CSCC09F
Programming on the Web

Client-Side Programming

Client-side scripting, JavaScript overview,
JavaScript language, DOM

Client-Side Scripting

- advantages :
  - Improved UI responsiveness – avoid page-reload from server, eg:
    - SPA view-change
    - list filtering, sorting
    - client-side data validation
    - local code often 10-100 times faster than server interaction
  - potential for "offline" Web apps (supported in HTML5)
  - offloads processing to client that is usually less loaded than server

- disadvantages:
  - scripts increasing size and time for initial-page downloads
  - scripts are readable, expose source code (IP), though can obfuscate
  - security risk posed by downloaded code; e.g. much of today's malware utilizes client-side scripting to enter your computer

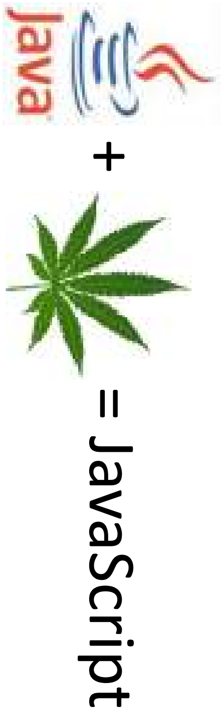
JavaScript Origins

- Netscape originated *LiveScript*, later renamed JavaScript at the last minute.
  - Invented by Brendan Eich at Netscape
  - Perceived competition with Sun Microsystems' Java-applets for client-side programming
- Microsoft has a similar language called JScript
- JavaScript and Java are complementary
  - JavaScript
    - cannot draw, multi-thread, network, or do I/O (but note HTML5's API's add support for all of these)
  - Java (Applets, for client side, no longer covered in C09)
    - cannot interact with browser or control page content
  - JavaScript and Java working together on the same document, could jointly accomplish task neither could independently

History and Java Relationship

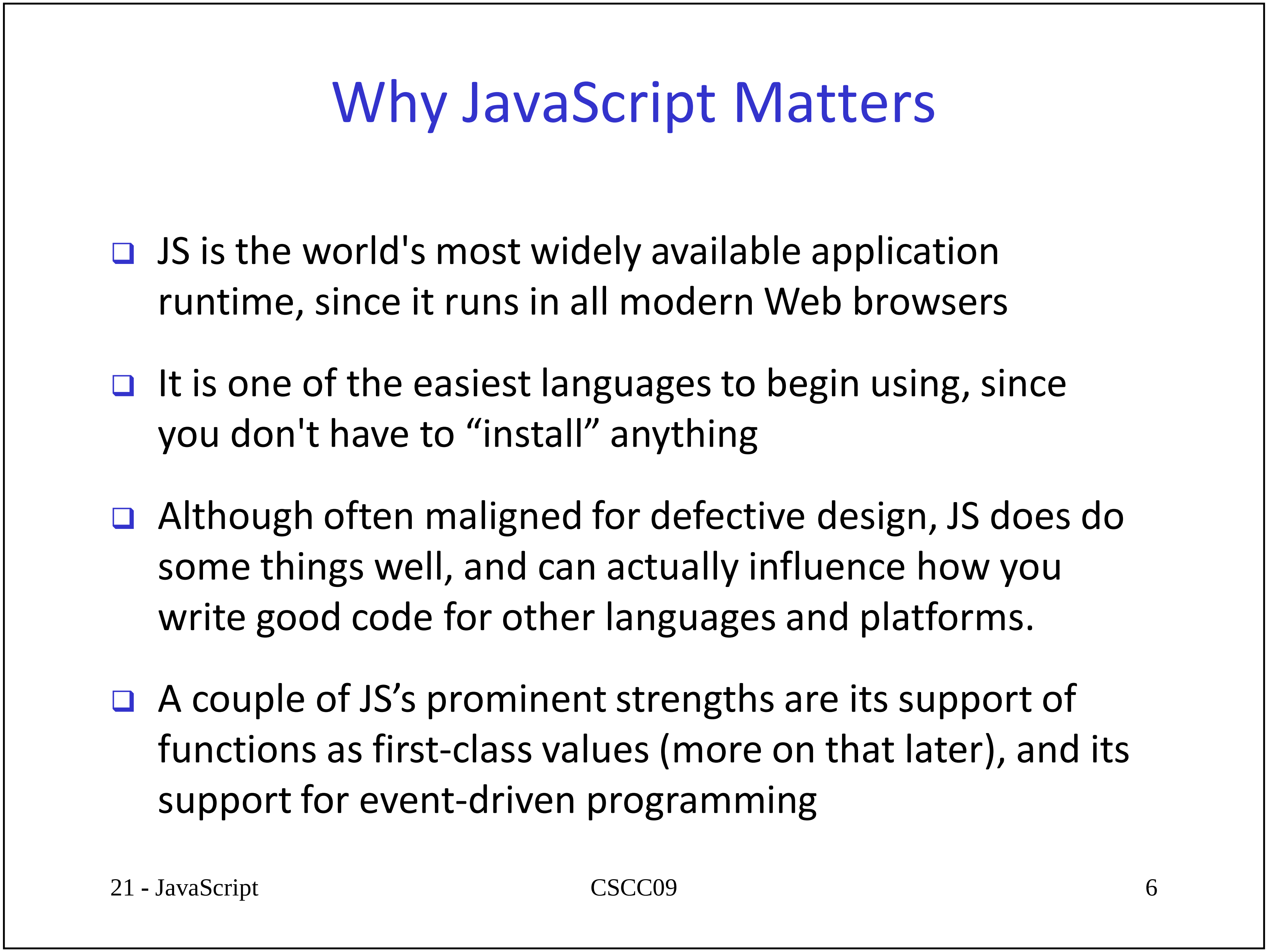
- *"JS had to "look like Java" only less so, be Java's dumb kid brother or boy-hostage sidekick. Plus, I had to be done in ten days, or something worse than JS would have happened."* - Brandon Eich

- As it happens, there is a lot of syntactic overlap with Java, e.g. comments, if-stmts, for-loops, while-stmts are the same as in Java
- Some things are very different, e.g. data typing rules, class/object handling, handling of Boolean values, variable scoping, treatment of functions, "first-class" regular-expressions



