# RESOLUCIÓN PROPUESTA EJERCICIO DE CLASE



p: (-5, 2)  
r: (9, 2)  
s: (4, 4)

Distancia entre p y q: 12.083045973594572

INFO ABOUT RIGHTS

2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

```
<script type="text/javascript">
  function Punto(x, y) {
    this.x = x;
    this.y = y;

    this.cambiar = (x, y) => {
      this.x = x;
      this.y = y;
    };

    this.copia = () => (new Punto(this.x, this.y));
    this.suma = (p2) => (new Punto(this.x + p2.x, this.y + p2.y));
    this.toString = () => (` ${this.x} , ${this.y} `);
    this.obtenerDistancia = function (p2) {
      return Math.sqrt(
        Math.pow(Math.abs(this.x - p2.x), 2) +
        Math.pow(Math.abs(this.y - p2.y), 2)
      );
    };
  };

  //Prueba de los métodos y construcciones
  var p = new Punto(1, 2);
  var q = new Punto(6, -3);
  //modificamos coordenadas de p
  p.cambiar(-5, 2);
  //r será una copia de p
  var r = p.copia();
  r.x = 9;
  //comprobamos que r y p son puntos distintos
  document.write("p: " + p.toString() + "<br/>");
  document.write("r: " + r.toString() + "<br/>");

  //s es un nuevo punto que toma la suma de coordenadas
  //de p y r
  var s = p.suma(r);
  document.write("s: " + s.toString() + "<br/>");
  //Obtener distancia entre p y q
  document.write("Distancia entre p y q: " + p.obtenerDistancia(q) + "<br/>");
</script>
```

# PROPUESTA EJERCICIO DE CLASE



Los objetos de tipo Array poseen numerosos métodos que resultan muy útiles. Considero que es interesante añadir un método a todos los arrays en JavaScript que permita realizar el cálculo de la media aritmética de sus elementos. Es por tanto, que se realiza esta propuesta “indecente” de modificar el prototipo de los arrays para añadir la mejor de las funcionalidades en el análisis de datos.

INFO ABOUT RIGHTS

2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

# RESOLUCIÓN PROPUESTA EJERCICIO DE CLASE

## Prototitpo Array

El array es: [1,2,3,4,5,6]  
La media del array es: 3.5

El segundo array es: [10,13,16,21,28,19,45,32,20]  
La media del array es: 22.666666666666668

```
1  <!DOCTYPE html>
2  <html lang="es">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta http-equiv="X-UA-Compatible" content="IE=edge">
7    <meta name="viewport" content="width=device-width, initial-scale=1.0">
8    <title>Prototipo Array</title>
9  </head>
10
11 <body>
12   <h1>Prototitpo Array</h1>
13   <script type="text/javascript">
14     Array.prototype.media = function () {
15       let suma = this.reduce((anterior, actual) => (anterior + actual));
16       let tamaño = this.length;
17       return suma / tamaño;
18     }
19
20     let a1 = [1, 2, 3, 4, 5, 6];
21     document.write("El array es: ["+a1+"]<br/>");
22     document.write("La media del array es: "+a1.media()+"<br/>");
23     let a2 = [10, 13, 16, 21, 28, 19, 45, 32, 20];
24     document.write("<br/>El segundo array es: ["+a2+"]<br/>");
25     document.write("La media del array es: "+a2.media()+"<br/>");
26   </script>
27 </body>
28
29 </html>
```



# PROPUESTA EJERCICIO DE CLASE



Desarrolla una función constructora que permita crear objetos que nos sirvan para representar las figuras de ordenadores de sobremesa en el juego de los Sims los cuales poseen como propiedades:

- Marca (es de tipo texto)
- Modelo (es de tipo texto)
- Memoria RAM (es de tipo numérico y representa los GB de capacidad)
- Capacidad del disco duro (es de tipo numérico que indica los TB de capacidad)
- Pulgadas de pantalla (es de tipo numérico que indica los inches de la diagonal)

Como métodos dispondrá:

- toString → Devuelve en forma de tabla los detalles del objeto ordenador.

Al crear cualquier objeto con la función ordenador se pueden indicar todos los valores, pero por defecto (sin ser obligado indicarlos) se toman como valores 17 pulgadas, 8GB de RAM y 3TB de disco duro. Exclusivamente las propiedades necesarias serán la marca y el modelo.

Crea otro constructor que represente los portátiles, los cuales hereden todo de los ordenadores de sobremesa, pero añade la propiedad autonomía que es un número que expresa las horas de vida de la batería. Estos objetos se construyen igual que los sobremesa pero pudiendo añadir la autonomía (por defecto, 4 horas). Por defecto, los portátiles son de 15 pulgadas y discos de 1 TB.

