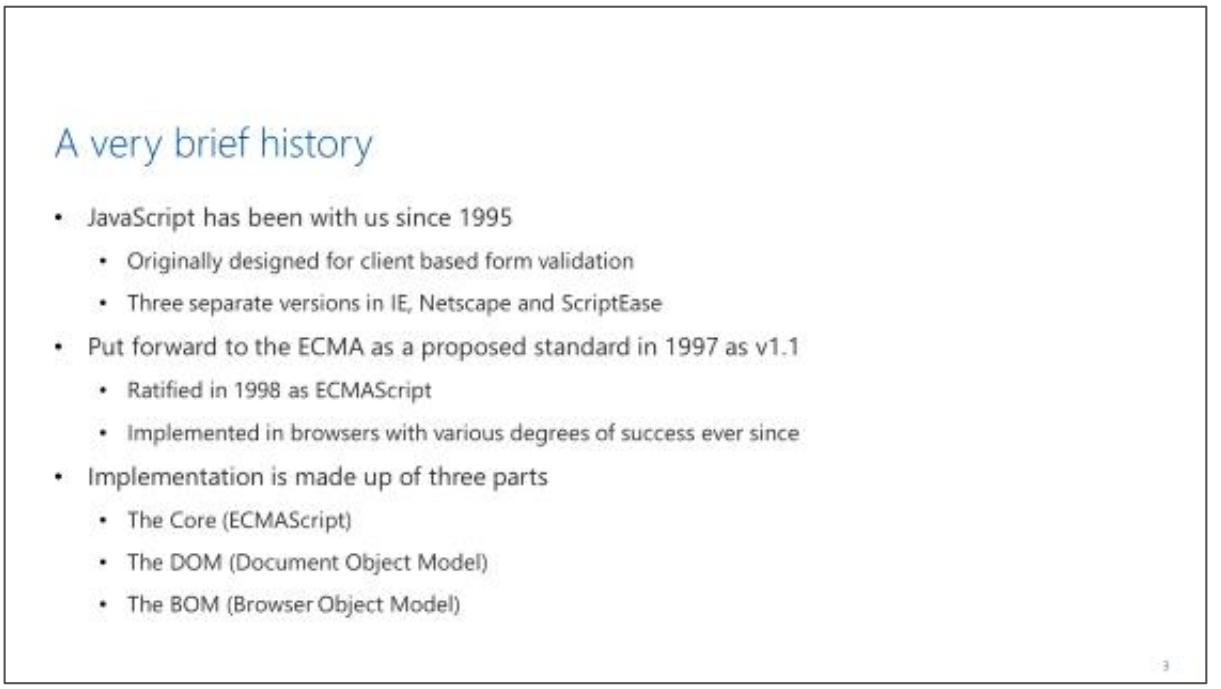
Introduction to JavaScript

PROGRAMMING WITH JAVASCRIPT

Introduction

- A very brief history
- What is JavaScript?
- How to place script in a web page
  - Embedding
  - Linking
  - <noscript>
- Visual Studio
- Chrome Developer tools

