*

Lecture 25

Introduction to AJAX

and the migration toward applications

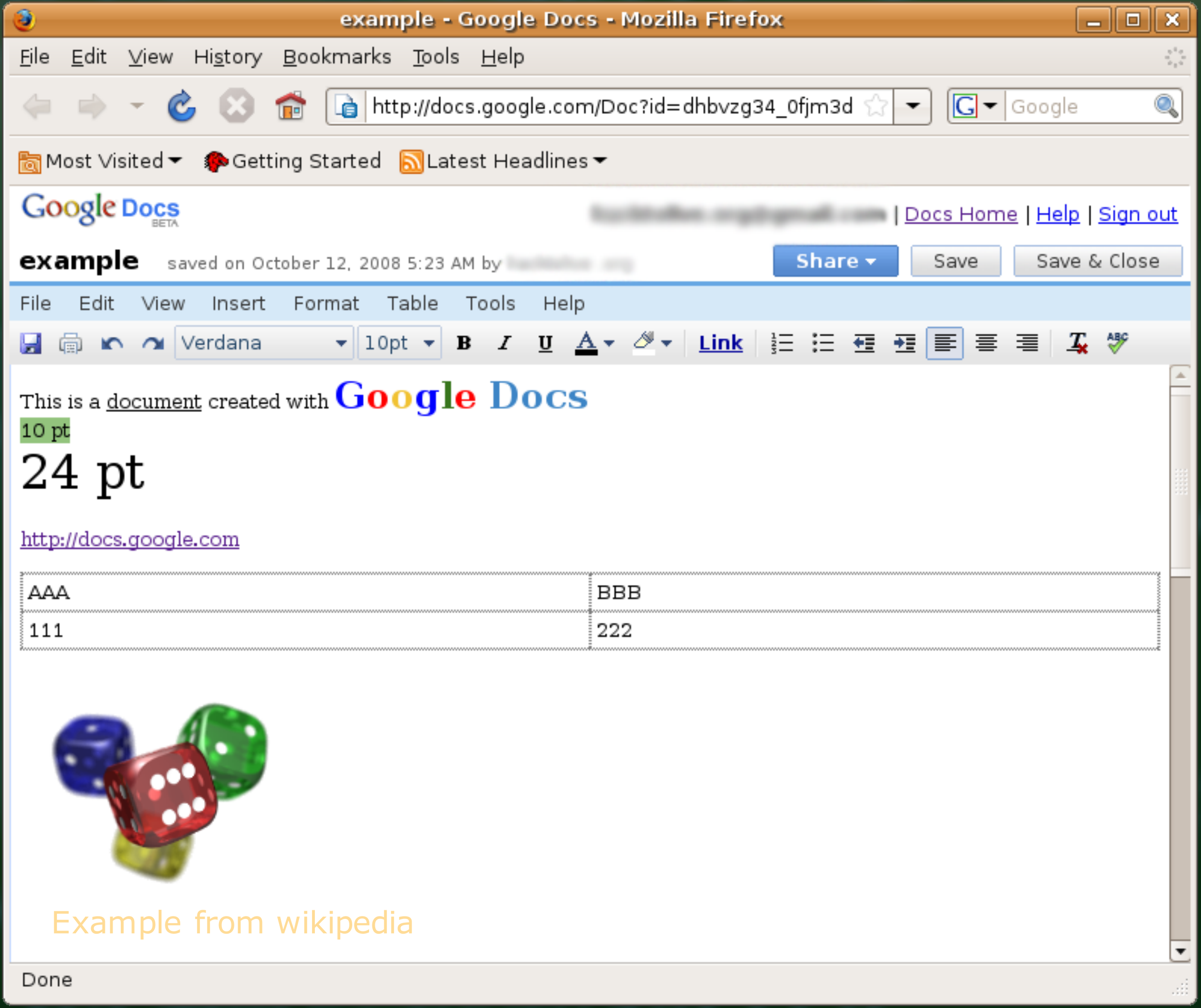
* Course logo spider web photograph from Morguefile openstock photograph by Gabor Karpati, Hungary.

AJAX – Breaking the Promise

- Before AJAX
  - A server serves a page to a client!
- After AJAX
  - Widget level event driven programming.
  - Server analogous to App backend.
  - Client analogous to interactive GUI.
- AJAX once meant …
  - Asynchronous JavaScript and XML
- The XML part has faded away.

Outward Examples

- Google Apps
- MapQuest
- Facebook
- Really, almost all modern complex sites.


example - Google Docs - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://docs.google.com/Doc?id=dhbvzg34_0fjm3d
Google
Most Visited
Getting Started
Latest Headlines
Google Docs BETA
Docs Home | Help | Sign out
Share
Save
Save & Close
File Edit View Insert Format Table Tools Help
Verdana
10pt
Link
Done

example saved on October 12, 2008 5:23 AM by

This is a document created with Google Docs

10 pt

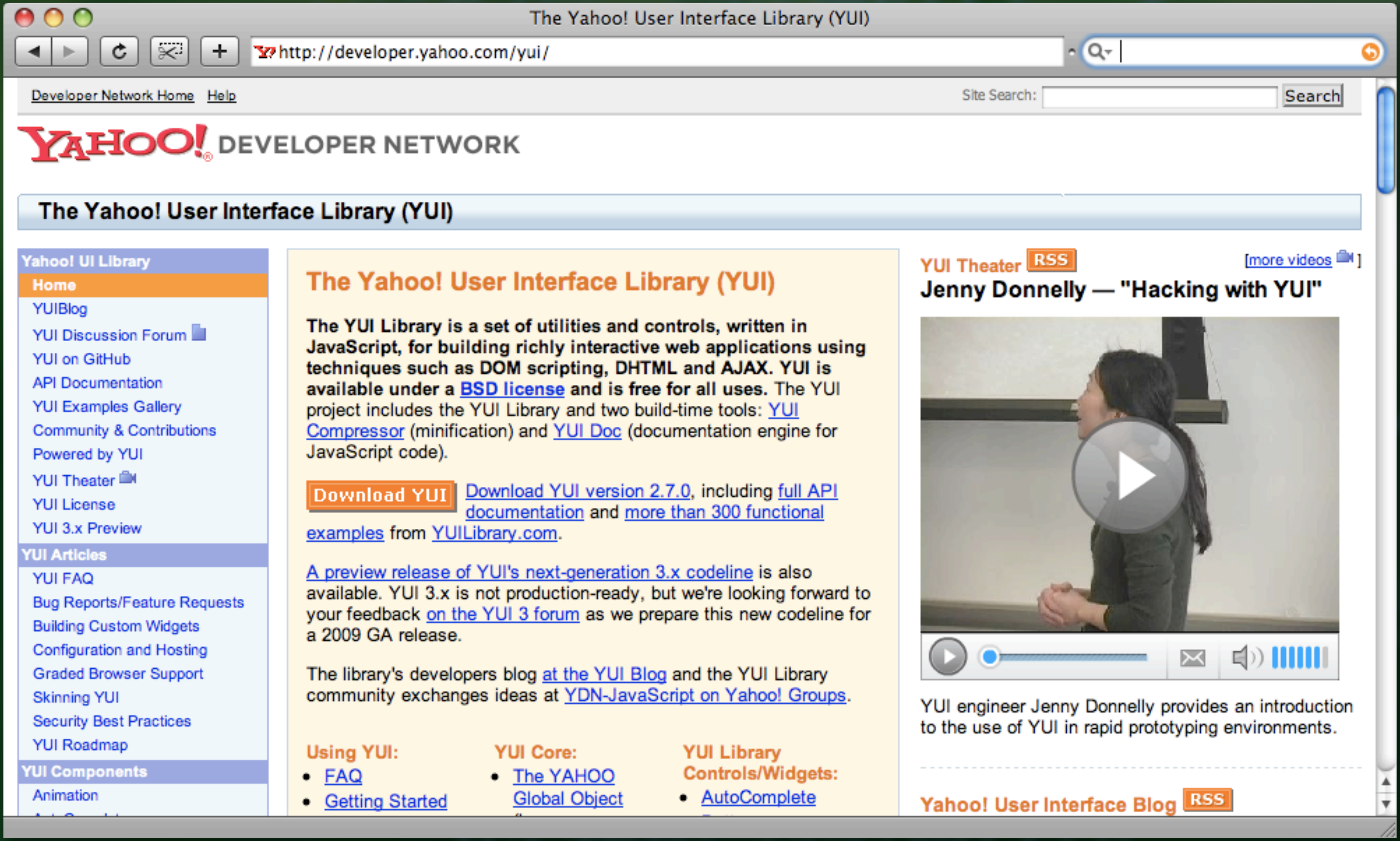
24 pt

http://docs.google.com

AAA	BBB
111	222

Example from wikipedia

Support Libraries Abound


The Yahoo! User Interface Library (YUI)
http://developer.yahoo.com/yui/
Developer Network Home
Help
Site Search:
Search
YAHOO! DEVELOPER NETWORK

The Yahoo! User Interface Library (YUI)

The Yahoo! User Interface Library (YUI)

The YUI Library is a set of utilities and controls, written in JavaScript, for building richly interactive web applications using techniques such as DOM scripting, DHTML and AJAX. YUI is available under a BSD license and is free for all uses. The YUI project includes the YUI Library and two build-time tools: YUI Compressor (minification) and YUI Doc (documentation engine for JavaScript code).

Download YUI — Download YUI version 2.7.0, including full API documentation and more than 300 functional examples from YUILibrary.com.

