MALMÖ UNIVERSITY

INLEDANDE WEBBPROGRAMMERING MED JAVASCRIPT

INTRODUCTION TO WEB PROGRAMING USING JAVASCRIPT

ME152A

L5: 1. HTTP - XMLHTTPREQUEST

2. FORMS AND FORM FIELDS

OUTLINE

- What did we learn so far?
- HTTP
- Forms and Form Fields

WHAT DID WE LEARN SO FAR?

- Higher-order functions
- Regular expressions
- JavaScript and HTML
- DOM
- Events

HTTP

- The Hypertext Transfer Protocol (HTTP), already mentioned in in previous lecture, is the mechanism through which data is requested and provided on the World Wide Web.

- The default port for HTTP traffic is 80
  - When you type http://mah.se/Forskning/ the browser first looks up the address of the server associated with http://mah.se and open a TCP connection on port 80

HTTP: REQUEST / RESPOND

Request:

```
GET /17_http.html HTTP/1.1
Host: mah.se
User-Agent: Your browser's name
```

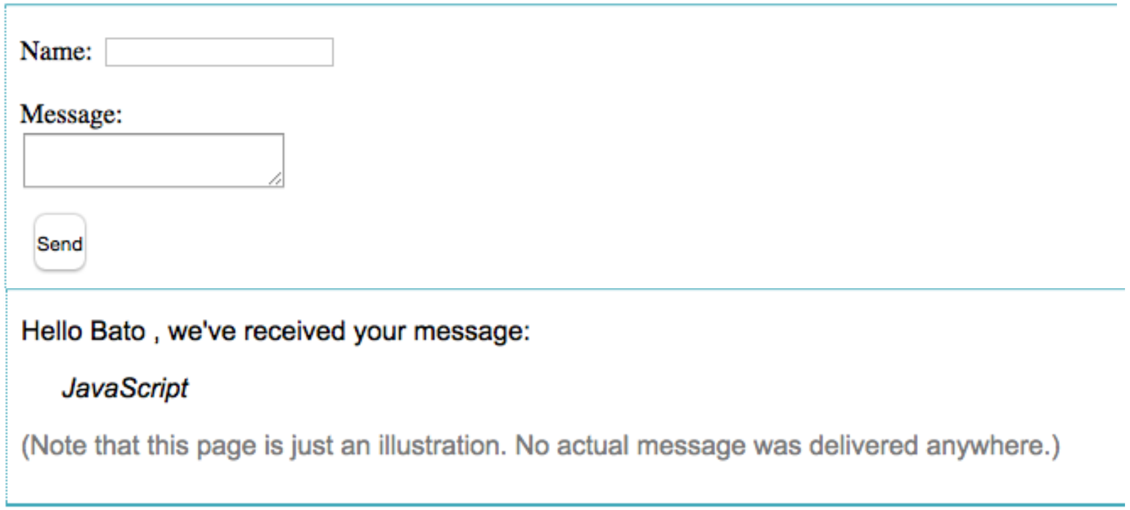
Respond:

```
 HTTP/1.1 200 OK
 Content-Length: 65585
 Content-Type: text/html
 Last-Modified: Wed, 09 Apr 2014 10:48:09 GMT

<!doctype html>
... the rest of the document
```

BROWSERS AND HTTP

- HTML pages may include *forms*, which allow the user to fill out information and send it to the server. This is an example of a form:

```
<form method="GET" action="example/message.html">
  <p>Name: <input type="text" name="name"></p>
  <p>Message:<br><textarea name="message"></textarea></p>
  <p><button type="submit">Send</button></p>
</form>
```



Hello Bato , we've received your message:

