

[Introduccion.](#)

[Que es JavaScript?](#)

[Orientado a prototipos.](#)

[Consola de Chrome.](#)

[console.log](#)

[Tipos de datos.](#)

[Comentarios](#)

[Tipo de datos Object.](#)

[String](#)

[Date](#)

[Algunas propiedades y métodos.](#)

[Array](#)

[Algunas propiedades y métodos.](#)

[Funciones.](#)

[Funciones anónimas.](#)

[Ejemplo integrador.](#)

[Closures.](#)

[Typeof, arguments, try & catch.](#)

[Typeof](#)

[Arguments.](#)

[Ejemplo integrador.](#)

[Try & Catch](#)

[Test del capítulo.](#)

[Prácticas.](#)

Introduccion.

El objetivo del capítulo uno del curso es hacer un primer acercamiento al lenguaje JavaScript, ver las características principales del lenguaje, la diferencia con otros lenguajes, realizar los primeros ejemplos y comenzar a entender del concepto de Closures (fundamental para desarrollar en JavaScript como un profesional).

Que es JavaScript?

JavaScript es un lenguaje de programación interpretado, variant (las variables pueden cambiar su tipo de datos en runtime) y orientado a prototipos.

Fué creado por Netscape Communications Corp y actualmente mantenido por la fundación Mozilla.

Su principal uso es en navegadores webs, aunque actualmente está tomando una gran fuerza el server side (Node.js). También encontramos JavaScript aplicaciones Desktop, Mobile, Robótica, documentos PDF, Widgets, etc.

Orientado a prototipos.

Un lenguaje orientado a prototipos es aquel lenguaje en el cual los objetos no se obtienen de la instancia de una clase (lenguaje Java por ejemplo), sino de la clonación de un objeto preexistente.

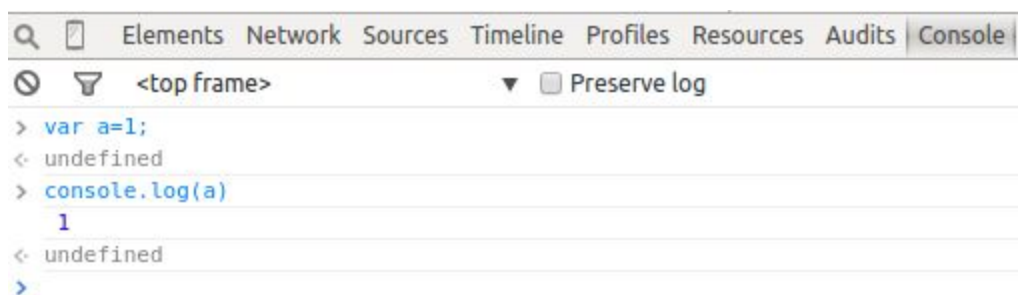
Otra característica interesante es que los objetos pueden mutar en runtime, es decir, que podemos agregarle propiedades y métodos durante la ejecución.

Consola de Chrome.

Google Chrome cuenta con el motor de JavaScript más usados actualmente desarrollado por Google, el motor V8, el cual es también la pieza fundamental de Node.js, MongoDB y otras tantas tecnologías basadas en JavaScript. El motor V8 es gratuito (cualquiera puede bajarlo), de desarrollo constante y con una fuerte comunidad por detrás.

Los navegadores actuales más populares (Chrome, FireFox, Internet Explorer) cuentan con una consola de desarrollo la cual nos permite ejecutar código JavaScript sin problemas, ahorrando mucho tiempo.

Para activar al consola de desarrollo de Chrome solo debemos presionar F12 en el navegador y hacer click en la solapa "Consola".



Declaramos una variable "a", asignamos el valor "1" y usando el método "log" del objeto "console" podemos ver el contenido de la variable "a".