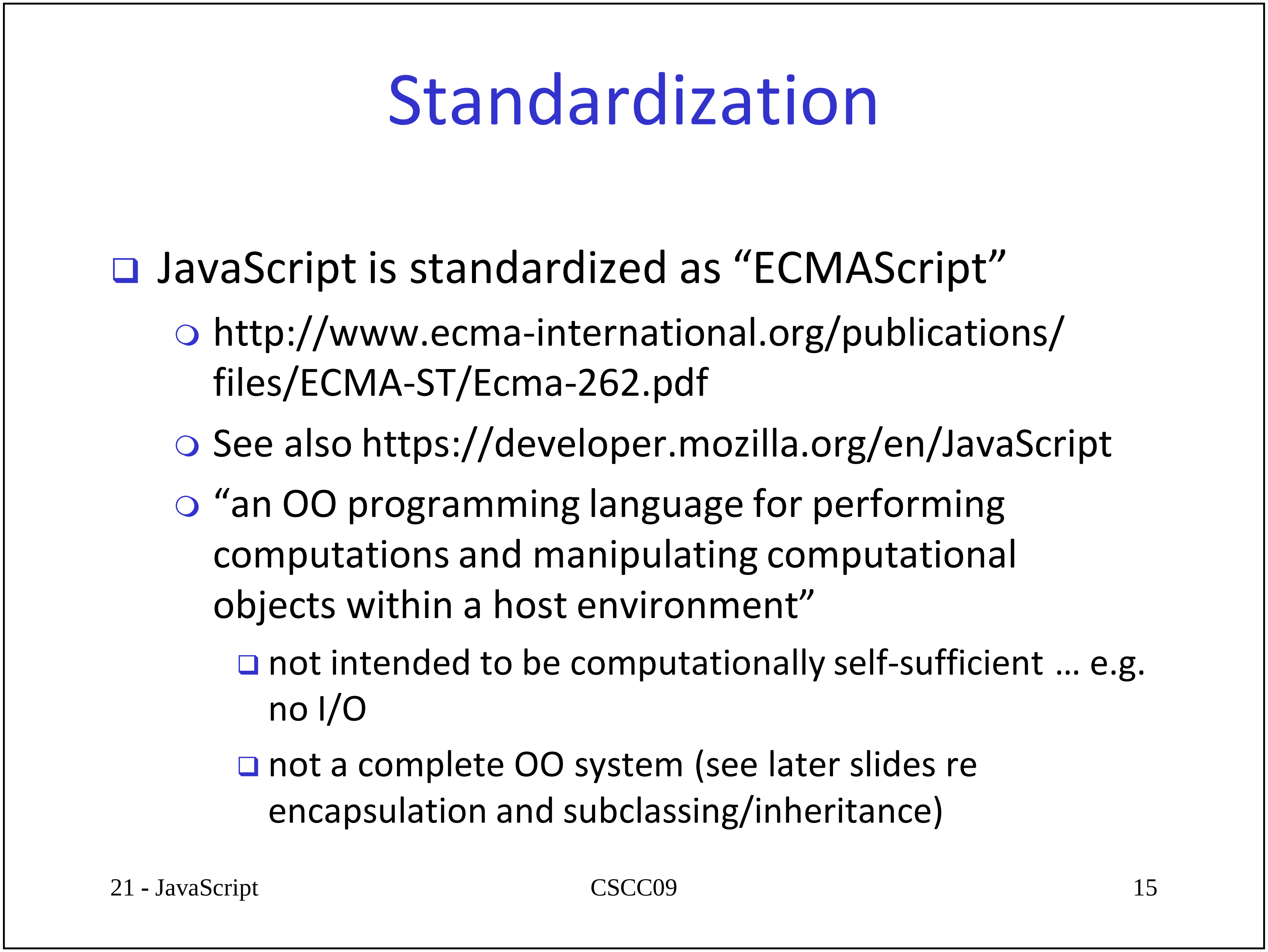
JavaScript the Language

- Early on, was considered a defective, poorly-designed language (what do you expect in 10 days?)
- Later it was dismissed as a simple language that you could pick up “on the job”
- Biggest hurdle faced by beginner developers – not bothering to actually learn the language before writing code – sounds absurd, but is surprisingly common!
  - “Javascript is the only language where good programmers believe they can use it effectively, without learning it.“ Douglas Crockford
  - Maybe feasible for HTML or CSS, but not for JS
- JavaScript now recognized as more powerful and more sophisticated than originally thought



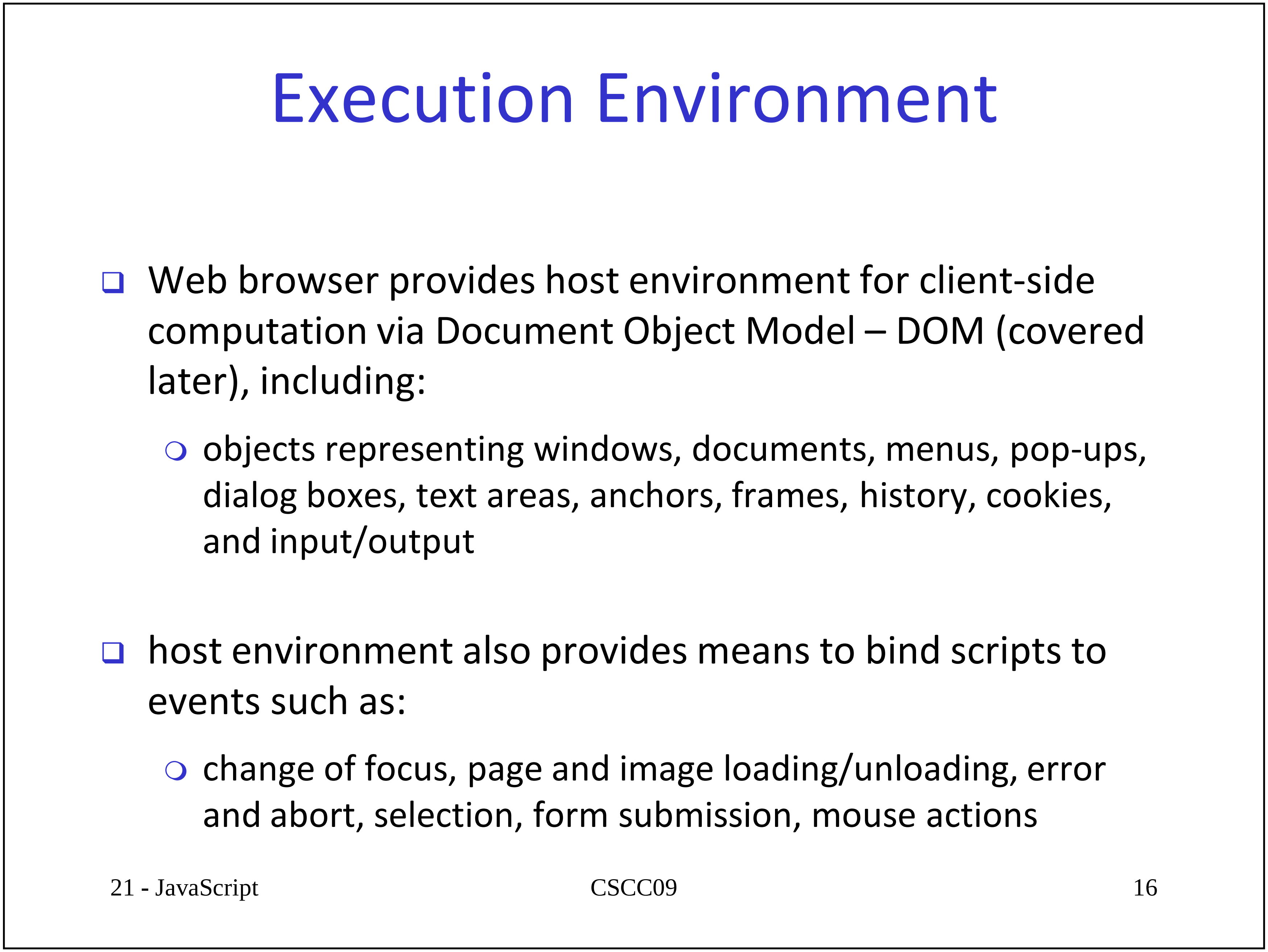
Why JavaScript Matters

- ❑ JS is the world's most widely available application runtime, since it runs in all modern Web browsers
- ❑ It is one of the easiest languages to begin using, since you don't have to “install” anything
- ❑ Although often maligned for defective design, JS does do some things well, and can actually influence how you write good code for other languages and platforms.
- ❑ A couple of JS’s prominent strengths are its support of functions as first-class values (more on that later), and its support for event-driven programming