A preview release of YUI's next-generation 3.x codeline is also available. YUI 3.x is not production-ready, but we're looking forward to your feedback on the YUI 3 forum as we prepare this new codeline for a 2009 GA release.

The library's developers blog at the YUI Blog and the YUI Library community exchanges ideas at YDN-JavaScript on Yahoo! Groups.

Using YUI:
- FAQ
- Getting Started

YUI Core:
- The YAHOO Global Object

YUI Library Controls/Widgets:
- AutoComplete

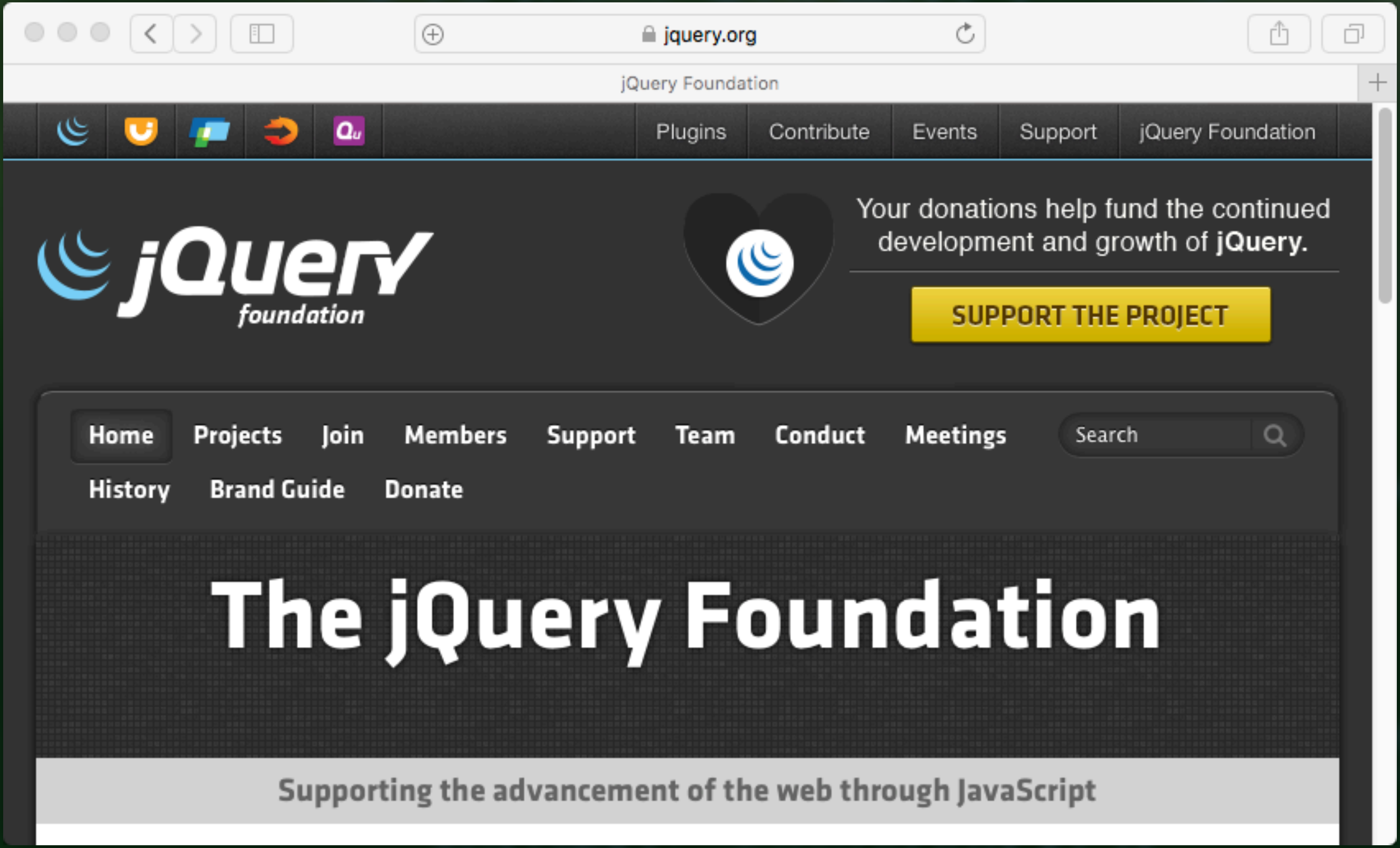
YUI Theater RSS [more videos]

Jenny Donnelly — "Hacking with YUI"

YUI engineer Jenny Donnelly provides an introduction to the use of YUI in rapid prototyping environments.

Yahoo! User Interface Blog RSS

And One is now Dominant



jquery.org

jQuery Foundation

Plugins Contribute Events Support jQuery Foundation

jQuery foundation

Your donations help fund the continued development and growth of **jQuery.**

SUPPORT THE PROJECT

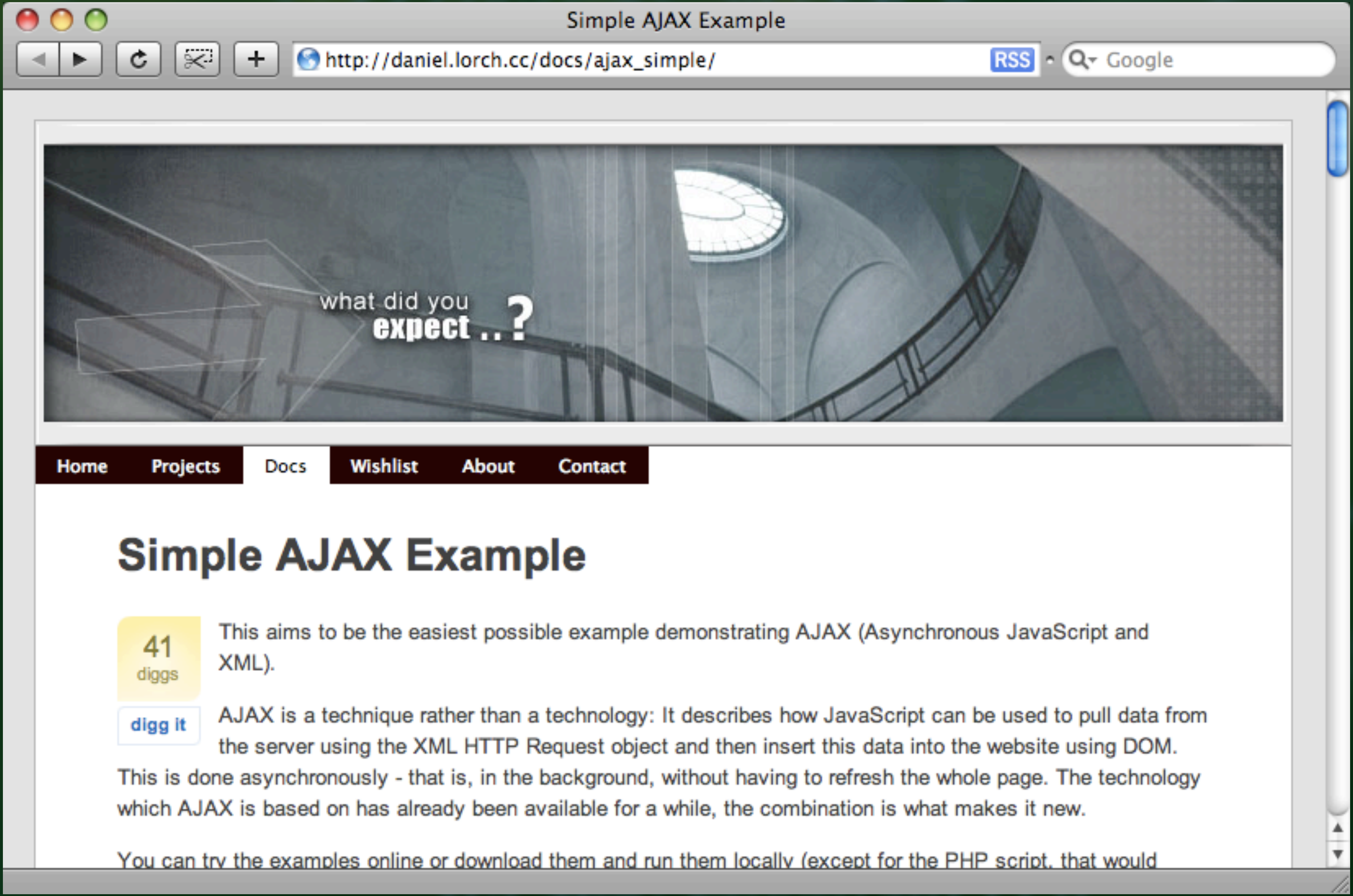
Home Projects Join Members Support Team Conduct Meetings Search

