

파트 1

코어 자바스크립트

JS

Ilya Kantor

갱신일 2025년 12월 21일

최신 버전의 튜토리얼은 <https://ko.javascript.info>에서 확인할 수 있습니다..

본 튜토리얼은 지속적으로 업데이트 되고있습니다. 잘못된 내용이 발견되었다면 GitHub 저장소로 이동해 양식에 맞추어 [이슈를 만들어주세요](#)..

- [소개](#)
 - [자바스크립트란?](#)
 - [매뉴얼과 명세서](#)
 - [코드 에디터](#)
 - [개발자 콘솔](#)

파트1에선 기본 문법부터 객체 지향 프로그래밍과 같은 고급 개념까지 다양한 내용을 학습합니다.

호스트 환경에 종속되지 않는 코어 자바스크립트에 집중할 예정입니다.

소개

자바스크립트 언어와 호스트 환경에 대해 소개합니다.

자바스크립트란?

자바스크립트(JavaScript)가 언어로서 지닌 특징에 대해 알아보겠습니다. 이어서 자바스크립트로 무엇을 할 수 있을지, 다른 기술들이 자바스크립트를 어떻게 활용하고 있는지도 이야기해 보겠습니다.

정의

*자바스크립트*는 '웹페이지에 생동감을 불어넣기 위해' 만들어진 프로그래밍 언어입니다.

자바스크립트로 작성한 프로그램을 *스크립트(script)* 라고 부릅니다. 스크립트는 웹페이지의 HTML 안에 작성할 수 있는데, 웹페이지를 불러올 때 스크립트가 자동으로 실행됩니다.

스크립트는 특별한 준비나 컴파일 없이 보통의 문자 형태로 작성할 수 있고, 실행도 할 수 있습니다.

이런 관점에서 보면 자바스크립트는 [자바\(Java\)](#) 와는 매우 다른 언어라고 할 수 있습니다.

왜 자바스크립트인가요?

처음 자바스크립트가 만들어졌을 때는 'LiveScript'라는 이름으로 불렸습니다. 그런데, 당시 자바의 인기가 아주 높은 상황이었습니다. 관련인들은 자바스크립트를 자바의 '동생' 격인 언어로 홍보하면 도움이 될 것이라는 의사결정을 내리고 이름을 바꿨습니다.

이름은 자바에서 차용해 왔지만, 자바스크립트는 자바와는 독자적인 언어입니다. 꾸준히 발전을 거듭하면서 [ECMAScript](#) 라는 고유한 명세를 갖춘 독립적인 언어가 되었죠. 자바스크립트는 자바와 아무런 연관이 없습니다.

자바스크립트는 브라우저뿐만 아니라 서버에서도 실행할 수 있습니다. 이 외에도 [자바스크립트 엔진\(JavaScript engine\)](#) 이라 불리는 특별한 프로그램이 들어 있는 모든 디바이스에서도 동작합니다.

브라우저엔 '자바스크립트 가상 머신'이라 불리는 엔진이 내장되어 있습니다.

엔진의 종류는 다양한데, 엔진마다 특유의 코드네임이 있습니다. 아래처럼 말이죠.

- [V8](#) – Chrome과 Opera에서 쓰입니다.
- [SpiderMonkey](#) – Firefox에서 쓰입니다.
- IE는 버전에 따라 'Trident'나 'Chakra'라 불리는 엔진을 사용합니다. 'ChakraCore'는 Microsoft Edge에 사용되며, 'SquirrelFish'는 Safari에 사용됩니다.

위의 코드네임은 개발 관련 글에서 종종 언급되기 때문에 기억해 두는 것이 좋습니다. 본 튜토리얼에서도 해당 코드네임을 사용할 예정입니다. "X라는 기능은 V8에서만 지원합니다."라는 식으로 말이죠. 이런 문장을 만나면 Chrome과 Opera에서만 이 기능을 지원한다고 이해하시면 됩니다.

i 엔진은 어떻게 동작하나요?

엔진이 어떻게 동작하는지 이해하려면 상당한 시간을 쏟아부어야 합니다. 하지만 기본 원리는 다음과 같이 간단합니다.