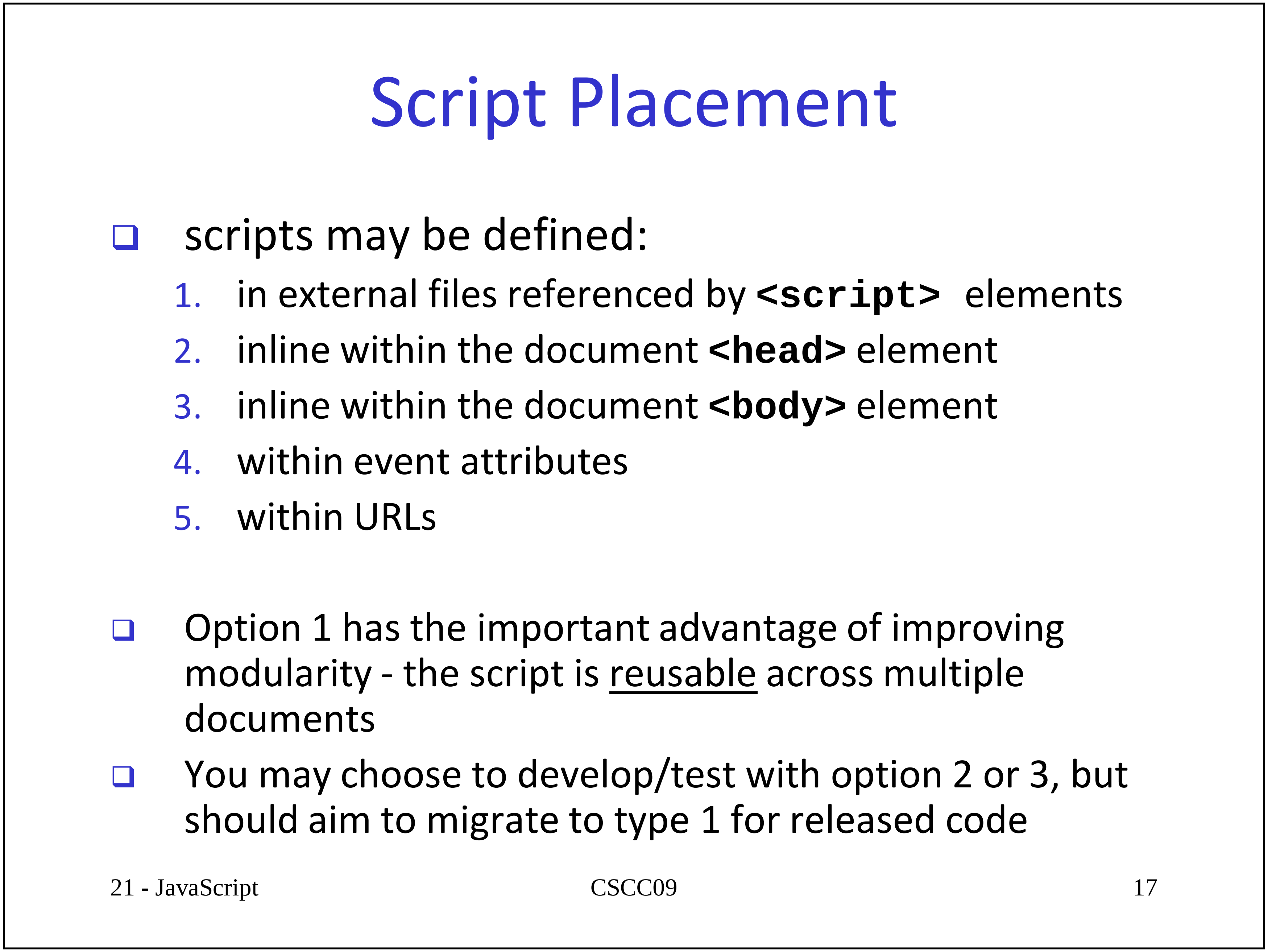
Standardization

- JavaScript is standardized as “ECMAScript”
  - http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
  - See also https://developer.mozilla.org/en/JavaScript
  - “an OO programming language for performing computations and manipulating computational objects within a host environment”
    - not intended to be computationally self-sufficient … e.g. no I/O
    - not a complete OO system (see later slides re encapsulation and subclassing/inheritance)



Execution Environment

- Web browser provides host environment for client-side computation via Document Object Model – DOM (covered later), including:
  - objects representing windows, documents, menus, pop-ups, dialog boxes, text areas, anchors, frames, history, cookies, and input/output
- host environment also provides means to bind scripts to events such as:
  - change of focus, page and image loading/unloading, error and abort, selection, form submission, mouse actions



Script Placement

- scripts may be defined:
  1. in external files referenced by **`<script>`** elements
  2. inline within the document **`<head>`** element
  3. inline within the document **`<body>`** element
  4. within event attributes
  5. within URLs

- Option 1 has the important advantage of improving modularity - the script is reusable across multiple documents
- You may choose to develop/test with option 2 or 3, but should aim to migrate to type 1 for released code

Script Placement

- scripts may be defined:
  1. in external files referenced by <script> elements
  2. inline within the document <head> element
- option 1 syntax:

```
<script type="text/javascript" src="script.js">
</script>
```

  - note the following deprecated form (still in wide circulation):

```
<script language="javascript" src="script.js"> ...
```

- option 2 syntax:

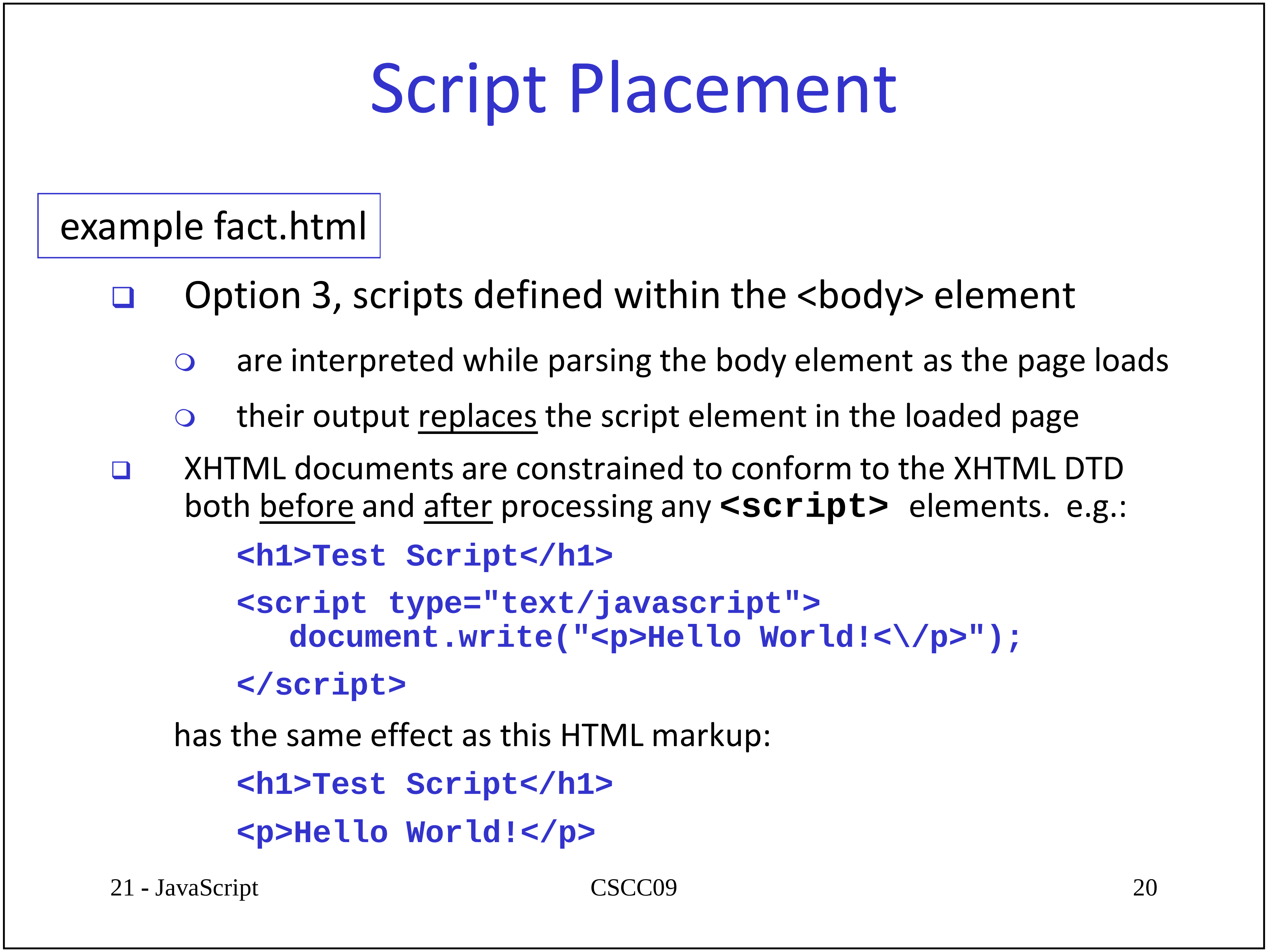
```
<script type="text/javascript" >
   … inline script code
</script>
```

Script Placement

- alternative syntax for browsers without script support (or scripts turned off) – why necessary?

```
<script type="text/javascript">
   <!--
    … javascript code …
     // end of script -->
</script>
<noscript>
    … alternate content …
</noscript>
```

(example **noscript.html**)



Script Placement

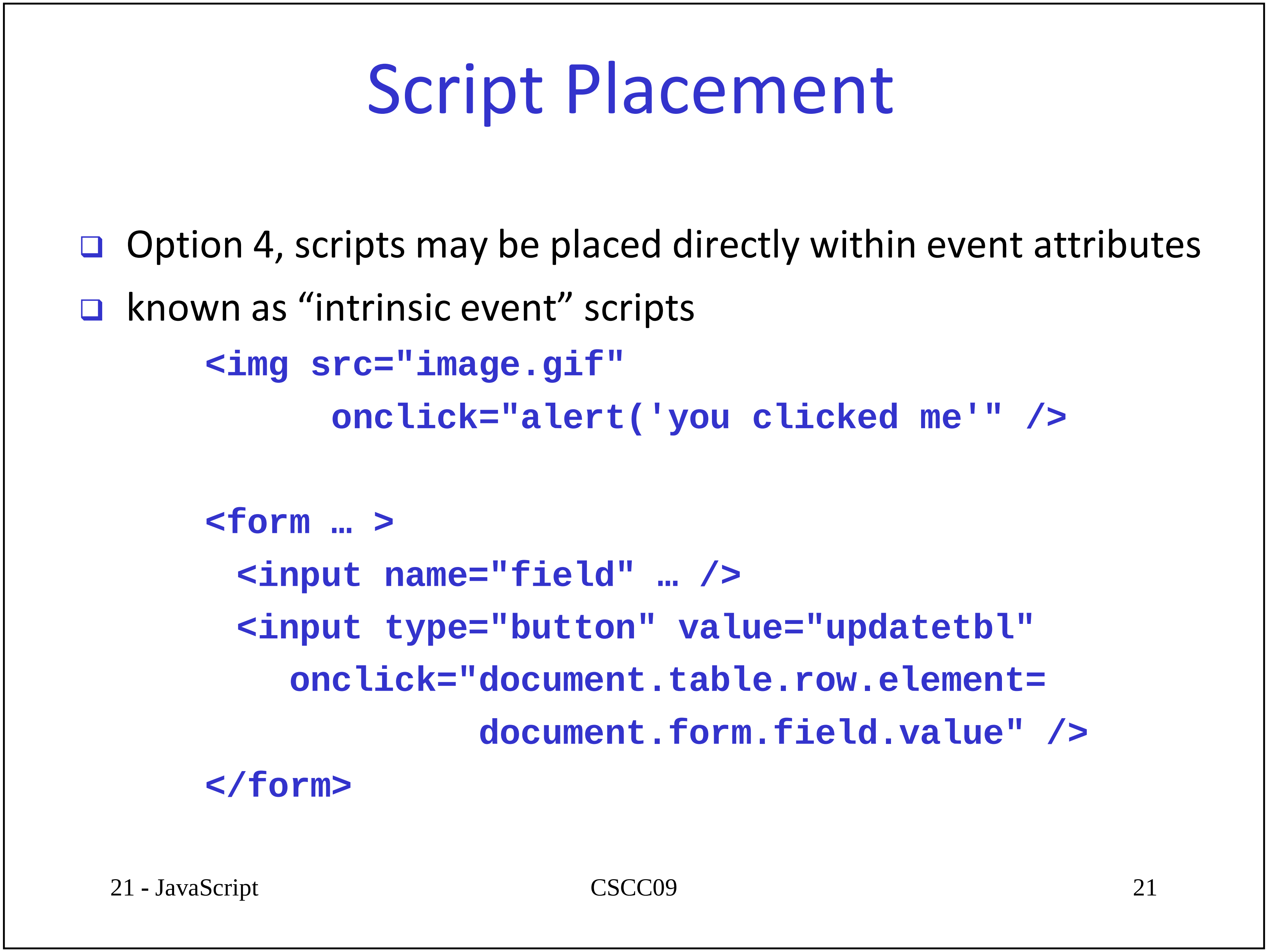
example fact.html

- Option 3, scripts defined within the <body> element
  - are interpreted while parsing the body element as the page loads
  - their output replaces the script element in the loaded page
- XHTML documents are constrained to conform to the XHTML DTD both before and after processing any **`<script>`** elements. e.g.:

```
<h1>Test Script</h1>
<script type="text/javascript">
   document.write("<p>Hello World!<\/p>");
</script>
```

has the same effect as this HTML markup:

```
<h1>Test Script</h1>
<p>Hello World!</p>
```