History Brand Guide Donate

The jQuery Foundation

Supporting the advancement of the web through JavaScript

Yes, we are getting closer, but first …

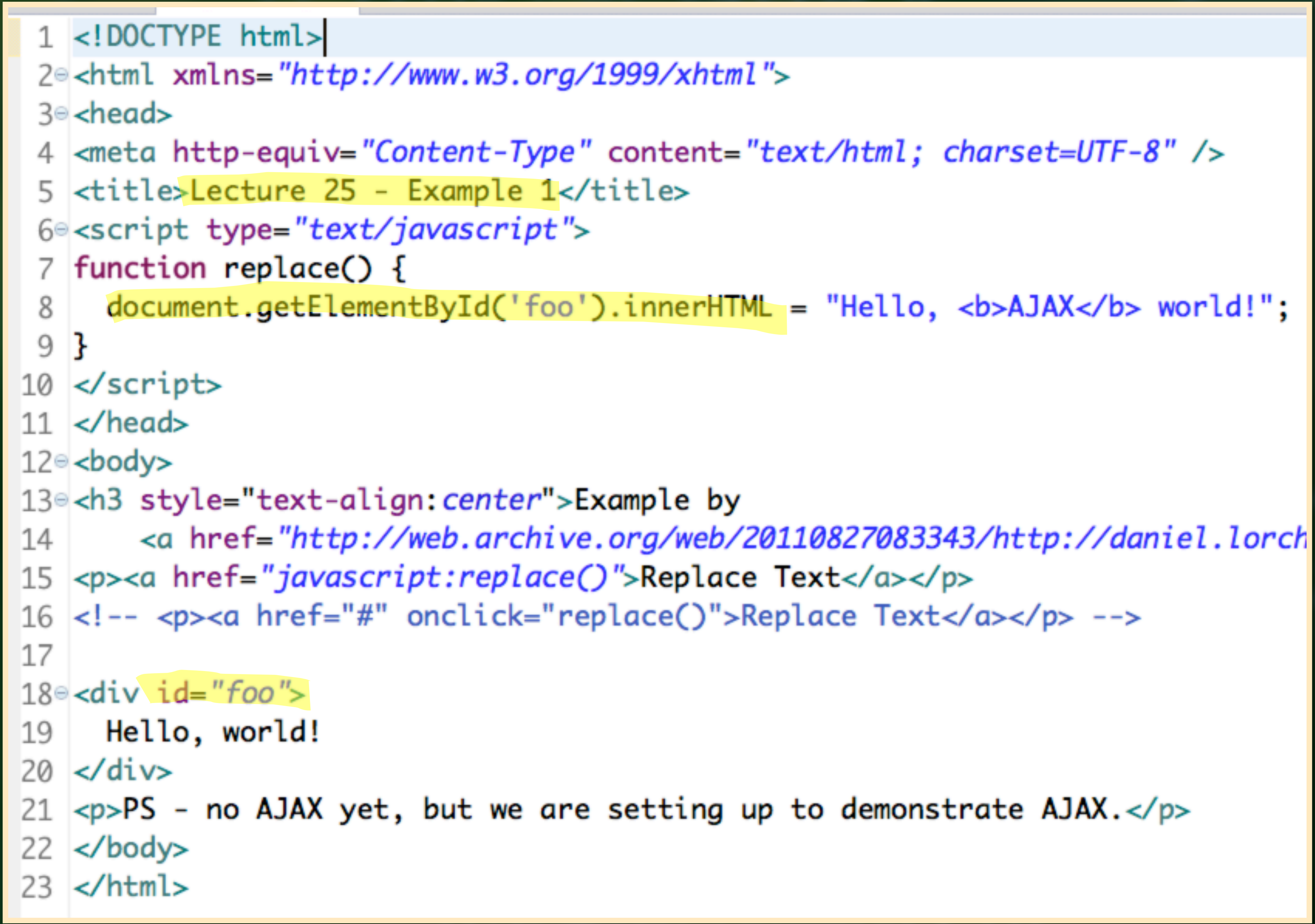


Simple AJAX Example

This aims to be the easiest possible example demonstrating AJAX (Asynchronous JavaScript and XML).

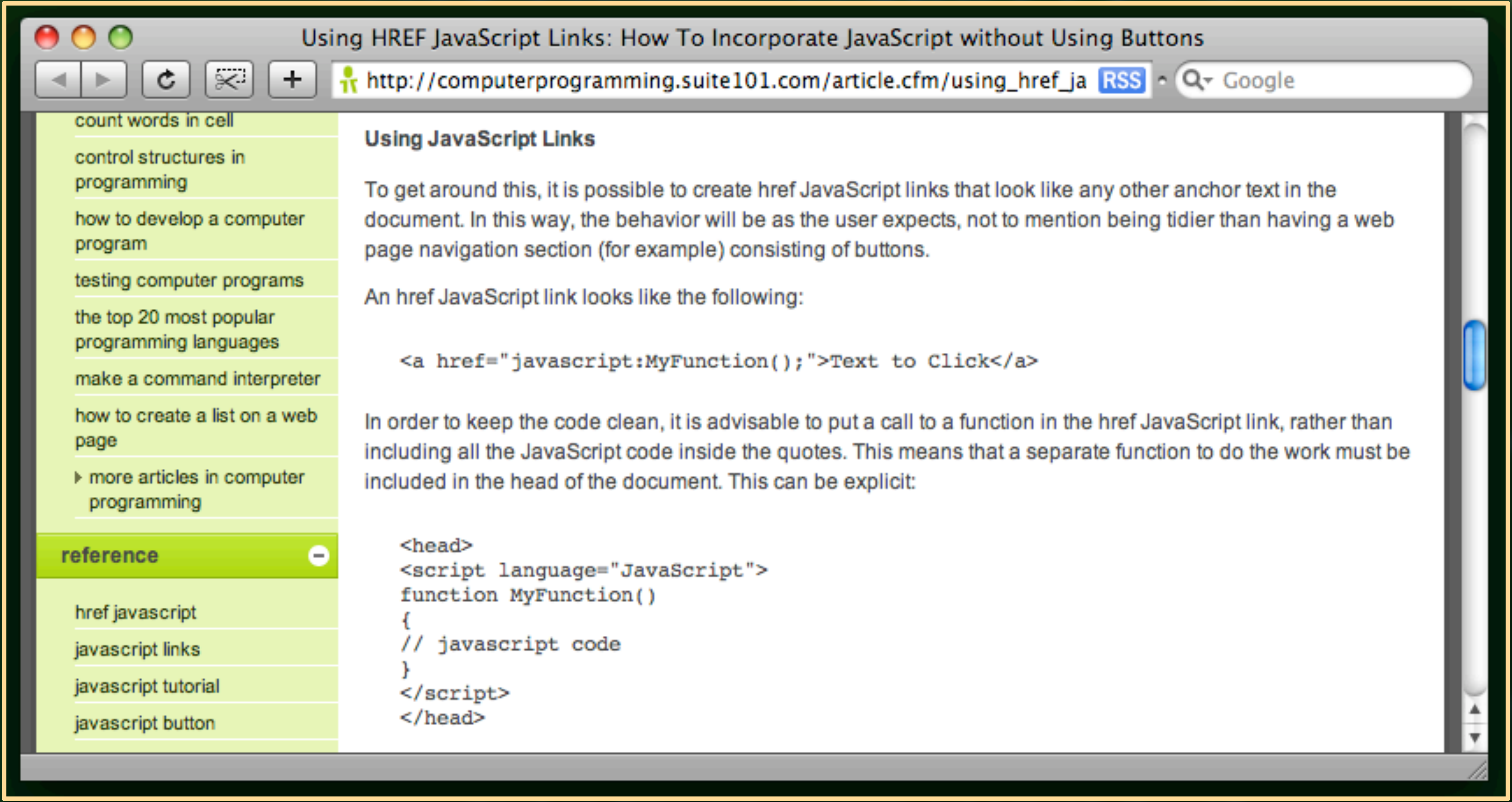
AJAX is a technique rather than a technology: It describes how JavaScript can be used to pull data from the server using the XML HTTP Request object and then insert this data into the website using DOM. This is done asynchronously - that is, in the background, without having to refresh the whole page. The technology which AJAX is based on has already been available for a while, the combination is what makes it new.

You can try the examples online or download them and run them locally (except for the PHP script, that would



```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture 25 - Example 1</title>
<script type="text/javascript">
function replace() {
  document.getElementById('foo').innerHTML = "Hello, <b>AJAX</b> world!";
}
</script>
</head>
<body>
<h3 style="text-align:center">Example by
    <a href="http://web.archive.org/web/20110827083343/http://daniel.lorch
<p><a href="javascript:replace()">Replace Text</a></p>
<!-- <p><a href="#" onclick="replace()">Replace Text</a></p> -->

<div id="foo">
  Hello, world!
</div>
<p>PS - no AJAX yet, but we are setting up to demonstrate AJAX.</p>
</body>
</html>
```

JavaScript as a link


Using HREF JavaScript Links: How To Incorporate JavaScript without Using Buttons
http://computerprogramming.suite101.com/article.cfm/using_href_ja
RSS
Google
count words in cell
control structures in programming
how to develop a computer program
testing computer programs
the top 20 most popular programming languages
make a command interpreter
how to create a list on a web page
more articles in computer programming
reference
href javascript
javascript links
javascript tutorial
javascript button

Using JavaScript Links

To get around this, it is possible to create href JavaScript links that look like any other anchor text in the document. In this way, the behavior will be as the user expects, not to mention being tidier than having a web page navigation section (for example) consisting of buttons.