1. 엔진(브라우저라면 내장 엔진)이 스크립트를 읽습니다(파싱).
2. 읽어 들인 스크립트를 기계어로 변환합니다(컴파일).
3. 기계어로 변환된 코드가 실행됩니다. 기계어로 변환되었기 때문에 실행 속도가 빠릅니다.

엔진은 프로세스 각 단계마다 최적화를 진행합니다. 심지어 컴파일이 끝나고 실행 중인 코드를 감시하면서, 이 코드로 흘러가는 데이터를 분석하고, 분석 결과를 토대로 기계어로 변환된 코드를 다시 최적화하기도 합니다. 이런 과정을 거치면 스크립트 실행 속도는 더욱 더 빨라집니다.

브라우저에서 할 수 있는 일

모던 자바스크립트는 ‘안전한’ 프로그래밍 언어입니다. 메모리나 CPU 같은 저수준 영역의 조작을 허용하지 않습니다. 애초에 이러한 접근이 필요치 않은 브라우저를 대상으로 만든 언어이기 때문이죠.

자바스크립트의 능력은 실행 환경에 상당한 영향을 받습니다. [Node.js](#) ↗ 환경에선 임의의 파일을 읽거나 쓰고, 네트워크 요청을 수행하는 함수를 지원합니다.

브라우저 환경에선 웹페이지 조작, 클라이언트와 서버의 상호작용에 관한 모든 일을 할 수 있습니다.

브라우저에서 자바스크립트로 할 수 있는 일은 다음과 같습니다.

- 페이지에 새로운 HTML을 추가하거나 기존 HTML, 혹은 스타일 수정하기
- 마우스 클릭이나 포인터의 움직임, 키보드 키 눌림 등과 같은 사용자 행동에 반응하기
- 네트워크를 통해 원격 서버에 요청을 보내거나, 파일 다운로드, 업로드하기([AJAX](#) ↗ 나 [COMET](#) ↗ 과 같은 기술 사용)
- 쿠키를 가져오거나 설정하기. 사용자에게 질문을 건네거나 메시지 보여주기
- 클라이언트 측에 데이터 저장하기(로컬 스토리지)

브라우저에서 할 수 없는 일

브라우저는 보안을 위해 자바스크립트의 기능에 제약을 걸어놓았습니다. 이런 제약은 악성 웹페이지가 개인 정보에 접근하거나 사용자의 데이터를 손상하는 것을 막기 위해 만들어졌습니다.

몇 가지 제약사항을 소개해 드리겠습니다.

- 웹페이지 내 스크립트는 디스크에 저장된 임의의 파일을 읽거나 쓰고, 복사하거나 실행할 때 제약을 받을 수 있습니다. 운영체제가 지원하는 기능을 브라우저가 직접 쓰지 못하게 막혀있기 때문입니다.

모던 브라우저를 사용하면 파일을 다룰 수 있습니다. 하지만 접근은 제한되어 있습니다. 사용자가 브라우저 창에 파일을 '끌어다 두거나' `<input>` 태그를 통해 파일을 선택할 때와 같이 특정 상황에서만 파일 접근을 허용합니다.

카메라나 마이크 같은 디바이스와 상호 작용하려면 사용자의 명시적인 허가가 있어야 합니다. 자바스크립트가 활성화된 페이지라도 사용자 몰래 웹 카메라를 작동 시켜 수집한 정보를 [국가안보국\(NSA\)](#)과 같은 곳에 몰래 전송할 수 없습니다.