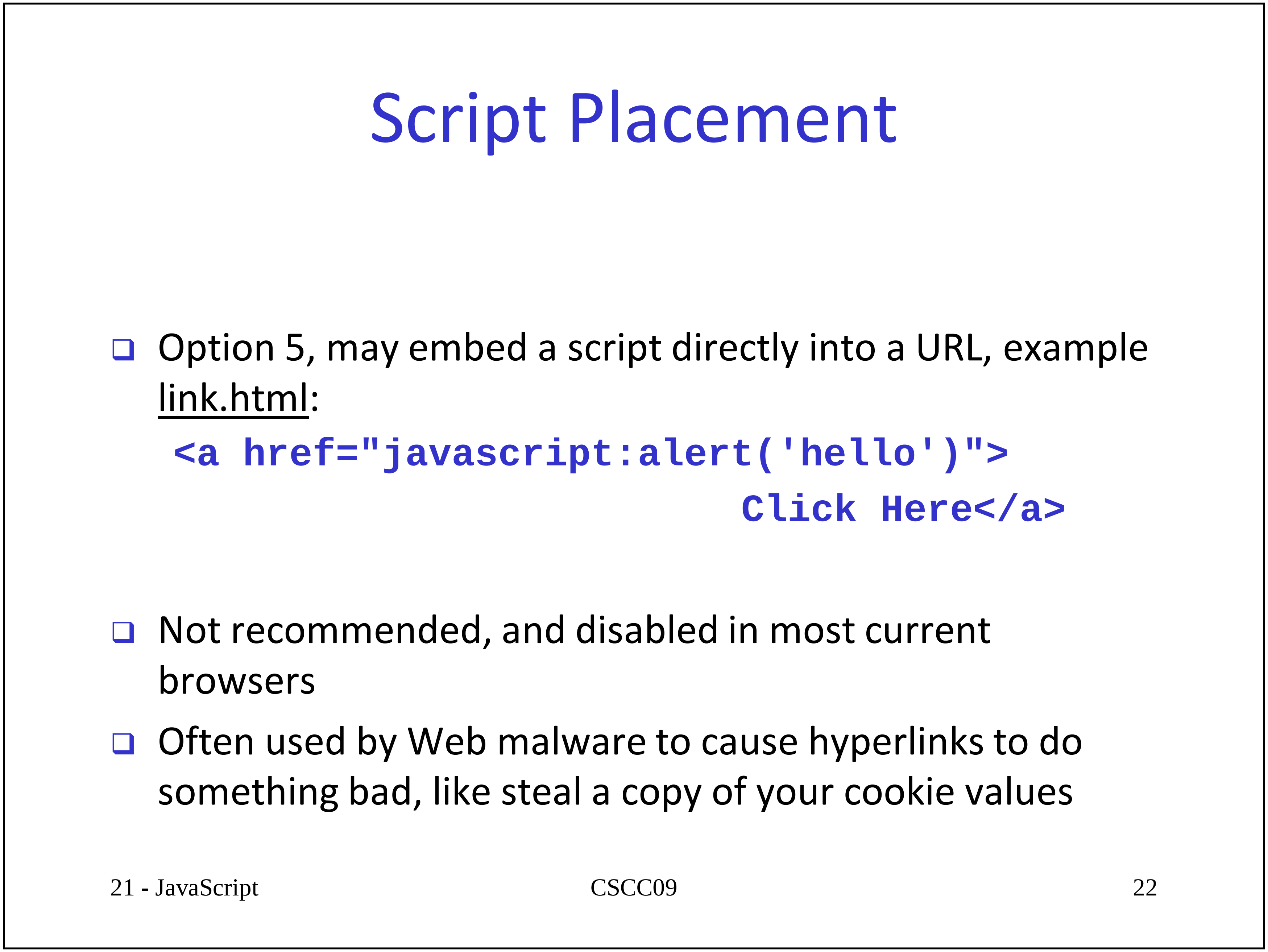


Script Placement

- Option 4, scripts may be placed directly within event attributes
- known as “intrinsic event” scripts

```
<img src="image.gif"
      onclick="alert('you clicked me'" />

<form … >
  <input name="field" … />
  <input type="button" value="updatetbl"
    onclick="document.table.row.element=
             document.form.field.value" />
</form>
```



Script Placement

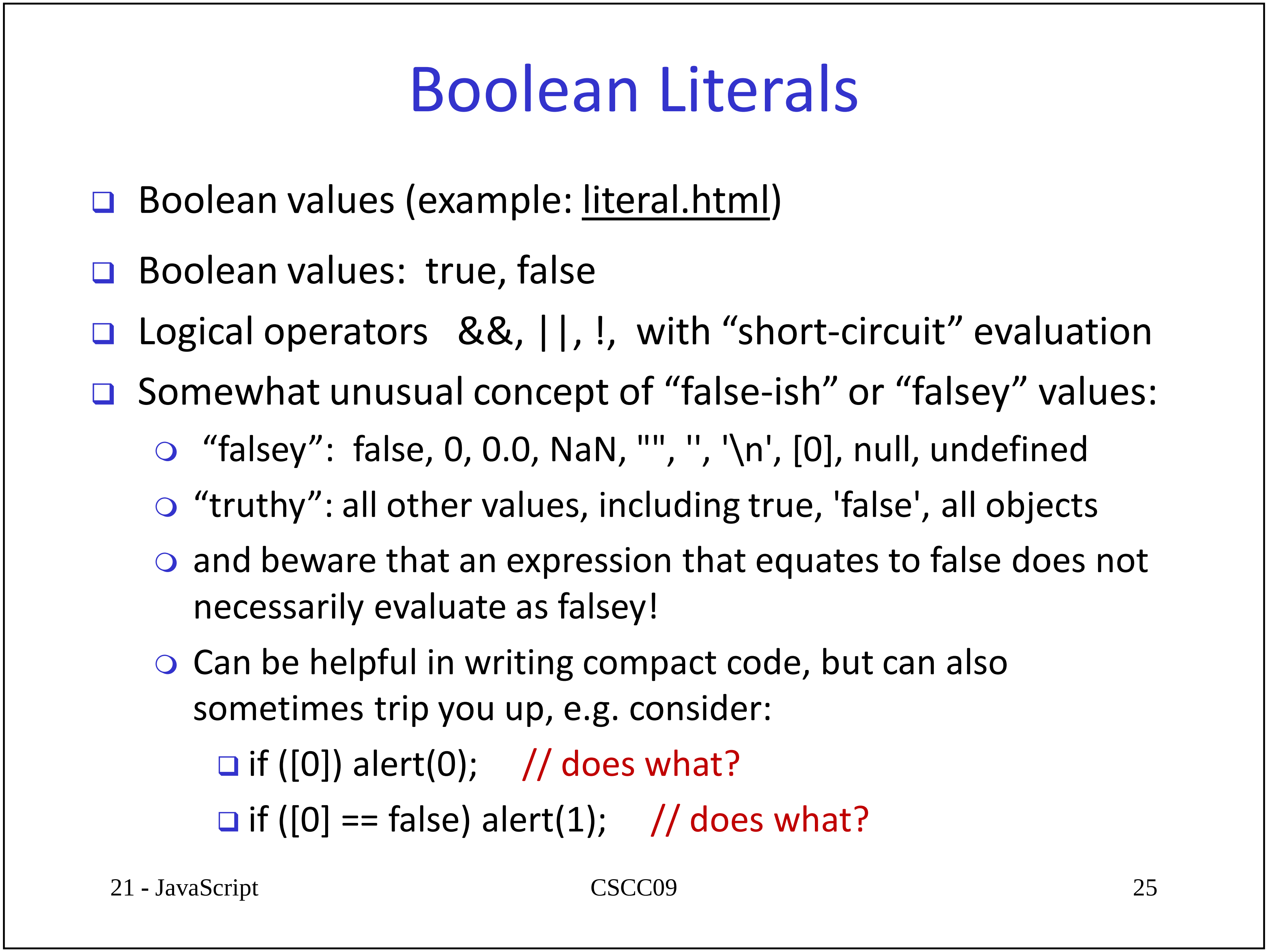
- Option 5, may embed a script directly into a URL, example link.html:

```
<a href="javascript:alert('hello')">
                                    Click Here</a>
```