La consola de Chrome nos ofrece una serie de herramientas orientada a los desarrolladores Frontend, aunque inicialmente nos será de perfecta ayuda para aprender los conceptos fundamentales de JavaScript.

console.log

Tanto en la consola de desarrollo de Chrome como en Node.js tenemos disponible el object "console" el cual nos ofrece una serie de métodos que nos permitirán "ver" el contenido de las variables. Parece algo simple, pero la realidad es que cuando debemos analizar propiedades circulares, objetos o el body de una función es una herramienta extremadamente útil y potente. El método "log" soporta N argumentos, por lo que podemos en una sola llamada ver el valor de varias variables al mismo tiempo.

```
console.log(a,b,c,d);
```

Tipos de datos.

Los tipos de datos en JavaScript se dividen en dos grupos, los primitivos y los objetos.

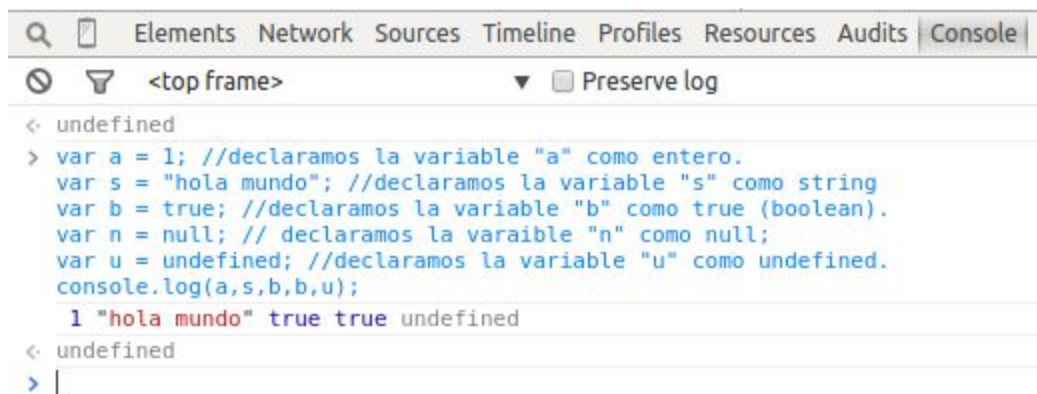
Primitivos: son cadenas (Strings), Booleanos (true/false), Números, null y undefined.

Si bien no es estrictamente necesario declarar las variables antes de usarla, es una excelente idea hacerlo para no encontrarnos con problemas o bugs en nuestra aplicación.

La palabra reservada "var" se utiliza para declarar las variables.

Comentarios

Los comentarios en JavaScript son en base C, por lo que usamos "//" para comentar una línea o "/* ... */" para comentar un bloque.



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The log shows the output of a `console.log` statement with five arguments: `a`, `s`, `b`, `b`, and `u`. The output is displayed as `1 "hola mundo" true true undefined`. The console also shows the source code of the script that was executed, which declares and logs the variables `a`, `s`, `b`, `n`, and `u`.

```
var a = 1; //declaramos la variable "a" como entero.
var s = "hola mundo"; //declaramos la variable "s" como string
var b = true; //declaramos la variable "b" como true (boolean).
var n = null; // declaramos la variable "n" como null;
var u = undefined; //declaramos la variable "u" como undefined.
console.log(a,s,b,b,u);
```

Tipo de datos Object.

Todo lo que no es nativo y es creado por el usuario es considerado un Object, por ejemplo una función. JavaScript también nos ofrece una serie de Objects como Date (fechas), Arrays, Error, etc.

Los objetos en JavaScript son lo más importante ya que son el TODO dentro del lenguaje.

String

El objeto nativo String se utiliza para el manejo de cadena de caracteres. Cuenta con propiedades y método para poder operar sobre strings de manera simple.

