# Javascript Evolution

Netscape- After the success of Netscape Navigator, Marc Andreessen felt the web still wasn't complete.

The “glue language” would not any relation to Java, except a bit of similarity in syntax.



Brendan Eich was employed in May 1995 to help with creating this language and he created a prototype in 10 days.

## **Early Days**

After the development of the prototype, this newly created language was called Mocha.

A better release was made in September 1995 and a name change was made as well, with the new name being LiveScript.

The final name was agreed upon after the beta 3 release in December 1995: the language would be called JavaScript.

It should be known, though, that this did not bring about any copyright infringements as Sun’s and Netscape had entered into an agreement earlier.

The name JavaScript, however, did help the language feed on the popularity of the Java programming language itself.

Netscape passed JavaScript to the European Computer Manufacturers Association (ECMA) for standardization in 1995.

The **ECMAScript** specification is a standardized specification of a scripting language developed by [**Brendan Eich**](https://en.wikipedia.org/wiki/Brendan_Eich) of [Netscape](https://en.wikipedia.org/wiki/Netscape_Communications_Corporation) in 1995; initially it was named **Mocha**, later renamed to **LiveScript**, and finally **JavaScript**.

The ECMAScript specification is a standardized specification of a scripting language. There are six editions of ECMA-262 published. Version 6 of the standard is codenamed "Harmony". TypeScript is aligned with the ECMAScript6 specification.

<http://es6-features.org/#Constants>

Ecma is a standards organization for information and communication systems. It acquired its current name in 1994, when the **European Computer Manufacturers Association (ECMA)** changed its name to reflect the organization's global reach and activities. As a consequence, the name is no longer considered an acronym and no longer uses full capitalization.

In **1996**, Microsoft developed a JavaScript implementation in ASP and .NET on the Internet Information Services (IIS) web server and it was named Jscript.

Jscript served as a reverse-engineered implementation of Netscape’s JavaScript.

About two years after passing JavaScript to ECMA, the first standard version of JavaScript, ECMAScript 1, was released in **June 1997**.

**1998** - ECMAScript 2 was released the next year, with minimal changes to the previous version to keep up with the ISO standard for the language.

While JavaScript was now beginning to be adopted by programmers, it was difficult building websites for Netscape Navigator and Microsoft’s Internet Explorer.

Programmers intending to build cross-platform websites started having issues with developing such websites with JavaScript as the browser reacted differently to code for development.

In **December 1999**, 18 months after the release of ECMAScript 2, **ECMAScript 3** was released with lots of changes. ECMAScript 3 saw the introduction of the language’s regular expression and exception handling features.

Immediately after the release of ECMAScript 3, plans to come up with ECMAScript 4 began in 2000. However, the whole process died down with the closure of this project confirmed in 2003 after ECMA released an interim report containing some of the functionality intended for ECMAScript 4.

**In 2005, Eich and Mozilla joined ECMA so as to be able to assist E4X (ECMAScript for XML**), which then lead to a collaboration with Macromedia. Macromedia decided to adopt JavaScript and use E4X in ActionScript 3.

**2006 – Ajax was introduced**. Later that year, the adoption of JavaScript skyrocketed further after Jesse James Garrett’s paper was released where he described the technologies JavaScript supported as the backbone for the web, introducing the term Ajax as a way of building modern websites.

While **ECMAScript 4 was abandoned**, the successor to ECMAScript 3 was finally released in December **2009**. This was a decade after the release of ECMAScript 3 and was called **ECMAScript 5** and came with lots of new features including **support for the parsing of JSON files**.

In **2009** – commonJS project is created

In **2013**, plans were made for the release of the ECMAScript 6 but just as in the case of ECMAScript 4, the process slowed down. However, the project did not die out completely as it was released in June 2015.

**2015 - ECMAScript 6 was renamed to ECMAScript 2015 (ES2015)**, and this naming pattern has continued for the latest releases of the JavaScript standard.

2016 - ECMAScript 2016 (ES2017)

2017 – ECMAScript 2017

2018 - ECMAScript 2018 is the latest version of JavaScript, with new features such as asynchronous iterators, asynchronous generators, and new regular expression features.

**2019 - ECMAScript 2019** (or ES2019 for short) will be released. The new **features** include Object. fromEntries() , trimStart() , trimEnd() , flat() , flatMap() , description property for symbol objects, optional catch binding

**2020 - ECMAScript 2020 (approved)** introduces multiple features ranging from a new import() facility for loading modules to a new BigInt type for working with arbitrary precision integers

<https://www.infoworld.com/article/3538809/ecmascript-2020-spec-for-javascript-approved.html>

**TypeScript**

TypeScript adopts its basic language features from the ECMAScript5 specification, i.e., the official specification for JavaScript. TypeScript language features like Modules and classbased orientation are in line with the EcmaScript 6 specification. Additionally, TypeScript also embraces features like generics and type annotations that aren’t a part of the EcmaScript6 specification

TypeScript is a relatively new language. It was launched for public use in October 2012, as the version 0.8 of the language. It was a result of two years of developed at Microsoft, with Anders Hejlsberg, the lead architect of c#, as well as the creator of Delphi and Turbo Pascal working on the project too.

Not long after its release, the language was praised by Miguel de Icaza, the Mexican programmer. However, he also criticized the absence of IDE support for TypeScript excluding Microsoft Visual Studio which is unavailable on Linux and OS X.

This lack of IDE support issue was tackled and by 2013, TypeScript support was present in other IDEs, such as Eclipse using a plug-in provided by Palantir Technologies. Several text editors were also released with support for TypeScript, such as Sublime, Vim and Emacs.

