



CEU

*Universidad
San Pablo*



JavaScript

Sistemas Web I

Grado en Ingeniería de Sistemas de Información

Álvaro Sánchez Picot

alvaro.sanchezpicot@ceu.es

v20221003

Introducción



- JavaScript (JS)
- Creado originalmente para que las webs estuvieran vivas
- No está relacionado con Java
- Los programas son scripts
- Basado en la [especificación ECMAScript \(ES\)](#)
- Hace falta un JavaScript engine para ejecutarlo:
 - V8 en Chrome y Opera
 - SpiderMonkey en Firefox
 - Chackra / ChackraCore en IE / Microsoft Edge
 - Nitro y SquirrelFish en Safari

Introducción

Características principales:

- Multiplataforma
- Orientado a objetos
- Completa integración con HTML / CSS
- Las cosas simples son sencillas
- Soporte de casi todos los navegadores
- Habilitado por defecto en los navegadores
- Tipado dinámico
- Tipado débil
- Versión de servidor: node.js

Introducción

- JS en el navegador no tiene acceso a funciones del SO
 - Está muy limitada la escritura de archivos
- Diferentes pestañas / ventanas en general no se conocen entre ellas
 - “Same Origin Policy”
- Una web puede comunicarse con el servidor del que vino
 - Está muy limitada la recepción de datos de otros dominios

Introducción

Lenguajes transpiled

- Lenguajes que tienen otra sintaxis y se convierten a JS
- Ejemplos:
 - [CoffeScript](#): sintaxis más corta
 - [TypeScript](#): tipos de datos strictos. Desarrollado por Microsoft
 - [Flow](#): tipos de datos. Desarrollado por Facebook
 - [Dart](#): lenguaje standalone. Desarrollado por Google.
 - [Brython](#): Python 3 transpiler para JS.

Introducción

Compatibilidad:

- No todas las funcionalidades están soportadas por todos los navegadores
- <https://caniuse.com/>
- <https://kangax.github.io/compat-table/es6/>

Introducción

- IDE
 - [Visual Studio Code](#)
 - [Webstorm](#)
- Editores ligeros
 - [Atom](#)
 - [Sublime Text](#)
 - [Notepad++](#)

Introducción

Consola de desarrollo:

- Google Chrome (Ctrl + Shift + I)
- Firefox, Edge (F12)
- Safari: Dentro de opciones > Avanzado, habilitar el menú de desarrollo.
Cmd + Opt + C

Introducción

- Código incrustado en el HTML

```
<script>  
  alert( 'Hello, world!' );  
</script>
```

- Script externo:

```
<script src="/path/to/script.js"></script>
```

Comentarios

- De una línea con //

```
//Esto es un comentario
```

- Multilínea con /* */ (No se pueden anidar)

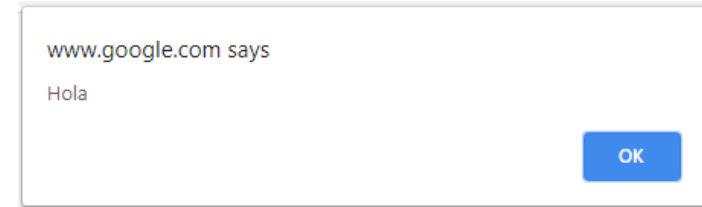
```
/*  
Comentario multilínea  
*/
```

Interacción

- alert

- Le muestra un mensaje al usuario

```
alert("Hola");
```



- prompt

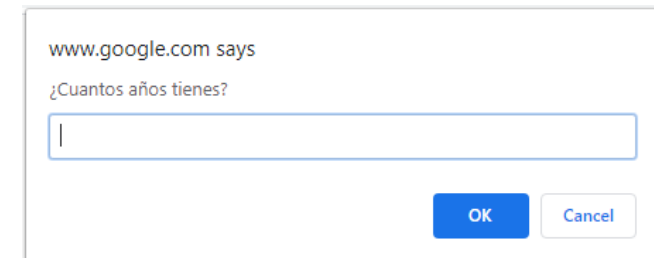
- Le pide al usuario que introduzca información

```
//prompt(title, [default]);
```

```
let result = prompt("¿Cuántos años tienes?", "");
```

```
console.log(result);
```

```
console.log(typeof(result)); //string
```



Interacción

- confirm

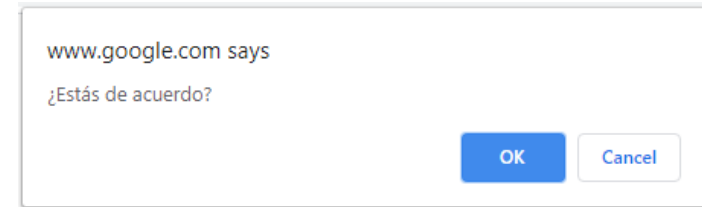
- Le solicita al usuario que confirme

```
//result = confirm(question);  
let str = confirm("¿Estás de acuerdo?");  
console.log(str) //true o false
```

- console.log

- Imprime un mensaje por la consola del navegador

```
//console.log(mensaje);  
console.log("Esto se imprime en la consola");
```



Esto se imprime en la consola

Statements

- Declaraciones
- Separadas por ';' aunque se puede omitir en un salto de línea, a veces (no recomendable)

```
alert('Hello'); alert('World');
```

```
alert('Hello')
```

```
alert('World')
```

```
alert('Hello');
```

```
alert('World');
```

Statements

- Statements

```
//Funciona correctamente  
console.log(3 +  
1  
+ 2);
```

```
//TypeError: Cannot read property '2' of undefined  
alert("There will be an error")  
[1, 2].forEach(alert)
```

- [Reglas ASI](#) (Automatic Semicolon Insertion)

Variables

- Almacenamiento de datos con nombre

```
let message;  
message = 'Hello!';  
console.log(message);
```

- Declaración múltiple:

```
let user = 'John',  
    age = 25,  
    message = 'Hello';
```

Variables

- Puede cambiar el tipo de datos, aunque mucho cuidado

```
let message;  
console.log(typeof(message)) //undefined  
message = 'Hello!';  
console.log(typeof(message)) //string  
message = 1234  
console.log(typeof(message)) //number
```


Variables

- Sintaxis
 - Solo pueden contener letras, dígitos o \$ o _
 - El primer carácter no puede ser un dígito
 - Se recomienda usar lower camelCase para múltiples palabras
unaVariableConUnNombreLargo
 - [Lista de palabras reservadas](#)

Constantes

- Variable que no cambia de valor

```
const myBirthday = '18.04.1982';
```

- No se puede reasignar su valor

```
const myBirthday = '23.05.1983';
```

```
//SyntaxError: Identifier 'myBirthday' has already been declared
```

- Tiene que asignarse un valor al declararlas

```
const errorAlDeclarar;
```

```
//SyntaxError: Missing initializer in const declaration
```

