
# 자바스크립트란?

## 자바스크립트의 탄생

1995년 당시 약 90%의 시장 점유율로 웹 브라우저 시장을 지배하고 있던 넷스케이프 커뮤니케이션즈(Netscape comunications)는 정적인 HTML을 동적으로 표현하기 위해 경량의 프로그래밍 언어를 도입하기로 결정했다. 그래서 탄생한 것이 브렌던 아이크(Brendan Eich)가 개발한 자바스크립트이다.

자바스크립트는 1996년 3월 넷스케이프 커뮤니케이션즈의 웹 브라우저인 Netscape Navigator 2에 탑재되었고 “Mocha”로 명명되었다. 그해 9월 “LiveScript”로 이름이 변경되었고, 12월 “JavaScript”로 최종 명명되었다.

이렇게 탄생한 자바스크립트는 현재 모든 브라우저의 표준 프로그래밍 언어가 되었다. 그러나 자바스크립트가 순탄하게 성장했던 것은 아니다. 자바스크립트가 탄생한 뒤 얼마 지나지 않아 자바스크립트의 파생 버전인 JScript가 출시되어 자바스크립트는 위기를 맞게 된다.

## 자바스크립트의 파편화와 표준화

1996년 8월, 마이크로소프트는 자바스크립트의 파생 버전인 “JScript”를 Internet Explorer 3.0에 탑재하였다. 그런데 문제는 JScript와 자바스크립트가 표준화되지 못하고 적당히 호환되었다는 것이다. 즉, 자사 브라우저의 시장 점유율을 점유하기 위해 자사 브라우저에서만 동작하는 기능을 경쟁적으로 추가하기 시작했다는 것이다. 이로 인해 브라우저에 따라 웹 페이지가 정상 동작하지 않는 **크로스 브라우징 이슈**가 발생하기 시작했고 모든 브라우저에서 동작하는 웹 페이지를 개발하는 것은 무척 어려워졌다.

이에 자바스크립트의 파편화를 방지하고 모든 브라우저에서 동일하게 동작하는 표준화된 자바스크립트에 대한 필요성이 제기되기 시작했다. 이를 위해 1996년 11월, 넷스케이프 커뮤니케이션즈는 컴퓨터 시스템의 표준을 관리하는 비영리 표준화 기구인 ECMA 인터내셔널에 자바스크립트의 표준화를 요청하였다.

1997년 7월, ECMA-262라 불리는 표준화된 자바스크립트 초판(ECMAScript 1)의 명세(specification)가 완성되었고 상표권 문제로 자바스크립트는 **ECMAScript**로 명명되었다. 이후 1999년 ECMAScript 3(ES3)이 공개되었고 10년 만인 2009년 출시된 ECMAScript 5(ES5)는 HTML5와 함께 출현한 표준안이다.

2015년 ECMAScript 6(ECMAScript 2015)가 공개되었고 범용 프로그래밍 언어로서 갖추어야 할 let/const 키워드, 화살표 함수, 클래스, 모듈 등과 같은 기능들을 대거 도입하는 큰 변화가 있었다. ES6 이후의 버전업은 작은 기능의 추가 레벨로 매년 공개할 것으로 예고되었다. ECMAScript 버전별 특징은 아래와 같다.

| 버전 | 출시년도 | 특징 |
| --- | --- | --- |
| ES1 | 1997 | 초판 |
| ES2 | 1998 | ISO/IEC 16262 국제 표준과 동일한 규격을 적용 |
| ES3 | 1999 | 정규 표현식, try...catch 예외 처리 |
| ES5 | 2009 | HTML5와 함께 출현한 표준안. JSON, strict mode, 접근자 프로퍼티(getter, setter), 향상된 배열 조작 기능 (forEach, map, filter, reduce, some, every) |
| ES6 (ECMAScript 2015) | 2015 | let, const, class, 화살표 함수, 템플릿 리터럴, 디스트럭처링 할당, spread 문법, rest 파라미터, Symbol, Promise, Map/Set, iterator/generator, module import/export |
| ES7 (ECMAScript 2016) | 2016 | 지수(**) 연산자, Array.prototype.includes, String.prototype.includes |
| ES8 (ECMAScript 2017) | 2017 | async/await, Object 정적 메소드 (Object.values, Object.entries, Object.getOwnPropertyDescriptors) |
| ES9 (ECMAScript 2018) | 2018 | Object Rest/Spread 프로퍼티 |