Método / Propiedad	Descripción
charAt()	Retorna el carácter de la posición pasada como argumento.
concat(S1, S2, S...N)	Une dos o más Strings y retorna un nuevo string.
indexOf(C)	Retorna la posición de la primera aparición de/los caracteres pasados como argumento. En caso de no encontrar nada retorna -1.
lastIndexOf()	Retorna la posición de la última aparición de/los caracteres pasados como argumento.
replace()	Reemplaza la primer aparición del carácter o string por el pasado como segundo argumento. Retorna un nuevo string.
split()	Convierte un string en un array dividiendo por el patrón pasado como argumento. Si no se pasan argumentos asume que el patrón es el carácter de espacio.
substr(init, N)	Retorna un nuevo string extrayendo N cantidad de caracteres desde la posición indicada.
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Convierte los caracteres del string en minúsculas.

toUpperCase()	Convierte los caracteres del string en mayúsculas.
trim()	Elimina los espacios en blanco a los costados de un string.

Date

El object nativo Date se utiliza para operar con fechas/horas del sistema.

Al ser un Object podemos obtener una "copia" (clonar) usando el operador "new".

Si invocamos al object Date como función nos devolverá en formato de string la fecha y hora actual. Si invocamos como instancia (clon) anónimo (sin asignar la copia a una variable) retornara un Object Date y mostrará en la consola su valor.

Si lo clonamos y asignamos la copia a una variable podemos llamar a los métodos del Object para mostrar la fecha actual en la consola.

```
Q Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log
> Date
< function Date() { [native code] }
> Date()
< "Sun Jan 04 2015 22:28:02 GMT-0300 (ART)"
> new Date()
< Sun Jan 04 2015 22:28:08 GMT-0300 (ART)
> var d= new Date()
< undefined
> d.getDate() + "/" + d.getMonth() + "/" + d.getFullYear();
< "4/0/2015"
> |
```

Algunas propiedades y métodos.

Método / Propiedad	Descripción.
getDate()	Retorna el día del mes (de 1-31).
getDay()	Retorna el día de la semana(from 0-6).
getFullYear()	Retorna el año en formato de 4 dígitos.
getHours()	Retorna la hora (0-23).

getMilliseconds()	Retorna los milisegundos (0-999).
getMinutes()	Retorna los minutos (0-59).
getMonth()	Retorna el mes. Arranca de cero, por lo que debemos sumarle 1 (0-11).
getSeconds()	Retorna los segundos (0-59).
getTime()	Retorna el timestamp.
setDate()	setea el día del mes.
setFullYear()	Setea el año, formato 4 dígitos.
setHours()	Setea la hora.
setMinutes()	Setea los minutos.
setMonth()	Setea el mes.
setSeconds()	Setea los segundos.

Array

Los array en JavaScript son objetos que cuentan con propiedades y métodos como en Java o C#, de longitud variable y fácil uso.

Formas de declarar un array:

```
Array(1,2,3); //Array  
[1,2,3]; //array  
new Array(1,2,3); //nuevo array (object).
```



