

JavaScript

Programación Web2

Un poco de historia

- JavaScript nace en medio de “la guerra de los navegadores” del lado de [Netscape Navegador](#)
- Inicialmente se popularizó en la programación frontend (lado del cliente); aunque en sus inicios, también fué usado por Netscape para la programación del lado del servidor (backend)

<https://docs.oracle.com/cd/E19957-01/816-6411-10/intro.htm#1022274>

- Hoy en día se utiliza en una amplia variedad de dispositivos y plataformas lo que lo ha convertido en el lenguaje más usado para el desarrollo web.



Brendan Eich

JavaScript Incrustado

```
<!DOCTYPE html>

<html lang="es">

  <head>

    <title>Ejemplos de JavaScript</title>

    <meta charset="UTF-8">

    <script>console.log("Hola Mundo");</script>

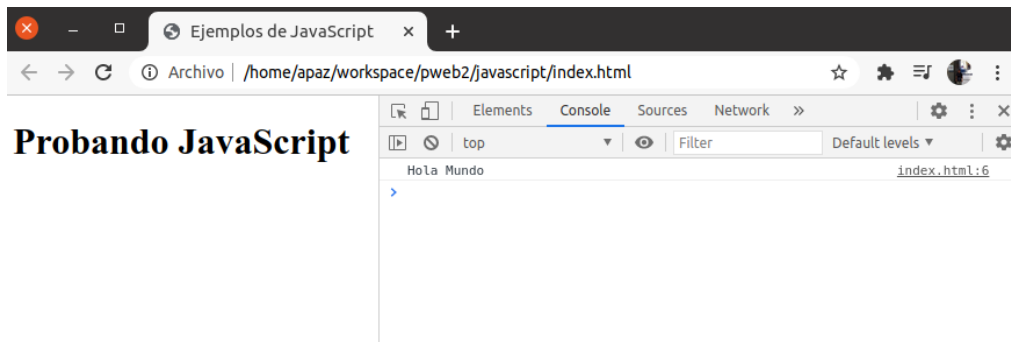
  </head>

  <body>

    <h1>Probando JavaScript</h1>

  </body>

</html>
```



Abrir Consola de JavaScript

- **Shift + CTRL + J**
- **Shift + CTRL + K (firefox)**
- **Option + ⌘ + J**

Tipos de Datos

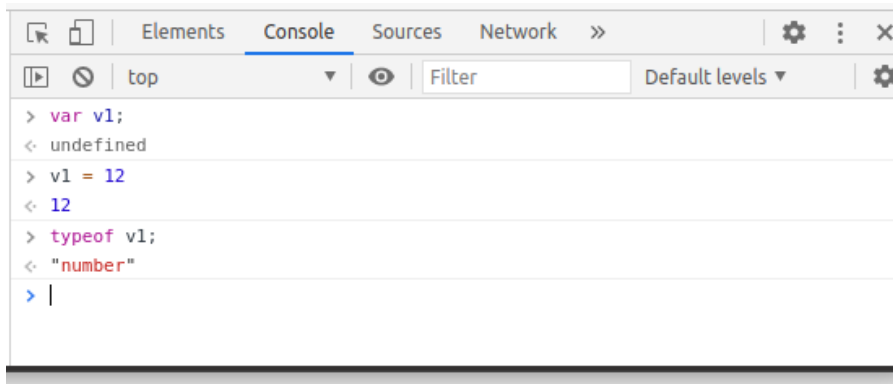
- Primitivos

- Numerico
- String
- Boolean
- Null
- Undefined
- Symbol

- Derivados

- Objetos
- Funciones
- Arreglos
- Expresiones regulares

Para probar el tipo de datos una variable puede usar el operador `typeof`



```
> var v1;  
< undefined  
  
> v1 = 12  
< 12  
  
> typeof v1;  
< "number"  
  
> |
```

== VS ===

De manera básica == compara valores aunque no sean del mismo y para conseguirlo usa funciones de conversión de tipo automáticas, mientras que === compara sólo si las variables son del mismo tipo.