The TypeScript 0.8 was followed by a newer version, the TypeScript 0.9 which was released in 2013. The newer edition had an additional feature- support for generics. TypeScript 0.9 was succeeded by TypeScript 1.0 which was released at the Build 2014. The second update for Visual Studio 2013 provides default support for TypeScript.

A New compiler for TypeScript was announced by its development team in July 2014. The team claimed that the new compiler would bring five times gains in performance with itself. Around the same time, the source code for TypeScript, which was originally hosted by CodePlex, was shifted to Github.

**Around 30 versions released so far, latest being 3.9 released on May 2020. Version 4.0 beta is now available.**

<https://www.typescriptlang.org/>

<https://time.graphics/line/155170>

**Angular Evolution**

**2010 – Angular JS released**

AngularJs built on the concept of “scope” and controllers,

**2016 – Angular 2 Released**

* Angular is a complete **rewrite** of AngularJS
* An Angular application and its architecture are different from AngularJS. The main building elements for Angular are **modules, components, templates, metadata, data binding, directives, services, and dependency injection**

Angular 2 also recommends using the TypeScript language, which introduces these features

* Support reactive programming using **RxJS**

**Angular 3**

Angular 3 was not launched due to misalignment in Angular’s router packaging which was already distributed as v3.3.0 at that time.

2017 - Angular 4

* Introducing **HttpClient**, a smaller, easier to use, and more powerful library for making HTTP Requests.
* New **router life cycle** events for Guards and Resolvers.
* Allowed adding BrowserAnimationsModule in NgModule.
* Angular 4 is compatible with TypeScript 2.1 and TypeScript 2.2.
* **Angular Universal**: The vast majority of the Angular Universal code has been merged into Angular core.

### 2017 – Angular 5

support for [progressive web apps](https://en.wikipedia.org/wiki/Progressive_web_app), a build optimizer and improvements related to Material Design

* **@angular/http** is replaced with **@angular/common/http** library.
* An improved compiler that supports incremental compilation meaning faster rebuilds.
* RxJS (reactive programming library) has been updated to version 5.5.2 or later.

### 2018 – Angular 6

Features like : ng update, ng add, Angular Elements, Angular Material + CDK Components, Angular Material Starter Components, CLI Workspaces, Library Support, Tree Shakable Providers, Animations Performance Improvements, and RxJS v6.

* Added a new command to manage npm dependencies — **ng update, ng add**
* Angular Material + CDK Components.
* Angular Elements
* Ivy Renderer
* Service Worker
* Schematics

### **2018 - Version 7**

Angular 7 was released on October 18, 2018. Updates regarding Application Performance, Angular Material & CDK, Virtual Scrolling, Improved Accessibility of Selects, now supports Content Projection using web standard for custom elements, and dependency updates regarding Typescript 3.1, RxJS 6.3, Node 10

* **Drag & Drop** API has been added.
* Supported **Virtual Scrolling**.
* CLI Prompts have been added.
* Angular Elements now supports content projection using web standards for custom elements.

### **2019 - Angular 8**

### Featuring Differential loading for all application code, Dynamic imports for lazy routes, Web workers, TypeScript 3.4 support, and Angular Ivy as an opt-in preview. Angular Ivy opt-in preview includes:

* Generated code that is easier to read and debug at runtime
* Faster re-build time
* Improved payload size
* Improved template type checking
* Route Configurations use Dynamic Imports using loadChildren.
* Added Web Worker Support.
* Much-anticipated Ivy compiler as an opt-in feature.

### **2020 - Angular 9**

Version 9 moves all applications to use the [Ivy compiler and runtime](https://angular.io/guide/ivy) by default. Angular has been updated to work with TypeScript 3.6 and 3.7. In addition to hundreds of bug fixes, the Ivy compiler and runtime offers numerous advantages:

* Smaller bundle sizes
* Faster testing
* Better debugging
* Improved CSS class and style binding
* Improved type checking
* Improved build times, enabling AOT on by default
* Improved [Internationalization](https://en.wikipedia.org/wiki/Internationalization_and_localization)

### **2020 - Angular 10** was released on June 24, 2020.

* New date range picker.
* Warnings about CommonJS imports
* Optional stricter settings that can improve maintainability, help catch bugs, and enable advanced optimizations.
* New versions supported of TypeScript 3.9, TSLib 2.0, TSLint 6.

**New Default Browser Configuration**

eprecating support for older browsers including IE 9, 10, and [Internet Explorer Mobile](https://en.wikipedia.org/wiki/Internet_Explorer_Mobile)

# Life Cycle Hooks

[Hooks for the Component](https://codecraft.tv/courses/angular/components/lifecycle-hooks/#_hooks_for_the_component)

**constructor**

This is invoked when Angular creates a component or directive by calling new on the class.

**ngOnChanges**

Invoked **every** time there is a change in one of th input properties of the component.

**ngOnInit**

Invoked when given component has been initialized.  
This hook is only called **once** after the first ngOnChanges

**ngDoCheck**

Invoked when the change detector of the given component is invoked. It allows us to implement our own change detection algorithm for the given component.

**ngOnDestroy**

This method will be invoked just before Angular destroys the component.  
Use this hook to unsubscribe observables and detach event handlers to avoid memory leaks.

**Hooks on children**

These hooks are only called for components and not directives.

#### **Note**

We will cover the difference between Components and Directives in the next section.

**ngAfterContentInit**

Invoked after Angular performs any content projection into the component’s view (see the previous lecture on Content Projection for more info).

**ngAfterContentChecked**

Invoked each time the content of the given component has been checked by the change detection mechanism of Angular.

**ngAfterViewInit**

Invoked when the component’s view has been fully initialized.

**ngAfterViewChecked**

Invoked each time the view of the given component has been checked by the change detection mechanism of Angular.