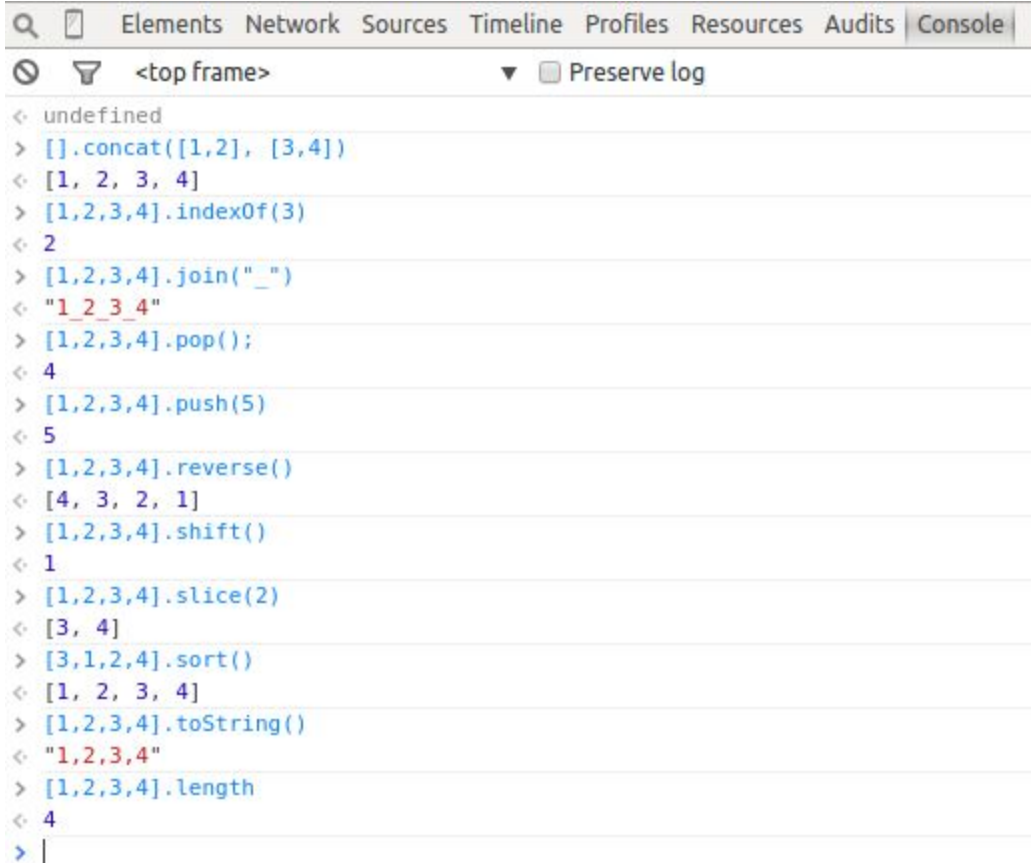
```
> Array(1,2,3); //Array  
< [1, 2, 3]  
> [1,2,3]; //array  
< [1, 2, 3]  
> new Array(1,2,3); //nuevo array (object).  
< [1, 2, 3]  
> |
```

Algunas propiedades y métodos.

Método/Propiedad	Descripción
concat()	Une dos o más arrays, retornando un nuevo array con las partes unidas.
indexOf()	Busca dentro de los elementos un valor y si lo encuentra retorna su posición dentro del array. Si no encuentra datos retorna -1.
join()	Une todos los elementos del array y los une en un string separando cada elemento por "," por defecto (o el carácter/string que pasemos como argumento).
pop()	Elimina el último elemento del array y retorna dicho elemento.
push()	Agrega un nuevo elemento en el Array y retorna la nueva longitud.
reverse()	Invierte el orden de los elementos de un array.
shift()	Elimina el primer elemento de un Array y lo retorna.
slice(N)	Selecciona desde la posición N de un array retornando un nuevo array con los elementos.
sort()	Ordena los elementos de un Array.
toString()	Convierte el Array en un string.
length	Devuelve la longitud de un Array

```
[].concat([1,2], [3,4]);  
[1,2,3,4].indexOf(3);  
[1,2,3,4].join("_");  
[1,2,3,4].pop();  
[1,2,3,4].push(5);  
[1,2,3,4].reverse();  
[1,2,3,4].shift();  
[1,2,3,4].slice(2);  
[3,1,2,4].sort();  
[1,2,3,4].toString();
```

```
[1,2,3,4].length;
```



```
Q [1,2,3,4].length;
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log
< undefined
> [].concat([1,2], [3,4])
< [1, 2, 3, 4]
> [1,2,3,4].indexOf(3)
< 2
> [1,2,3,4].join("_")
< "1_2_3_4"
> [1,2,3,4].pop();
< 4
> [1,2,3,4].push(5)
< 5
> [1,2,3,4].reverse()
< [4, 3, 2, 1]
> [1,2,3,4].shift()
< 1
> [1,2,3,4].slice(2)
< [3, 4]
> [3,1,2,4].sort()
< [1, 2, 3, 4]
> [1,2,3,4].toString()
< "1,2,3,4"
> [1,2,3,4].length
< 4
> |
```

Funciones.

Las funciones en JavaScript son un mundo muy interesante y complejo, es por eso lo que iremos abordando de a poco, avanzando en cada capítulo hasta llegar a un máximo de comprensión.

Declaracion.