En general para las comparaciones de igualdad en if, while o for deberíamos usar ===, a menos, que la conversión automática de tipos sea algo deseable.

```
> var v1 = '1';  
< undefined  
  
> v1  
< "1"  
  
> var v2  
< undefined  
  
> v2 = 1  
< 1  
  
> v1 == v2  
< true  
  
> v1 === v2  
< false
```

<https://stackoverflow.com/questions/359494/which-equals-operator-vs-should-be-used-in-javascript-comparisons>

Programación orientada a objetos

A diferencia de Java, que se necesitan clases para crear objetos; en JavaScript se pueden crear objeto directamente a partir de la declaración de sus campos.

Dos objetos pertenecerán a la misma clase si ambos poseen los mismos campos

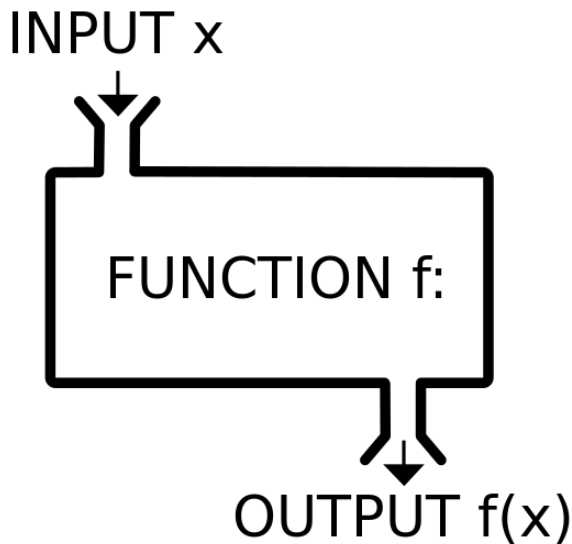
La programación orientada a objetos en JavaScript *puede ser un poco frustrante de entender*, debido a que su implementación es algo distinta a la de lenguajes como Java o C++.

```
> var persona = {  
  nombre: 'Alfredo',  
  apellidos: 'Paz Valderrama',  
  edad: 47,  
}  
< undefined  
-----  
> persona.nombre  
< "Alfredo"
```

Programación funcional

- Hacer programación funcional en JavaScript puede ser más sencillo, el concepto es simple: la función recibe una entrada, hace un proceso y devuelve un valor de salida

```
function saluda(name){  
  return "¡Hola " + name + ", bien vendio al curso de PWEB2!";  
}
```



Funciones de “alto orden” (high order functions)

Son funciones que pueden recibir una función como parámetro o devolver una función como respuesta.

La reutilización de código y generalización no sólo se puede hacer a nivel de clases y objetos

```
> function accion(ver){  
  return function (suj){  
    return suj + " sabe " + ver;  
  }  
}  
< undefined  
> let cocina = accion("cocinar");  
< undefined  
> cocina("Alfredo");  
< "Alfredo sabe cocinar"
```


Filter

Es una forma de iterar sobre los valores de un “grupo” de valores de los que se desea seleccionar sólo algunos: los que cumplan con alguna condición.

Recibe una función de filtro

Devuelve un nuevo grupo de valores con aquellos elementos que cumplieron con la condición del filtro

```
> let num = [1, 2, 3, 4];  
< undefined  
  
> let par = num.filter(function(n){ return n % 2 == 0; });  
< undefined  
  
> par  
< ▶ (2) [2, 4]
```

Map

Es otra forma de iterar sobre un conjunto de valores que se desea cambiar

Recibe una función, normalmente anónima

Devuelve un nuevo grupo de valores, donde la función se aplicó la función a cada valor.

```
> let num = [1, 2, 3, 4];  
< undefined  
  
> let dob = num.map(function(n){ return 2*n; });  
< undefined  
  
> dob  
< ▶ (4) [2, 4, 6, 8]
```