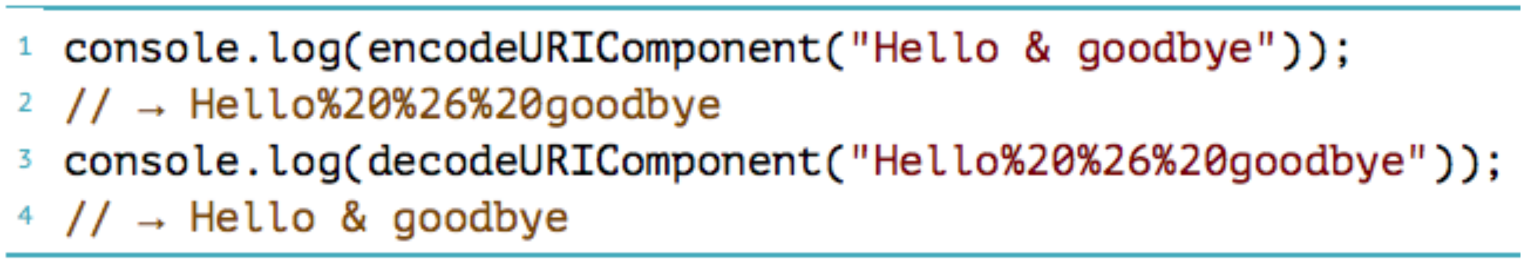
JavaScript

(Note that this page is just an illustration. No actual message was delivered anywhere.)

```
GET /example/message.html?name=Bato&message=JavaScript%3F HTTP/1.1
```

BROWSERS AND HTTP (CONT’D)

- JavaScript provides the `encodeURIComponent` and `decodeURIComponent` functions to encode and decode this format.



```
console.log(encodeURIComponent("Hello & goodbye"));
// → Hello%20%26%20goodbye
console.log(decodeURIComponent("Hello%20%26%20goodbye"));
// → Hello & goodbye
```

BROWSERS AND HTTP (CONT'D)

- If we change the method attribute of the HTML form in the example to POST, the HTTP request made to submit the form will use the POST method and put the query string in body of the request, rather than adding it to the URL.

```
POST /example/message.html HTTP/1.1
Content-length: 24
Content-type: application/x-www-form-urlencoded

name=Bato&message=JavaScript%3F
```

GET OR POST?

- GET is simpler and faster than POST, and can be used in most cases.
- However, always use POST requests when:
  - A cached file is not an option (update a file or database on the server).
  - Sending a large amount of data to the server (POST has no size limitations).
  - Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

http://www.w3schools.com/ajax/ajax_xmlhttprequest_send.asp

AJAX

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- You can:
  - Update a web page without reloading the page
  - Request data from a server - after the page has loaded
  - Receive data from a server - after the page has loaded
  - Send data to a server - in the background

- **NOTE:** AJAX is a misleading name. You don't have to understand XML to use AJAX.