La forma tradicional de declarar una funcion en JavaScript es la siguiente:

```
function say(word){
  console.log(word);
}
```

En el ejemplo anterior declaramos la funcion "say" la cual espera el argumento "word". En el cuerpo de la función mostramos en la consola el valor pasado como argumento.

Si ejecutamos en la consola el ejemplo deberíamos ver algo como lo siguiente.

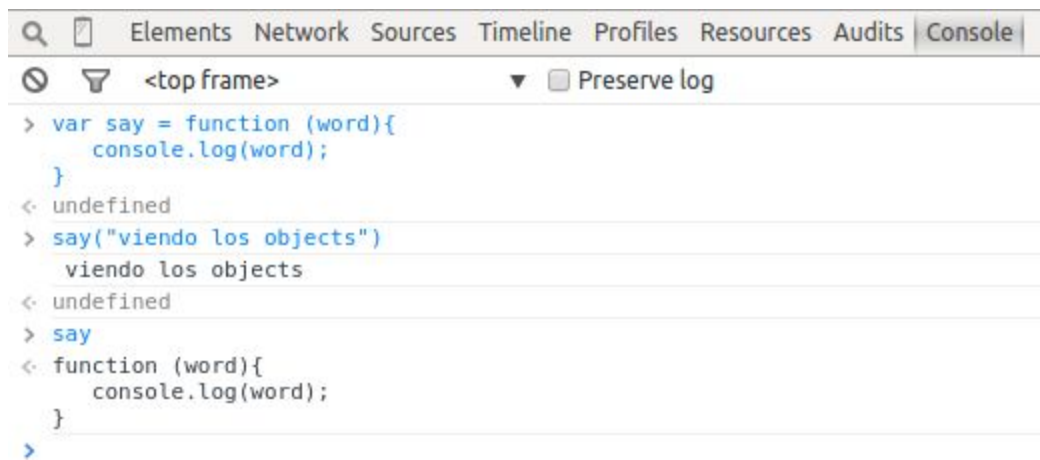
En el capítulo anterior hablamos de los tipos de datos Object, y se mencionó que una función es un tipo de dato.

Ahora bien, la forma de declarar una variable como Object del tipo Function es la siguiente

```
var say = function (word){  
    console.log(word);  
}
```

En el ejemplo declaramos la variable "say" cuyo valor es una function.

Si en la consola queremos ver el contenido de "say" veremos que nos muestra el "body" de la función.



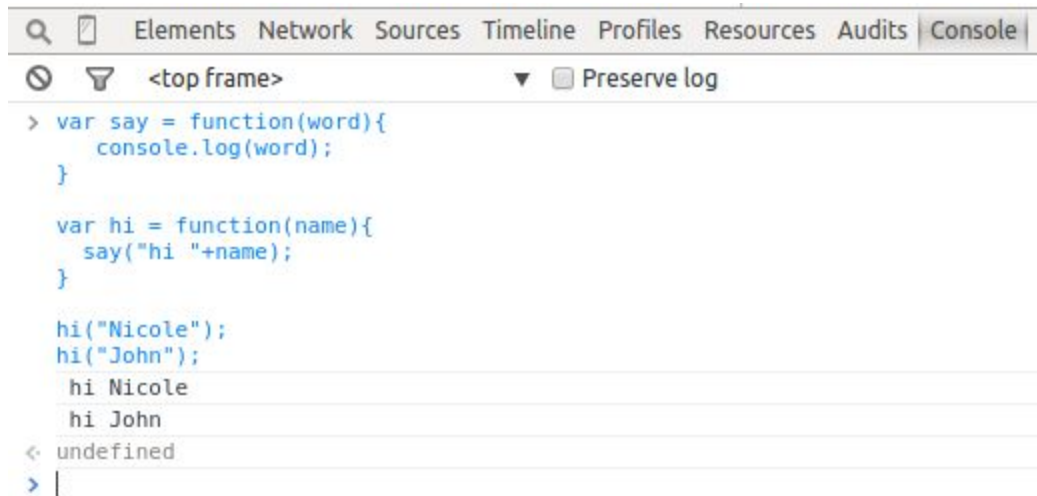
Las funciones puede recibir como argumento cualquier tipo de dato (primitivo u Objects), lo que quiere decir que por ejemplo podemos recibir una function como argumento.