## 자바스크립트 성장의 역사

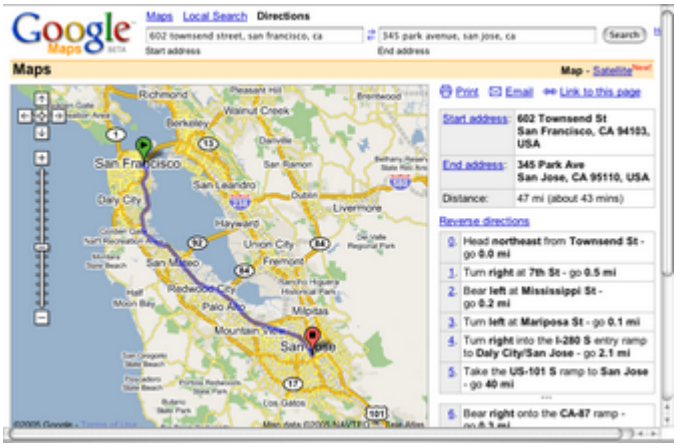
초창기 자바스크립트는 웹 페이지의 보조적인 기능을 수행하기 위해 한정적인 용도로 사용되었다. 이 시기에 대부분 로직은 주로 웹 서버에서 실행되었고 브라우저는 서버로부터 전달받은 HTML과 CSS를 단순히 렌더링하는 수준이었다.

1999년, 자바스크립트를 이용해서 **비동기적(Asynchronous)**으로 서버와 브라우저가 데이터를 교환할 수 있는 통신 기능인 **Ajax(Asynchronous JavaScript and XML)**가 XMLHttpRequest이라는 이름으로 등장했다.

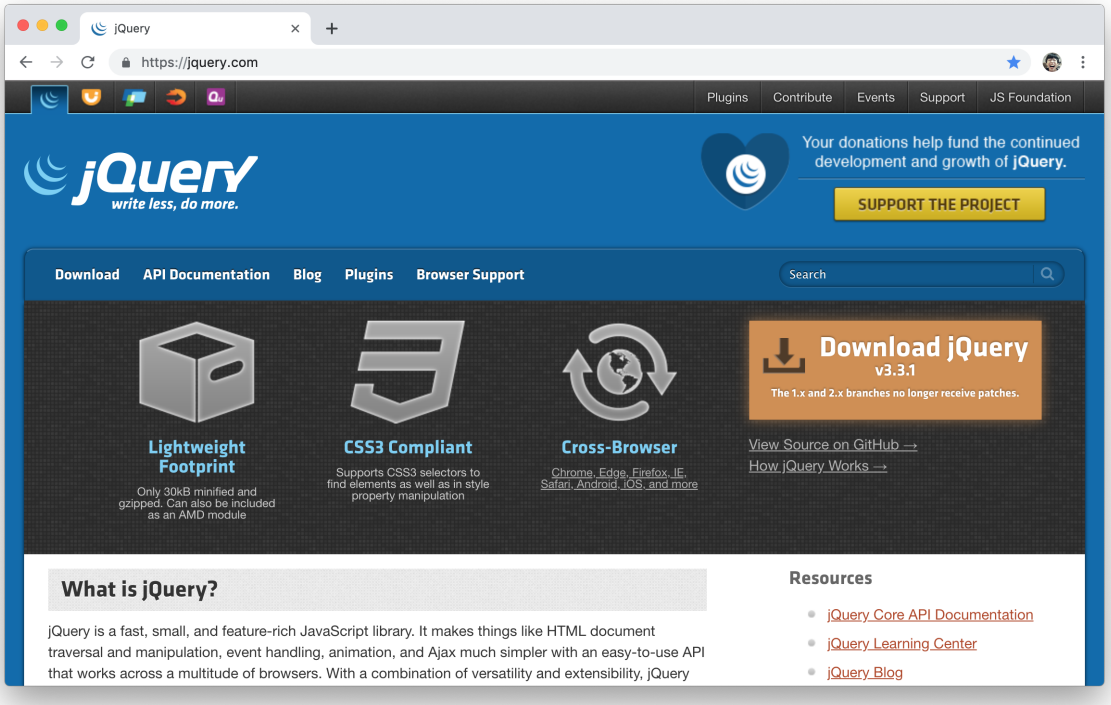
이전의 웹 페이지는 서버로부터 완전한 HTML을 전송 받아 웹 페이지 전체를 렌더링하는 방식으로 동작했다. 따라서 화면이 전환되면 서버로부터 새로운 HTML을 전송 받아 웹 페이지 전체를 처음부터 다시 렌더링하였다. 이는 변경이 없는 부분까지 포함된 HTML을 서버로부터 다시 전송 받기 때문에 불필요한 데이터 통신이 발생하고, 변경이 없는 부분까지 처음부터 다시 렌더링해야 하기 때문에 퍼포먼스 측면에서도 불리한 방식이다. 이로 인해 화면 전환이 일어나면 화면이 순간적으로 깜박이는 현상이 발생하고 이는 웹 애플리케이션의 한계로 받아들여 졌다.