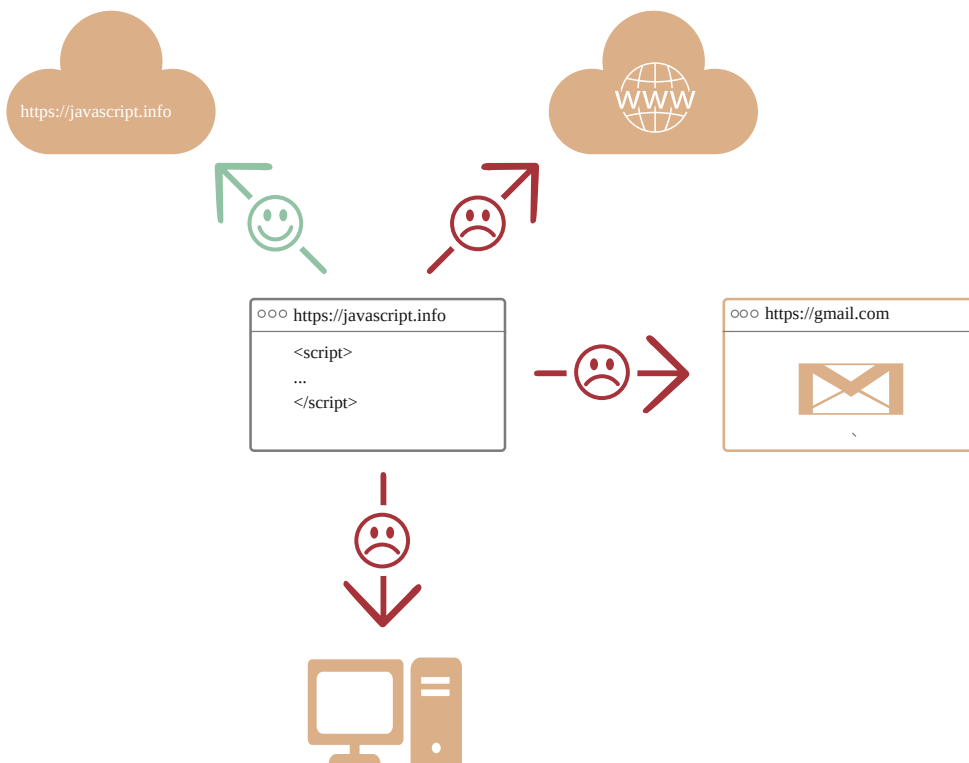
- 브라우저 내 탭과 창은 대개 서로의 정보를 알 수 없습니다. 그런데 자바스크립트를 사용해 한 창에서 다른 창을 열 때는 예외가 적용됩니다. 하지만 이 경우에도 도메인이나 프로토콜, 포트가 다르다면 페이지에 접근할 수 없습니다.

이런 제약사항을 '동일 출처 정책(Same Origin Policy)'이라 부릅니다. 이 정책을 피하려면 두 페이지는 데이터 교환에 동의해야 하고, 동의와 관련된 특수한 자바스크립트 코드를 포함하고 있어야 합니다. 자세한 사항은 추후 학습하도록 하겠습니다.

다시 한번 강조하지만, 이런 제약사항은 사용자의 보안을 위해 만들어졌습니다.

`http://anysite.com`에서 받아온 페이지가 `http://gmail.com`에서 받아온 페이지 상의 정보에 접근해 중요한 개인정보를 훔치는 걸 막기 위함입니다.

- 자바스크립트를 이용하면 페이지를 생성한 서버와 쉽게 정보를 주고받을 수 있습니다. 하지만 타 사이트나 도메인에서 데이터를 받아오는 건 불가능합니다. 가능하다 할지라도 원격 서버에서 명확히 승인을 해줘야 합니다(HTTP 헤더 등을 이용). 이 역시 보안을 위해 만들어진 제약사항입니다.



브라우저 환경 밖, 예를 들어 서버라면 이러한 제약은 존재하지 않을 것입니다. 다만, 모던 브라우저에선 추가 권한 허가를 요청하는 플러그인이나 익스텐션 설치가 허용됩니다.

자바스크립트만의 강점

자바스크립트엔 다양한 장점이 있지만 여기서 *세 가지*만 언급해 보도록 하겠습니다.

- HTML/CSS와 완전히 통합할 수 있음
- 간단한 일은 간단하게 처리할 수 있게 해줌
- 모든 주요 브라우저에서 지원하고, 기본 언어로 사용됨

이 세 가지 모두를 지원하는 브라우저 연관 기술은 자바스크립트뿐입니다.

이런 특징 때문에 자바스크립트는 브라우저 인터페이스를 만들 때 가장 널리 사용되고 있습니다.

이 외에도 자바스크립트를 이용해 서버나 모바일 앱 등을 만드는 것도 가능합니다.

자바스크립트 '너머의' 언어들

자바스크립트 문법은 모든 사람의 요구를 충족시키진 못합니다. 사람마다 각기 다른 기능을 원하기 때문이죠.

프로젝트마다 요구사항이 천차만별이기 때문에 이는 당연한 현상입니다.