Reduce

Es otra forma de iterar sobre un grupo de elementos y aplicar una misma fórmula a todos los elementos del grupo dos a dos, hasta conseguir un sólo valor “resumen” de todos ellos

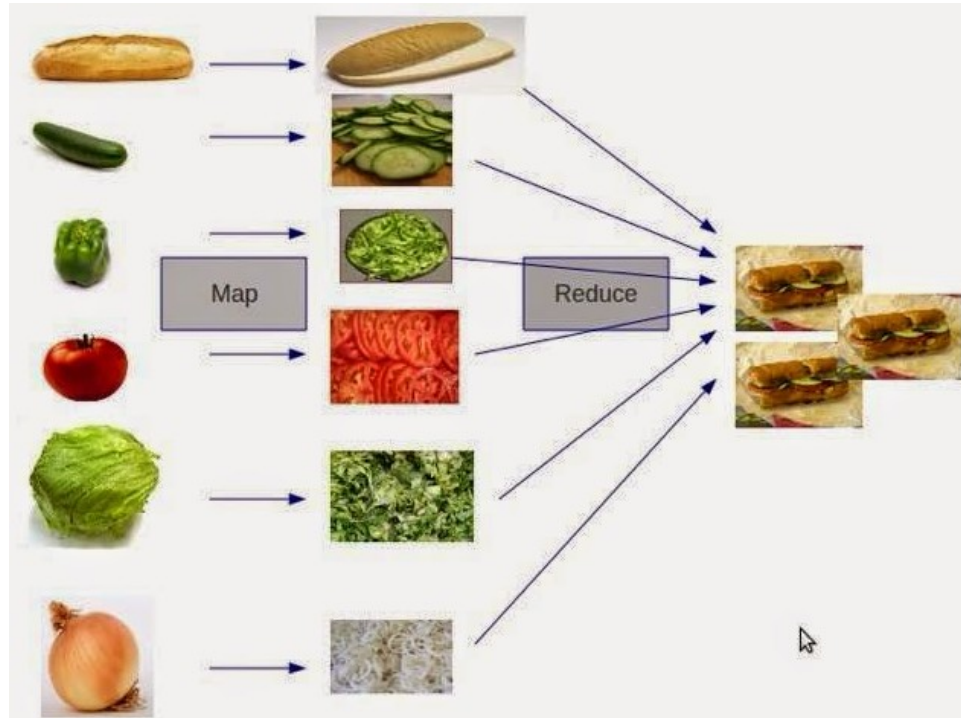
Entrada: Una función que recibe dos valores y les aplica una operación.

Salida: Un sólo valor productor de haber aplicado la operación a todos los elementos del grupo.

```
> let num = [1, 2, 3, 4];  
< undefined  
  
> let sum = num.reduce(function(a, b){ return a + b; });  
< undefined  
  
> sum  
< 10
```

Map Reduce

<https://www.datasciencecentral.com/forum/topics/what-is-map-reduce>



Aprender más sobre programación funcional

Revise el siguiente material y prepare una exposición de 5 minutos para la próxima clase:

- An introduction to functional programming (Mary Rose Cook)
 - <https://codewords.recurse.com/issues/one/an-introduction-to-functional-programming>
 - Don't iterate over lists. Use map and reduce.
 - Write declaratively, not imperatively
 - Use functions
 - Remove state
 - Use pipelines

ECMAScript

- Durante la guerra de los navegadores y con el objetivo de permitir páginas web que funcionen bien y distintos navegadores, se acordó estandarizar al lenguaje de JavaScript por un tercero [ECMA](#).
- ECMAScript describe un lenguaje de programación cuya implementación más conocidas es JavaScript.
- Es posible que JavaScript aún no implemente algunas de las más recientes características de ECMAScript.
- Los navegadores usan “motores” de JavaScript para interpretar nuestros códigos:
 - [V8](#) para Chrome
 - [SpiderMonkey](#) para Firefox

var, let, const

- La declaración de variables con var suele tener algunos problemas, por ejemplo :

```
>> var v1 = 1;
← undefined
>> var v1 = 2;
← undefined
>> console.log(v1);
2
```