# RESOLUCIÓN PROPUESTA EJERCICIO DE CLASE

## Los Sims

Marca	HP
Modelo	EliteDisplay
Ram	8 GB
SSD	3 TB
Pantalla	23 inch

Marca	Apple
Modelo	Macbook Air
Ram	8 GB
SSD	1 TB
Pantalla	13 inch

Autonomía: 8 horas

Marca	Dell
Modelo	Inspiron AIO
Ram	8 GB
SSD	3 TB
Pantalla	17 inch

Marca	Acer
Modelo	Aspire
Ram	8 GB
SSD	1 TB
Pantalla	15 inch

Autonomía: 4 horas

```
<script type="text/javascript">
function Ordenador(marca, modelo, ram = 8, disco = 3, pulgadas = 17) {
    this.marca = marca;
    this.modelo = modelo;
    this.ram = ram;
    this.disco = disco;
    this.pulgadas = pulgadas;
    this.toString = function () {
        return `<table>
            <tr><td>Marca</td><td>${this.marca}</td></tr>
            <tr><td>Modelo</td><td>${this.modelo}</td></tr>
            <tr><td>Ram</td><td>${this.ram} GB</td></tr>
            <tr><td>SSD</td><td>${this.disco} TB</td></tr>
            <tr><td>Pantalla</td><td>${this.pulgadas} inch</td></tr>
        </table>`;
    }
}

function Portatil(marca, modelo, ram = 8, disco = 1, pulgadas = 15, autonomia = 4) {
    this.__proto__ = new Ordenador(marca, modelo, ram, disco, pulgadas);
    this.autonomia = autonomia;
    this.toString = function () {
        return `${this.__proto__.toString()}
            <p><small>Autonomía: ${this.autonomia} horas</small></p>`;
    }
}

var o1 = new Ordenador("HP", "EliteDisplay", 8, 3, 23,);
var o2 = new Ordenador("Dell", "Inspiron AIO");
var p1 = new Portatil("Apple", "Macbook Air", 8, 1, 13, 8);
var p2 = new Portatil("Acer", "Aspire");
document.write(o1.toString() + "<br/>");
document.write(o2.toString() + "<br/>");
document.write(p1.toString() + "<br/>");
document.write(p2.toString() + "<br/>");
</script>
```



# PROPUESTA EJERCICIO DE CLASE



Realiza un aplicativo web en HTML5 y JS que pida una fecha en formato *día/mes/año*. En caso de existir un fallo en el formato ha de pedir de nuevo que meta la fecha. No se comprobará de nuevo la fecha, pero si que la nueva fecha posea dos dígitos para el día, dos para el mes y dos para el año. Se pedirá una segunda fecha, y se validará también. Finalmente se crearán dos fechas de tipo Date, y se mostrará la diferencia entre ambas.

[www.safecreative.org/work](http://www.safecreative.org/work)



Román-Herrera, J.C.

# RESOLUCIÓN PROPUESTA EJERCICIO DE CLASE



INFO ABOUT RIGHTS

2110259625010

[www.safecreative.org/work](http://www.safecreative.org/work)

```
<script type="text/javascript">
function getFecha(fechaString) {
  const VAL_FECHA_G = /(\d{2})\/(\d{2})\/(\d{4})/;
  let res = VAL_FECHA_G.exec(fechaString);
  if (res == null)
    return null;
  else {
    //exec extraerá en un array cada apartado entre
    //paréntesis. Empezando por el elemento 1
    // Así en el 1 tendremos el día, en el 2 el mes
    // y en el 3 el año
    //Los objetos Date requieren pasar este orden al revés
    //Al mes hay que restarle 1
    return new Date(res[3], res[2] - 1, res[1]);
  }
}

const VAL_FECHA = /\d{2}\/\d{2}\/\d{4}/;
var fecha1;
var fecha2;
do {
  fecha1 = prompt("Escriba la primera fecha (formato dd/mm/yyyy)");
} while (VAL_FECHA.test(fecha1) == false);

do {
  fecha2 = prompt("Escriba la segunda fecha (formato dd/mm/yyyy)");
} while (VAL_FECHA.test(fecha2) == false);

//getFecha devuelve una fecha, de esa fecha usamos el
//método getTime para obtener los milisegundos desde 1970
//que representa esa fecha
let msFecha1 = getFecha(fecha1).getTime();
let msFecha2 = getFecha(fecha2).getTime();
//restamos los milisegundos y les pasamos a días
let diferencia = (msFecha1 - msFecha2) / (1000 * 60 * 60 * 24);
document.write(`La diferencia en días de esa fecha es ${diferencia}`);
</script>
```

 safe creative  
2110019392312  
INFO ABOUT RIGHTS

# PROPUESTA EJERCICIO DE CLASE



Realiza un aplicativo web en HTML5 y JS que permite fichar a un trabajador. El tiempo que este tiene para realizar esta obligación es durante la jornada laboral de 08:00 a 14:00 de L-V, durante este periodo tendrá que aparecer un mensaje que diga, “Fiche aquí y ahora”. En caso de que se intente acceder al aplicativo fuera de su jornada laboral, ha de mostrar un mensaje que diga “No puedes fichar ahora, inténtelo más tarde”

creativecommons.org/work

# RESOLUCIÓN PROPUESTA EJERCICIO DE CLASE

```
1  <!DOCTYPE html>
2  <html lang="es">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta http-equiv="X-UA-Compatible" content="IE=edge">
7    <meta name="viewport" content="width=device-width, initial-scale=1.0">
8    <title>Fichar</title>
9  </head>
10
11 <body>
12   <script>
13     let fecha = new Date();
14     let hora = fecha.getHours();
15     let dia = fecha.getDay();
16     if (hora >= 8 && hora <= 14 && (dia == 6 || dia == 0)) {
17       document.write("<h1>FICHAR</h1>");
18       document.write("<p>Fiche aquí y ahora</p>");
19     }
20     else {
21       document.write("<h1>FICHAR</h1>");
22       document.write("<p>No puedes fichar ahora, intentelo más tarde</p>");
23     }
24   </script>
25 </body>
26
27 </html>
```