http://www.tutorialspoint.com/ajax/what_is_ajax.htm

http://www.w3schools.com/ajax/default.asp

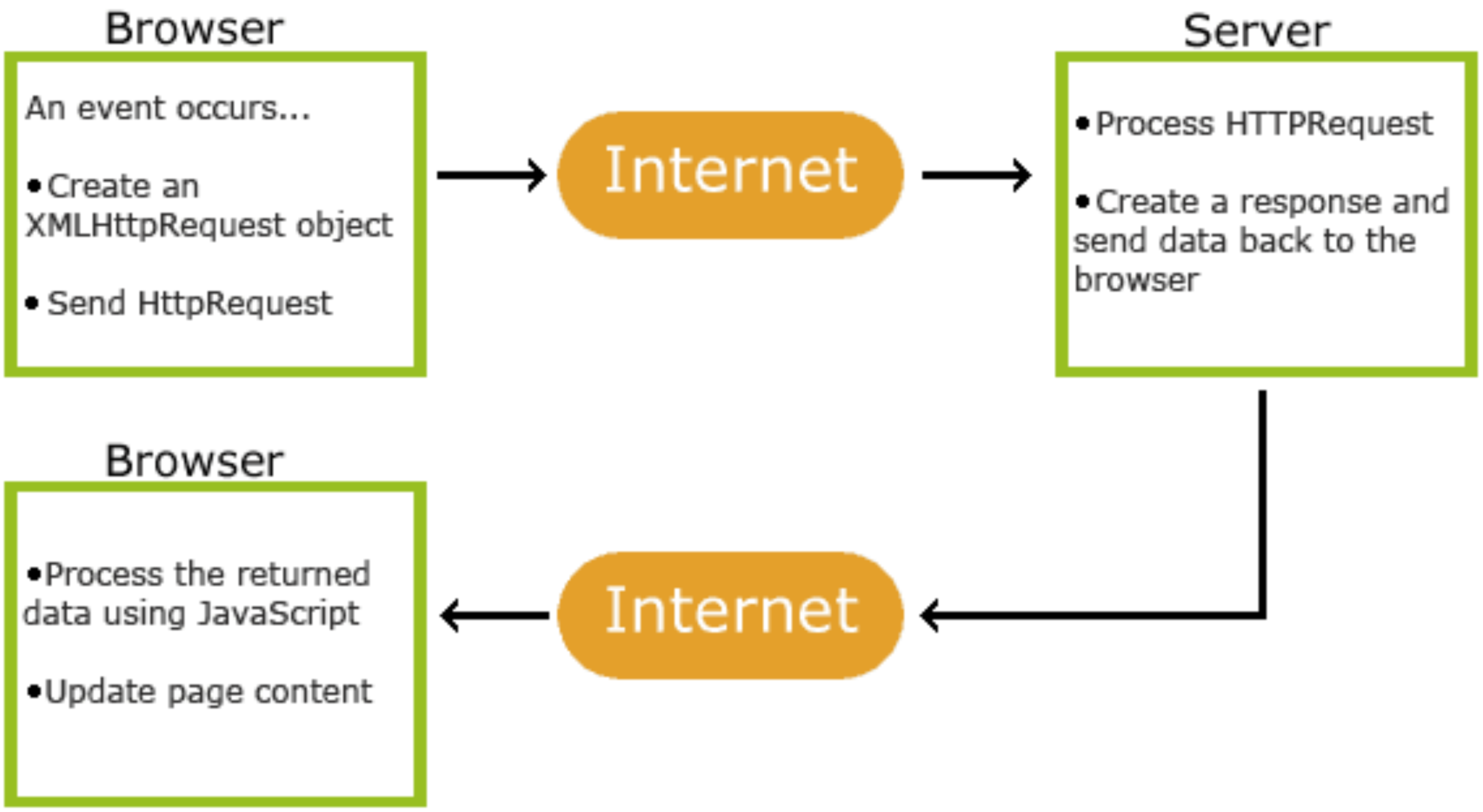
AJAX (CONT'D)

- AJAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
- Examples of applications using AJAX: Google Maps, Gmail, YouTube, and Facebook.

http://www.tutorialspoint.com/ajax/what_is_ajax.htm

http://www.w3schools.com/ajax/default.asp

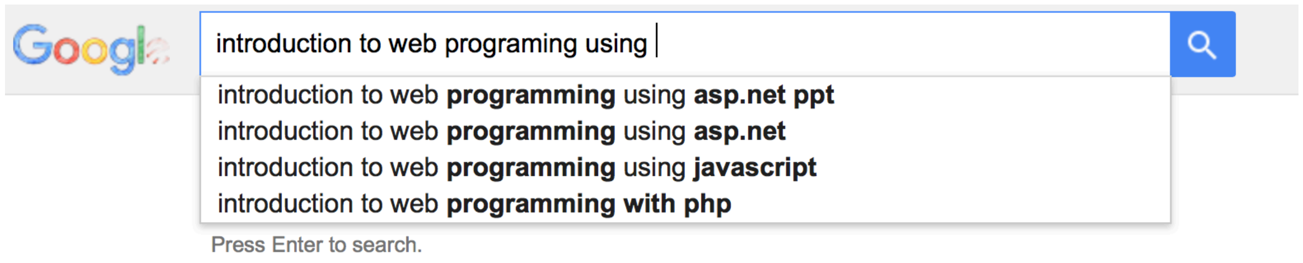
AJAX (CONT'D)


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

http://www.w3schools.com/ajax/ajax_intro.asp

AJAX IS BASED ON INTERNET STANDARDS

- AJAX is based on internet standards, and uses a combination of:
  - XMLHttpRequest object (to retrieve data from a web server)
  - JavaScript/DOM (to display/use the data)
- AJAX was made popular in 2005 by Google, with Google Suggest.
- Suggest is using AJAX to create a very dynamic web interface: When you start typing in Google's search box, a JavaScript sends the letters off to a server and the server returns a list of suggestions.


introduction to web programing using

introduction to web **programming** using **asp.net ppt**
introduction to web **programming** using **asp.net**
introduction to web **programming** using **javascript**
introduction to web **programming with php**

Press Enter to search.