An href JavaScript link looks like the following:

```
<a href="javascript:MyFunction();">Text to Click</a>
```

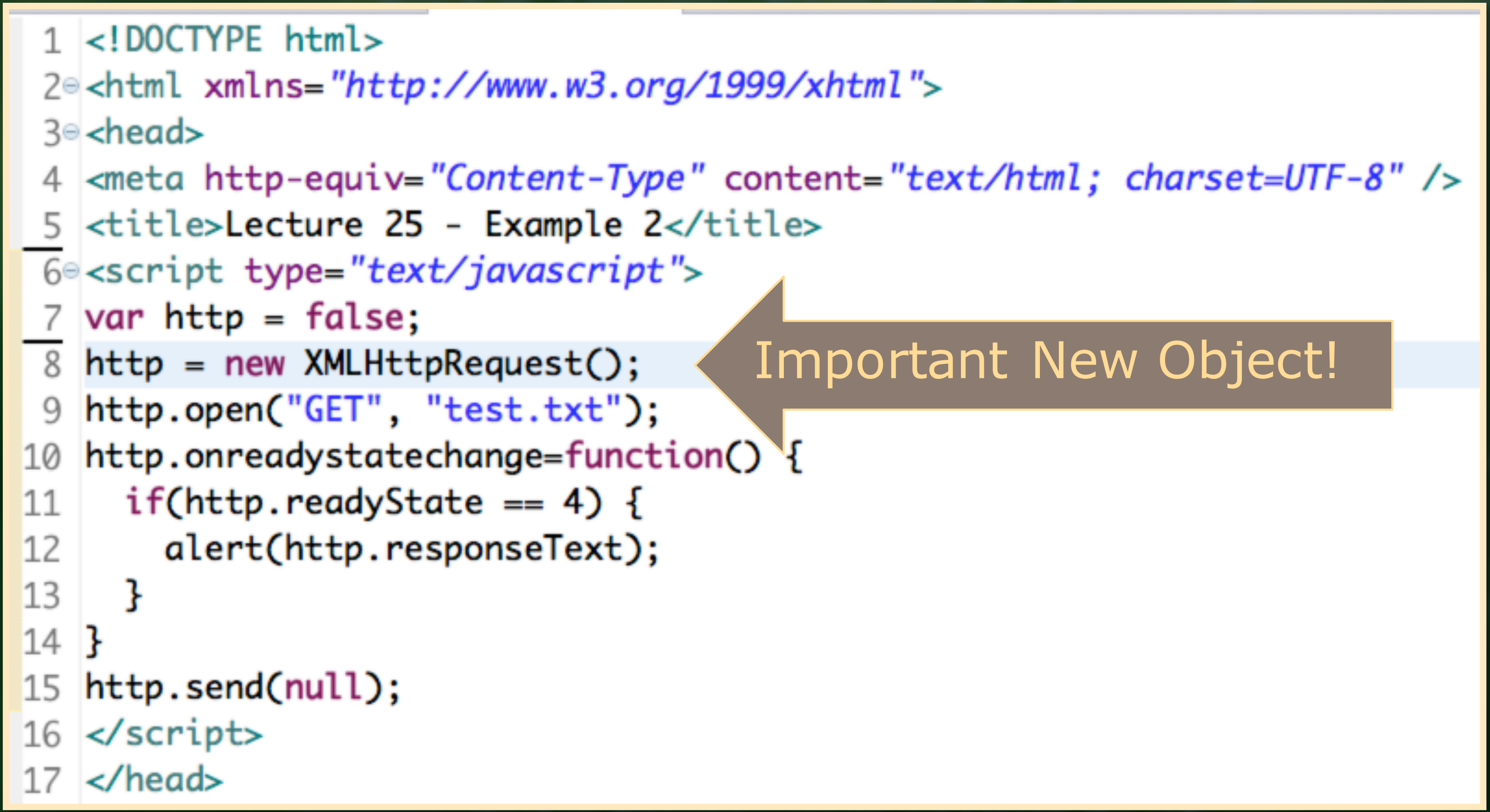
In order to keep the code clean, it is advisable to put a call to a function in the href JavaScript link, rather than including all the JavaScript code inside the quotes. This means that a separate function to do the work must be included in the head of the document. This can be explicit:

```
<head>
<script language="JavaScript">
function MyFunction()
{
// javascript code
}
</script>
</head>
```

Other Things to Note

- The div has a name (id actually)
- The document is an object/element
- Document contains other elements
- Elements have ‘innerHTML’
    - Not ‘inner peace’
- When/where does the html get set?

Example 2 -XMLHttpRequest

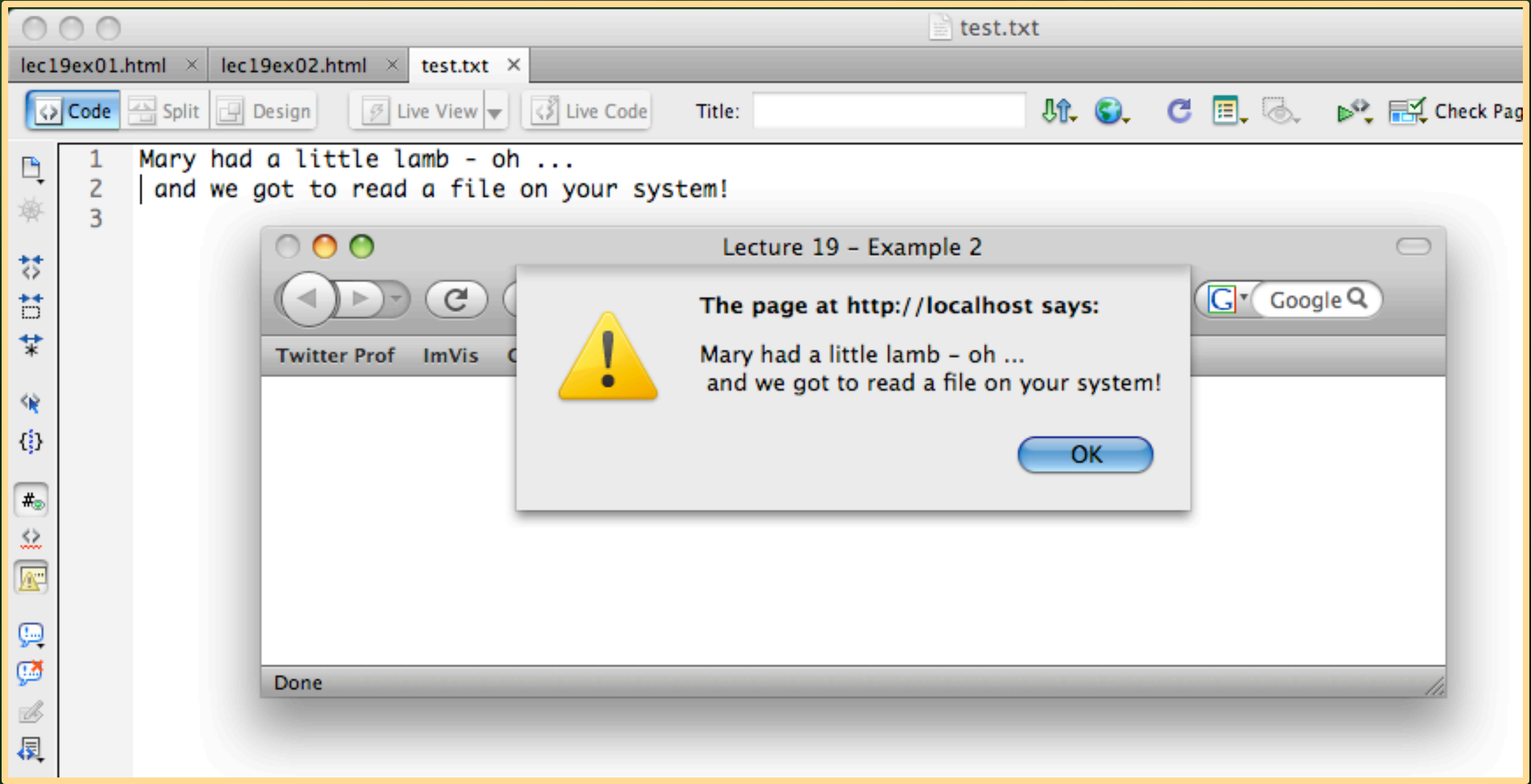


```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture 25 - Example 2</title>
<script type="text/javascript">
var http = false;
http = new XMLHttpRequest();
http.open("GET", "test.txt");
http.onreadystatechange=function() {
  if(http.readyState == 4) {
    alert(http.responseText);
  }
}
http.send(null);
</script>
</head>
```

Important New Object!

Note example does not support antiquated IE6 ActiveX alternative.

Example 2 – in action



test.txt

lec19ex01.html lec19ex02.html test.txt

Code Split Design Live View Live Code Title: Check Pag

```
Mary had a little lamb - oh ...
 and we got to read a file on your system!

