Overview

- History.
- Supported programming paradigms.
- Partial list of features.
- JS platforms.

Brief History

- Written by Brendan Eich within 10 days in May, 1995 at Netscape Communications. Name chosen based on marketing considerations to cash in on the popularity of Java.
- Microsoft released `jscript` in 1996. Incompatibilities introduced ("embrace and extend").
- Standardized as `EcmaScript` in June 1997 (ECMA-262).
- JavaScript got a bad reputation and was regarded as a poor programming language used for doing trivial things in the browser. Complexities caused by browser incompatibilities and the browser **Document Object Model** (DOM) were blamed on the language.
- Changed with the emergence of **Asynchronous JAvascript with Xml** (AJAX) in 2005.

Brief History Continued

- **JavaScript Object Notation** (JSON) popularized by Douglas Crockford emerged as a popular alternative to XML as a specification for data interchange between heterogeneous systems.
- Renaissance in js development. Browser incompatibilities and DOM complexities hidden by the use of libraries like `prototype`, `jquery` and `dojo`.
- `Node.js` released by Ryan Dahl in 2009. Popularized the use of js on the server.
- Succession of different ECMA standards: es 3, es 5. Currently, evolving as an "evergreen" language with standard updates being released yearly: es 2015 ... 2019.
- Allows use of a single programming language across the entire web stack. Most popular programming language in terms of deployments.

Multi-Paradigm Language

JavaScript supports programming using multiple paradigms:

- Imperative / Procedural.
- Object-oriented.
- Functional.

Imperative Programming

- Imperative programming gives a step-by-step description to a computer **how to solve** a problem.
- AKA **procedural programming** as program is a how-to *procedure* to solve a problem.
- Involved mutating shared state.
- Maps very closely to operation of computer; relatively low abstraction.

Object-Oriented Programming

- Package data and functions associated with that data together into *objects*.
- Objects can provide encapsulation.
- Usually some kind of inheritance (JS uses a non-traditional model of inheritance).
- Classes are not essential to JS OO programming.

Functional Programming

JS supports multiple degrees of functional programming:

- **Pure functions**: the result of a function depends only on its arguments; it is not allowed to do anything other than return a result. It cannot change any global state, including I/O.
  It is possible to write pure functions even in a language like C.
- **First-class functions**. Functions can be treated like any other datatype: stored in variables, passed between functions.
- **Lack of mutation**; clumsy but not impossible in JS. Forced in languages like Haskell.

Some Language Features

- Dynamically typed: variables are untyped, but values have types. Permits the use of **duck typing**.
- Initially interpreted, now compiled using techniques like runtime compilation.
- Possible to evaluate strings representing code at runtime using `eval()`.
- Allows reentering a function invocation using *generators*.
- Borrows concepts from Scheme, Perl and Self.
- Standard library is highly asynchronous.

JavaScript Platforms

JavaScript runs on two main platforms:

1. **Browser**: Platform provides interfaces to numerous browser technologies like the *Document Object Model* DOM and browser storage.
2. **Server**: Exemplified by `nodejs`. Platform provides access to filesystem, processes, etc.