Constantes

- Para constantes conocidas recomendable que solo contengan mayúsculas y las palabras separadas por _

```
const COLOR_RED = "#F00";
```

- Para otras constantes usar lower camelCase

```
const pageLoadTime = /* time taken by a webpage to load */;
```

let vs var

- En general no usar var
- var no tiene scope de bloque

```
if (true) {  
    var test = true;  
}  
alert(test); // true, the variable lives after if
```

```
if (true) {  
    let test = true;  
}  
alert(test); // ReferenceError: test is not defined
```

let vs var

- var tiene scope de función

```
function sayHi() {  
  if (true) {  
    var phrase = "Hello";  
  }  
  console.log(phrase); // works  
}  
sayHi();  
console.log(phrase); // ReferenceError: phrase is not defined
```

let vs var

- var se puede redeclarar, let no

```
let user;  
let user;  
// SyntaxError: Identifier 'user' has already been declared  
----  
var user = "Pete";  
var user = "John";  
console.log(user); // John
```

let vs var

- Con var no hace falta poner var al declarar la variable

```
user = "Pete";  
console.log(user); // Pete
```

let vs var

- las variables var se pueden declarar después de su uso (hoisting)

```
//Válido en modo normal
```

```
numero = 1234;
```

```
console.log(numero);
```

```
-----
```

```
//No válido en modo estricto
```

```
'use strict'
```

```
numero2 = 1234;
```

```
console.log(numero2);
```

```
//ReferenceError: numero2 is not defined
```


let vs var

- las variables var se pueden declarar después de su uso (hoisting)

```
//Válido en modo estricto
```

```
'use strict'
```

```
numero3 = 1234;
```

```
console.log(numero3); //1234
```

```
var numero3;
```

```
-----
```

```
numero4 = 1234;
```

```
console.log(numero4);
```

```
let numero4;
```

```
//ReferenceError: Cannot access 'numero4' before initialization
```

Tipos de datos – Number

- Tanto para integers como coma flotante
- Valores enteros entre $-(2^{53}-1)$ y $(2^{53}-1)$
 - BigInt para valores fuera de ese rango (terminado en n)
`const bigInt = 1234567890123456789012345678901234567890n;`
- Valores numéricos especiales
 - Infinity, -Infinity
`console.log(1/0); //Infinity`
`console.log(-1/0); //-Infinity`
`let a = Infinity`
 - NaN (Not a number)
`console.log("texto" / 2); //NaN`

Tipos de datos – String

- Declaración

```
let str = "Hello";
```

```
let str2 = 'Single quotes are ok too';
```

- Template literals ([no soportado en Internet Explorer](#))

```
let str = `Hello`;
```

```
let phrase = `can embed another ${str}`;
```

```
let text = `the result is ${1 + 2}`
```

- No hay tipo caracter (char)

Tipos de datos – boolean

- Declaración

```
let nameFieldChecked = true;  
let ageFieldChecked = false;
```

- Pueden ser el resultado de una comparación

```
let isGreater = 4 > 1;
```

Tipos de datos – null

- Declaración

```
let age = null;
```

- No es una referencia a un objeto que no existe
- Valor especial que representa nada, vacío, valor desconocido
- typeof devuelve object por compatibilidad histórica aunque no lo sea

```
typeof(null); //object
```

Tipos de datos – undefined

- Representa que no se ha asignado ningún valor
- Declaración

```
let age;
```

```
console.log(age); //undefined
```

```
let nombre = undefined; //No deberíamos hacer esto
```

Tipos de datos – Conversiones

- Conversión a String usando String()

```
console.log(String("23")); //'23'
```

```
console.log(String(true)); //'true'
```

```
console.log(String(false)); //'false'
```

```
console.log(String(undefined)); //'undefined'
```

```
console.log(String(null)); //'null'
```

Tipos de datos – Conversiones

- Conversión a Number usando Number()

```
console.log(Number("23") + 1); //24
```

```
console.log(Number(" 23 ")); //23
```

```
console.log(Number("")); //0
```

```
console.log(Number("trece")); //NaN
```

```
console.log(Number(true)); //1
```

```
console.log(Number(false)); //0
```

```
console.log(Number(undefined)); //NaN
```

```
console.log(Number(null)); //0
```


Tipos de datos – Conversiones

- Conversión a Number usando +

```
let apples = "2";
```

```
let oranges = "3";
```

```
console.log(+apples + +oranges); //5
```

Tipos de datos – Conversiones

- Conversión a boolean usando Boolean()

```
console.log(Boolean(0)); //false
```

```
console.log(Boolean(1)); //true
```

```
console.log(Boolean(11234)); //true
```

```
console.log(Boolean("")); //false
```

```
console.log(Boolean("Hola")); //true
```

```
console.log(Boolean("0")); //true
```

```
console.log(Boolean(" ")); //true
```

```
console.log(Boolean(null)); //false
```

```
console.log(Boolean(undefined)); //false
```

Tipos de datos – Conversiones

- Conversiones explícitas

```
console.log("2" + 2); //22
```

```
console.log(4 + 5 + "px"); //9px
```

```
console.log("$" + 4 + 5); // $45
```

```
console.log("2" / "5"); //0.4
```

```
console.log(6 - "2"); //4
```

```
console.log(false + 34); //34
```

```
console.log(true + 34); //35
```

```
console.log(true + "34"); //true34
```

Operadores

- Suma +
- Resta -
- Multiplicación *
- División /
- Resto %
`console.log(5 % 2); //1`
- Exponente **
`console.log(2 ** 3); //8`
- Incremento / decremento ++/--

Operadores

- Operadores de bit
 - AND &
 - OR |
 - XOR ^
 - NOT ~
 - LEFT SHIFT <<
 - RIGHT SHIFT >>
 - ZERO-FILL RIGHT SHIFT >>>
- Orden de prioridad

Comparaciones

- Mayor que >
- Menor que <
- Mayor o igual que >=
- Menor o igual que <=
- Igual ==
- Distinto !=

Comparaciones

- Strings
 - Letra a letra
 - Según el orden [Unicode](#)

```
console.log('Z' > 'A'); // true
```

```
console.log('Glow' > 'Glee'); // true
```

```
console.log('Bee' > 'Be'); // true
```

```
console.log('a' > 'A'); // true
```

```
console.log('Á' > 'a'); // true
```

Comparaciones

- Tipos distintos
 - Se convierten a números

```
console.log('2' > 1); // true
```

```
console.log('01' == 1); // true
```

```
console.log(true == 1); // true
```

```
console.log(false == 0); // true
```

```
console.log('0' == 0); // true
```

```
console.log(Boolean('0') == Boolean(0)); // false
```

```
console.log(null == undefined); //true
```


Comparaciones

- Tipos distintos
 - Si no se puede convertir a número devuelve false

```
console.log('Hola' > 34); // false
```

```
console.log('Hola' < 34); // false
```

```
console.log('Hola' == 34); // false
```