- Not recommended, and disabled in most current browsers
- Often used by Web malware to cause hyperlinks to do something bad, like steal a copy of your cookie values

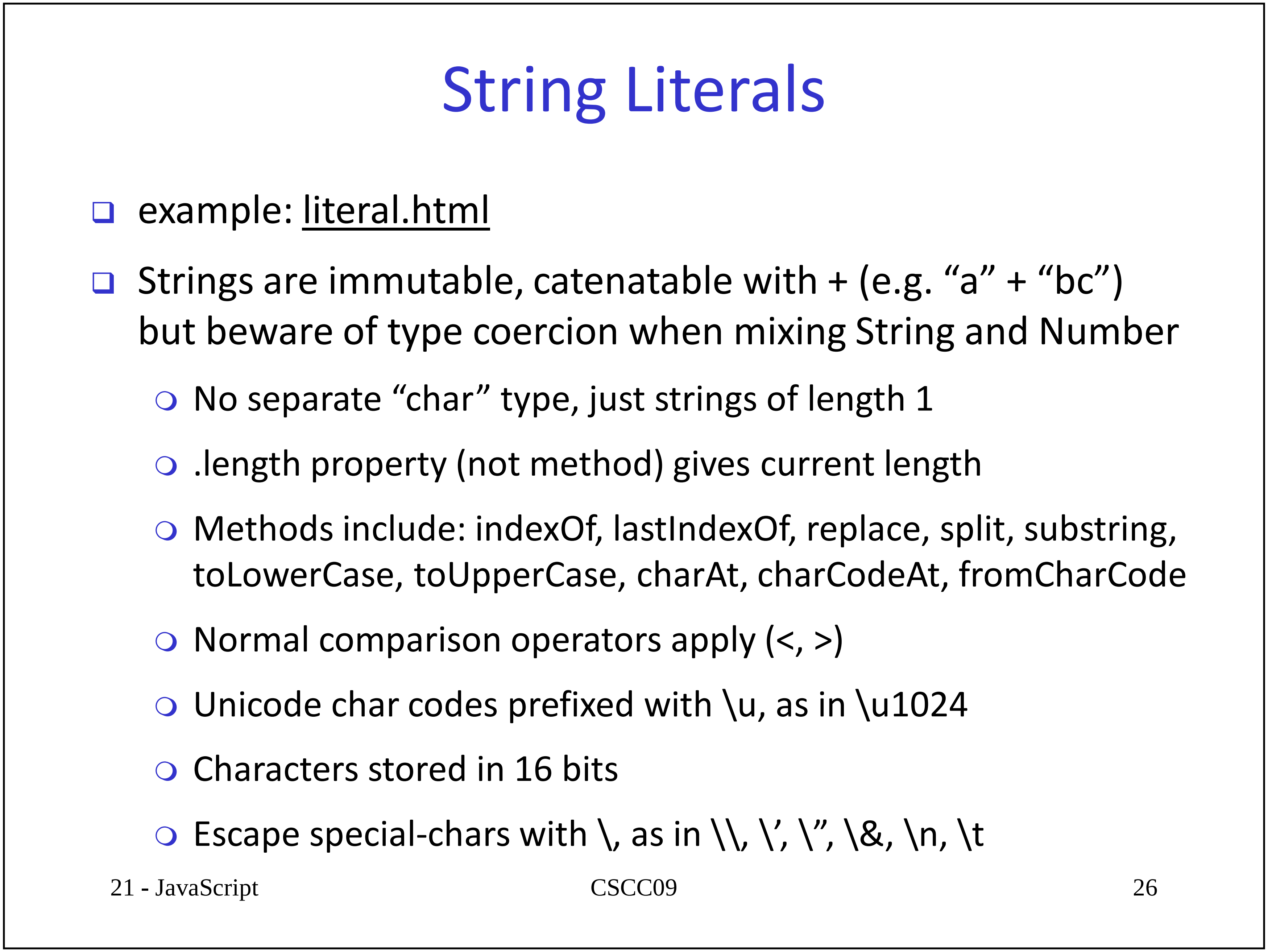
Data Typing

- Dynamically typed
  - different than Java or C … more like Python, and takes some ideas from functional languages like Scheme
  - a variable can hold any type of value:
    - number (8-byte IEEE fp)
    - string
    - boolean
    - function (first-class data type)
    - Object (DOM or JS)
    - array (whose elements can be of mixed types)
  - … and can hold values of different types at different times during execution - beware! ⚠