Ajax의 등장은 이전의 패러다임을 획기적으로 전환했다. 즉, 웹 페이지의 변경이 필요 없는 부분은 다시 렌더링하지 않고, 서버로부터 필요한 데이터만을 전송 받아 변경이 필요한 부분만을 한정적으로 렌더링하는 방식이 가능해진 것이다. 이로 인해 웹 브라우저에서도 데스크톱 애플리케이션과 유사한 빠른 퍼포먼스와 부드러운 화면 전환이 가능케 되었다.

2005년, 구글이 발표한 **Google Maps**는 웹 애플리케이션 개발 프로그래밍 언어로서 자바스크립트의 가능성을 확인하는 계기를 마련했다. 웹 브라우저에서 자바스크립트와 Ajax를 기반으로 동작하는 Google Maps가 데스크톱 애플리케이션과 비교해 손색이 없을 정도의 퍼포먼스와 부드러운 화면 전환 효과를 보여준 것이다.



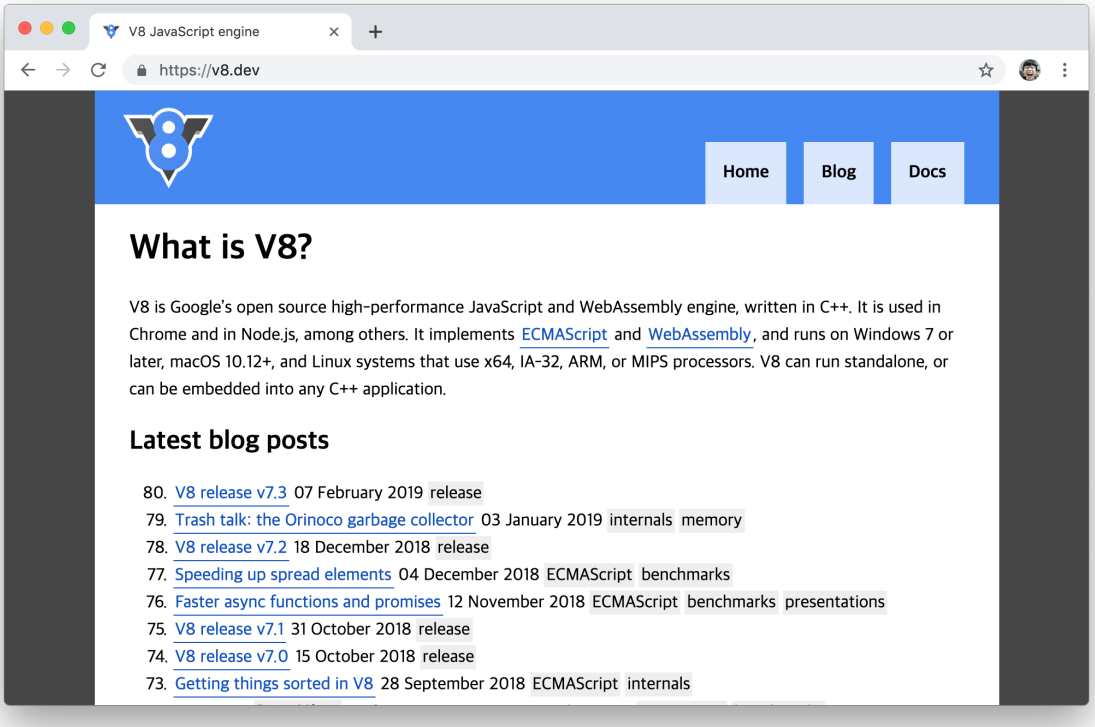

2006년, **jQuery**의 등장으로 다소 번거롭고 논란이 있던 DOM(Document Object Model)을 보다 쉽게 제어할 수 있게 되었고 크로스 브라우징 이슈도 어느 정도 해결되었다. jQuery는 넓은 사용자 층을 순식간에 확보했다. 이로 인해 당시 다소 까다로운 자바스크립트보다 배우기 쉽고 직관적인 jQuery를 더 선호하는 개발자가 양산되기도 했다.