Comparaciones

- Tipos distintos
 - Para que no se haga la conversión, usar la igualdad estricta ===
 - Si los tipos son distintos se devuelve false con ===

```
console.log('01' === 1); // false
```

```
console.log(true === 1); // false
```

```
console.log(false === 0); // false
```

```
console.log('0' === 0); // false
```

```
console.log(null === undefined) //false
```

Comparaciones

- Tipos distintos
 - Casos extraños con null y undefined comparado con 0
 - Tratar estos casos con cuidado

```
console.log(null > 0); // false
```

```
console.log(null == 0); // false
```

```
console.log(null >= 0); // true
```

```
console.log(undefined > 0); // false
```

```
console.log(undefined < 0); // false
```

```
console.log(undefined == 0); // false
```

Condicionales – if

- Se evalúa la expresión y se convierte a boolean

```
if (year == 2015) console.log('You are right!');
```

```
----
```

```
if (year == 2015) {  
    console.log('That's correct!');  
    console.log('You're so smart!');  
} else if (year == 2016){  
    console.log();  
} else {  
    console.log();  
}
```

Condicionales – if

- Operador ?

```
let accessAllowed = (age > 18) ? true : false;
```

```
let accessAllowed2 = age > 18 ? true : false; //No recomendado
```

```
let message = (age < 3) ? 'Hi, baby!' :
```

```
  (age < 18) ? 'Hello!' :
```

```
  (age < 100) ? 'Greetings!' :
```

```
  'What an unusual age!';
```

```
(company == 'Netscape') ?
```

```
  alert('Right!') : alert('Wrong.');// No recomendado
```

Condicionales – Operadores lógicos

- OR ||
 - Se evalúa de izquierda a derecha
 - Devuelve el primer el valor del primer operando que se evalúe a true
 - Si se llega al final, se devuelve el valor del último operando

```
let firstName = "";  
let lastName = "";  
let nickName = "SuperCoder";  
console.log( firstName || lastName || nickName || "Anonymous");  
// SuperCoder
```

- Cortocircuito

```
true || alert("not printed");  
false || alert("printed");
```

Condicionales – Operadores lógicos

- AND &&
 - Se evalúa de izquierda a derecha
 - Devuelve el primer el valor del primer operando que se evalúe a false
 - Si se llega al final, se devuelve el valor del último operando

```
alert( 1 && 0 ); // 0
```

```
alert( 1 && 5 ); // 5
```

```
alert( null && 5 ); // null
```

```
alert( 0 && "no matter what" ); // 0
```

- Cortocircuito

```
true && alert("printed");
```

```
false && alert("not printed");
```

Condicionales – Operadores lógicos

- NOT !

- Convierte el operando a booleano
- Devuelve el valor inverso

```
alert( !true ); // false
```

```
alert( !0 ); // true
```

- A veces se usa doble para convertir a booleano en vez de Boolean()

```
alert( !! "non-empty string" ); // true
```

```
alert( !!null ); // false
```


Condicionales – Operadores lógicos

- Nullish coalescing ??

- `a ?? b` → devuelve `a` si está definido (no es `null` ni `undefined`) y si no `b`
- Provee un valor por defecto a una variable
- No es compatible con todos los navegadores ([comprobar](#)).

```
result = (a !== null && a !== undefined) ? a : b;
```

```
result = a ?? b
```

- Por motivos de seguridad no se puede usar junto con `&&` o `||` sin paréntesis

```
let x = 1 && 2 ?? 3; // SyntaxError: Unexpected token '??'
```

- [Más información](#) sobre operadores

Condicionales – switch

- Reemplaza múltiples if con igualdad estricta ===
- Empieza ejecutando desde la primera condición true
- break para terminar la ejecución del switch
- default equivalente al else

```
switch (a) {  
  case 4:  
    console.log( 'Solo se ejecuta el 4' );  
    break;  
  case 5:  
    console.log( 'Se ejecuta el 5 y el default' );  
    break;  
  default:  
    console.log( "Default" );  
}
```

Condicionales – Ejercicio

- ¿Cuál es el resultado para cada uno?

```
alert( 1 && null && 2 );
```

```
alert( null || 2 && 3 || 4 );
```

```
if (-1 || 0) alert( 'first' );
```

```
if (-1 && 0) alert( 'second' );
```

```
if (null || -1 && 1) alert( 'third' );
```

```
alert( alert(1) && alert(2) );
```

Bucles

- while se ejecuta mientras la condición sea true

```
let i = 3;  
while (i) {  
    alert( i );  
    i--;  
}
```

```
let i = 3;  
while (i) alert(i--);
```

Bucles

- do..while se ejecuta al menos una vez

```
let i = 0;  
do {  
  console.log( i );  
  i++;  
} while (i < 3);
```

Bucles

- for

```
for (let i = 0; i < 3; i++) {  
    console.log(i);  
}  
console.log(i) //ReferenceError: i is not defined  
-----  
let i = 0;  
for (; i < 3;) {  
    alert( i++ );  
}
```

Bucles

- break
 - Fuerza la terminación del bucle

```
let sum = 0;
while (true) {
  let value = +prompt("Enter a number", '');
  if (!value) break;
  sum += value;
}
console.log( 'Sum: ' + sum );
```

Bucles

- continue
 - Fuerza al bucle a seguir con la siguiente iteración

```
for (let i = 0; i < 10; i++) {  
  if (i % 2 == 0) continue;  
  console.log(i); // 1, then 3, 5, 7, 9  
}
```


Bucles

- etiquetas (labels)
 - Identifica el bucle
 - Referencias para break / continue
 - No se pueden usar para saltar a cualquier lado

outer:

```
for (let i = 0; i < 3; i++) {  
  for (let j = 0; j < 3; j++) {  
    let input = prompt(`Value at coords (${i},${j})`, '');  
    if (!input) break outer;  
    // do something with the value...  
  }  
}  
  
alert('Done!');
```

Funciones

- Principales bloques de construcción
- Reutilización del código
- Las variables declaradas dentro no son accesibles fuera

```
function showMessage() {  
    let mensaje = 'Hello everyone!';  
    console.log(mensaje);  
}  
showMessage();  
showMessage();  
console.log(mensaje); //ReferenceError: mensaje is not defined
```

Funciones

- Las variables exteriores son accesibles y modificables dentro

```
let userName = 'John';  
function showMessage() {  
    userName = "Bob";  
    let message = 'Hello, ' + userName;  
    console.log(message);  
}  
console.log( userName ); // John  
showMessage();  
console.log( userName ); // Bob
```

Funciones

- Si se declara la misma variable dentro y fuera, la de dentro oscurece (shadows) la de fuera
- Sucede lo mismo en condicionales y bucles

```
let userName = 'John';  
function showMessage() {  
    let userName = "Bob";  
    let message = 'Hello, ' + userName;  
    console.log(message);  
}  
console.log( userName ); // John  
showMessage();  
console.log( userName ); // John
```