The keystone of AJAX is the XMLHttpRequest object.

http://www.w3schools.com/ajax/ajax_intro.asp

XMLHTTPREQUEST

- The interface through which browser JavaScript can make HTTP requests is called XMLHttpRequest
- XMLHttpRequest is an API that provides client functionality for transferring data between a client and a server. It provides an easy way to retrieve data from a URL without having to do a full page refresh.
- XMLHttpRequest was originally designed by Microsoft and adopted by Mozilla, Apple, and Google. It's now being standardized at the WHATWG.
- XMLHttpRequest can be used to retrieve any type of data, not just XML, and it supports protocols other than HTTP (including file and ftp).

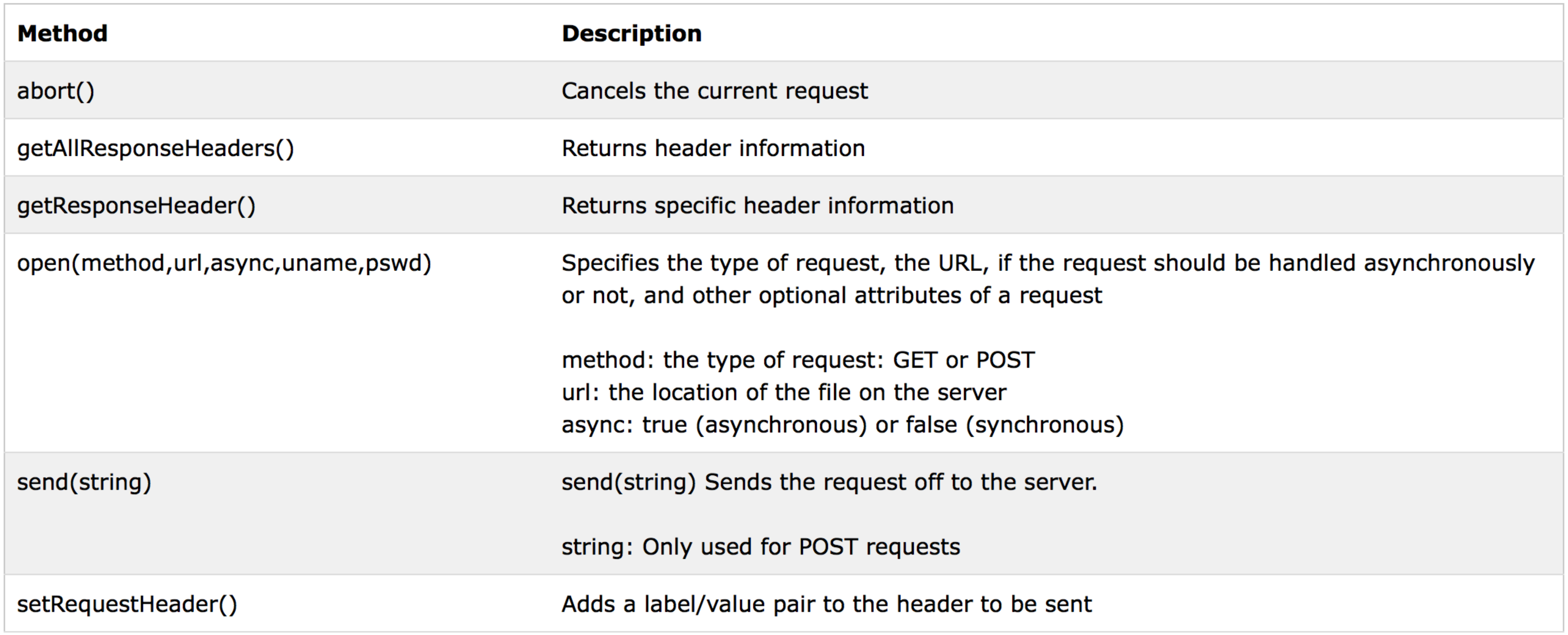
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpReque

THE XMLHTTPREQUEST OBJECT

- The XMLHttpRequest object is used to exchange data with a server behind the scenes.
- The XMLHttpRequest object is **the developers dream**, because you can:
  - Update a web page without reloading the page
  - Request data from a server after the page has loaded
  - Receive data from a server after the page has loaded
  - Send data to a server in the background

http://www.w3schools.com/xml/dom_http.asp

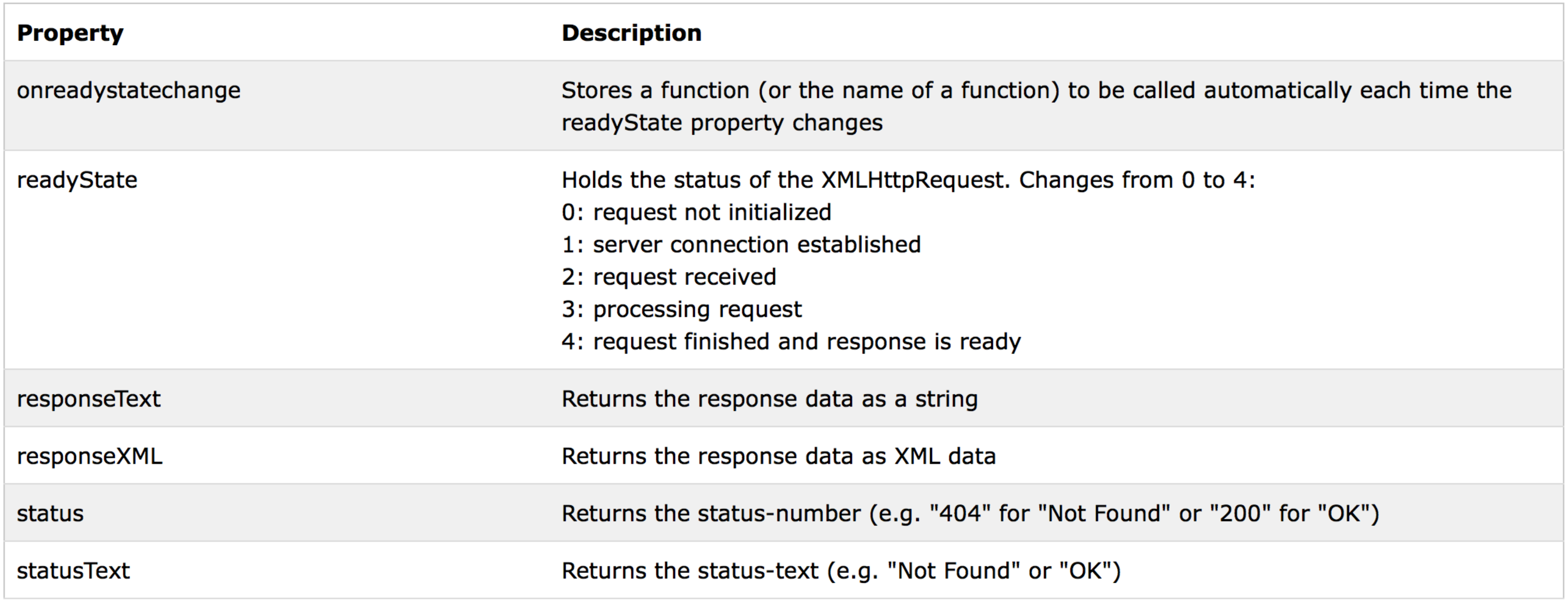
THE XMLHTTPREQUEST OBJECT METHODS



Method	Description
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method,url,async,uname,pswd)	Specifies the type of request, the URL, if the request should be handled asynchronously or not, and other optional attributes of a request method: the type of request: GET or POST url: the location of the file on the server async: true (asynchronous) or false (synchronous)
send(string)	send(string) Sends the request off to the server. string: Only used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

