

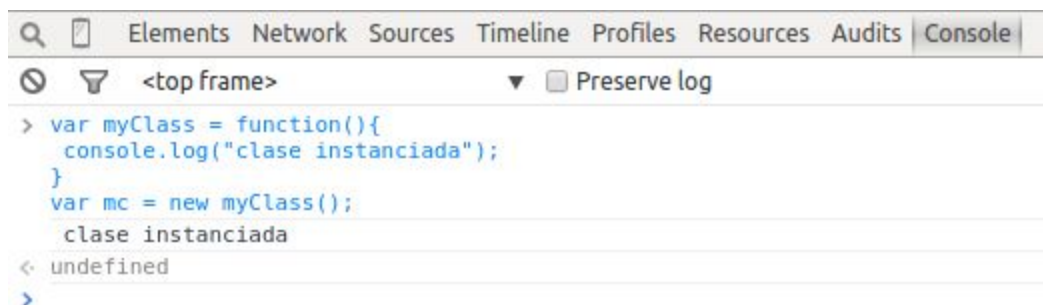
[Class](#)[Prototype](#)[This](#)[Referencias](#)[Extendiendo.](#)[JSON](#)[Delete](#)[Referencia circular.](#)[Object](#)[Métodos](#)[Ejercicios](#)[\\$Class](#)[Union](#)[Solucion](#)[Test del capítulo.](#)

## Class

En JavaScript las clases tienen diferencia con respecto a lenguajes como Java, PHP, C++ o C#. Si recordamos los capítulos anteriores, en JavaScript la forma de obtener una "instancia" es clonar un objeto preexistente, y las funciones son objetos. Bajo estos dos conceptos, una instancia es el resultado de clonar un objeto del tipo "function". Para clonar un objeto usamos el operador "new".

Las clases en JavaScript no tienen constructores propiamente dicho ya que el cuerpo de la función es el constructor en sí.

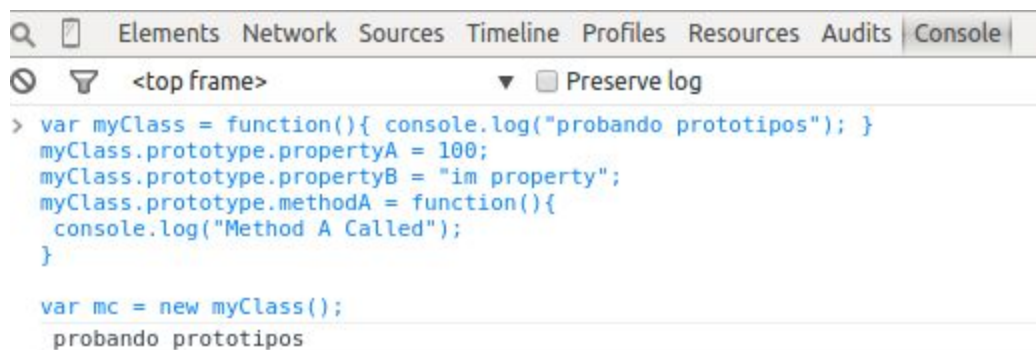
```
var myClass = function(){  
  console.log("clase instanciada");  
}  
var mc = new myClass();
```



## Prototype

Los prototipos son propiedades de la propiedad "prototype" que se transformarán en propiedades de todas las instancias de una función. Todos los objetos en JavaScript tienen prototipos.

```
var myClass = function(){}
myClass.prototype.propertyA = 100;
myClass.prototype.propertyB = "im property";
myClass.prototype.methodA = function(){
  console.log("Method A Called");
}
```



En el ejemplo anterior hemos creado una clase a la cual le asignamos propiedades y métodos usando el objeto prototype (que es que una propiedad del tipo Object).

La forma de acceder a las propiedades o métodos de los objetos es por medio del operador "." o poner el nombre entre []

Object.property  
Object["property"]

## This

"this" representa el contexto de la función (o clase), dando acceso a sus propiedades y métodos. Puede ser usado tanto para declarar como capturar valores.