Funciones – Parámetros

- Se puede llamar a la función con menos parámetros y entonces se consideran undefined

```
function showMessage(from, text) {  
    console.log(from + ': ' + text);  
}  
showMessage('Ann', 'Hello!'); //Ann: Hello!  
showMessage("Ann"); //Ann: undefined
```

Funciones – Parámetros

- Se puede incluir un valor por defecto

```
function showMessage(from, text = "no hay texto") {  
    alert( from + ": " + text );  
}  
  
showMessage('Ann', 'Hello!'); //Ann: Hello!  
showMessage("Ann"); // Ann: no hay texto
```

Funciones – Parámetros

- El valor por defecto puede ser una llamada a otra función

```
function showMessage(from, text = anotherFunction()) {  
    // anotherFunction() only executed if no text given  
    // its result becomes the value of text  
}
```

Funciones – Parámetros

- Múltiples parámetros

```
function foo() {  
    for (let i = 0; i < arguments.length; i++) {  
        console.log(arguments[i]);  
    }  
}  
  
foo('Hola');  
foo(1,2,3,4,5,6,7,8,9);
```


Funciones – Parámetros

- Múltiples parámetros

```
function my_log(x, ...args) {  
    console.log(x, args, ...args);  
}  
my_log('Hola', 'qué', 'tal');
```

Funciones – return

- Para devolver un resultado en la función
- Si no existe, la función devuelve undefined
- Se puede usar vacío (return;) para terminar la ejecución de la función

```
function checkAge(age) {  
    if (age >= 18) {  
        return true;  
    }  
    return confirm('¿Tienes permiso?');  
}  
  
let age = prompt('How old are you?', 18);  
let valido = checkAge(age);
```

Funciones – Expresiones

- Se puede asignar una función a una variable
- Añadir el ; al final

```
let sayHi = function() {  
    console.log( "Hello" );  
};  
console.log(sayHi); //se imprime el código de la función  
console.log(sayHi());// Hello\nundefined
```

Funciones – Expresiones

- Una declaración de función se puede usar antes, una expresión, no

```
sayHi("John"); // Hello, John
```

```
saludar("John"); // ReferenceError: saludar is not defined
```

```
function sayHi(name) {
```

```
    alert( `Hello, ${name}` );
```

```
}
```

```
let saludar = function(name) {
```

```
    alert( `Hello, ${name}` );
```

```
};
```

Funciones – Arrows

- Equivalente a funciones lambda
- Funciones anónimas
- Forma concisa de declarar una función

```
let sum = (a, b) => a + b;
```

- Código casi equivalente a

```
let sum = function(a, b) {  
    return a + b;  
};
```

Funciones – Arrows

- Pueden usar múltiples líneas con {}, en ese caso necesitan return

```
let sum = (a, b) => {  
  let result = a + b;  
  return result;  
};  
console.log( sum(1, 2) ); // 3
```

Arrays

- Crear arrays:

```
let fruits = ['Apple', 'Banana'];
```

- Acceder a un elemento

```
console.log(fruits[0]);
```

- Longitud

```
console.log(fruits.length);
```

- Recorrerlo

```
fruits.forEach(function(item, index, array) {  
    console.log(item, index);  
});
```

Arrays

- Añadir un elemento

```
let newLength = fruits.push('Orange');
```

- Eliminar el último elemento

```
let last = fruits.pop();
```

- Buscar un índice de un elemento

```
let pos = fruits.indexOf('Banana');
```

- Eliminar un elemento

```
let removedItem = fruits.splice(pos, 1);
```

- [Más información](#)

Objetos

- Almacenar colecciones de varios datos

```
let user = new Object(); // "object constructor" syntax
```

```
let user = {}; // "object literal" syntax
```

- Se pueden inicializar con datos mediante clave: valor separados por coma

```
let user = {  
  name: "John",  
  age: 30  
};
```

- Para acceder a la información

```
console.log(user.name);  
console.log(user["name"]);
```

Objetos

- Se puede añadir nuevas propiedades

```
let user = {  
  name: "John",  
  age: 30  
};  
user.isAdmin = true;
```

- Se pueden borrar las propiedades con delete
delete user.age;

Objetos

- Una propiedad puede tener más de una palabra, pero entonces hay que usar comillas

```
let user = {  
  name: "John",  
  age: 30,  
  "likes birds": true  
};
```

- A esas propiedades hay que acceder con []

```
console.log(user["likes birds"]);  
console.log(user["age"]);
```

Objetos

- Comprobar si existe una propiedad

```
let user = {};
```

```
console.log(user.noSuchProperty === undefined); // true
```

- Se puede usar el operador *in* que devuelve true si existe

```
console.log("noSuchProperty" in user); // false
```

```
let key = "age";
```

```
console.log(key in user); // false
```

Objetos

- Recorrer objetos

```
let user = {  
  name: "John",  
  age: 30,  
  isAdmin: true  
};  
for (let key in user) {  
  alert(key); // name, age, isAdmin  
  alert(user[key]); // John, 30, true  
}
```

Objetos

- Se pueden anidar los objetos

```
let user = {  
  name: "John",  
  birthday: {  
    year: 1990,  
    month: "November"  
  }  
};  
console.log(user.birthday.year);  
console.log(user["birthday"]["month"]);
```

API

- Hay un montón de interfaces y APIs predefinidas ([lista](#))
- Propiedad [onload](#)
 - Para cuando el recurso se ha cargado

```
window.addEventListener('load', (event) => {  
    init();  
});  
//Alternativamente pero no recomendable  
window.onload = function() {  
    init();  
};
```

API

- Hay un montón de interfaces y APIs predefinidas ([lista](#))
 - Propiedad [onclick](#)

```
<button id="boton">Click me</button>
```

```
-----
```

```
var element = document.getElementById("boton");  
element.addEventListener("click", myScript);  
//Alternativamente pero no recomendable  
element.onclick = function(){myScript};
```


API

- Acceder al documento

```
let markup = document.documentElement.innerHTML;
```

- Acceder a elementos

```
let myElement = document.getElementById("intro");
```

```
let x = document.getElementsByTagName("p");
```

```
let x = document.getElementsByClassName("intro");
```

```
let x = document.querySelectorAll("p.intro");
```