http://www.w3schools.com/xml/dom_http.asp

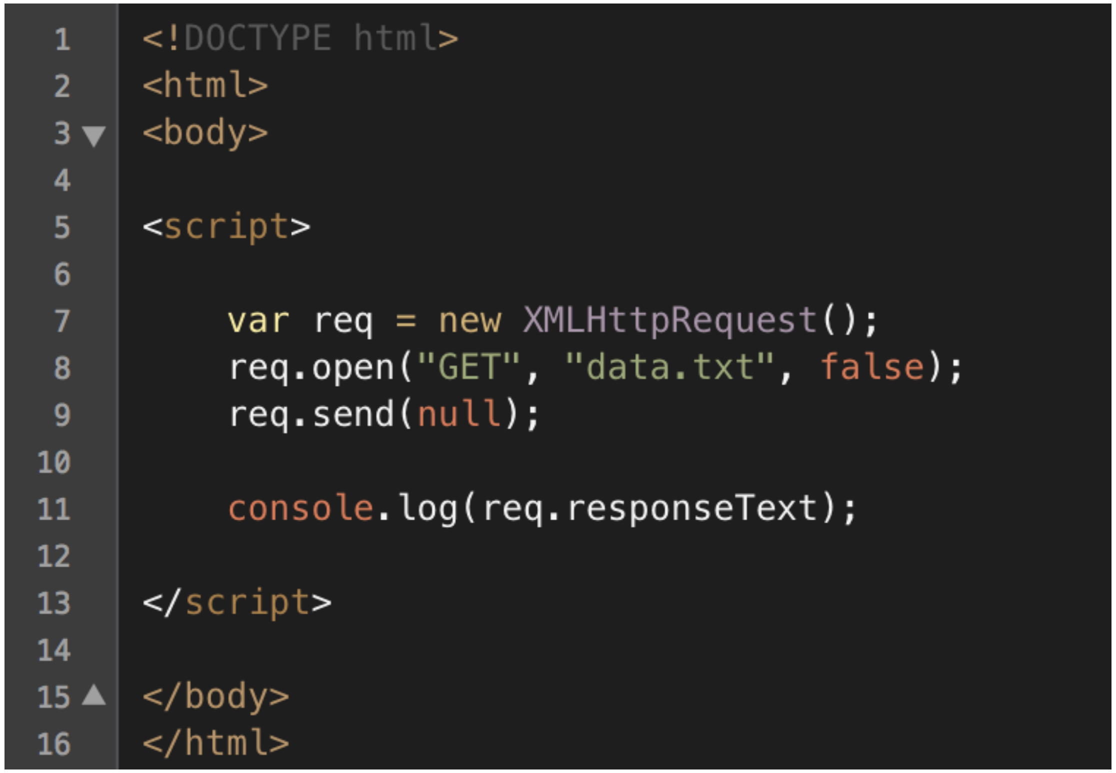
THE XmlHttpRequest OBJECT PROPERTIES



Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number (e.g. "404" for "Not Found" or "200" for "OK")
statusText	Returns the status-text (e.g. "Not Found" or "OK")