```

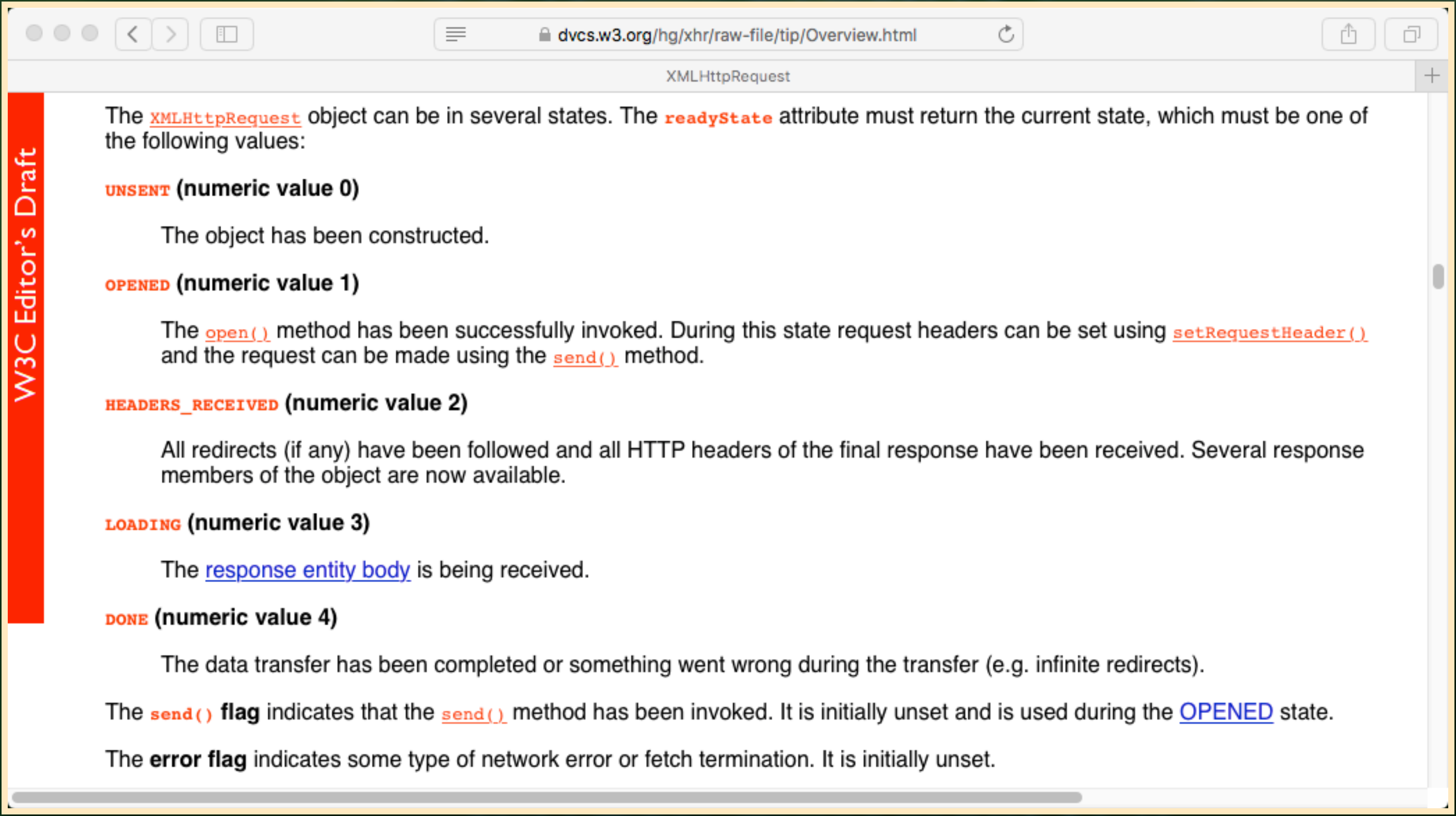
The page at http://localhost says:

Mary had a little lamb – oh ...
 and we got to read a file on your system!

Example 2 - Comments

- There is a var/object ‘http’
- Establishes a connection to a server.
- Notice the use of
  - Open
  - ‘Get’
  - Events
  - Send null - close

readyState Change



The XMLHttpRequest object can be in several states. The readyState attribute must return the current state, which must be one of the following values:

UNSENT (numeric value 0)

The object has been constructed.

OPENED (numeric value 1)

The open() method has been successfully invoked. During this state request headers can be set using setRequestHeader() and the request can be made using the send() method.

HEADERS_RECEIVED (numeric value 2)

All redirects (if any) have been followed and all HTTP headers of the final response have been received. Several response members of the object are now available.

LOADING (numeric value 3)

The response entity body is being received.

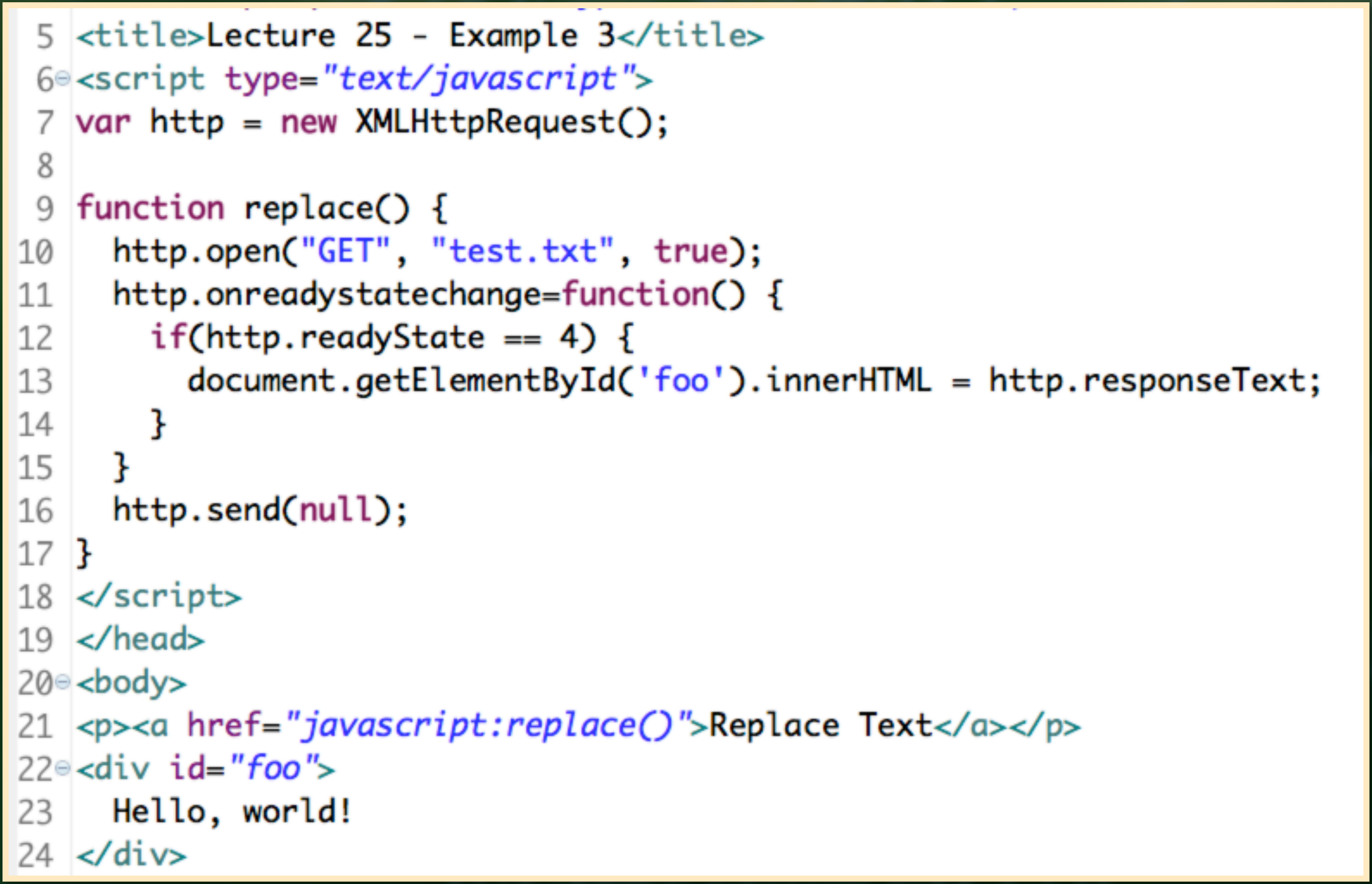
DONE (numeric value 4)

The data transfer has been completed or something went wrong during the transfer (e.g. infinite redirects).

The **send() flag** indicates that the send() method has been invoked. It is initially unset and is used during the OPENED state.

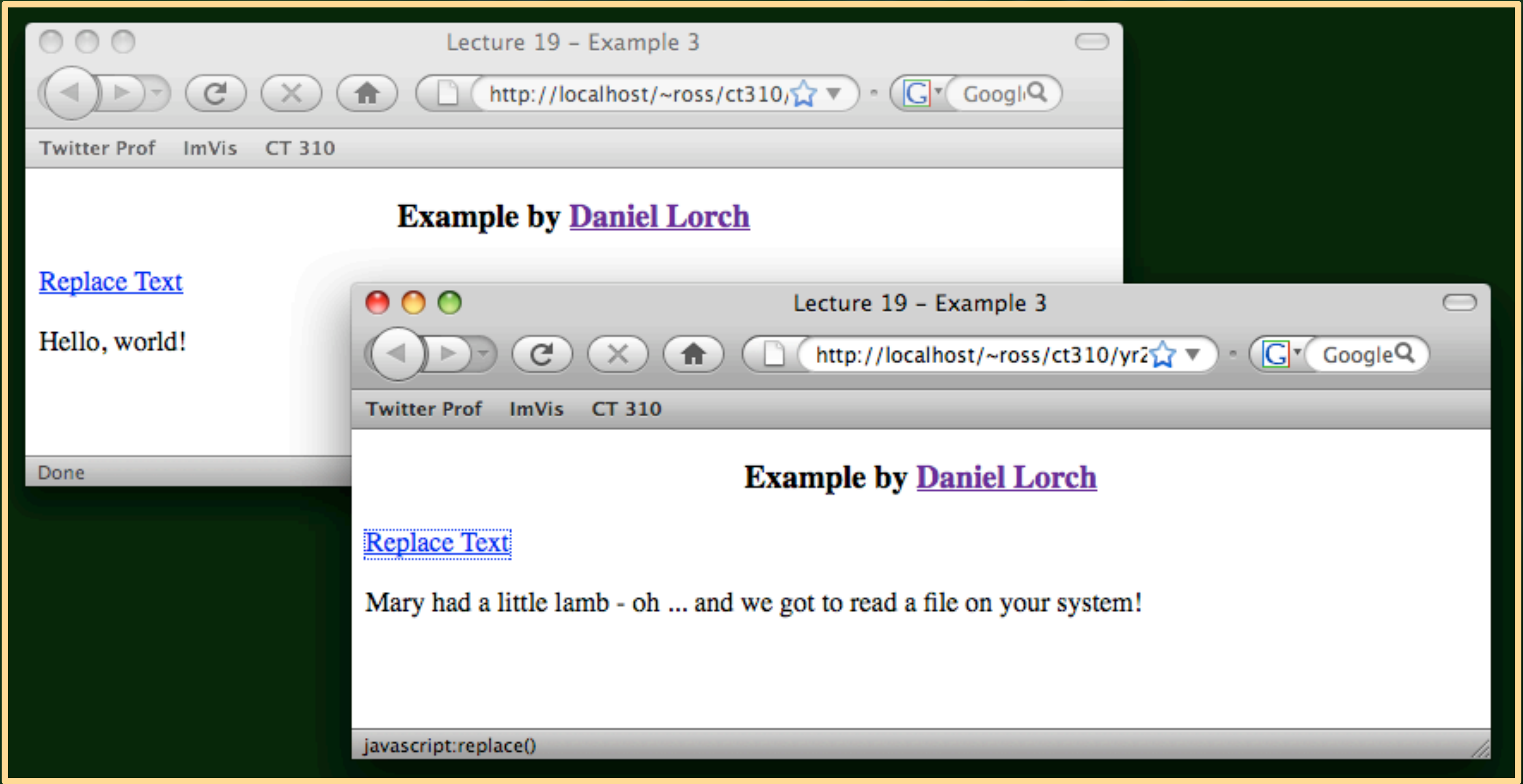
The **error flag** indicates some type of network error or fetch termination. It is initially unset.