2

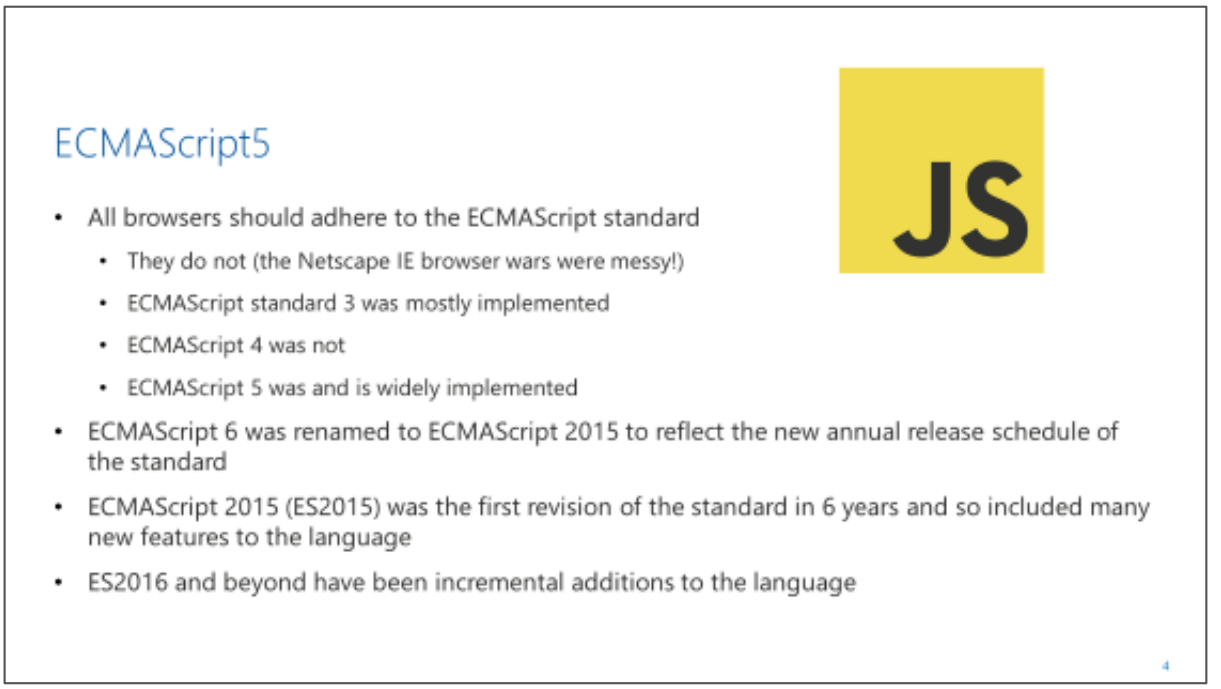


Whether you are here as a new web programmer or an old hand server guru forced to consider the front end in a different way because of those dashed, new fangled MVC approaches, JavaScript is a core part of any web developer's arsenal. JavaScript is supported in almost every browser today and increasingly in a universal way; but, as we will see, older browsers cause us real issues with the way we code and what we can do.

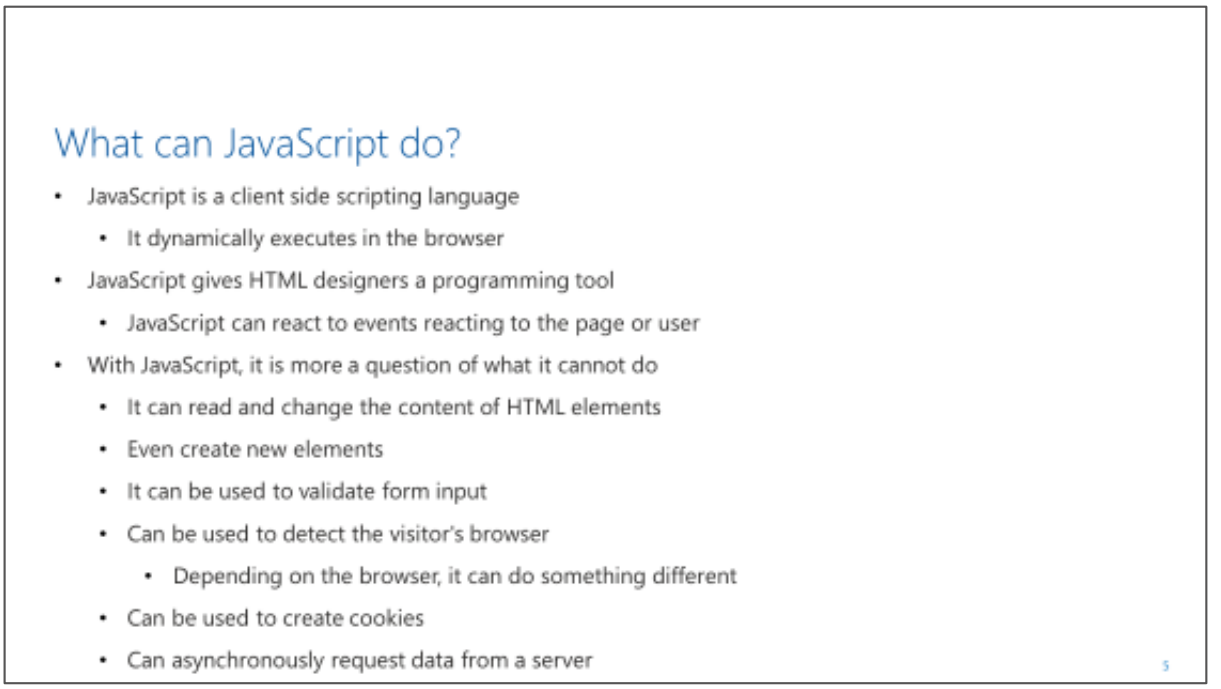
JavaScript is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

JavaScript was formalised in the ECMAScript language standard and is primarily used in the form of client-side JavaScript, implemented as part of a Web browser in order to provide enhanced user interfaces and dynamic websites. This enables programmatic access to computational objects within a host environment.