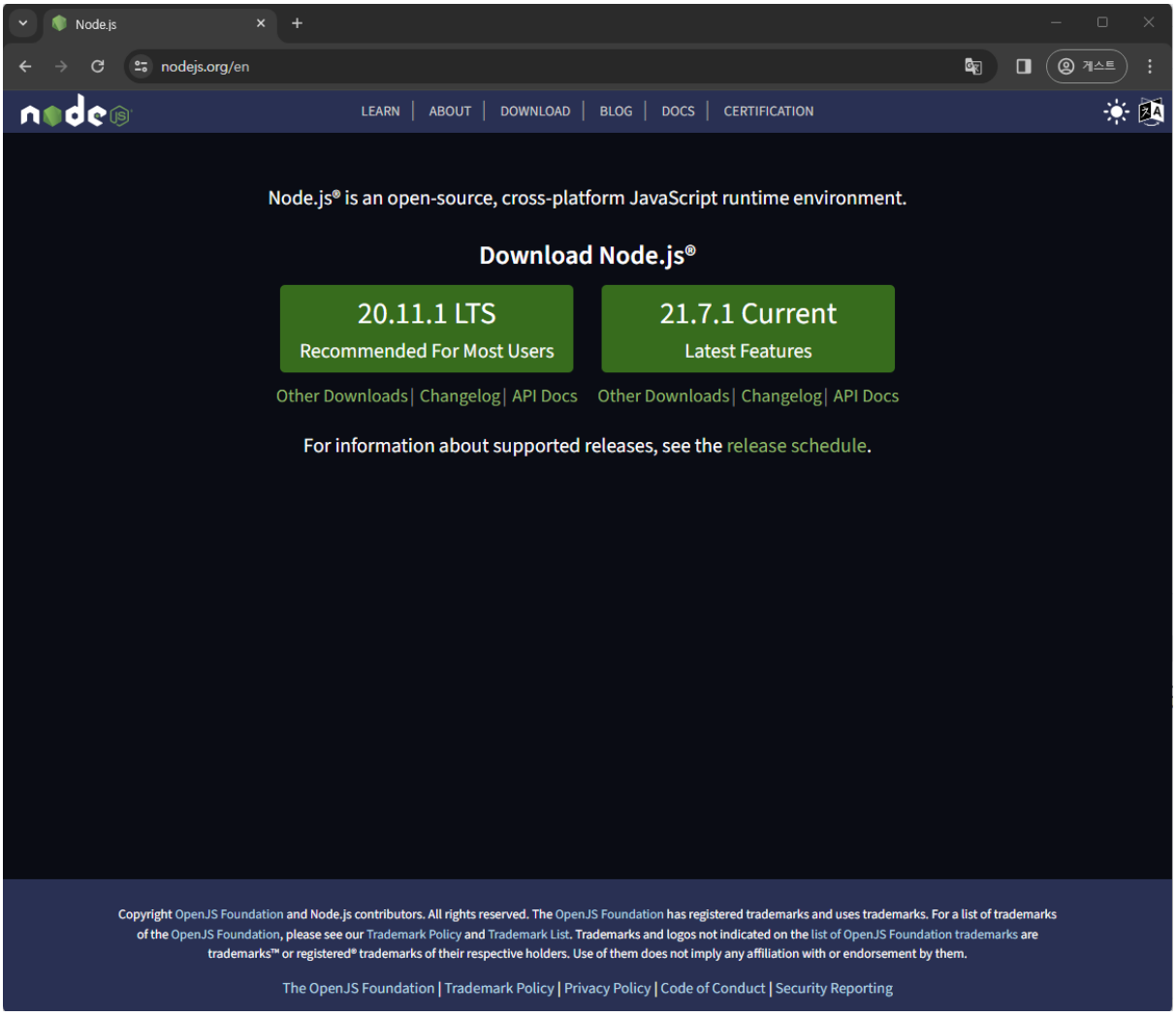

Google Maps를 통해 가능성이 확인된 자바스크립트로 웹 애플리케이션을 구축하려는 시도가 늘어가면서 보다 빠르게 동작하는 자바스크립트 엔진이 요구되었다. 2008년 등장한

구글의 **V8 자바스크립트 엔진**은 이러한 요구에 부합하는 빠른 성능을 보여 주었다. V8 자바스크립트 엔진의 등장으로 자바스크립트는 데스크톱 애플리케이션과 유사한 사용자 경험(user experience)을 제공할 수 있는 웹 애플리케이션 개발 프로그래밍 언어로 정착하게 되었다.

V8 자바스크립트 엔진으로 촉발된 자바스크립트의 발전으로 말마암아 과거 웹 서버에서 수행되던 역할들이 클라이언트(브라우저)로 이동하였고, 이로써 웹 애플리케이션에서 프런트엔드 영역이 주목받는 계기가 되었다.




2009년, 브라우저에서만 동작하던 자바스크립트를 브라우저 이외의 환경에서 동작시킬 수 있는 자바스크립트 실행 환경인 **Node.js**의 등장으로 자바스크립트는 웹 브라우저를 벗어나 서버 사이드 애플리케이션 개발에서도 사용되는 범용 프로그래밍 언어가 되었다. 웹 브라우저에서만 동작하는 반쪽짜리 프로그래밍 언어 취급을 받던 자바스크립트는 이제 프런트엔드 영역은 물론 백엔드 영역까지 아우르는 웹 프로그래밍 언어의 표준으로 자리잡고 있다.