Dentro de la definición de una clase podemos crear propiedades y métodos gracias al operador this.

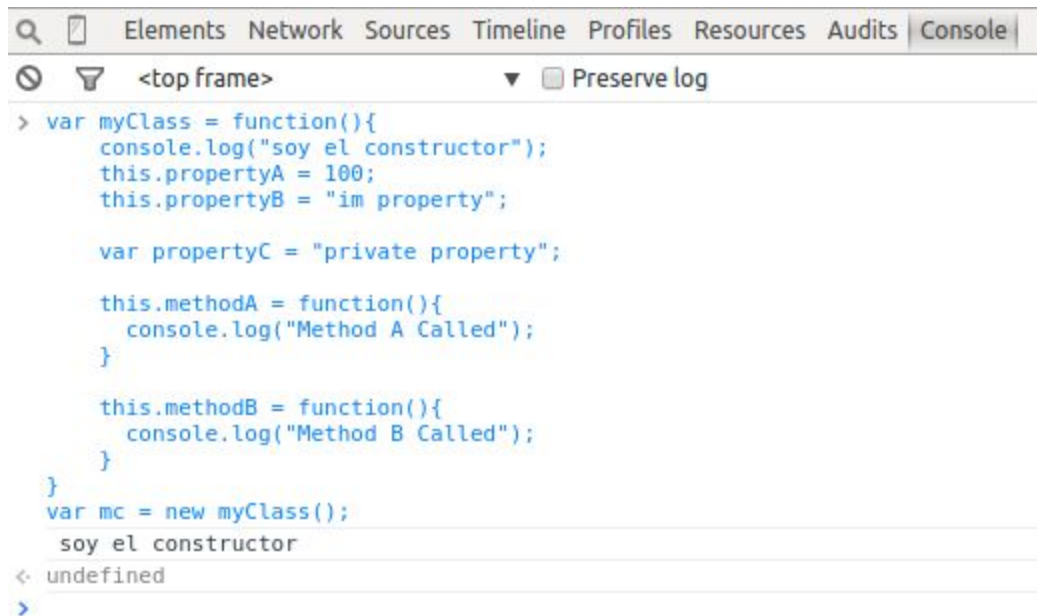
```
var myClass = function(){
  this.propertyA = 100;
  this.propertyB = "im property";

  var propertyC = "private property";

  this.methodA = function(){
```

```
        console.log("Method A Called");
    }

    this.methodB = function(){
        console.log("Method B Called");
    }
}
var mc = new myClass();
```



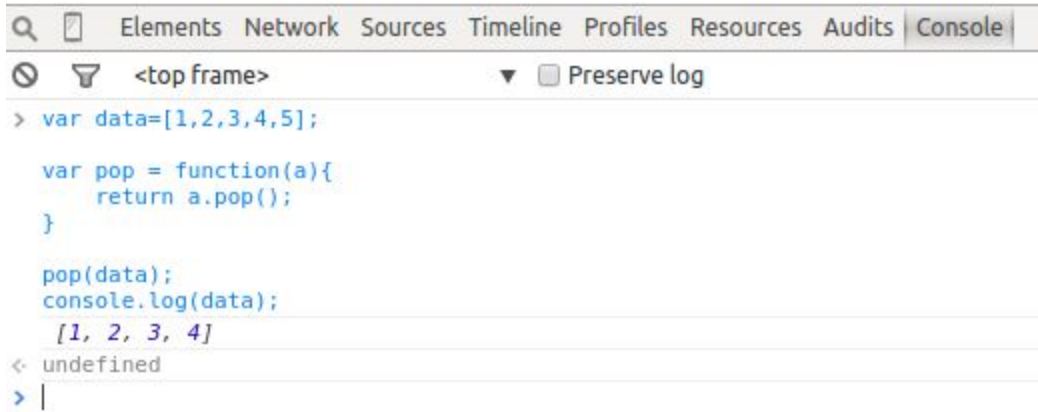
## Referencias

Los Array y Objetos en JavaScript se pasan como referencia en las funciones. Esto quiere decir que si pasamos un objeto definido como argumento de función, todas las operaciones que realizemos sobre el objeto dentro del cuerpo de la función impactarán directamente sobre el objeto inicial.

```
var data=[1,2,3,4,5];

var pop = function(a){
    return a.pop();
}
```

```
pop(data);
console.log(data);
```