이로 인해 근래엔 브라우저에서 실행 되기 전에 자바스크립트로 *트랜스파일(transpile, 변환)* 할 수 있는 새로운 언어들이 많이 등장했습니다.

최신 툴을 사용하면 트랜스파일을 빠르고 명확하게 수행할 수 있습니다. 최신도구는 자바스크립트 이외의 언어로 작성한 코드를 '보이지 않는 곳에서' 자바스크립트로 자동 변환해줍니다.

자바스크립트로 트랜스파일이 가능한 언어 몇 가지를 소개해 드리겠습니다.

- [CoffeeScript](#) ➦ 는 자바스크립트를 위한 'syntactic sugar'입니다. 짧은 문법을 도입하여 명료하고 이해하기 쉬운 코드를 작성할 수 있습니다. Ruby 개발자들이 좋아합니다.
- [TypeScript](#) ➦ 는 개발을 단순화 하고 복잡한 시스템을 지원하려는 목적으로 '자료형의 명시화(strict data typing)'에 집중해 만든 언어입니다. Microsoft가 개발하였습니다.
- [Flow](#) ➦ 역시 자료형을 강제하는데, TypeScript와는 다른 방식을 사용합니다. Facebook이 개발하였습니다.
- [Dart](#) ➦ 는 모바일 앱과 같이 브라우저가 아닌 환경에서 동작하는 고유의 엔진을 가진 독자적 언어입니다. Google이 개발하였습니다.

이 외에도 자바스크립트로 트랜스파일 할 수 있는 언어는 다양합니다. 개발 언어로 이런 언어 중 하나를 택한다고 하더라도 자신이 무엇을 하고 있는지 이해하려면 결국엔 자바스크립트를 알아야 합니다.

요약

- 자바스크립트는 브라우저에서만 쓸 목적으로 고안된 언어이지만, 지금은 다양한 환경에서 쓰이고 있습니다.

- 오늘날 자바스크립트는 브라우저 환경에서 가장 널리 사용되는 언어로 자리매김하였습니다. HTML/CSS와 완전한 통합이 가능합니다.
- 자바스크립트로 '트랜스파일'할 수 있는 언어는 많습니다. 각 언어마다 고유한 기능을 제공하죠. 자바스크립트에 숙달한 뒤에 이 언어들을 살펴볼 것을 추천드립니다.

매뉴얼과 명세서

본 *튜토리얼*은 자바스크립트를 기본부터 차근차근 배울 수 있도록 만들어졌습니다. 그런데 어느 정도 자바스크립트가 익숙해지면 튜토리얼 이외의 자료가 필요한 시점이 옵니다.

명세서

[ECMA-262 명세서\(specification\)](#) 는 자바스크립트와 관련된 가장 심도 있고 상세한 정보를 담고 있는 공식 문서입니다. 이 명세서에서 자바스크립트라는 언어를 정의합니다.

ECMA-262 명세서의 고유한 형식 때문에 명세서를 처음 접하는 사람은 그 내용을 이해하기가 쉽지 않습니다. 자바스크립트에 관한 정보를 얻을 수 있는 가장 신뢰할 만한 자료이긴 하지만 일상적인 참고 자료로는 적합하지 않죠.

ECMA-262명세서는 새로운 버전이 매년 나옵니다. 공식 버전이 나오기 이전의 최신 초안은 <https://tc39.es/ecma262/> 에서 확인할 수 있습니다.

갓 명세서에 등록된 기능이나 '등록되기 바로 직전'에 있는 기능(스테이지(stage)3 상태의 기능), 제안 목록은 <https://github.com/tc39/proposals> 에서 확인할 수 있습니다.

본 튜토리얼의 **두 번째 대 단원**에서 브라우저와 관련된 명세서를 다룰 예정이므로, 만약 브라우저에서 돌아가는 기능을 구현하는 개발자라면 해당 내용을 확인해 보시기 바랍니다.

매뉴얼

- Mozilla 재단이 운영하는 **MDN JavaScript Reference**엔 다양한 예제와 정보가 있습니다. 특정 함수나 메서드에 대한 깊이 있는 정보를 얻고 싶다면 이 사이트가 제격입니다.