Example 3 code



```
<title>Lecture 25 - Example 3</title>
<script type="text/javascript">
var http = new XMLHttpRequest();

function replace() {
  http.open("GET", "test.txt", true);
  http.onreadystatechange=function() {
    if(http.readyState == 4) {
      document.getElementById('foo').innerHTML = http.responseText;
    }
  }
  http.send(null);
}
</script>
</head>
<body>
<p><a href="javascript:replace()">Replace Text</a></p>
<div id="foo">
  Hello, world!
</div>
```

Example 3 in action



Example by Daniel Lorch

Replace Text

Hello, world!

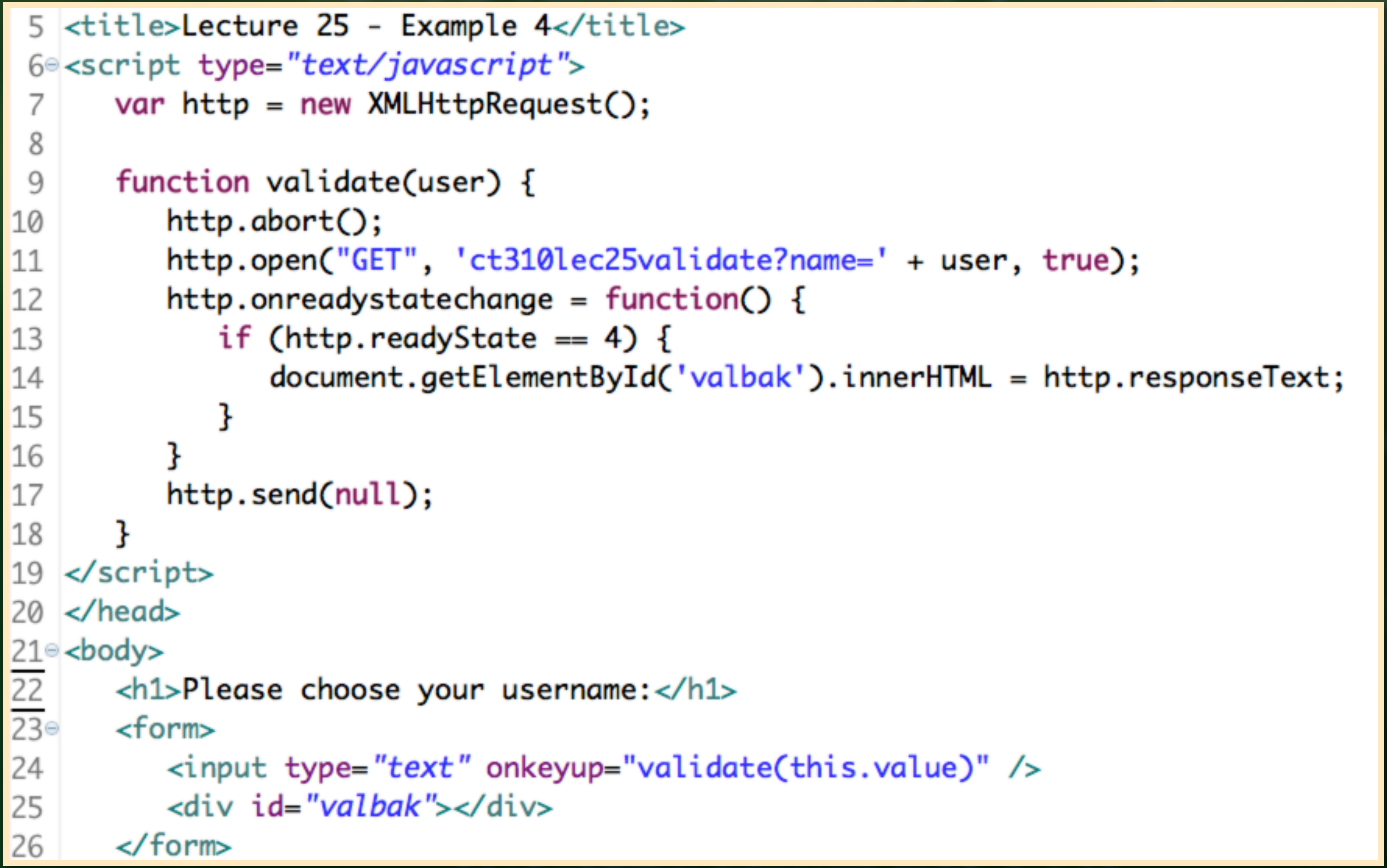
Example by Daniel Lorch

Replace Text

Mary had a little lamb - oh ... and we got to read a file on your system!

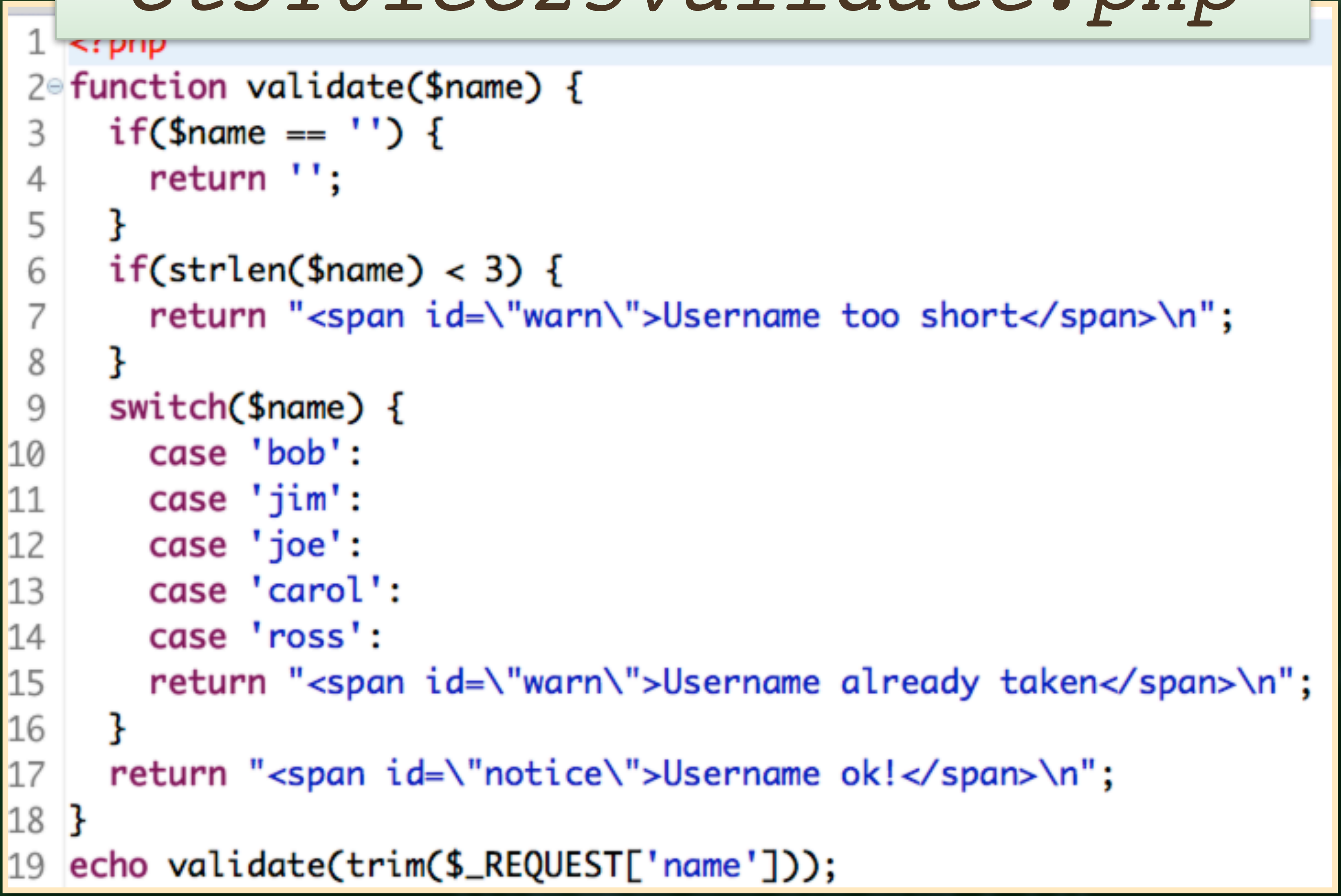
javascript:replace()