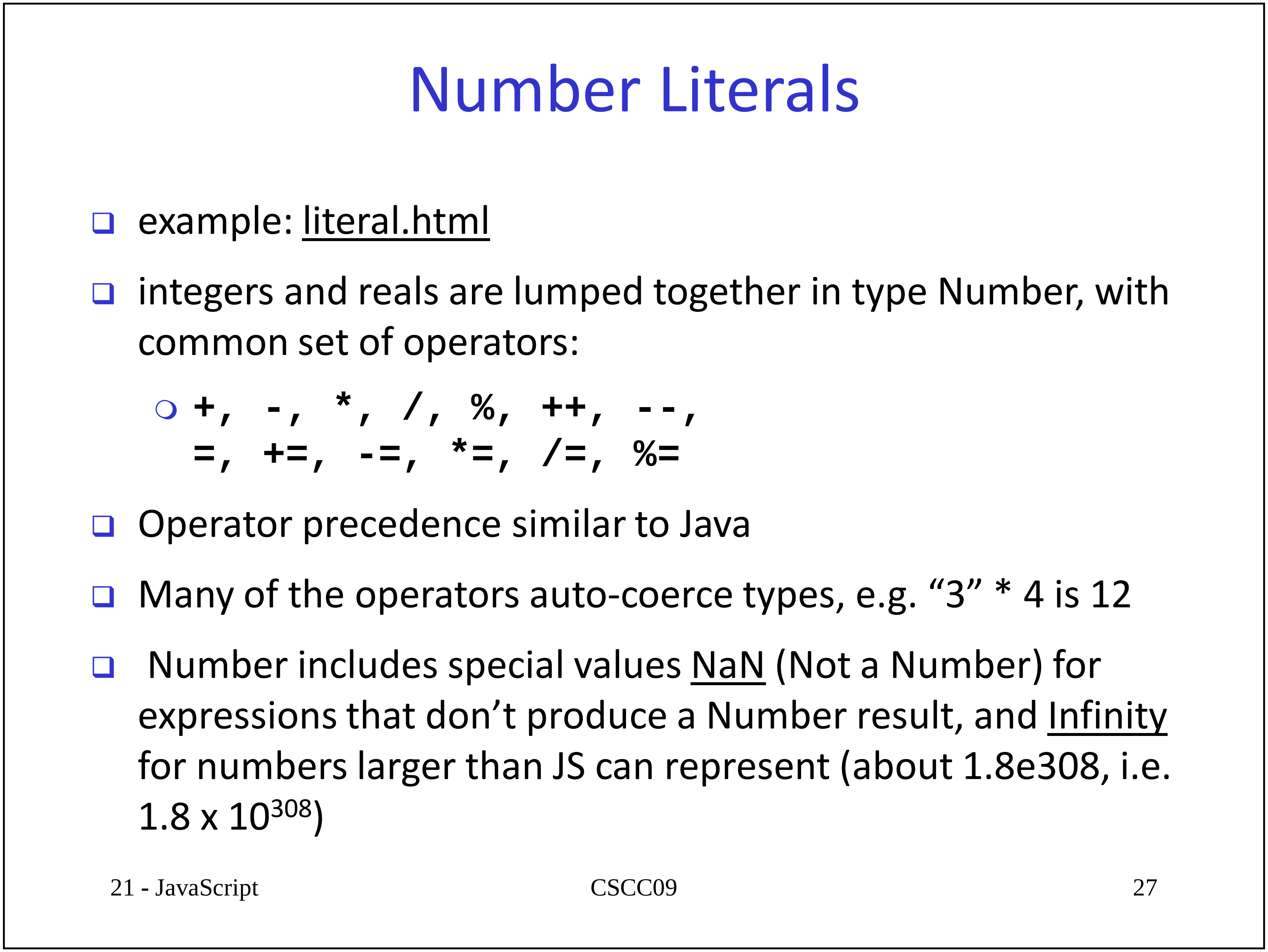
Boolean Literals

- Boolean values (example: literal.html)
- Boolean values: true, false
- Logical operators &&, ||, !, with “short-circuit” evaluation
- Somewhat unusual concept of “false-ish” or “falsey” values:
  - “falsey”: false, 0, 0.0, NaN, "", '', '\n', [0], null, undefined
  - “truthy”: all other values, including true, 'false', all objects
  - and beware that an expression that equates to false does not necessarily evaluate as falsey!
  - Can be helpful in writing compact code, but can also sometimes trip you up, e.g. consider:
    - if ([0]) alert(0); // does what?
    - if ([0] == false) alert(1); // does what?