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

위 사이트에 들어가서 원하는 내용을 직접 검색하는 것도 좋지만, 가끔은 검색 엔진을 이용해 내용을 찾는 게 더 나을 때도 있습니다. Google 검색 엔진에 접속해 'MDN [원하는 용어]'를 입력해 봅시다. `parseInt` 함수에 대한 정보를 얻고 싶다면 <https://google.com/search?q=MDN+parseInt> 같이 검색하는 식으로 말이죠.

호환성 표

자바스크립트는 끊임없이 발전하는 언어입니다. 새로운 기능이 정기적으로 추가되죠.

특정 브라우저나 엔진이 내가 사용하려는 기능을 지원하는지 확인할 땐, 아래 두 사이트가 좋습니다.

- <http://caniuse.com> 에선 브라우저가 특정 기능을 지원하는지 (표 형태로) 확인할 수 있습니다. 암호화 관련 기능인 cryptography를 특정 브라우저에서 사용할 수 있는지 아닌지

를 보려면 <http://caniuse.com/#feat=cryptography> ↗ 를 확인하면 됩니다.

- <https://kangax.github.io/compat-table> ↗ 에선 자바스크립트 기능 목록이 있고, 해당 기능을 특정 엔진이 지원하는지 여부를 거대한 표를 통해 보여줍니다.

실제 개발을 하다 보면 위에 언급 드린 자료가 아주 유용할 겁니다. 메서드나 함수 관련 정보, 브라우저 지원 여부 등은 의사결정을 내릴 때 꼭 필요한 정보이기 때문입니다.

말씀드린 사이트나 이 페이지를 기억해 놓았다가 특정 기능에 대한 상세한 정보가 필요할 때 방문해 보시길 바랍니다.

코드 에디터

개발자는 코드 에디터(code editor)에서 가장 많은 시간을 보냅니다.

코드 에디터는 크게 통합 개발 환경(IDE)과 경량 에디터로 나뉘는데, 많은 개발자가 둘 중 하나를 택해 작업합니다.

통합 개발 환경

통합 개발 환경 ↗ (Integrated Development Environment, IDE)은 강력한 에디터입니다. 보통 '프로젝트 전체'를 관장하는 다양한 기능을 제공합니다. 이름에서 알 수 있듯이 IDE는 단순한 에디터가 아닙니다. '개발 환경'을 쾌적하게 해주는 통합 환경을 제공합니다.

IDE를 이용하면 수많은 파일로 구성된 프로젝트를 불러오고, 파일 간의 탐색 작업이 수월해 집니다. 단순히 열려있는 파일뿐만 아니라 전체 프로젝트에 기반한 자동 완성 기능도 사용할 수 있습니다. 여기에 더하여 [git](#) ↗ 과 같은 버전 관리 시스템, 테스트 환경 등, '프로젝트 수준'의 작업도 IDE에서 할 수 있습니다.

아직 어떤 IDE를 사용할지 결정하지 못했다면, 아래 두 옵션을 고려해 보시길 바랍니다.

- [Visual Studio Code](#) ↗ (크로스 플랫폼, 무료)
- [WebStorm](#) ↗ (크로스 플랫폼, 유료)

Windows 사용자라면 'Visual Studio'라는 IDE를 들어보셨을 겁니다. Visual Studio는 'Visual Studio Code'와는 다릅니다. 'Visual Studio'는 .NET 플랫폼 개발에 쓰이는 유료 에디터로, Windows에서만 사용할 수 있습니다. 자바스크립트도 지원합니다. Visual Studio의 무료 버전인 [Visual Studio Community](#) ↗ 도 있으니 참고하시기 바랍니다.

상당수의 IDE가 유료이긴 하지만 개발자 연봉 대비 무시할 만한 수준입니다. 체험 기간을 이용해 자신에게 맞는 IDE를 찾아 구매하는 것을 권유 드립니다.

경량 에디터

'경량 에디터(lightweight editor)'는 IDE만큼 많은 기능을 제공하진 않지만, 속도가 빠르고 단순하다는 장점이 있습니다.

경량 에디터는 파일을 열고 바로 수정하고자 할 때 주로 사용됩니다.

'경량 에디터'와 'IDE'의 가장 큰 차이점은 IDE는 프로젝트 레벨에서 작동한다는 점입니다. IDE는 구동 시 불러와야 할 데이터가 많고, 필요하다면 구동 시 프로젝트 구조를 분석하는 일

등도 합니다. 파일 하나만 수정하고 싶다면 경량 에디터를 사용하는 게 훨씬 빠릅니다.