Veamos algunos ejemplos.

```
var say = function(word){  
    console.log(word);  
}
```

```
var hi = function(name){  
    say("hi "+name);  
}
```

```
hi("Nicole");  
hi("John");
```



```
> var say = function(word){
    console.log(word);
}

var hi = function(name){
    say("hi "+name);
}

hi("Nicole");
hi("John");
hi Nicole
hi John
< undefined
> |
```

Una de las características de los arrays en JavaScript es que pueden almacenar cualquier objeto en cada una de sus posiciones, ya sean tipos de datos primitivos como objetos.

Funciones anónimas.

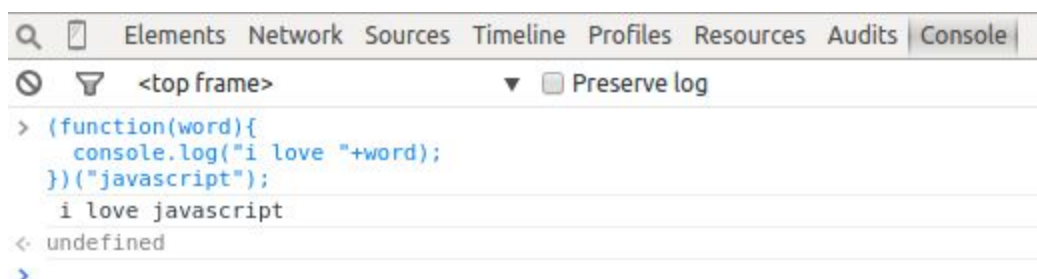
JavaScript soporta el manejo de funciones anónimas, las cuales son fundamentales para entender el concepto de callbacks (que veremos más adelante).

Algo importante a saber es que cualquier cosa entre () será ejecutada por JavaScript.

Las funciones anónimas son funciones que se pueden almacenar en una posición de memoria (por ejemplo, en una posición de array) pero que no pueden ser ejecutadas en forma referencial a un nombre ya que carece del mismo.

JavaScript soporta el paso de funciones anónimas como argumentos de funciones (una función que espera como argumento otra función). Ésta es la clave de los callbacks.

```
(function(word){
    console.log("i love "+word);
})("javascript");
```



```
> (function(word){
    console.log("i love "+word);
})("javascript");
i love javascript
< undefined
>
```

En el ejemplo anterior hemos creado una función anónima, la cual es ejecutada inmediatamente una vez de haber sido creada. Cuando ejecutamos pasamos un string como argumento, el cual es recibido como la variable "word".

Como bien habíamos mencionado anteriormente, los Arrays en JavaScript pueden almacenar cualquier objeto en sus posiciones, entre los cuales se encuentran las funciones.

Ejemplo integrador.

En este ejemplo veremos como las funciones anónimas son usadas para ser pasadas como argumentos.

```
var functions = []; //declaramos un array vacio

/*
Declaramos la function add, la cual espera un argumento "f".
*/
var add = function(f){
    functions.push(f);
}

/*
Declaramos la function run.
*/
var run = function(){
    for(var x=0; x<functions.length; x++){ //iteramos sobre el array "functions"
        functions[x](); //ejecutamos cada una de las posiciones del array
    }
}

add(function(){
    console.log("soy la func 1");
});

add(function(){
    console.log("soy la func 2");
});

add(function(){
    console.log("soy la func 3");
});

run();
```

```
Q  Elements Network Sources Timeline Profiles Resources Audits Console
<  <top frame>  Preserve log

> var functions = [];

var add = function(f){
  functions.push(f);
}

var run = function(){
  for(var x=0; x<functions.length; x++){
    functions[x]();
  }
}

add(function(){
  console.log("soy la func 1");
});

add(function(){
  console.log("soy la func 2");
});

add(function(){
  console.log("soy la func 3");
});

run();
soy la func 1
soy la func 2
soy la func 3
< undefined
> |
```

Closures.

