Primera librería.

En éste capítulo nos enfocaremos a escribir nuestro primera librería, la cual nos ayudará a lo largo de todo el curso a extender las propiedades y métodos de los objetos nativos en JavaScript (además de crear nuestros propios objetos útiles).
Lo interesante de la librería es que nos será igual de útil tanto en los navegadores como en Node.js, ya que el lenguaje es el mismo (con sus ventajas y carencias).

Objetivos.

El objetivo será construir un archivo .js en el cual una vez cargado (tanto en los navegadores como en node.js) extienda automáticamente los prototypes de los objetos nativos de JavaScript, como así también generar nuevos objetos que nos permita realizar operaciones más complejas.

Escenario de trabajo.

Para poder avanzar con nuestra librería necesitaremos crear 2 archivos, un archivo .html y otro .js.

El contenido de nuestro archivo index.html debería ser algo como lo siguiente

```
<html>
<head>
  <title>Curso de Node.js</title>
  <script type="text/javascript" src="lib.js"></script>

</head>
<body>
 <h1>Curso de Node.js</h1>
</body>
</html>
```

También crearemos un archivo llamado lib.js con el siguiente contenido

```
console.log("Cargando librería...");
```

Ahora para poder ver si nuestra librería empieza a funcionar solo debemos abrir nuestro archivo index.html con Chrome (o nuestro navegador favorito).

A partir de ahora trabajaremos con el archivo lib.js. Recordemos que para ver los nuevos cambios solo deberemos darle F5 en chrome (actualizar la página).

Todo lo que carguemos en la librería será accesible desde la consola de Chrome.

Lo primero que debemos hacer ahora es ponerle un nombre a nuestra librería, creando un objeto que la represente. También debemos auto ejecutar el código javascript para que automáticamente podamos contar con los nuevos prototypes de los objetos en cuanto cargamos el archivo index.html en el navegador.

```
/**
Name: $_$
Version: 0.0.1
*/

(function(){
console.log('Soy el código auto ejecutable');
})();
```

Arrays

En los capítulos anteriores aprendimos a trabajar con Arrays de manera simple, vimos el concepto de referencias, algunas propiedades, etc. Ahora realizaremos ejercicios más avanzado en los cuales los Arrays son claves, como así también nuevos métodos.

Prototypes

El objetivo es agregar nuevos prototypes al objeto Array de JavaScript, para eso debemos tener en cuenta:

- Que no exista previamente el prototype a crear.
- Que las operaciones funcionen sin importar el tipo de dato almacenado en los Arrays.
- Todos los prototipos que creemos deberán contar con el método version.

Veremos un ejemplo de requerimiento de prototype y el código que lo resuelve.

Prototype min.

El método min retorna el valor mínimo contenido dentro del Array, teniendo en cuenta sólo los valores del tipo numérico.
En caso de que no existan valores numericos retornará null.

Solución

```
(function(){
  Array.prototype.min = function(){
        var p=null;
        for(var x=0; x<this.length; x++) {
                if(typeof this[x] == "number"){
                 p = p==null ? this[x] : ( p<this[x] ? p : this[x]);
                }
        }//end for
        return p;
 }

})();
```

Solo nos estaría faltando agregar el método "version" al método "min".

Prototype max.

El prototype max retorna el valor máximo contenido dentro del Array, teniendo en cuenta sólo los valores del tipo numérico.
En caso de que no existan valores numericos retornará null.

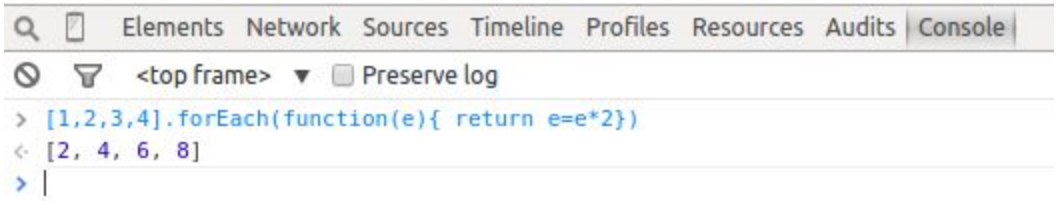
Si ambos prototypes funcionan bien, el siguiente ejemplo debería arroja un “2” como resultado.

```
var a=[2,6,123,667,21,66734,124,22];
var b=[5,1123,543,"",111,-123,229,1];
var c=[-23,1,2,6,723,22,1,73,2,663,1];

[a.min(), b.min(), c.min()].max();
```

Prototype forEach.