In recent years, it has found usage in non-web programming architectures, such as Acrobat files and Windows 8 Metro applications, demonstrating the amazing versatility of this incredibly flexible and powerful language and server-side programming with tools such as Node.js.



The 6th edition, officially known as ECMAScript 2015, was finalised in June 2015. This update adds significant new syntax for writing complex applications, including classes and modules, but defines them semantically in the same terms as ECMAScript 5 strict mode. Other new features include iterators and for/of loops, Python-style generators and generator expressions, arrow functions, binary data, typed arrays, collections (maps, sets and weak maps), promises, number and math enhancements, reflection, and proxies (metaprogramming for virtual objects and wrappers).



JavaScript gives HTML designers and developers a programming tool. It can react to events reacting to the page or user.

JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element.

JavaScript can manipulate HTML elements. JavaScript can read and change the content of an HTML element.

JavaScript can be used to validate data. JavaScript can be used to validate form input.

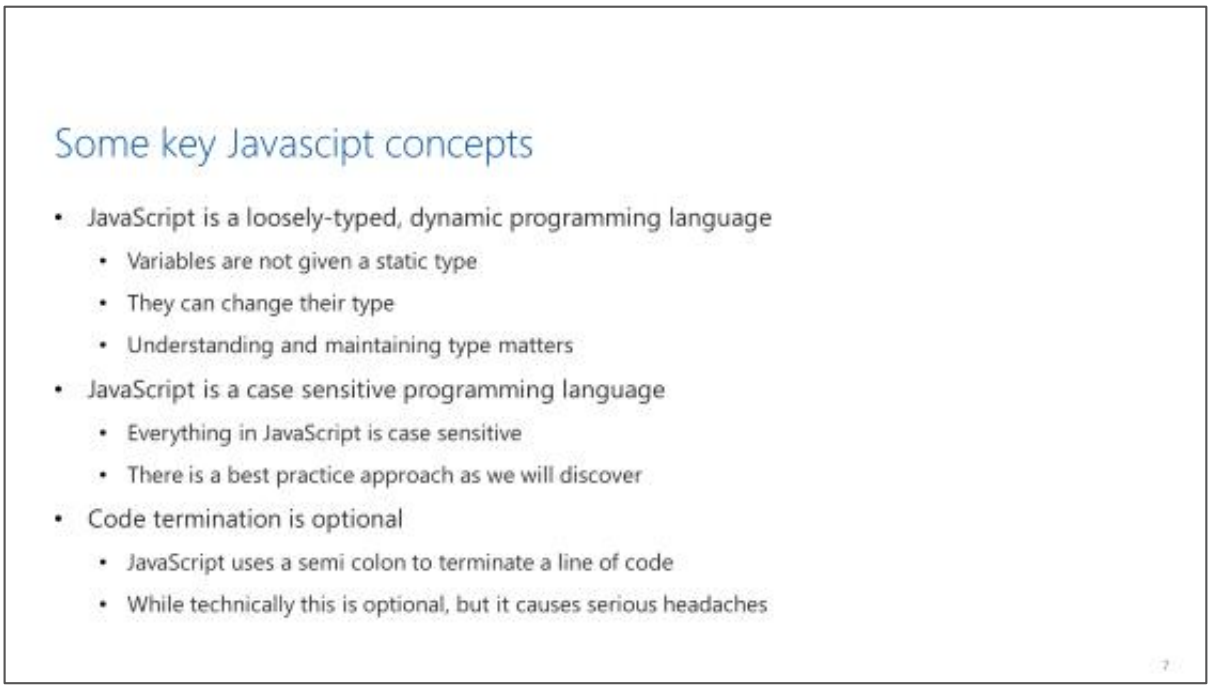
JavaScript can be used to detect the visitor's browser. JavaScript can be used to detect the visitor's browser, and – depending on the browser – load another page specifically designed for that browser.