Los closures suelen ser uno de los temas más difíciles de entender en JavaScript, pero la realidad es que son un concepto simple.

En JavaScript podemos tener N funciones dentro de otra función, y cada función "contenida" tiene acceso a las variables de las funciones de nivel superior (contenedora), a eso se le llama closure.

Veamos el ejemplo:

```
var main = function(){
  var name = "John";

  function child1(){
    console.log("hi "+name);
  }
}
```

```
function child2(){
  console.log("bye "+name);
}

function change(){
  name = "Rita";
  child1();
  child2();
}

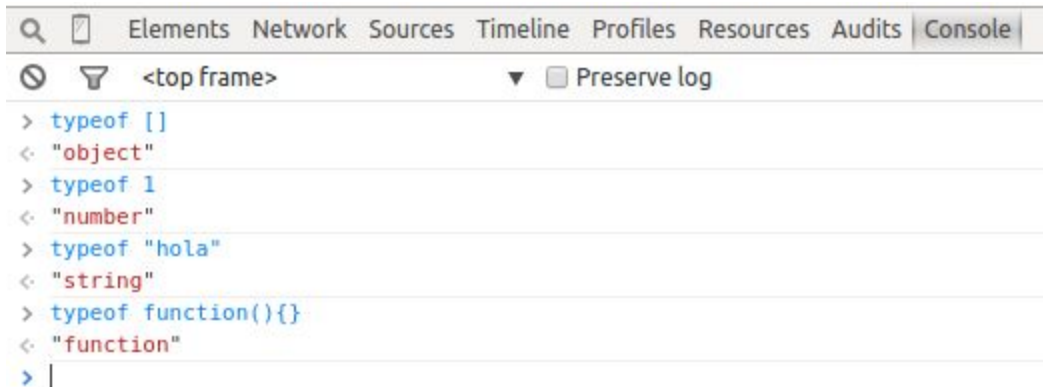
child1();
child2();
change();
}
main();
```

Typeof, arguments, try & catch.

Para terminar el capítulo 1 veremos 3 temas simples pero extremadamente útiles en JavaScript.

Typeof

typeof es una palabra reservada que nos devolverá el tipo de datos del argumento que pasemos. No es una función, por lo cual no se ejecuta entre paréntesis.



```
> typeof []
< "object"
> typeof 1
< "number"
> typeof "hola"
< "string"
> typeof function(){}
< "function"
> |
```

Arguments.

Arguments es una propiedad que tendremos disponible dentro de cualquier función, y representa la N cantidad de argumentos pasados a la función.

En JavaScript si bien podemos definir que esperamos uno o más argumentos en la función, no quiere decir que debemos respetar la cantidad (podemos pasar menos o más argumentos de los esperados).

La variable "arguments" es un array con el total de argumentos pasados a una función.

```
var run = function(){
  console.log(arguments);
}

run(1,"hola", 2,3, "rita", 5);
```



Ejemplo integrador.

```
var run = function(){
  for(var x=0; x<arguments.length; x++) {
    if(typeof arguments[x] == "function" ) arguments[x]();
  }
}

run([
  , Date
  , function(){
    console.log("Yo si soy una function!");
  }
  , 6635
  , "soy un lindo texto"
  , function(){
    console.log("soy otra function!");
  }
  , "fin"
]);
```