- Con let eso provocaría un error sintáctico detectable.

```
>> let v1 = 1;
← undefined
>> let v1 = 2;
❗ ▶ Uncaught SyntaxError: redeclaration of let v1
   <anonymous> debugger eval code:1
   [Saber más]
```

- El caso de const es igual al de let, pero el valor de la variable no se puede cambiar

Ámbito de una variable (environment)

Cuando se usa **var**, el ámbito puede ser un poco extraño respecto a lenguajes como java, en cambio **let** se comporta de manera más familiar

```
>> var v1 = 1;
```

```
← undefined
```

```
>> if(true){  
    var v1 = 2;  
}
```

```
← undefined
```

```
>> console.log(v1);
```

```
2
```

```
>> let v1 = 1;
```

```
← undefined
```

```
>> if( true ){  
    let v1 = 2;  
}
```

```
← undefined
```

```
>> console.log(v1);
```

```
1
```

```
>> if(true){  
    var v1 = 1;  
}
```

```
← undefined
```

```
>> console.log(v1);
```

```
1
```

```
>> if(true){  
    let v1 = 1;  
}
```

```
← undefined
```

```
>> console.log(v1);
```

```
❗ ▶ Uncaught ReferenceError: v1 is not defined  
    <anonymous> debugger eval code:1  
    [Saber más]
```


Plantillas de Strings (template literals)

En Perl se podía incrustar código HTML en medio de un script usando <<< e incluso poner variables.

En JavaScript se puede conseguir un efecto similar usando las comillas invertidas

```
> var nombre = 'Alfredo';  
< undefined  
-----  
> var saludo = `Hola ${nombre}  
esta es otra línea`;  
< undefined  
-----  
> console.log(saludo);  
Hola Alfredo  
esta es otra línea
```

Funciones flecha (arrow)

Con esta sintaxis se pueden escribir funciones como valor de manera más sencilla

```
> let cuadrado1 = function (x){  
  return x*x;  
};  
↵ undefined  
  
> let cuadrado2 = x => {  
  return x*x;  
};  
↵ undefined  
  
> let cuadrado3 = x => x*x;  
↵ undefined  
  
> console.log(cuadrado1(2));  
4  
↵ undefined  
  
> console.log(cuadrado3(2));  
4  
↵ undefined
```

Array destructing, property shorthand, shorthand method names

Estas nuevas características de JavaScript permiten escribir códigos más legibles

```
> let [a, b] = [3, 5];
```

```
< undefined
```

```
> console.log(b);
```

```
5
```

```
< undefined
```

```
> console.log(a);
```

```
3
```

```
> [b, a] = [a, b]
```

```
< ▶ (2) [3, 5]
```

```
> console.log(b);
```

```
3
```

```
< undefined
```

```
> console.log(a);
```

```
5
```

```
> let coord = { a, b };
```

```
< undefined
```

```
> console.log(coord);
```

```
▶ {a: 5, b: 3}
```

```
> let p = {  
  nombre: 'Alfredo',  
  apellido: 'Paz-Valderrama',  
  saludo: () => {  
    console.log('Hola');  
  }  
}
```

```
< undefined
```

```
> p.saludo()
```

```
Hola
```

Modulos

En JavaScript se pueden usar archivos separados a modo de bibliotecas de código

El archivo html incluye un archivo de javaScript con type module	El archivo “test.js” usa funciones declaradas en otro archivo	El archivo “mate.js” es una biblioteca de dos funciones
<pre><!DOCTYPE html> <html lang="es"> <head> <title>Ejemplos de JavaScript</title> <meta charset="UTF-8"> <script type='module' src='test.js'></script> </head> <body> <h1>Probando JavaScript</h1> </body> </html></pre>	<pre>import {suma, resta} from './mate.js'; console.log(suma(1,2), resta(1,2));</pre>	<pre>let suma = (a, b) => a + b; let resta = (a, b) => a - b; export { suma, resta };</pre>