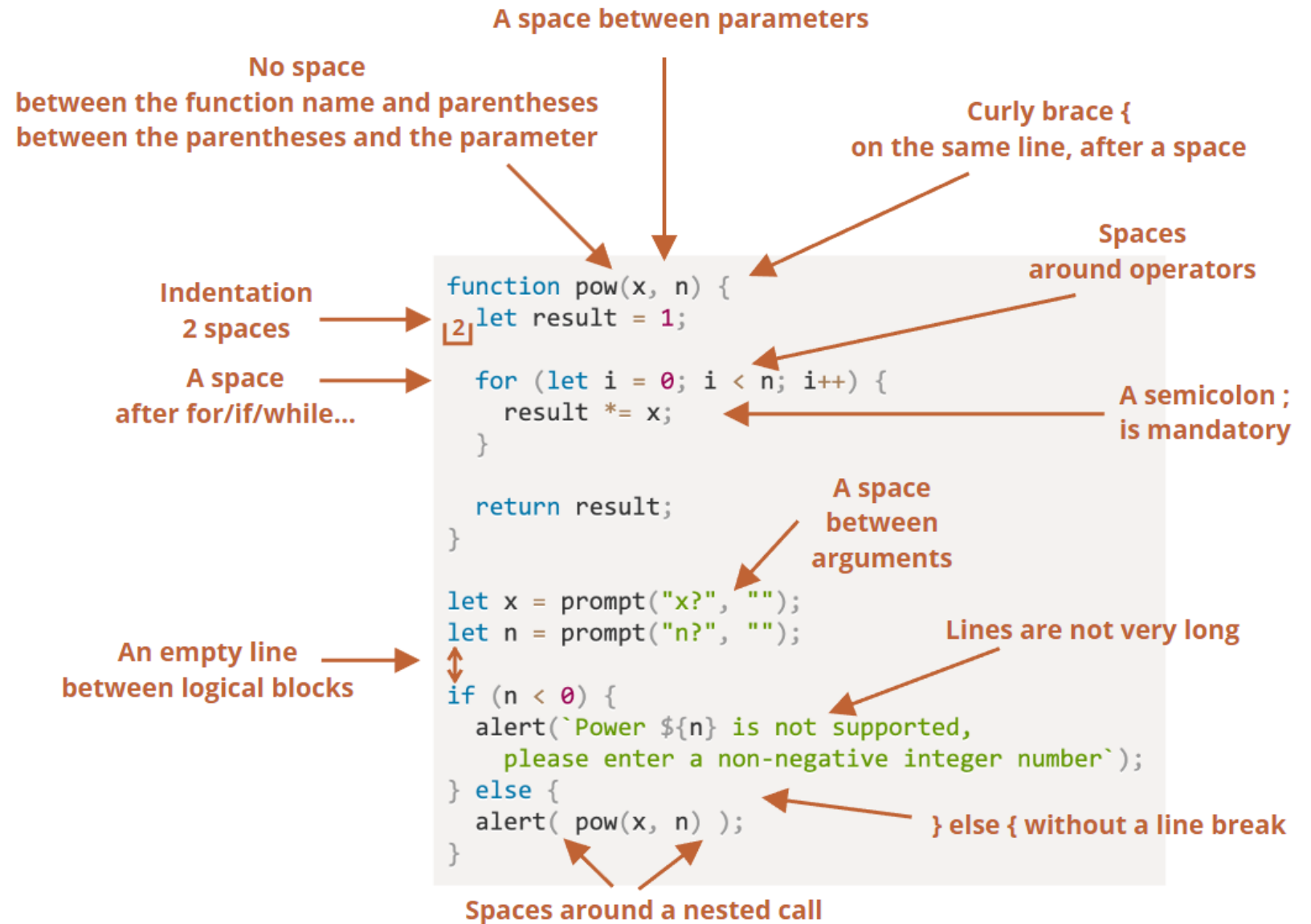
- Añadir elementos nuevos

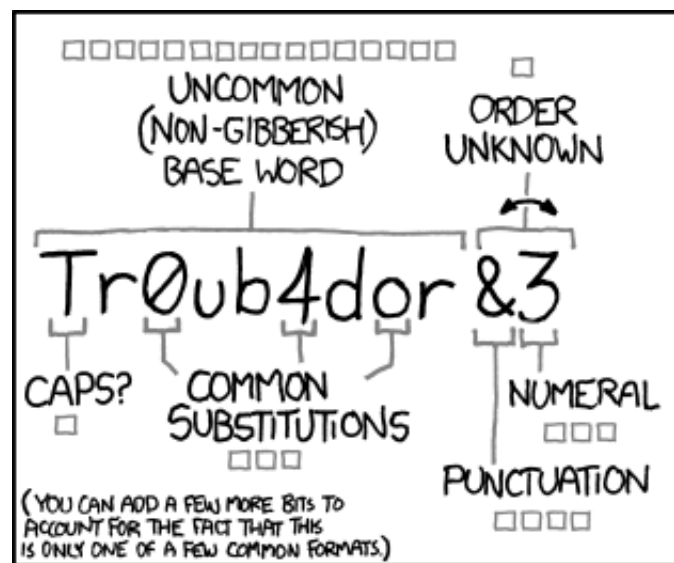
```
let element = document.getElementById("new");
```

```
element.appendChild(tag);
```

Estilo

Fuente





~28 BITS OF ENTROPY

$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$

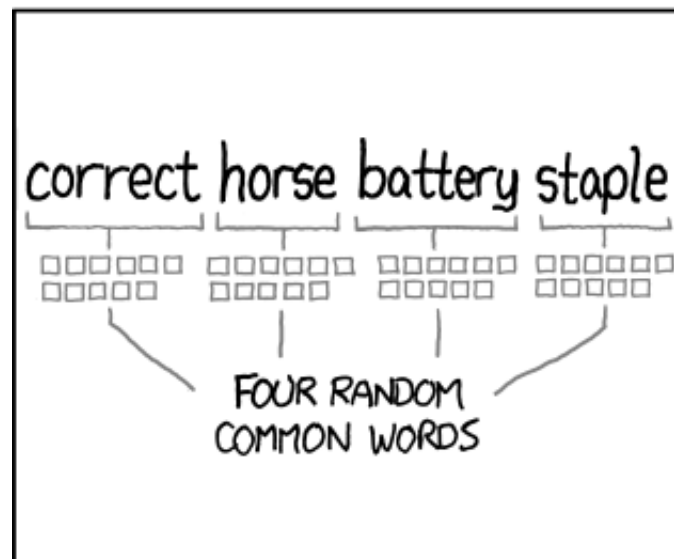
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

DIFFICULTY TO GUESS: **EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?

AND THERE WAS SOME SYMBOL...

DIFFICULTY TO REMEMBER: **HARD**



~44 BITS OF ENTROPY

$2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$

DIFFICULTY TO GUESS: **HARD**

THAT'S A BATTERY STAPLE.

CORRECT!

DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT

THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

JS – Ejercicio 1



- Diseña una web que genere una contraseña
- Busca un diccionario y guarda la información en una variable
- Usa el diccionario para generar la contraseña con varias palabras aleatorias
- Añade alguna opción de configuración:
 - Número de palabras
 - Comenzar cada palabra en mayúsculas
 - No repetir palabras
 - ...

JS – Ejercicio 1

Número de palabras (1-10):

Generar Contraseña

Número de palabras (1-10):

Generar Contraseña

La contraseña generada es: AcrataAfacaAbsitAgape

JS – Ejercicio 2



Esta página web evita que seleccionemos el texto

- ¿Qué está pasando?
- ¿Cómo lo evitamos?
- ¿Por qué no aparece el cursor de seleccionar texto?