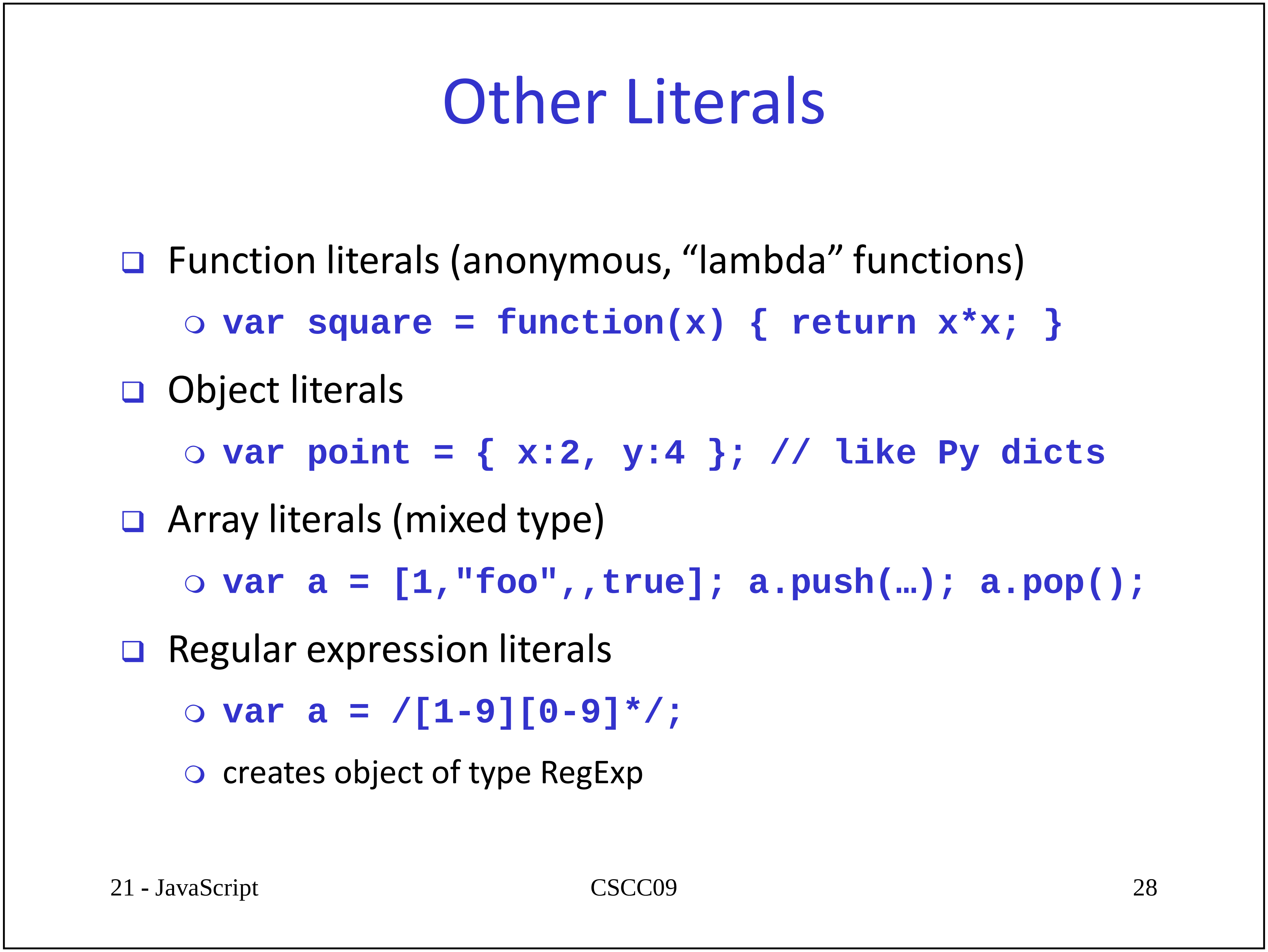
String Literals

- example: literal.html
- Strings are immutable, catenatable with + (e.g. “a” + “bc”) but beware of type coercion when mixing String and Number
  - No separate “char” type, just strings of length 1
  - .length property (not method) gives current length
  - Methods include: indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase, charAt, charCodeAt, fromCharCode
  - Normal comparison operators apply (<, >)
  - Unicode char codes prefixed with \u, as in \u1024
  - Characters stored in 16 bits
  - Escape special-chars with \, as in \\, \’, \”, \&, \n, \t



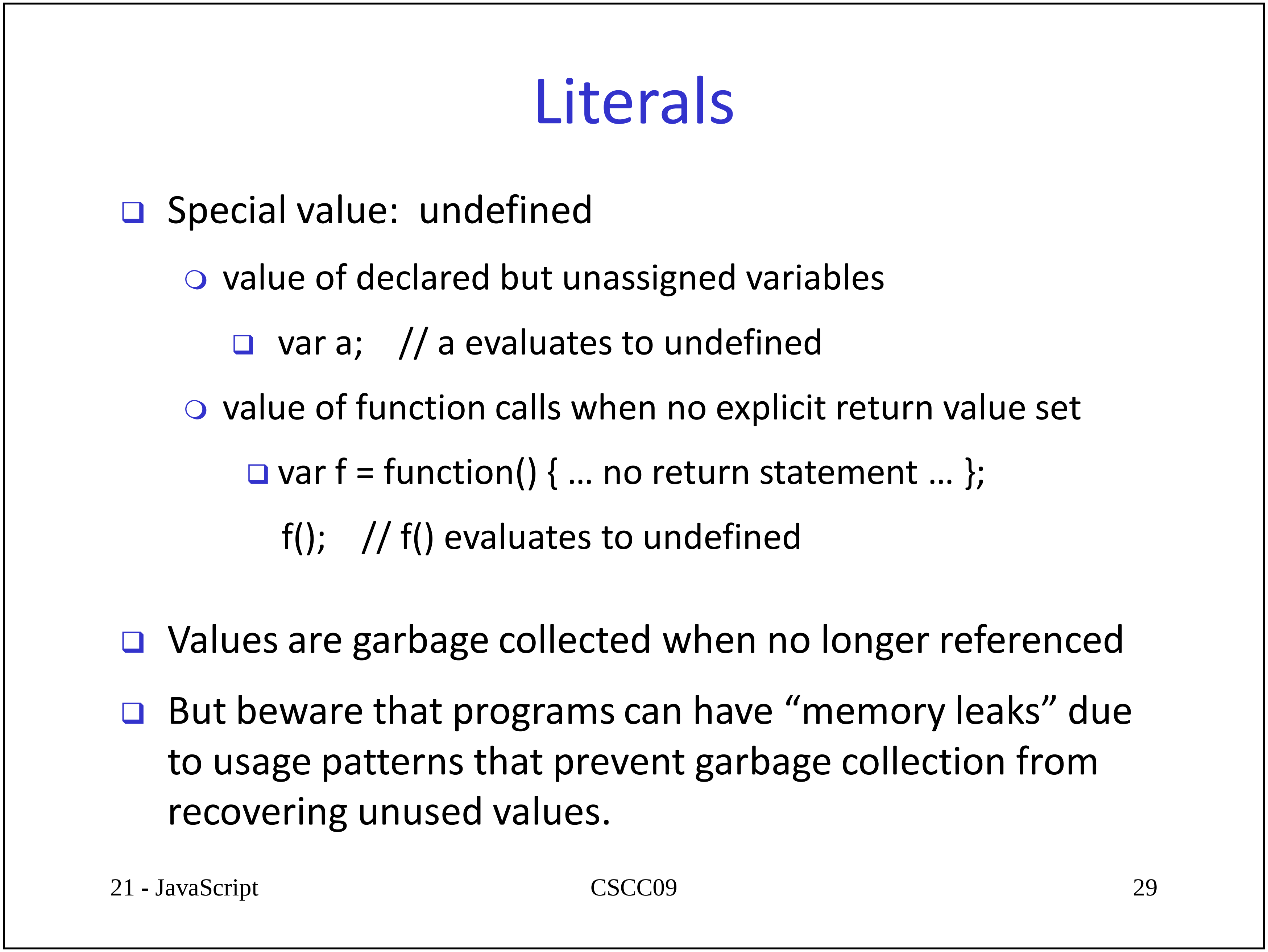
Number Literals

- example: literal.html
- integers and reals are lumped together in type Number, with common set of operators:
  - `+, -, *, /, %, ++, --,`
    `=, +=, -=, *=, /=, %=`
- Operator precedence similar to Java
- Many of the operators auto-coerce types, e.g. “3” * 4 is 12
- Number includes special values NaN (Not a Number) for expressions that don’t produce a Number result, and Infinity for numbers larger than JS can represent (about 1.8e308, i.e. 1.8 x 10^{308})



Other Literals

- Function literals (anonymous, “lambda” functions)
  - `var square = function(x) { return x*x; }`
- Object literals
  - `var point = { x:2, y:4 }; // like Py dicts`
- Array literals (mixed type)
  - `var a = [1,"foo",,true]; a.push(…); a.pop();`
- Regular expression literals
  - `var a = /[1-9][0-9]*/;`
  - creates object of type RegExp



Literals

- Special value: undefined
  - value of declared but unassigned variables
    - var a;   // a evaluates to undefined
  - value of function calls when no explicit return value set
    - var f = function() { … no return statement … };

      f();   // f() evaluates to undefined
- Values are garbage collected when no longer referenced
- But beware that programs can have “memory leaks” due to usage patterns that prevent garbage collection from recovering unused values.

Variable Declarations

- Variable declaration; note no data-typing ⚠
  - var i = 12,   msg = “hello”, test;   // what is the value of test?
- If you omit a variable declaration (keyword var):
  - automatically declared in global scope (beware! can lead to very hard-to-debug errors) ⚠
  - `"use strict";` instructs browser to flag this as an error
- Beware: “code-hoisting” and no “block-level” scope. ⚠ What will this function alert? Example block.html

```
function test() {
    if ( 1 == 1 )  { var j = 12; }
    alert(j);    // displays what?
}
```

Variable Scoping

- What will the following program alert?

```
// global scope
var items = [/* some list elements */];
for (var i = 0; i < 10; i++) {
    subLoop();
}

function subLoop() {
    // scope of function subLoop
    for (i = 0; i < 10; i++) {

        alert(i);

    }
}
```