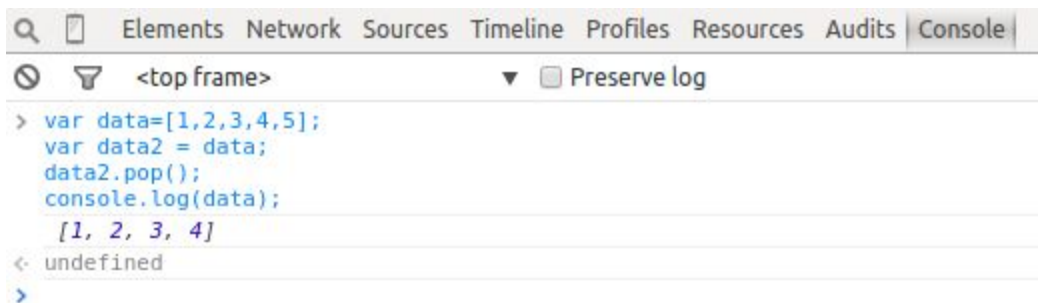
```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log
> var data=[1,2,3,4,5];

var pop = function(a){
  return a.pop();
}

pop(data);
console.log(data);
[1, 2, 3, 4]
< undefined
> |
```

Lo mismo ocurre cuando asignamos variables a partir de arrays u objeto existentes, la nueva variable no es una copia, sino una referencia.

```
var data=[1,2,3,4,5];
var data2 = data;
data2.pop();
console.log(data);
```



```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log
> var data=[1,2,3,4,5];
var data2 = data;
data2.pop();
console.log(data);
[1, 2, 3, 4]
< undefined
>
```

## Extendiendo.

Las instancias en JavaScript parte de la clonación de un objeto (del tipo function normalmente). Cada objeto tiene un prototipo definido, el cual hereda la nueva instancia.

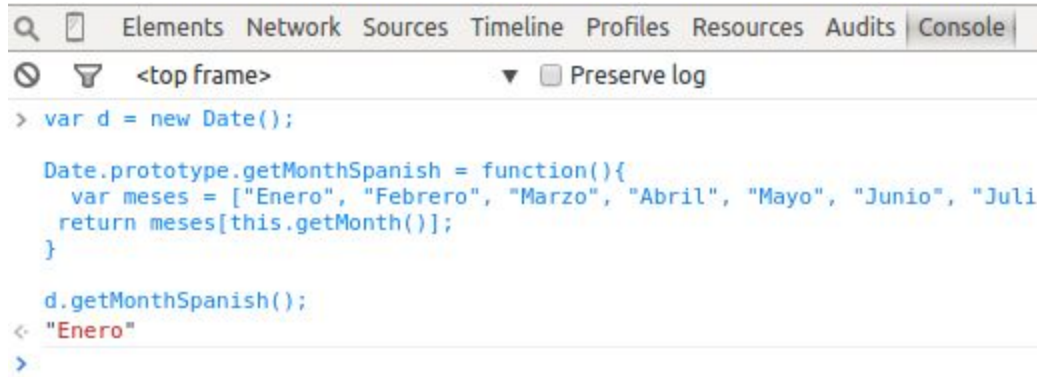
La instancia hace una referencia al prototype de su función padre, por lo cual si agregamos nuevas propiedades y métodos a la clase padre (a su prototype) los mismo quedarán disponibles en sus instancias (hijos).