Callback

- Funciones pasadas como argumento a otra función
- Se invocan cuando se completa alguna acción o rutina
- Pueden ser síncronas
- Habitualmente se usan asíncronamente

```
function greeting(name) {  
    alert('Hello ' + name);  
}  
  
function processUserInput(callback) {  
    let name = prompt('Please enter your name.');
```

callback(name);

```
}  
  
processUserInput(greeting);
```

Input – Teclado

- Eventos del teclado:
 - keydown: Cuando se pulsa una tecla
 - keypress: Mientras se pulsa (múltiples invocaciones). Deprecado
 - keyup: cuando se libera la tecla

```
document.addEventListener('keydown', logKey);  
function logKey(e) {  
    console.log(e.code);  
}
```


Input – Ratón

- Eventos del ratón:
 - `click`: cuando se pulsa y luego se libera el botón del ratón
 - `dblclick`: doble click
 - `mouseup`: cuando se libera el botón del ratón
 - `mousedown`: cuando se pulsa el botón del ratón
 - `mousemove`: mientras se mueve el ratón (múltiples invocaciones)

```
document.addEventListener('click', click);  
function click(e) {  
    console.log(e);  
}
```

Canvas

- Elemento HTML que nos permite dibujar gráficos

```
<canvas id="myCanvas" width="500" height="500">fallback content</canvas>
```

- Por defecto es un rectángulo blanco
- `fallback content`: Contenido recomendable añadir para que se muestre en navegadores antiguos que no soporten la etiqueta (pej. IE < 9)
- Necesita la etiqueta de cierre
- Para trabajar con canvas con JS necesitamos el contexto 2d:

```
var canvas = document.getElementById("myCanvas");
```

```
var ctx = canvas.getContext("2d");
```

Canvas – Plantilla

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Canvas</title>
</head>
<body>
<canvas width="500" height="500" id="myCanvas"></canvas>
<script type="text/javascript">
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
</script>
</body>
</html>
```

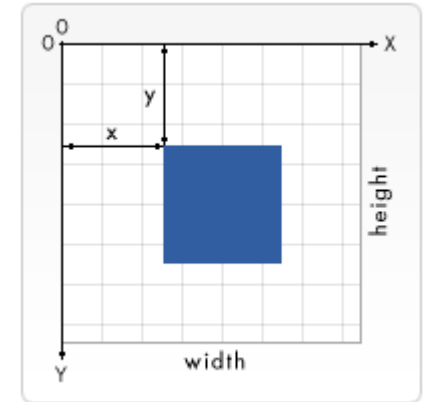
Canvas – formas

- Dibujar una línea:

```
ctx.moveTo(0, 0); //Arriba a la izquierda  
ctx.lineTo(200, 100); //Coordenadas en px  
ctx.stroke();
```

- Dibujar una figura:

```
ctx.moveTo(100, 200);  
ctx.lineTo(200, 200);  
ctx.lineTo(150, 100);  
ctx.lineTo(100, 200);  
ctx.stroke();  
//ctx.fill(); //rellena
```



Canvas – formas

- Dibujar un rectángulo:

```
//strokeRect(x, y, width, height)  
ctx.strokeRect(50, 50, 100, 50);
```

- Dibujar un rectángulo relleno:

```
//fillRect(x, y, width, height)  
ctx.fillRect(50, 50, 100, 50);
```

- Borrar un rectángulo:

```
//clearRect(x, y, width, height)  
ctx.fillRect(50, 50, 100, 50);  
ctx.clearRect(60, 60, 80, 30);
```



Canvas – formas

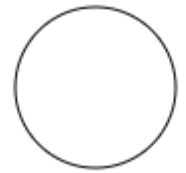
- Dibujar un círculo

```
//arc(x, y, radius, startAngle, endAngle, counterclockwise)
```

```
ctx.beginPath();
```

```
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
```

```
ctx.stroke();
```



- Dibujar un círculo relleno:

```
ctx.beginPath();
```

```
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
```

```
ctx.fill();
```



Canvas

- Añadir texto:

```
ctx.font = "30px Arial";  
ctx.fillText("Hello World", 10, 50);
```

Hello World

- Añadir strokeText:

```
ctx.font = '48px serif';  
ctx.strokeText('Hello world', 10, 50);
```

Hello world

Canvas

- Definir color

```
//fillStyle = color  
//strokeStyle = color  
ctx.fillStyle = 'orange';  
ctx.strokeStyle = 'blue';  
ctx.beginPath();  
ctx.arc(95, 50, 40, 0, 2 * Math.PI);  
ctx.stroke();  
ctx.fill();
```



Canvas

- Imágenes

```
//drawImage(image, x, y)
//drawImage(image, x, y, width, height)
let img = new Image();
img.onload = function() {
    ctx.drawImage(img, 0, 0);
};
img.src = 'image.png';
```

Canvas

- Y muchas más funcionalidades:
 - Transformaciones
 - Gradientes
 - Modificación de imágenes
 - Manipulación de píxeles
 - Efectos
 - Gráficos 3D
 - ...
- Y muchas librerías para facilitar el trabajo

Canvas - Animaciones

- Para dibujar un frame necesitamos:
 1. Limpiar el canvas
 2. Dibujar los elementos que queramos
- Para controlar las animaciones:
 - `setInterval(function[, delay]);`
 - Para invocar una función cada delay milisegundos
 - [Más información](#)
 - `setTimeout(function[, delay];`
 - Invoca una función una sola vez dentro de delay milisegundos
 - [Más información](#)

Canvas - Animaciones

- Para controlar las animaciones:
 - `requestAnimationFrame(callback);`
 - Ejecuta una animación antes del siguiente repaint del navegador
 - Se ejecuta una sola vez
 - La tasa de refresco es unas 60 veces por segundo
 - La tasa de refresco puede variar según el display
 - La tasa de refresco puede ser más lenta si el navegador está sobrecargado

Canvas - Animaciones