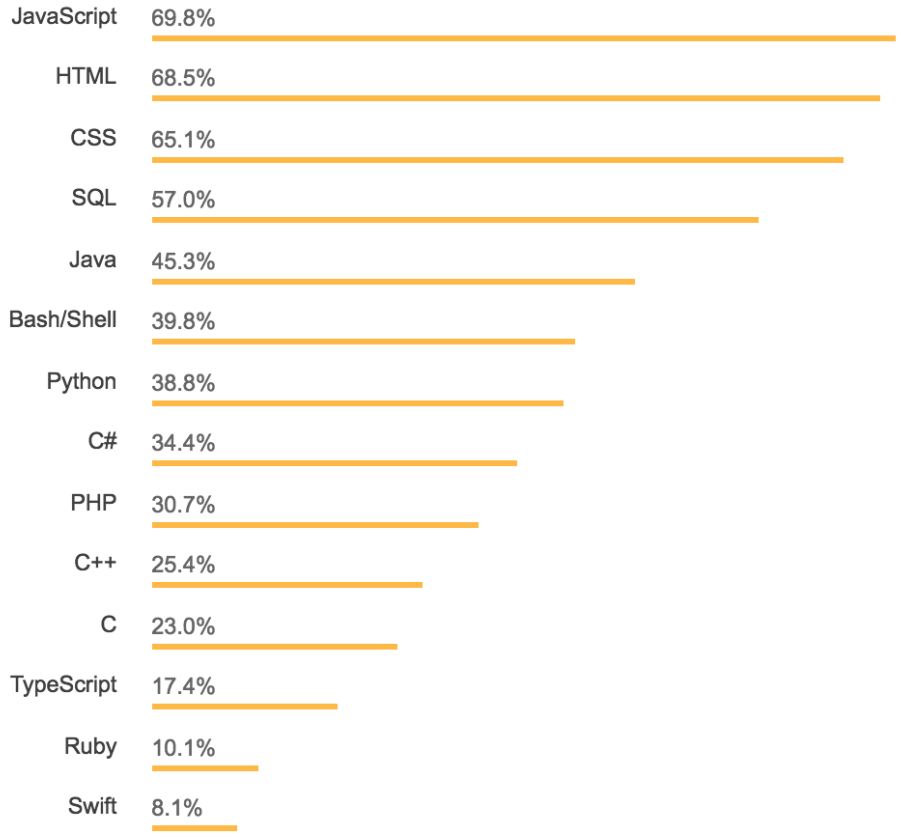



자바스크립트는 크로스 플랫폼을 위한 가장 중요한 언어로 주목받고 있다. 자바스크립트는 웹은 물론 모바일 하이브리드 앱(PhoneGap, Ionic), 서버 사이드(NodeJS), 데스크톱(Electron), 머신 러닝(TensorFlow.js), 로보틱스(Johnny-Five) 프로그래밍 언어로서 세계에서 가장 인기있는 프로그래밍 언어이다.

## Most Popular Technologies

### Programming, Scripting, and Markup Languages

All Respondents | Professional Developers




이제 웹 애플리케이션은 데스크톱 애플리케이션과 비교해도 손색없는 성능과 사용자 경험을 제공하는 것이 필수가 되었고, 개발 규모와 복잡도도 더불어 상승했다. 이전의 개발 방식으로는 복잡해진 개발 과정을 수행하기 어려워졌고, 이러한 필요에 따라 많은 패턴과 라이브러리가 출현하였다. 이는 개발에 많은 도움을 주었지만 유연하면서 확장이 쉬운 애플리케이션 아키텍처 구축을 어렵게 하였고 필연적으로 프레임워크가 등장하게 되었다. SPA(Single Page Application)가 대중화되면서 Angular, React, Vue.js 등 다양한 SPA 프레임워크/라이브러리 또한 많은 사용층을 확보하고 있다.

# JavaScript와 ECMAScript

ECMAScript는 자바스크립트의 표준 명세인 ECMA-262를 말하며 프로그래밍 언어의 타입, 값, 객체와 프로퍼티, 함수, 빌트인 객체 등 핵심 문법(core syntax)을 규정한다. 각 브라우저 제조사는 ECMAScript를 준수하여 브라우저에 내장되는 자바스크립트 엔진을 구현한다.