Esto lo podemos hacer con cualquier objeto, tanto nativos como los creados por el usuario.

```
var d = new Date();

Date.prototype.getMonthSpanish = function(){
    var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto",
    "Septiembre", "Octubre", "Noviembre", "Diciembre"];
    return meses[this.getMonth()];
}

d.getMonthSpanish();
```



## JSON

JavaScript Object Notation es la representación de objetos en javascript. Esta representación es totalmente válida y ejecutable por cualquier entorno JS (navegadores, Node.js, etc.) y es empleada para definir objetos, con sus propiedades y métodos.

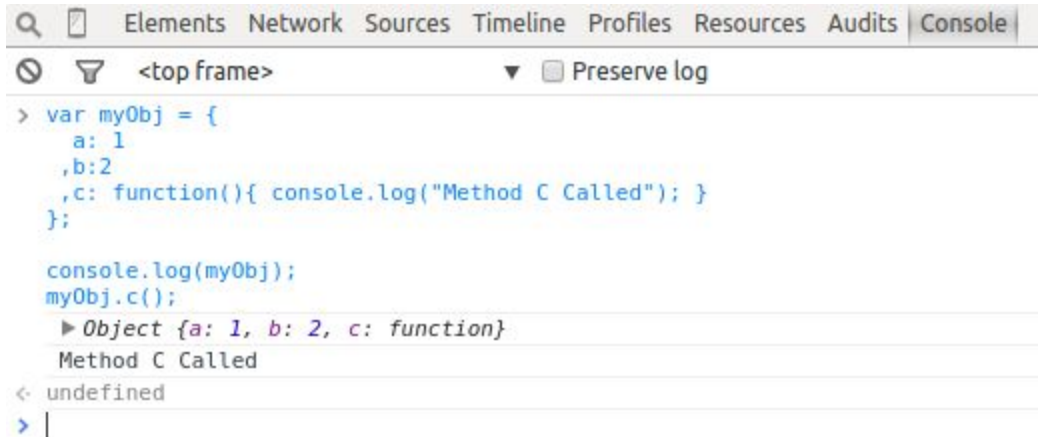
Los objetos definidos en JSON no se pueden clonar ya que no son funciones.

La forma de definir un json es por medio de los caracteres "{}" y es un json válido un objeto vacío.

Las propiedades son separadas entre sí por coma ",". Los valores y las propiedades se separan mediante ":". Las propiedades no deben terminar nunca con ";".

```
var myObj = {
  a: 1
  ,b:2
  ,c: function(){ console.log("Method C Called"); }
};

console.log(myObj);
myObj.c();
```



```
> var myObj = {
  a: 1
  ,b:2
  ,c: function(){ console.log("Method C Called"); }
};

console.log(myObj);
myObj.c();
▶ Object {a: 1, b: 2, c: function}
Method C Called
< undefined
> |
```

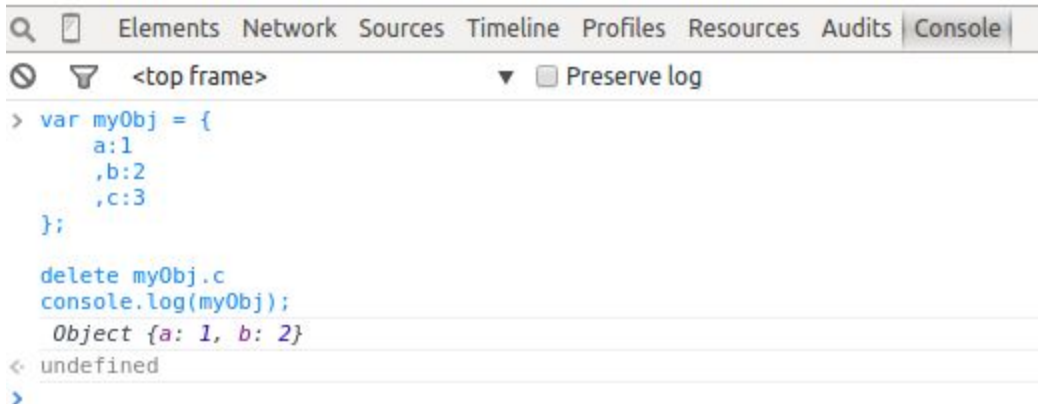
Las propiedades de los objetos pueden tener como valor cualquier objeto. En el caso de "c" su valor es una función, es lo que conocemos como "método".