The screenshot shows the Chrome DevTools Console with the 'Console' tab selected. The breadcrumb is '<top frame>' and there is a 'Preserve log' checkbox. The console contains the following JavaScript code and its output:

```
> var run = function(){
    for(var x=0; x<arguments.length; x++) {
        if(typeof arguments[x] == "function" ) arguments[x]();
    }

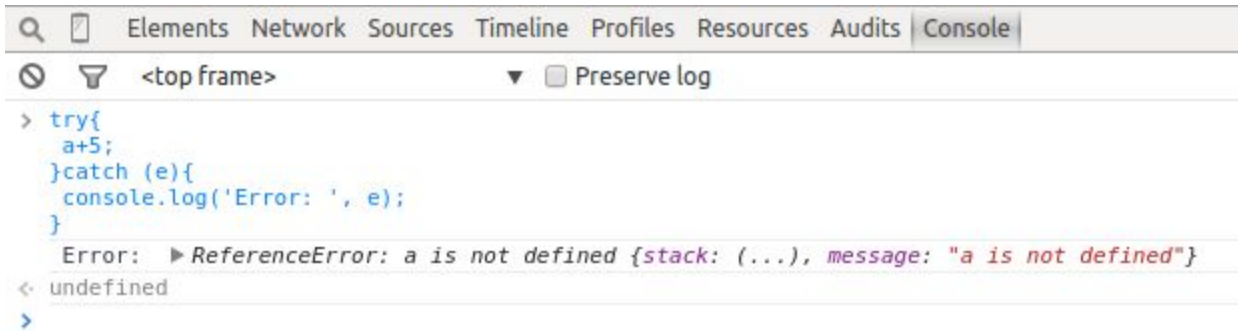
    run([
        Date
        , function(){
            console.log("Yo si soy una function!");
        }
        , 6635
        , "soy un lindo texto"
        , function(){
            console.log("soy otra function!");
        }
        , "fin"
    ]);
    Yo si soy una function!
    soy otra function!
    < undefined
>
```

Try & Catch

En JavaScript como en muchos otros lenguajes tenemos la posibilidad de capturar errores para poder decidir qué hacer con los mismos evitando que nuestra aplicación se cierre.

Ejemplo:

```
try{
    a+5;
}catch (e){
    console.log('Error: ', e);
}
```



```
> try{
  a+5;
}catch (e){
  console.log('Error: ', e);
}
Error: ▶ReferenceError: a is not defined {stack: (...), message: "a is not defined"}
< undefined
>
```

Test del capítulo.

Marque con una X la respuesta correcta.

JavaScript es un lenguaje:

- Orientado a objetos
- Orientado a datos
- Orientado a Prototipos
- Orientado a Instancias

Qué método del objeto Array une dos o más array devolviendo una copia nueva?

- join()
- slice()
- concat()
- union()
- sort()

Qué método del objeto Date retorna el año en formato de cuatro dígitos?

- getDate()
- getYear()
- getFullYear()
- getCompleteYear()

Que muestra en pantalla el siguiente script?

```
(function(a){
  for(var x=0; x<a.length; x++) console.log(a[x]);
})([2541,2,56,-10,221,76,25,-110].sort());
```

- El contenido del array en pantalla

- El contenido del array en pantalla en forma ordenada
- Los números del 1 al 10
- Genera un error

Que muestra en pantalla el siguiente script?

```
var fns = [];  
  
fns.push(function(){  
  console.log("say 5");  
});  
  
fns.push(function(){  
  console.log("say 2");  
});  
  
fns.push(function(){  
  console.log("say 22");  
});  
  
fns.push(function(){  
  console.log("say 76");  
});  
  
fns[      [0,2,1,3].sort().pop()-2    ]    ();
```

- say 5
- say 2
- say 22
- say 76
- Un Error
- Ninguno de los anteriores

Prácticas.

Escribir una función que reciba N argumentos. Iterando sobre los argumentos, Si solo si el tipo de dato del argumento es "function" almacenar dicho argumento dentro de un array (declarado dentro del cuerpo de la función). Iterar finalmente sobre el array declarado, ejecutando cada una de las funciones, teniendo en cuenta posibles errores en las funciones almacenadas.

```
var f= function(){
  var a = [];
  for(var x=0; x<arguments.length; x++){
    if(typeof arguments[x] == "function") a.push(arguments[x]);
  }

  for(var x=0; x<a.length; x++){
    try { a[x](); }
    catch(e){ console.log("alguien la bardio: ",e)}
  }//end for
}

f(1,"asads", [], function(){ console.log("hi");}, true, 1, function(){ asd*2;});
```