JavaScript can be used to create cookies. JavaScript can be used to store and retrieve information on the visitor's computer.

What can JavaScript do?

- JavaScript is a scripting language
- JavaScript gives front-end developers a programming tool
- JavaScript can react to events created by the page itself (page loaded) or the user (click)
- It can read and change the content of HTML elements, create new content, remove and hide elements
- It can be used to validate form input
- Can provide access to HTML 5 APIs such as indexeddb, geolocation, canvas and more
- Can be used to create cookies
- Can asynchronously request data from a server
- JavaScript is also widely used as a server-side language, via NodeJS
- JavaScript can be used to create desktop applications via tools, such as Electron
- JavaScript can be used to create native mobile applications via tools, such as React Native

6



As we begin our journey with JavaScript, we will get a few things locked down in our brainboxes that will save us a lot of heartache and pain as the course goes on.

JavaScript is a case-sensitive language. This means…

```
var numberOne = 5;
var NumberOne = 5;
```

… actually creates two separate variables! This is a key fact to remember as you code – be careful as we move on in the development of our code.

JavaScript will work without a semicolon terminating the code in many situations, but this will also cause you real issues as the course draws to a conclusion and we discuss minificiation of script. Please try, therefore, to remember that every instruction needs to be terminated.

The browser needs to know the data it is rendering is not just normal text and is actually JavaScript. We achieve this by using the <script> element. You may use as many script elements as you like on a page. Browsers today will assume that your script is client side JavaScript, if no further information is provided.

Script blocks are executed as soon as the browser reaches the code block in the page. The script will then be executed; we will see the importance of this concept as the course progresses.

Everything said about inline script processing is also true for linked scripts. The key consideration when using external script references is with performance. Each script file is a separate request to the server for the file. On first request to the page, this can cause some lag in a page rendering. Older browser, IE 6 being a notable example, can only make two asynchronous file requests from a server at a time, so the page load can be uneven. Very often, you compact your files together to create a single script file to assist with this; we will see this strategy later in the course.

The big benefit of this strategy of external linking is that script to be used on multiple pages can be managed centrally, making site maintenance much simpler and, as per usual, with asset caching the initial first cost leads to a quicker load on subsequent visits.

Everything said about inline script processing is also true for linked scripts. The key consideration when using external script references is with performance. Each script file is a separate request to the server for the file. On first request to the page, this can cause some lag in a page rendering. Older browser, IE 6 being a notable example, can only make two asynchronous file requests from a server at a time so the page load can be uneven. Very often, you compact your files together to create a single script file to assist with this; we will see this strategy later in the course.

The big benefit of this strategy of external linking is that script to be used on multiple pages can be managed centrally, making site maintenance much simpler and, as per usual, with asset caching the initial first cost leads to a quicker load on subsequent visits.

The <noscript> element

- Client-side scripting may not be available
- The <noscript> element will only render its content if
  - The browser does not understand <script>
  - The client has disabled script

The <noscript> element

```
<noscript>
        If you see this, you do not have JavaScript
enabled.
</noscript>
```

11

JavaScript has two ways of marking a comment: // starts a comment, which ends at the end of the current line; and multi-line comments, which can be created by inserting the comment text between /* (start-comment) and */ (end-comment).

Note that multi-line comments do not nest. This typical debugging method will cause problems:

```
/*  something wrong in this function
    commented out temporarily to isolate the problem

    function fred()
    {
       /* this function computes mortgage interest
       */
       x = ((y * 3) + 4 )/ 6.291 ^ 3.412
    }

*/
```

The first */ will close all comments. The second */ will be a syntax error.

Walkthrough – using Visual Studio Code

- We will be using Visual Studio Code on this course as our editor – free, open source and built using JavaScript!
- Not to be confused with Visual Studio

13

Walkthrough – Chrome development tools

- Most browsers have development and debugging tools
- Incredibly useful for testing and management

14

Review

- What is JavaScript?
  - A scripting language
- What is JavaScript for?
  - Building client-side, server-side, desktop and mobile applications
- How do we use JavaScript?
  - Linked or embedded

15

Exercise

- There is no exercise for this module

16