## Delete

Para eliminar una propiedad de un objeto podemos usar la palabra reservada delete.

```
var myObj = {
  a:1
  ,b:2
  ,c:3
};

delete myObj.c
console.log(myObj);
```



```
> var myObj = {
  a:1
  ,b:2
  ,c:3
};

delete myObj.c
console.log(myObj);
Object {a: 1, b: 2}
< undefined
>
```

## Referencia circular.

Una referencia circular es cuando una propiedad tiene como valor el objeto contenedor, por lo tanto iteramos en forma infinita sobre las propiedades del objeto.

```
var cir = {  
  a:1  
  ,b:2  
};  
cir.c = cir;
```



## Object

Hasta el momento hemos visto que las clases en JavaScript son clones de objetos, pero no hemos visto al objeto padre de todo esto, al objeto Object.

El objeto Object contiene propiedades y métodos que nos permiten operar sobre los demás objetos. Todos los objetos en JavaScript heredan el prototype de Object (decienden de object).

## Métodos

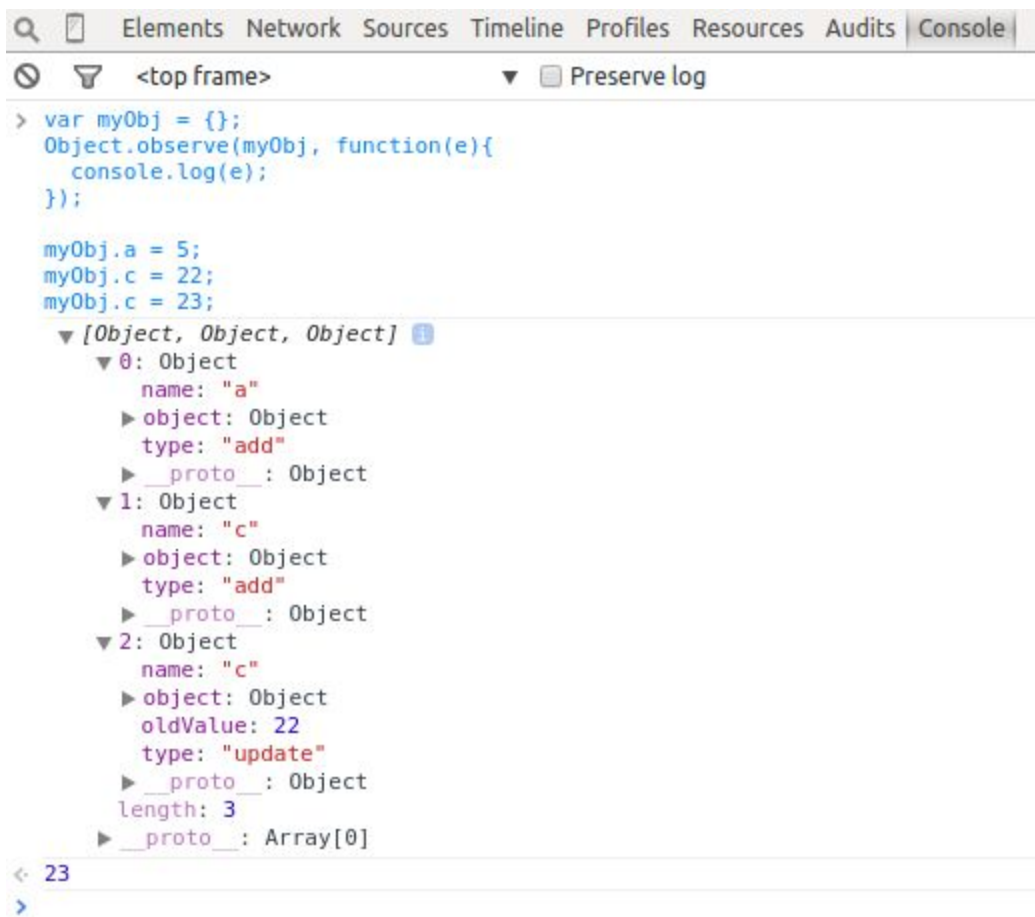
Método / Propiedad	Descripción
create(null)	Crea un objeto vacío.
freeze(o)	Impide que las propiedades puedan ser modificadas.
isFrozen(o)	Determina si un objeto ha sido freezado.
keys(o)	Retorna un array con todas las keys del objeto pasado como argumento.