자바스크립트는 일반적으로 프로그래밍 언어로서 기본 뼈대를 이루는 ECMAScript와 브라우저가 별도 지원하는 **클라이언트 사이드 Web API**, 즉 DOM, BOM, Canvas, XMLHttpRequest, Fetch, requestAnimationFrame, SVG, Web Storage, Web Component, Web worker 등을 아우르는 개념이다.

클라이언트 사이드 Web API는 ECMAScript와는 별도로 World Wide Web Consortium (W3C)에서 별도의 명세로 관리하고 있다. 클라이언트 사이드 Web API의 자세한 내용은 MDN web docs: Web API를 참고하기 바란다.

## 자바스크립트의 특징

자바스크립트는 HTML, CSS와 함께 웹을 구성하는 요소 중 하나로 **웹 브라우저에서 동작하는 유일한 프로그래밍 언어**이다. 자바스크립트는 기존의 프로그래밍 언어에서 많은 영향을 받았다. 기본 문법은 C, Java와 유사하고 Self에서는 프로토타입 기반 상속을, Scheme에서는 일급 함수의 개념을 차용했다.

자바스크립트는 개발자가 별도의 컴파일 작업을 수행하지 않는 **인터프리터 언어 (Interpreter language)**이다. 대부분의 모던 자바스크립트 엔진(Chrome의 V8, FireFox의 Spidermonkey, Safari의 JavaScriptCore, Microsoft Edge의 Chakra 등)은 인터프리터와 컴파일러의 장점을 결합하여 비교적 처리 속도가 느린 인터프리터의 단점을 해결했다. 인터프리터는 소스코드를 즉시 실행하고 컴파일러는 빠르게 동작하는 머신 코드를 생성하고 최적화한다. 이를 통해 컴파일 단계에서 추가적인 시간이 필요함에도 불구하고 보다 빠른 코드의 실행이 가능하다.