```
let canvas = document.getElementById("myCanvas");
let ctx = canvas.getContext("2d");
let x = 0, y = 0, speed = 4;
const width = 20, height = 20;

window.addEventListener("load", (event) => {
  init();
});

function init(){
  window.requestAnimationFrame(draw);
}

function draw(timestamp){
  console.log(timestamp);
  if(x < 0 || x > canvas.width - width){
    speed *= -1;
  }
  x += speed;
  ctx.clearRect(0, 0, canvas.width, canvas.height); //Limpiar el canvas
  ctx.fillRect(x, y, width, height);
  window.requestAnimationFrame(draw);
}
```

Canvas – Ejemplos

- [Sistema Solar](#)
- [Reloj](#)
- [Snake](#)
- [Tanks](#)
- [Basic Raycaster](#)

Canvas – Ejercicio



- Crea alguna animación o juego con canvas
- Experimenta con diferentes elementos que hemos visto
- Añade alguna imagen
- Reacciona al input del usuario
- Dibujar la animación independientemente del ratio de la tasa de refresco
- Puedes buscar más información sobre canvas [aquí](#)

Referencias

- Tutoriales:

<https://javascript.info/>

<https://www.w3schools.com/js/>

- Manuales:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

- API:

<https://developer.mozilla.org/en-US/docs/Web/API>

https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

JQUERY

Introducción

- Librería JavaScript
- fast, small and feature-rich
- Recorrer y manipular HTML y CSS
- Gestión de eventos
- Efectos y animaciones
- Ajax
- Compatible con muchos navegadores
- 70M+ de webs lo usan ([info](#))
- Web: <https://jquery.com/>



Introducción

- Se puede descargar ([link](#))
 - Archivos sin comprimir (.js), comprimido (.min.js), Map files (.min.map)
- Enlace a un CDN
 - Añadirlo al final de body

```
<body>
```

```
<!-- Contenido del body -->
```

```
<script
```

```
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

```
<!-- Aquí los scripts que usen jQuery -->
```

```
</body>
```

Plantilla

```
<!doctype html>
<html><head><meta charset="utf-8"><title>Demo</title>
<style type="text/css">
  p{padding: 10px;}
  #p1{padding: 150px;
    background-color: #e5eccc;}
  .rojo{color: red;}
</style>
</head><body>
  <h1>Prueba jQuery</h1>
  <p id="p1">Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam vel orci congue, malesuada lacus non, ultrices lorem. Aenean eu lorem rhoncus, luctus lectus sed, tempus erat. Pellentesque dignissim ex sit amet ullamcorper volutpat. Vestibulum luctus neque justo. Quisque in magna sit amet tellus porttitor sodales. Mauris condimentum mauris in commodo vehicula. Suspendisse commodo augue nec vulputate pellentesque. Aliquam non dignissim risus. Suspendisse condimentum hendrerit eros, id dapibus nibh. Sed ac ligula sapien. Morbi nec dolor urna. Duis vestibulum fringilla turpis, quis feugiat lacus consequat vel. Ut a nunc quis dui aliquam eleifend et vitae magna. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vivamus aliquam nulla est, non ornare ex malesuada in.</p>
  <p id="p2" class="rojo">Quam lacus suspendisse faucibus interdum posuere lorem ipsum dolor sit. Ultrices eros in cursus turpis massa tincidunt dui. Dolor sit amet consectetur adipiscing elit pellentesque habitant morbi tristique.</p>
  <p id="p3">Párrafo 3</p>
  <div><a href="https://www.google.es/">Enlace</a></div>
  <div><input type="text"></div>
  <div><button id="boton1">Botón 1</button>
  <button id="boton2">Botón 2</button></div>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</body></html>
```

Uso

- Acceso a jQuery mediante \$
 - Selector: elemento HTML sobre el que queremos actuar
 - Acción: acción a realizar sobre el elemento
- `$(selector).action()`
-
- jQuery también tiene acciones que no requieren selector
- `$.action()`

Selectores

- Window

`$(window)`

- Document

`$(document)`

- This

`$(this)`

- Selectores CSS (revisar sección CSS)

`$("p")`

`$(".rojo")`

`$("#p1")`

Selectores

- `:contains(texto)`: elementos que contengan texto
 - Cuidado que hay que alternar las comillas entre simples, dobles y template literals (estas últimas no pueden ir dentro)

```
$(":contains('ipsum')");
```

```
$(':contains("ipsum")');
```

```
$(`:contains("ipsum")`);
```

```
$(":contains(`ipsum`)"); //No devuelve el resultado esperado
```



- Se puede usar con otros selectores antes de los dos puntos

```
$("p:contains('ipsum')");
```

Selectores

- [Más información sobre los selectores](#)

Eventos

- `.click(handler)`: Se ejecuta la función handler cuando se hace click sobre el elemento

```
$("#p").click(function(){  
    console.log("Has pulsado ", $(this));  
});  
-----  
function info(){  
    console.log("Has pulsado ", $(this));  
}  
$("#boton1").click(info);
```

Eventos

- Cuidado con las funciones con argumentos

```
function info(mensaje){  
    console.log(mensaje);  
}
```

```
$("p").click(info("Hola"));
```

- Hay que usar una función anónima como wrapper

```
$("p").click(function(){  
    info("Hola");  
});
```



Eventos

- `$(handler)`: Se ejecuta cuando el Document Object Model ha terminado de cargar
 - Punto de comienzo para ejecutar JS
 - Antiguamente `$(document).ready(handler)` (Deprecated)
 - Pueden no haberse cargado los assets (imágenes, vídeos...)
 - [Más información](#)

```
$(function(){  
    console.log("Bienvenido");  
});
```

Eventos

- `.on("load", handler)`: Se ejecuta handler cuando se ha terminado de cargar el elemento.
 - En vez de load, otros nombres de evento: click, keyDown...
 - [Más información](#)

```
$(img).on("load", function(){  
    console.log("Se ha cargado la imagen");  
});
```

Eventos

- `$(window).on("load", handler)`: handler se ejecuta cuando se han terminado de cargar los assets de la página
 - Cuidado porque puede haber pasado unos segundos en los que el usuario ha interactuado con la página

```
$(window).on("load", function(){  
    console.log("Bienvenido. Ya está todo listo.");  
});
```

Eventos

- `.on(handler)`: Permite asociar múltiples acciones

```
$("#p").on({  
  mouseenter: function(){  
    $(this).css("background-color", "green");  
  },  
  mouseleave: function(){  
    $(this).css("background-color", "lightblue");  
  },  
  click: function(){  
    $(this).css("background-color", "yellow");  
  }  
});
```

Eventos

- Múltiples eventos de ratón, teclado, formularios...
- [Lista de eventos](#)

<u>Ratón</u>	<u>Teclado</u>	<u>Formularios</u>
.click()	.keydown()	.blur()
.dblclick()	.keypress()	.focus()
.hover()	.keyup()	.select()
.mousedown()		.submit()
.mouseup()		
.mouseenter()		

Eventos

- Todos los handlers disponen de información del evento que lo activó en un objeto evento
 - [Información](#)

```
$("#p").click(function(e){  
    console.log(e);  
});
```


Eventos

- Propiedades del objeto evento

`e.pageX, e.pageY` //Posición del ratón

`e.type` //Tipo de evento

`e.target` //El elemento HTML que inició el evento

`e.timestamp` //Tiempo desde que se cargó la página

- Funciones del objeto evento

`e.preventDefault()` //Previene la acción por defecto

`e.stopPropagation()` //Evita que el evento se propague a los hijos

CSS

- `.css(propertyname)`: Devolver el valor de una propiedad CSS
 - En caso de que haya multiples elementos solo devuelve el del primero
 - [Info](#)

```
$("#p").css("background-color");
```