Example 4 – Code 1



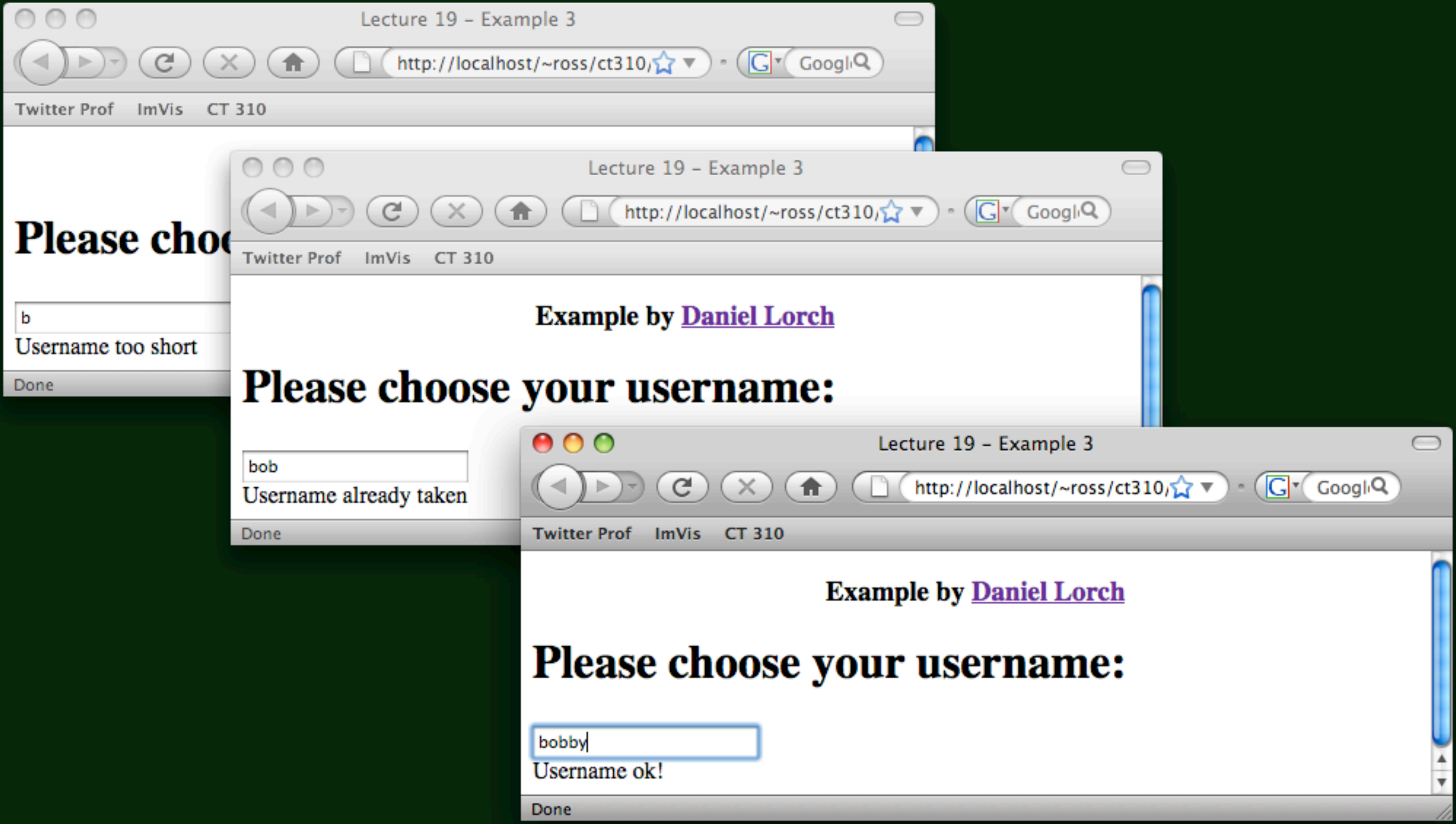
```
<title>Lecture 25 - Example 4</title>
<script type="text/javascript">
   var http = new XMLHttpRequest();

   function validate(user) {
      http.abort();
      http.open("GET", 'ct310lec25validate?name=' + user, true);
      http.onreadystatechange = function() {
         if (http.readyState == 4) {
            document.getElementById('valbak').innerHTML = http.responseText;
         }
      }
      http.send(null);
   }
</script>
</head>
<body>
   <h1>Please choose your username:</h1>
   <form>
      <input type="text" onkeyup="validate(this.value)" />
      <div id="valbak"></div>
   </form>
```

ct310lec25validate.php



```php
<?php
function validate($name) {
  if($name == '') {
    return '';
  }
  if(strlen($name) < 3) {
    return "<span id=\"warn\">Username too short</span>\n";
  }
  switch($name) {
    case 'bob':
    case 'jim':
    case 'joe':
    case 'carol':
    case 'ross':
    return "<span id=\"warn\">Username already taken</span>\n";
  }
  return "<span id=\"notice\">Username ok!</span>\n";
}
echo validate(trim($_REQUEST['name']));
```

Example 4 in action



Please cho

b

Username too short

Example by Daniel Lorch

Please choose your username:

bob

Username already taken

Example by Daniel Lorch

Please choose your username:

bobby

Username ok!

Cross-Site Scripting!

- In class (2012) I walked head long into an illegal use of cross-site scripting.
- On my laptop, i.e. **localhost**, I tried:

```
http.open("GET",
  'http:///www.cs.colostate.edu/' +
  '~ct310/yr2013sp/aplay/lec18/' +
  'validate.php?name=' +
   user, true);
```

Violates the "Same Origin Policy" !

Background & Summary

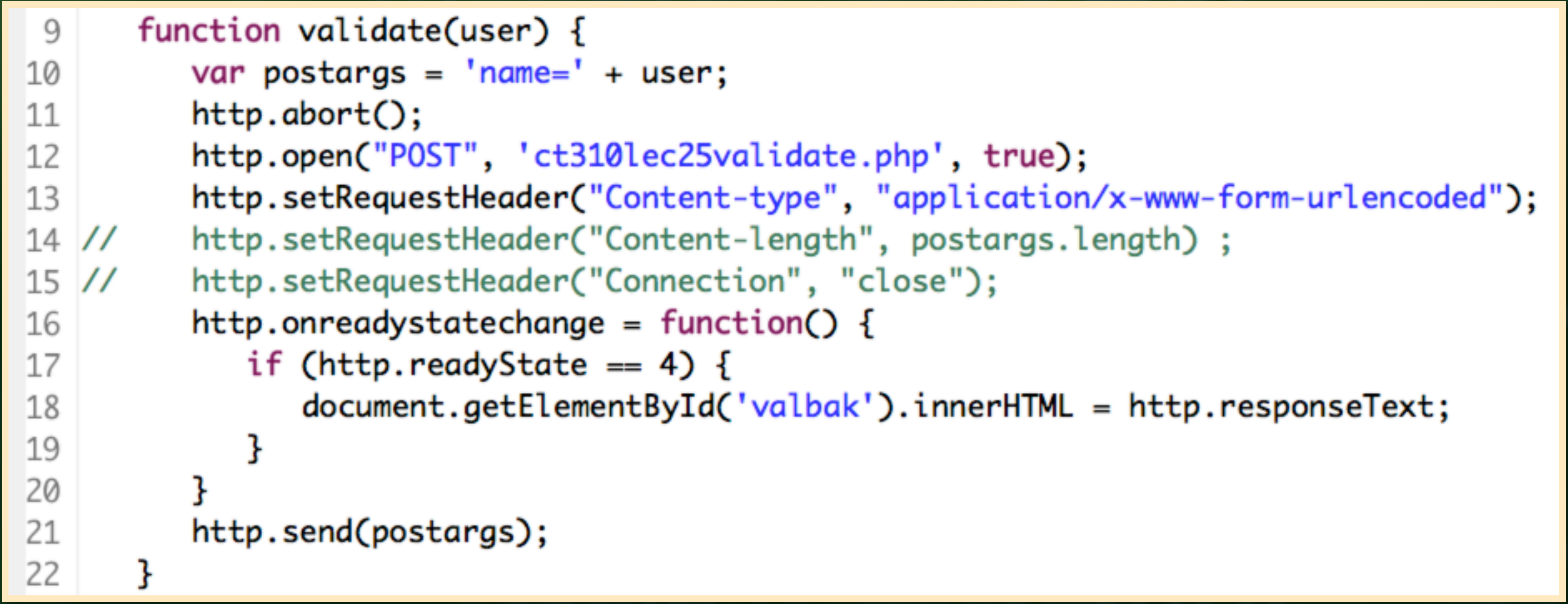
- Cross-site scripting attacks represent a large fraction of malicious behavior – stolen data/accounts/etc.
- In a nutshell, site A gets hacked, hook inserted to load JavaScript from site B, code from B then gains access to what A knows/does etc.