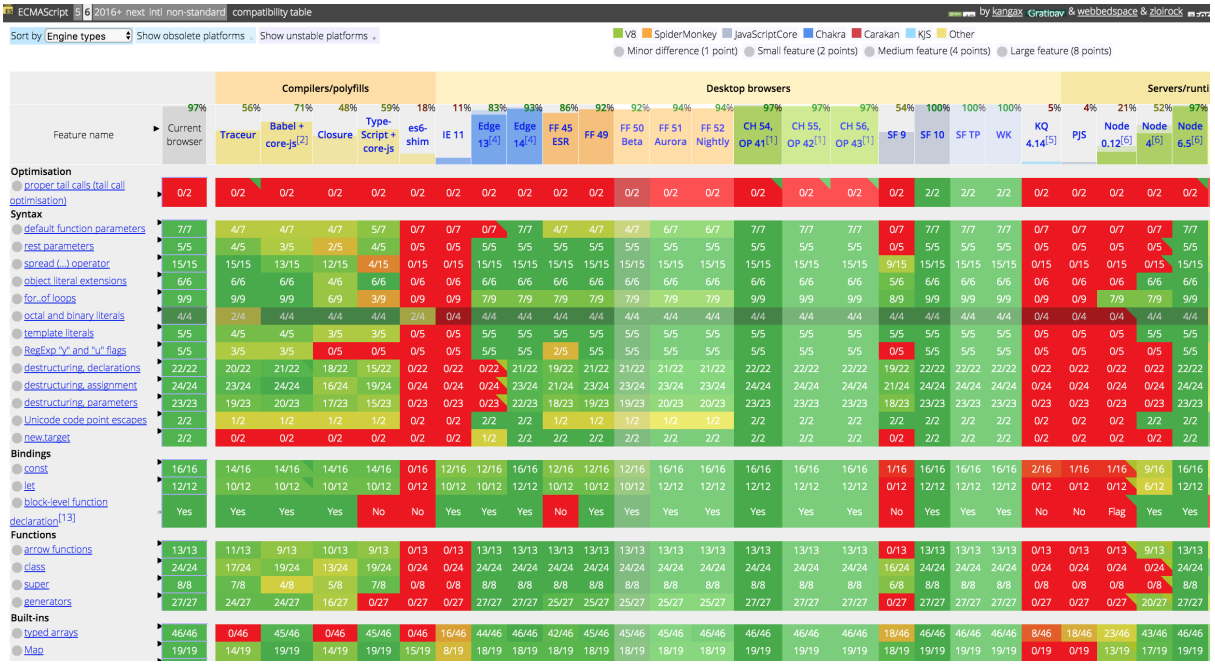
자바스크립트는 명령형(imperative), 함수형(functional), 프로토타입 기반(prototype-based) 객체지향 프로그래밍을 지원하는 멀티 패러다임 프로그래밍 언어다.

비록 다른 객체지향 언어들과의 차이점에 대한 논쟁들이 있긴 하지만, 자바스크립트는 강력한 객체지향 프로그래밍 능력을 지니고 있다. 간혹 클래스(ES6에서 새롭게 도입되었다), 상

속, 정보 은닉을 위한 키워드 private가 없어서 객체지향 언어가 아니라고 오해(자바스크립트는 가장 많은 오해를 받는 언어이다.)하는 경우도 있지만 자바스크립트는 클래스 기반 객체지향 언어보다 효율적이면서 강력한 **프로토타입 기반의 객체지향 언어**이다.

## ES6 브라우저 지원 현황

Internet Explorer를 제외한 대부분의 모던 브라우저는 ES6를 지원하고 있지만 100% 지원하고 있지는 않다. Node.js의 경우 v4부터 ES6를 지원하기 시작했다. ES6 지원 현황은 아래의 웹 사이트에서 확인할 수 있다.




Internet Explorer를 제외한 모던 브라우저의 ES6 지원 비율은 96~ 99%로 거의 100%에 육박하지만 Internet Explorer나 구형 브라우저는 ES6를 대부분 지원하지 않는다. 따라서 Internet Explorer나 구형 브라우저를 고려해야 하는 상황이라면 babel과 같은 트랜스파일러를 사용하여 ES6로 구현한 소스코드를 ES5 이하의 버전으로 다운그레이드할 필요가 있다. 또한 ES6에서 도입된 모듈 import/export는 아직 대부분의 브라우저가 지원하고 있지 않다. 따라서 프로젝트에서 모듈을 도입하려면 Webpack과 같은 모듈 번들러를 사용해야 한다.