Características

- Lenguaje de alto nivel
  - No nos preocupamos por ej. de gestión de memoria
- Orientado a objetos
  - Basado en objetos para almacenar distintos tipos de datos
- Multiplataforma, cliente - servidor
- Lenguaje interpretado
  - No se compila
  - Errores en tiempo de ejecución

Rol de JavaScript en el desarrollo web

HTML -> Maquetación, semántica (nombres)

```
<p>Párrafo</p>
```

CSS -> Diseño (adjetivos)

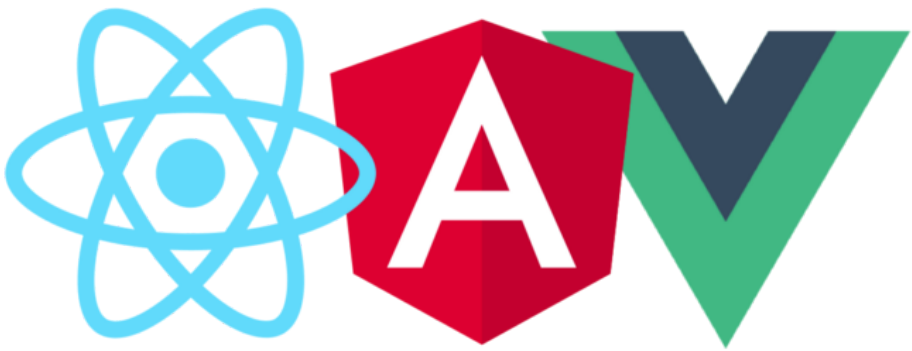
```
p {color: red};
```

JS -> Comportamiento (verbos o acciones)

```
p.hide();
```

Cliente web

- JavaScript
- Framewords de JavaScript
    - Están basados 100% en JavaScript
    - Pueden desaparecer, pero no JS
    - Algunos ejemplos:



- Angular
- React
- Vue

Aplicaciones móviles

- JS
- Ionic
- React native

Aplicaciones en servidor

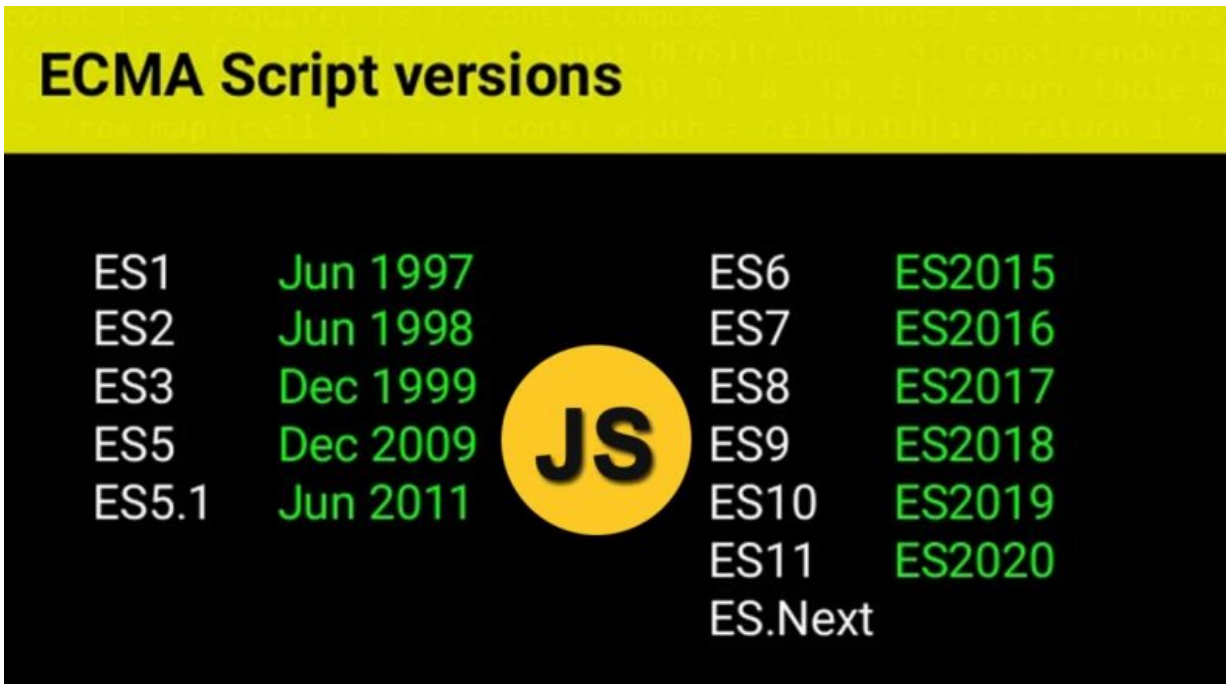
- node.js

Aplicaciones nativas de escritorio

- Electron
    - Ej. Visual Studio Code

Versiones de JavaScript

- Nos vamos a centrar en JS moderno, sin olvidar lo anterior.
- Las características de versiones antiguas siempre funcionan (backwards compatible)
- Las características nuevas funcionarán en función del motor de JS del navegador



ES.next

- Stage 1, 2, 3 ,4
- Los navegadores ya empiezan a implementar las opciones cuando están en Stage 3

¿Cómo desarrollamos en JS?

- Desarrollo:
    - Utilizamos la última versión de Google Chrome
    - Developer tools (F12)
    - Visual Studio Code con vitaminas:
        - eslint
        - prettier
        - Quokka.js
- Producción:
    - Babel para transpilar y polyfill del código a ES5
    - Compatibilidad con todos los navegadores (ver https://kangax.github.io/compat-table/es6/)
- Herramienta Obsidian (second brain)

Prácticas propuestas

Práctica 1 (4h)- juego adivinar un número

Manejar el DOM

Práctica 2 (4h)- app cuenta bancaria

Trabajar con datos (funciones como map, reduce, filter, some, every...).

Práctica 3 (4h)- app cuenta bancaria

Promesas, async-await... programación asíncrona.

Práctica 4 (4h)- app cuenta bancaria

Práctica con API ARASAAC repasando cosas anteriores.

Configuración entorno

Escribimos un Quokka.js con un código similar al siguiente:

```
console.log('kkk');
// el DOM devuelve texto!!!!!
const numIntentos = "5"
// let intentosRestantes = '5'
console.log('Llevas', numIntentos, "intentos")
console.log('Llevas '+ numIntentos + ' intentos')

// ES6
// LITERAL STRINGS
console.log(`Llevas ${numIntentos} intentos`)
// variable tipo texto pero ha permitido la
resta!!! -> type coarcing
console.log(`Te quedan ${numIntentos -1} intentos`)
// type-coarcing es propenso a errores:
console.log(`Te quedan ${numIntentos + 1}
intentos`)
```

¡Debemos visualizar errores!

La extensión ***eslint*** de Visual Studio Code los visualiza, pero debemos instalar también el paquete de npm eslint, que es el que realmente detecta los errores.

Para instalar paquetes de node necesitamos node instalado en el equipo, que a su vez, instala el gestor de paquetes npm.

```
npm install -g eslint (global)
npm init # (en el basedir de nuestro proyecto, ojo Windows team, fuera del terminal de vscode)
eslint --init # creación del fichero de inicialización
```

Herramienta formateo -> cambia formato código -> Instalamos extensión **Prettier**

Usamos una configuración eslint tipo AirBnB y nos quedó un fichero .eslintrc.json como este:

```
{
  "env": {
    "browser": true,
    "es2021": true
  },
  "extends": "airbnb-base",
  "overrides": [],
  "parserOptions": {
    "ecmaVersion": "latest",
    "sourceType": "module"
  },
  "rules": {
```

```
  }
}
```

Podemos añadir reglas a nuestro gusto, por ejemplo:

```
  "rules": {
    "semi": ["error", "never"],
    "no-console": "off"
  }
```