http://www.w3schools.com/xml/dom_http.asp

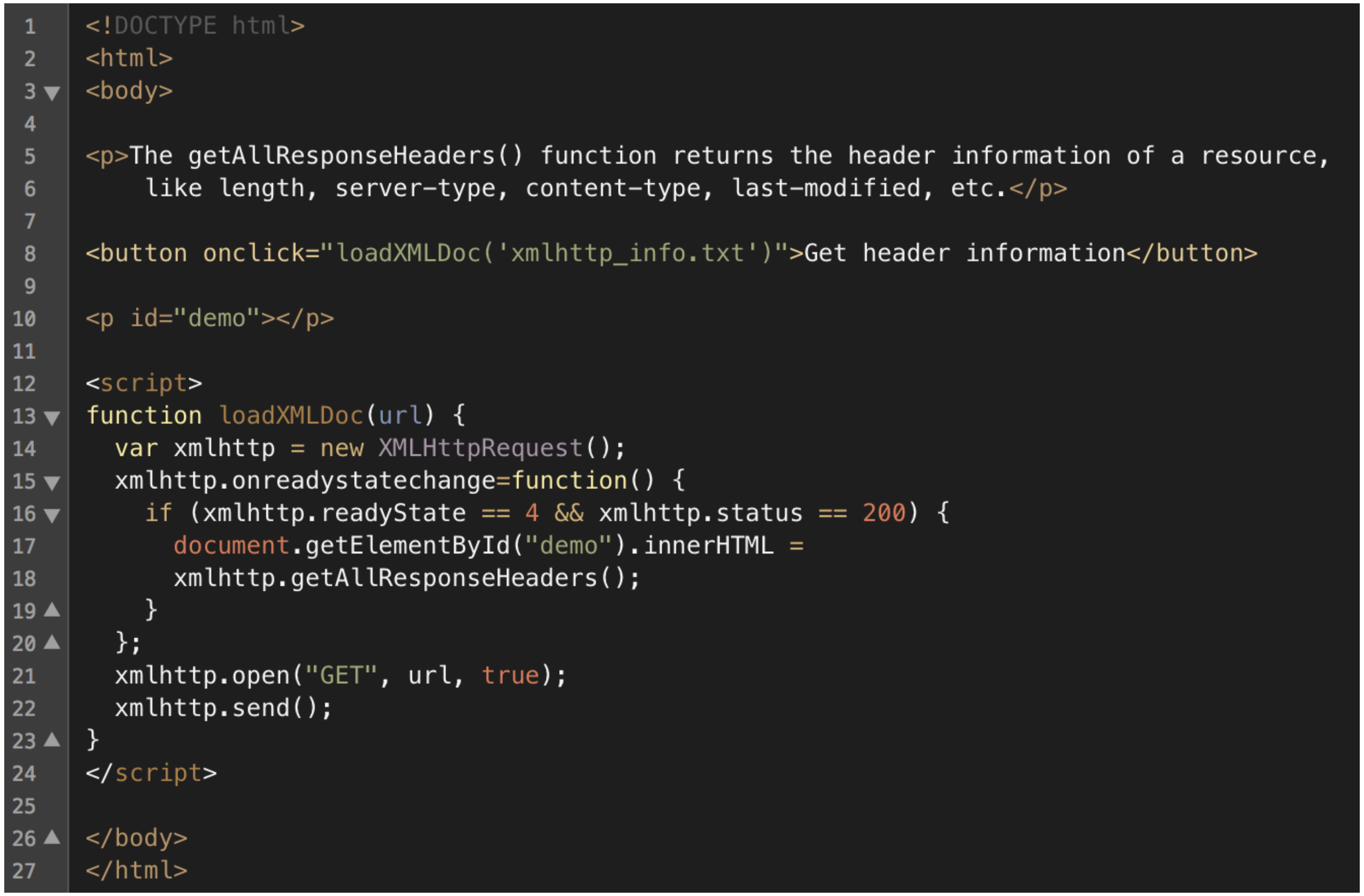
A SIMPLE XMLHTTPREQUEST EXAMPLE



```
<!DOCTYPE html>
<html>
<body>

<script>

    var req = new XMLHttpRequest();
    req.open("GET", "data.txt", false);
    req.send(null);

    console.log(req.responseText);

</script>

</body>
</html>
```