경량 에디터는 다양한 플러그인을 지원합니다. 디렉터리 레벨 문법 분석기나 자동완성기능 등을 플러그인을 설치해 사용할 수 있습니다. 플러그인을 사용하면 경량 에디터에서도 IDE 못지않게 다양한 기능을 사용할 수 있죠. 요즘엔 경량 에디터와 IDE 사이의 엄격한 구분이 사라져가는 추세입니다.

추천하는 에디터는 다음과 같습니다.

- [Atom](#) (크로스 플랫폼, 무료)
- [Visual Studio Code](#) (크로스 플랫폼, 무료)
- [Sublime Text](#) (크로스 플랫폼, 셰어웨어)
- [Notepad++](#) (Windows, 무료)
- [Vim](#) 이나 [Emacs](#) 도 에디터로 사용법만 잘 숙지하면 충분히 에디터 역할을 잘합니다.

논쟁하지 맙시다

위에 나열한 에디터는 저나 제가 훌륭하다고 생각하는 개발자들이 오랫동안 만족하며 사용하고 있는 것들을 추린 것입니다.

이 외에도 제가 모르는 훌륭한 에디터가 있을 수 있으니 여러분이 가장 좋아하는 것 하나를 택하시면 됩니다.

여타 툴과 마찬가지로 에디터를 선택하는 것은 프로젝트의 종류, 개발 습관, 개인 성향에 따라 다르므로 이에 관한 논쟁은 지양하도록 합시다.

개발자 콘솔

코드엔 에러가 항상 도사리고 있습니다. 당신이 [로봇](#) 이 아니라 사람이라면 분명 에러를 만들 겁니다. 그 누구도 예외가 아니죠.

그런데 브라우저는 스크립트에 문제가 있어서 에러가 발생해도 이를 사용자에게 직접 보여주지 않습니다. 에러가 발생했는지조차 모르면 에러를 고칠 수 없겠죠?

브라우저엔 '개발자 도구'라는 것이 내장되어 있습니다. 이 도구를 이용하면 에러를 확인할 수 있습니다. 스크립트에 대한 쓸만한 정보도 얻을 수 있죠.

대부분의 개발자는 Chrome이나 Firefox를 이용해 개발하는 걸 선호합니다. 두 브라우저에서 제공하는 개발자 도구가 굉장히 훌륭하기 때문이죠. 기타 브라우저들도 개발자 도구를 제공하고 독특한 기능이 있지만, 거의 Chrome이나 Firefox 기능을 '따라가는' 수준입니다. 그래서 개발자들은 Chrome이나 Firefox 중 '선호하는' 브라우저를 하나 택해 개발하다가 사용하고 있는 브라우저에 종속된 문제가 발생하면 다른 브라우저로 전환해 개발을 이어나가곤 합니다.

개발자 도구에서 지원하는 기능을 잘 활용하면 개발 효율이 상당히 올라갑니다. 이 챕터에선 개발자 도구를 열어 에러를 확인하고, 다양한 명령어를 입력해 보는 방법에 대해 소개하도록 하겠습니다.

Chrome

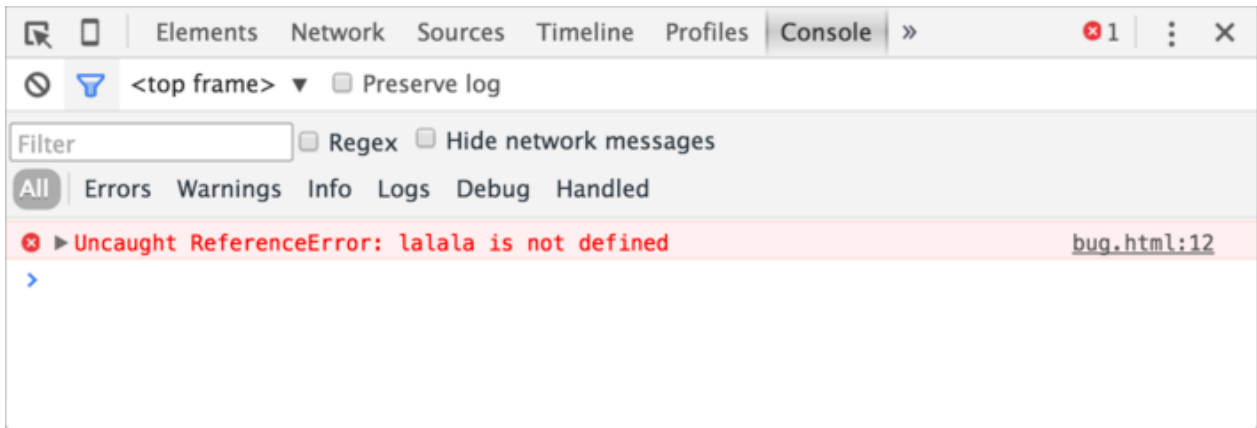
[bug.html](#)을 열어봅시다.