El prototype forEach recibe como argumento una función. Por cada elemento del Array se ejecuta la función, pasando como argumento el elemento en sí.
Retorna el Array.

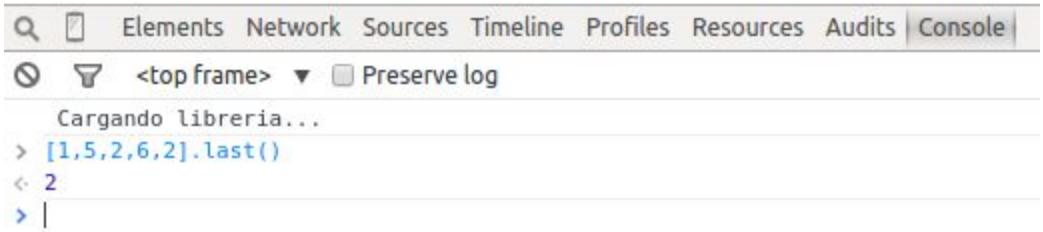
```
var prices=[54,6,123,667,21,634,124,22];

var dolar = 8.68 //depende de la internacionalizacion
prices.forEach(function(e){
console.log(e);
       return {
        pesos: e*dolar
        ,dolar: e
       }
});
```




```
([
[2,5,2,623,42,342,34,234]
,[123,123,12,3,123]
,[7,34,54,2345,234]
].forEach(function(a){
 return a.max()
})).max()
```

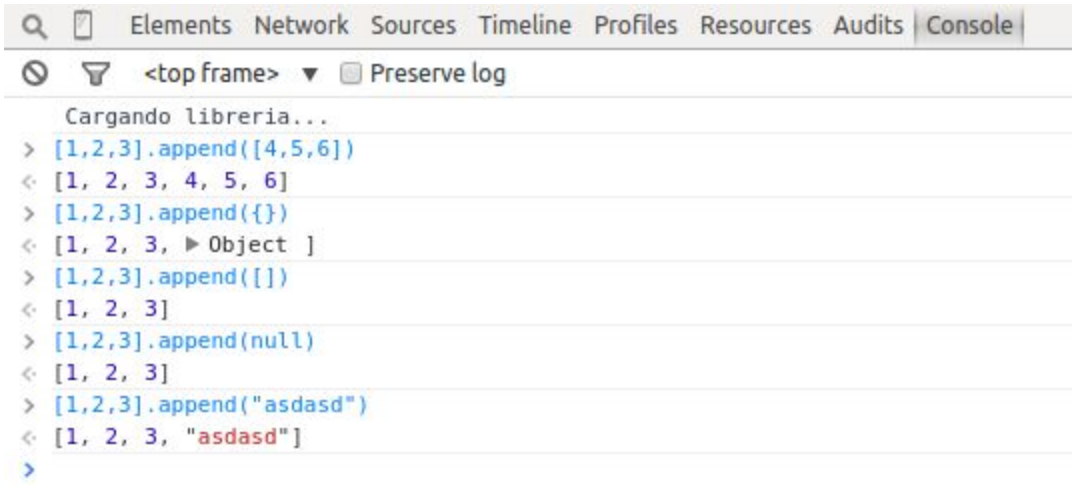
Prototype last.

El método last retorna el último elemento del Array. No recibe argumentos.



Prototype append.

El método append agregar el valor pasado como argumento al Array. Si el valor pasado como argumento es un Array deberá unir todos los valores del Array pasado como argumento al Array original.
Retorna el Array original (this).



Test del capítulo 3.

Que devuelve el siguiente script?

```
console.log(typeof {} == "undefined" ? ( typeof [] == "Array" ? true : false) : [1,2,3].pop())
```

- undefined
- Error
- true
- 55
- 3
- function

Cual de las siguientes afirmaciones es correcta?

- Un método de clase es heredado por todas sus instancias.
- Typeof retorna el tipo de objeto instanciado.
- Los prototipos pueden tener propiedades cuyo valor sea una función.