- `.css(propertyname,value)`: Cambiar el valor de una propiedad
- ```
$("#p").css("background-color", "yellow");
```

# CSS

- `.css({"propertyname": "value", "propertyname": "value", ...}):`  
Actualizar múltiples propiedades CSS a la vez

```
$("#p1").css({"background-color": "yellow",
 "color": "green",
 "font-size": "24px"
});
```

# CSS

- [.addClass\(clase\)](#) / [.removeClass\(clase\)](#): añade / elimina la clase de los elementos HTML

```
$("#p1").addClass("rojo");
```

- [.toggleClass\(clase\)](#): añade la clase a los elementos si no la tienen o la elimina si ya la tienen

```
$("#p").toggleClass("rojo");
```

# CSS

- Más comandos para manipular el CSS:
  - `.hasClass()`
  - `.height()`
  - `.width()`
  - `.position()`

# Efectos

- `.hide()`: Oculta el elemento
- `.show()`: Muestra el elemento

```
$("#boton1").click(function(){
 $("#p1").hide();
});
$("#boton2").click(function(){
 $("#p1").show();
});
```

# Efectos

- `.toggle()`: Oculta el elemento si está visible y lo muestra si está oculto

```
$("#boton1").click(function(){
 $("p").toggle();
});
```

# Efectos

- `.fadeIn()`: Muestra el elemento con efecto fade
- `.fadeOut()`: Oculta el elemento con efecto fade
- `.fadeToggle()`: Muestra/oculta el elemento con efecto fade

```
$("#p1").hide();
$("#boton1").click(function(){
 $("#p1").fadeIn();
 $("#p2").fadeOut();
 $("#p3").fadeToggle(5000);
});
```



# Efectos

- `slideDown()`: Desliza el elemento hacia abajo para mostrarlo
- `slideUp()`: Desliza el elemento hacia arriba para ocultarlo
- `slideToggle()`: Desliza el elemento hacia arriba/abajo

```
$("#boton1").click(function(){
 $("#p1").slideToggle();
});
```

# Efectos

- Parámetros de los efectos

```
$(selector).hide(speed,callback);
```

- speed
  - "slow"
  - "fast"
  - Un número que representa los ms
- callback: Función que se ejecuta cuando termina el efecto

```
$("#p1").hide(1000);
```

```
$("#p").toggle("slow", function(){
 console.log("Terminó el efecto");
});
```

# Efectos

- Se pueden encadenar múltiples efectos (chaining)
- Cuando termina uno se ejecuta el siguiente, excepto los que son instantáneos

```
$("#p1").css("color", "red")
 .slideUp(3000)
 .slideDown(3000);
```

- Varios efectos más
  - [Información](#)

# Manipulación del DOM

- .html(): devuelve el contenido del primer elemento HTML

```
console.log($("#body").html());
```

```
console.log($("#div").html());
```

- .html(contenido): modifica el contenido de todos los elementos HTML

```
$("#div").html("Sin contenido");
```

# Manipulación del DOM

- .text(): devuelve el texto combinado de todos los elementos y sus descendientes

- Ignora las etiquetas html

```
console.log($("#body").text());
```

- .text(texto): modifica el texto de todos los elementos HTML

- Trata las etiquetas html como texto

```
$("#p").text("Sin contenido");
```

# Manipulación del DOM

- .val(): devuelve el valor del primer elemento
  - Se usa con elementos de formulario (input, select, textarea...)

```
console.log($("#input").val());
```

- .val(valor): modifica el valor de todos los elementos HTML
- ```
$("#input").val("Prueba el texto");
```

Manipulación del DOM

- `.attr(nombreAtributo)`: devuelve el valor de un atributo HTML
`console.log($("#a").attr("href"));`
- `.attr(nombreAtributo, valor)`: modifica el valor del atributo
`$("#a").attr("href", "https://jquery.com/");`

Manipulación del DOM

- [.prepend\(contenido\)](#) / [.append\(contenido\)](#): añade contenido al principio / final dentro cada elemento HTML

```
$("#p1").prepend("<span>Al principio</span>");
```

```
$("#p1").append("<span>Al final</span>");
```

- Permiten añadir múltiple contenido

```
let txt1 = "<h1>Un título</h1>";
```

```
let txt2 = "<h1>Otro título</h1>";
```

```
$("#body").append(txt1, txt2);
```


Manipulación del DOM

- [.before\(contenido\)](#) / [.after\(contenido\)](#): añade contenido al antes / detrás de cada elemento HTML

```
$("#p1").before("<h2>Título</h2>");
```

```
$("#p1").after("<span>Comentario final</span>");
```

- Permiten añadir múltiple contenido

```
let txt1 = "<h2>Un título</h2>";
```

```
let txt2 = "<p>Párrafo de introducción</p>";
```

```
$("#p1").before(txt1, txt2);
```

Manipulación del DOM

- .remove(): elimina los elementos HTML y todo su contenido

```
$("#p").remove();
```

- .remove(selector): elimina los elementos HTML que cumplan el selector

```
$("#p").remove(":contains('Lorem')");
```

- .empty(): vacía los elementos HTML

```
$("#p").empty();
```

Manipulación del DOM

- .children(): Devuelve los hijos de los elementos HTML

```
$("body").children();
```

- .parent(): devuelve el elemento padre de los elementos HTML

```
$("a").parent();
```

- .siblings(): devuelve los hermanos de los elementos HTML

```
$("#p1").siblings();
```

Manipulación del DOM

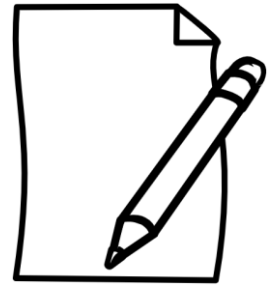
- [Más comandos para manipular el DOM](#)

jQuery – Ejercicio



- Crea una página principal de una web con el contenido que sea
- Añade un botón para que se registre el usuario:
 - Que oculte el contenido del container principal
 - Que aparezca un formulario que pida introducir:
 - Nombre y apellidos
 - email
 - teléfono
 - contraseña
 - Botón para enviar los datos
 - Botón de cancelar para eliminar el formulario y volver a mostrar el contenido principal

jQuery – Ejercicio



- Valida los campos cuando se pulse el botón
 - Nombre y apellidos
 - Que haya al menos dos palabras
 - Solo letras (extra: aceptar acentos)
 - email
 - Formato válido
 - teléfono
 - solo números, sin espacios
 - contraseña
 - Al menos 8 caracteres con al menos una mayúscula, una minúscula y un número
- Si todo es válido, que le aparezca un mensaje al usuario confirmándolo

Referencias

- Tutoriales:

<https://javascript.info/>

<https://www.w3schools.com/jquery/>

<https://learn.jquery.com/>

- API:

<https://api.jquery.com/>

- Listado de eventos:

https://www.w3schools.com/jquery/jquery_ref_events.asp