페이지 내 스크립트에 에러가 있는데, 일반적인 사용자 눈에는 이 에러가 보이지 않습니다. 개발자 도구를 열어 에러를 확인해 봅시다.

F12를 눌러봅시다. Mac 사용자라면 **Cmd+Opt+J**를 누르면 됩니다.

개발자 도구가 보일 겁니다. 개발자 도구를 처음 열어보셨다면 Console 패널이 기본으로 보입니다.

아래와 같이 말이죠.



화면 구성은 사용하고 있는 Chrome 버전에 따라 다릅니다. 버전이 바뀔 때 마다 구성이 조금씩 바뀌긴 하지만 큰 틀은 바뀌지 않습니다.

- 빨간색 에러 메시지가 보일 겁니다. 'lalala'가 정의되지 않았다(not defined)라는 메시지입니다.
- 에러 메시지 우측에 링크 `bug.html:12`가 있습니다. `bug.html`은 해당 에러가 발생한 파일, 12는 에러가 발생한 줄을 나타냅니다.

에러 메시지 아래에 파란색 기호 `>`가 있는데, 이 기호가 있는 곳엔 자바스크립트 명령어(command)를 입력할 수 있습니다. 이를 '커맨드 라인(command line)'이라 부릅니다. 커맨드 라인에 명령어(command)를 입력한 후 **Enter**를 누르면 해당 명령어가 실행됩니다.

자 이제 에러를 확인하는 방법을 알았습니다. 시작치곤 나쁘지 않네요. 에러를 확인하고 고치는 방법(디버깅)은 [Chrome으로 디버깅하기](#)에서 다루도록 하겠습니다.

i Multi-line input

보통은 한줄 짜리 명령어를 입력하고 **Enter**를 눌러 해당 명령어를 실행하는 작업을 많이 합니다.

명령어를 여러 줄에 걸쳐 작성하고 싶다면 **Shift+Enter**를 누르면 됩니다.

Shift+Enter를 누르면 명령어를 실행시키지 않고 줄 바꿈만 할 수 있기 때문에 자바스크립트 코드 조각을 입력하는 것도 가능해집니다.