Same Origin Policy

- Modern browsers impose strong constraints on AJAX behavior.
- Domain serving document must be the same domain used through XMLHttpRequest().
- Workarounds include JSONP,
  - Cross-origin resource sharing.

Post Not Get

Same as Example 4 but using POST.

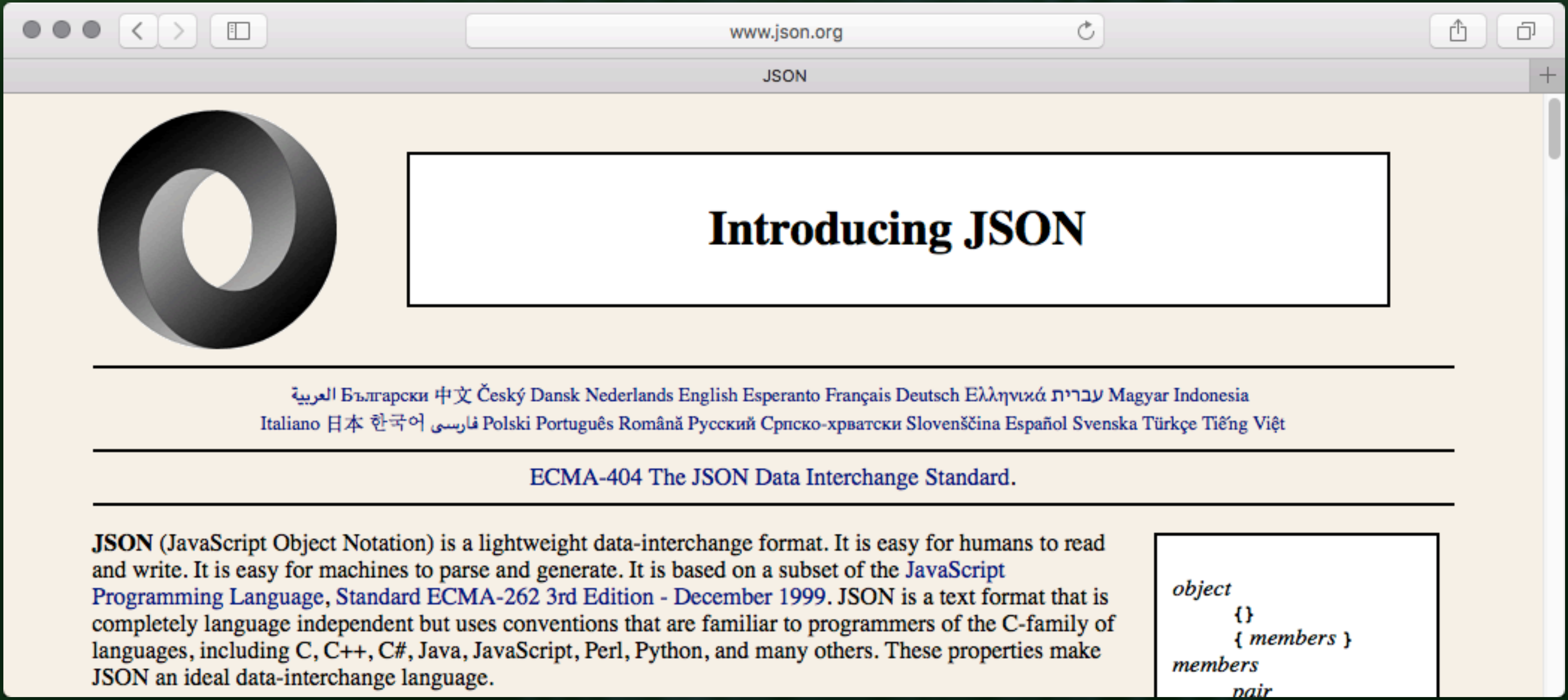


```
   function validate(user) {
      var postargs = 'name=' + user;
      http.abort();
      http.open("POST", 'ct310lec25validate.php', true);
      http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
//    http.setRequestHeader("Content-length", postargs.length) ;
//    http.setRequestHeader("Connection", "close");
      http.onreadystatechange = function() {
         if (http.readyState == 4) {
            document.getElementById('valbak').innerHTML = http.responseText;
         }
      }
      http.send(postargs);
   }
```

Bit more code; bit more secure.

Moving Data - JSON

- Moving data from server to client?
  - We've just seen plain text
  - JSON is common for structured data



Introducing JSON

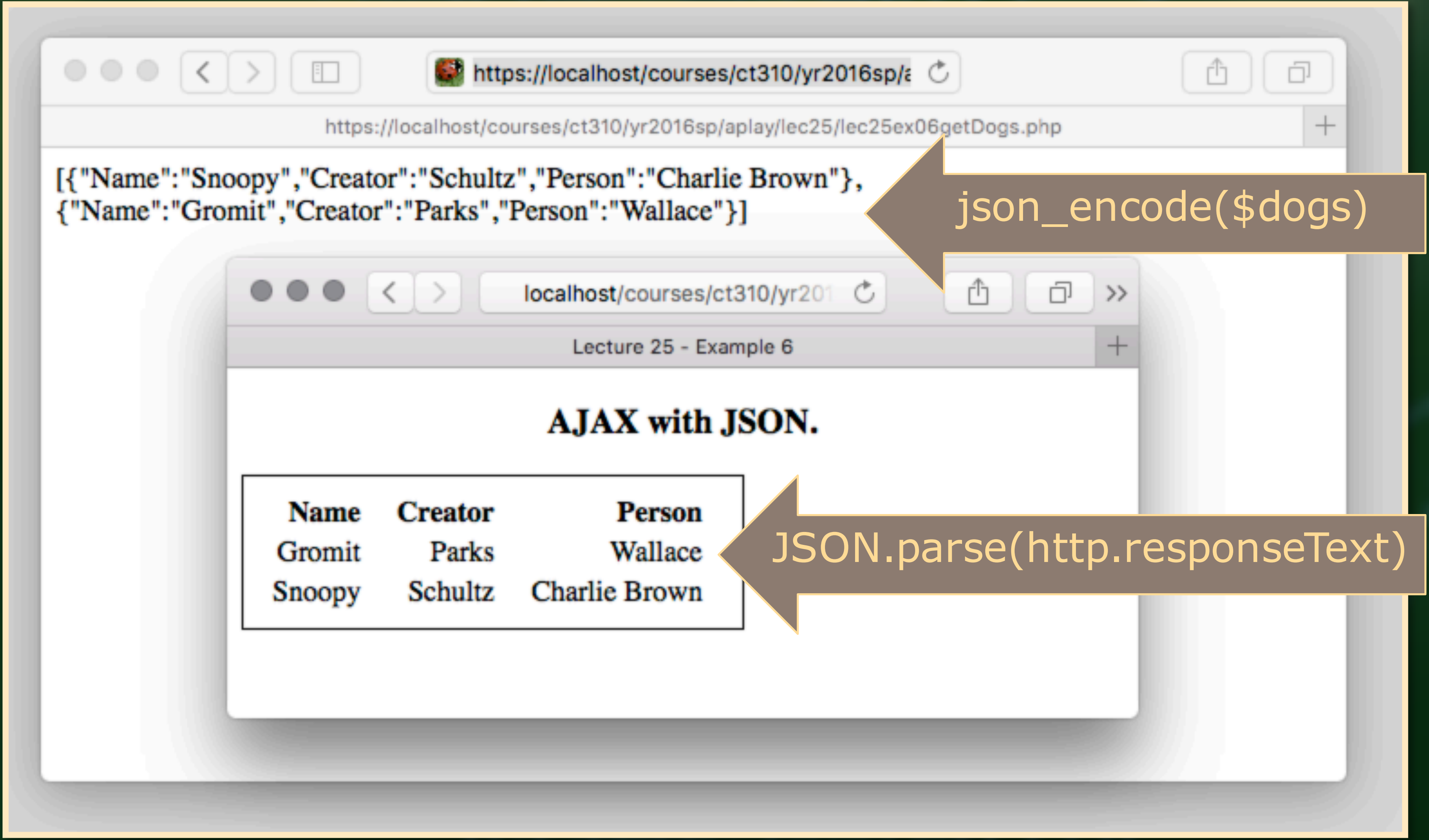
العربية Български 中文 Český Dansk Nederlands English Esperanto Français Deutsch Ελληνικά עברית Magyar Indonesia Italiano 日本 한국어 فارسی Polski Português Română Русский Српско-хрватски Slovenščina Español Svenska Türkçe Tiếng Việt

ECMA-404 The JSON Data Interchange Standard.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

```
object
    {}
    { members }
members
    pair
```

Example 6 – JSON Dogs


https://localhost/courses/ct310/yr2016sp/a
https://localhost/courses/ct310/yr2016sp/aplay/lec25/lec25ex06getDogs.php

```
[{"Name":"Snoopy","Creator":"Schultz","Person":"Charlie Brown"},
{"Name":"Gromit","Creator":"Parks","Person":"Wallace"}]
```

json_encode($dogs)

localhost/courses/ct310/yr201
Lecture 25 - Example 6

AJAX with JSON.

Name	Creator	Person
Gromit	Parks	Wallace
Snoopy	Schultz	Charlie Brown

JSON.parse(http.responseText)

Closing thoughts on AJAX

The Path Back to Computer Science

- In the beginning
  - HTML is simple and elegant …
  - Easy to learn and use
  - Far removed from CS complexities
- With AJAX, the circle closes
  - What do you need to understand?
  - Just about everything taught in CS