observe(obj, func)

Observamos en forma async si algo cambia en los objetos, cada vez que cambia se detona la func asociada (callback). El callback recibe como argumento un objeto con la descripción.

```
var myObj = {};  
Object.observe(myObj, function(e){  
  console.log(e);  
});
```

```
myObj.a = 5;  
myObj.c = 22;  
myObj.c = 23;
```



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The code being executed is:

```
> var myObj = {};  
Object.observe(myObj, function(e){  
  console.log(e);  
});  
  
myObj.a = 5;  
myObj.c = 22;  
myObj.c = 23;
```

The console output shows an array of three objects: `[Object, Object, Object]`. Each object has a `name` property and an `object` property (which is the `myObj` object). The first two objects have a `type` of `"add"`, and the third object has a `type` of `"update"` and an `oldValue` of `22`. The `length` of the array is 3.



## Ejercicios

### \$Class

Crear una clase llamada \$Class que tenga un método "convert", el cual recibe como argumento un Object (json) y devuelve una función.

La función que devolverá deberá tener las mismas propiedades y métodos que el objeto pasado como argumento al método convert de \$Class.

```
var clon = new $Class().convert({
  name: "John"
, hi: function(){
  console.log("hi ", this.name);
}
, bye: function(){
  console.log("bye ", this.name);
}
});

new clon().hi();
```

Para resolver este ejercicio es importante recordar de que manera se puede acceder a una propiedad (operadores . y []). También es necesario el uso de métodos del objeto Object.

### Union

A la clase \$Class agregarle un método llamada "union" el cual recibe como argumentos N cantidad de objetos, retornando finalmente un nuevo objetos con todas las propiedades y métodos de los objetos pasados como argumentos.

```
new $Class().union(
{a:1,b:2}
,{c: function(){} }
,{d:"hi every body"}
);
```

### Solucion

```
var $Class =function(){

  this.convert = function(obj){
    var keys = Object.keys(obj);
    var func = function(){};

    for(var x=0; x<keys.length; x++){
      func.prototype[keys[x]] = obj[keys[x]];
    }
  }
}
```

```
    }

    return func;
}

this.union = function(){

    var res = {};

    var union = function(obj){
        var keys = Object.keys(obj);
        for(var x=0; x<keys.length; x++) res[keys[x]] = obj[keys[x]];
    };
    for(var x=0; x<arguments.length; x++){
        union(arguments[x]);
    } //each arguments

    return res;
}
}
```

## Test del capítulo.

Marque con una X la respuesta correcta.

### El operador new sirve para?

- Obtener una instancia de clase
- Obtener un clon de un objeto
- Obtener una referencia circular

### El constructor de las clases en JavaScript es:

- La propiedad "construct"
- El cuerpo de la función
- No posee constructor

### Que retorna el siguiente script?

```
(Object.keys({a:1, b:2, c:3}).pop())
```

- undefined
- [1,2,3]
- "c"
- [a,b,c]

### Que retorna el siguiente script?

```
("hola a todos los programadores de javascript").split(" ").join("/")
```

- "hola/a/todos/los/programadores/de/javascript"
- "hola a todos los programadores de javascript"

- ["hola", "a", "todos", "los", "programadores", "de"]
- undefined

**Que retorna el siguiente script?**

```
new new function(){  
  return function(){  
    }  
}
```

- Object {}
- function (){}
- undefined
- Error