Firefox, Edge 및 기타 브라우저

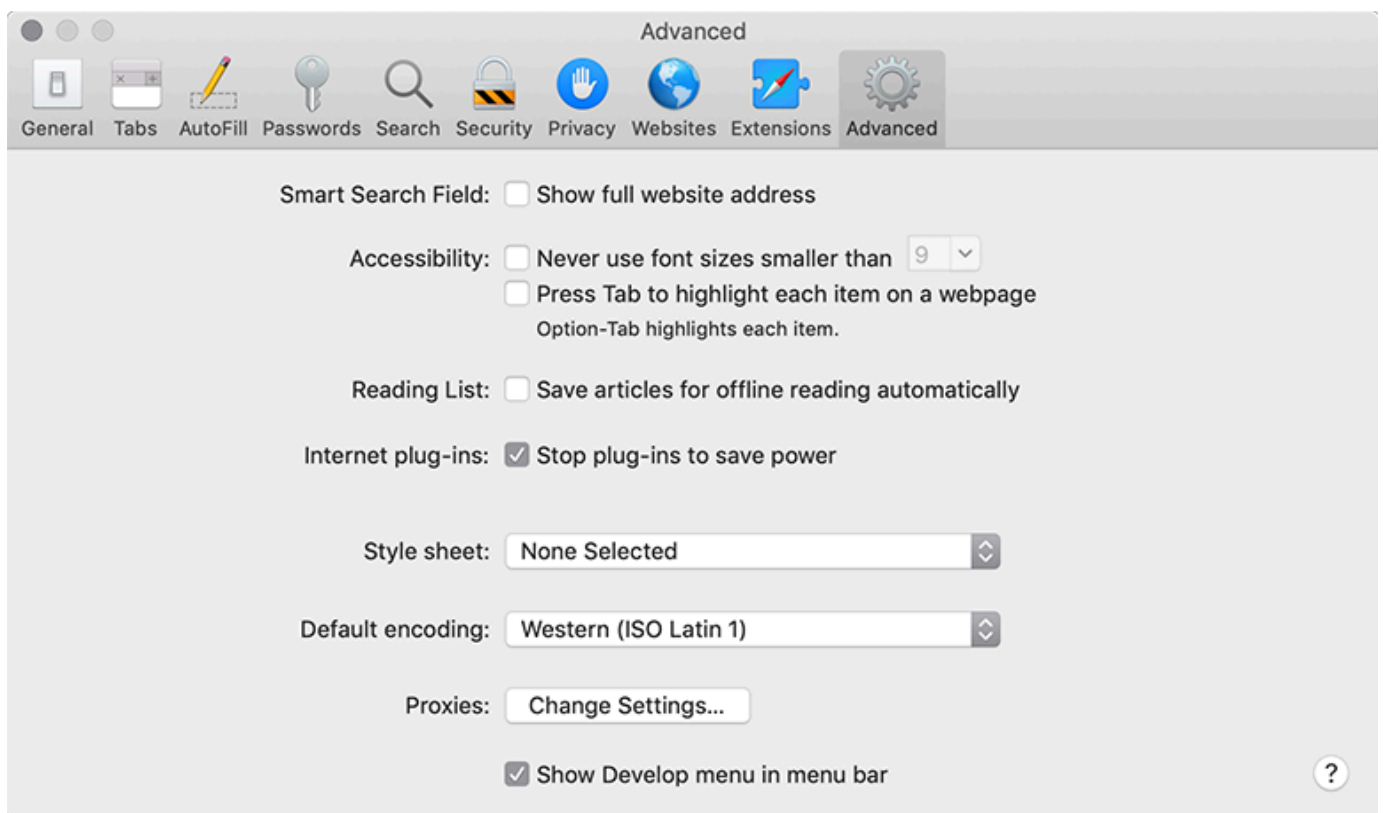
F12를 누르면 대부분의 브라우저에서 개발자 도구를 열 수 있습니다.

브라우저는 다르지만 개발자 도구 인터페이스는 거의 유사하기 때문에, 한 브라우저에 익숙해지면 다른 브라우저에 적응하는 건 어렵지 않습니다. Chrome을 이용해 개발자 도구에 입문해 보시길 바랍니다.

Safari

Mac 전용 브라우저인 Safari에서 개발자 도구를 사용하려면 '개발자 메뉴(Develop menu)'를 명시적으로 활성화해주어야 합니다.

환경설정(Preferences)의 '고급(Advanced)' 패널을 클릭한 후 '메뉴 막대에서 개발자용 메뉴 보기' 체크 박스를 체크해 개발자 도구를 활성화해봅시다.



이제 **Cmd+Opt+C**를 눌러 개발자 콘솔을 여닫을 수 있게 되었습니다. Safari 상단에 '개발자용(Develop)' 메뉴가 새로 생긴 것도 볼 수 있습니다. 개발자용 메뉴엔 다양한 명령어와 옵션이 있습니다.

요약

- 개발자 도구를 이용하면 에러를 확인하고, 명령어를 실행하고, 변수를 분석해보는 등의 일을 할 수 있습니다.
- Windows 사용자는 **F12**를 눌러 개발자 도구를 열 수 있습니다. Mac 사용자는 Chrome에선 **Cmd+Opt+J**, Safari에선 **Cmd+Opt+C**를 누르면 됩니다. Safari는 개발자 메뉴를 활성화해 줘야 개발자 도구를 사용할 수 있습니다.

이제 학습 환경은 잘 갖추었습니다. 다음 섹션부터는 자바스크립트에 대해 본격적으로 학습해 보도록 하겠습니다.