http://www.w3schools.com/xml/dom_http.asp

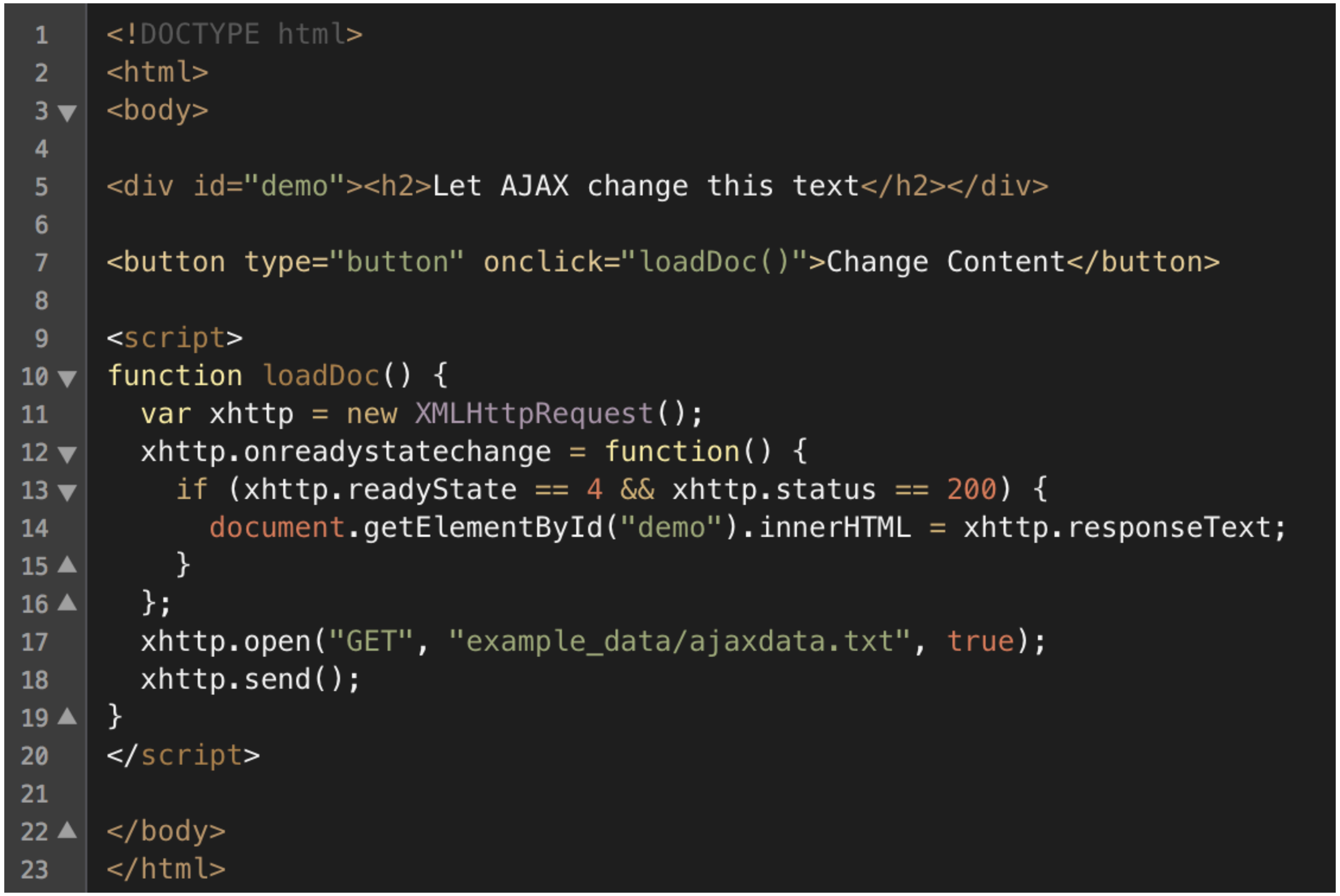
XMLHTTPREQUEST: getAllResponseHeaders()



```
<!DOCTYPE html>
<html>
<body>

<p>The getAllResponseHeaders() function returns the header information of a resource,
    like length, server-type, content-type, last-modified, etc.</p>

<button onclick="loadXMLDoc('xmlhttp_info.txt')">Get header information</button>

<p id="demo"></p>

<script>
function loadXMLDoc(url) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      document.getElementById("demo").innerHTML =
      xmlhttp.getAllResponseHeaders();
    }
  };
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
}
</script>

</body>
</html>
```

http://www.w3schools.com/xml/dom_http.asp

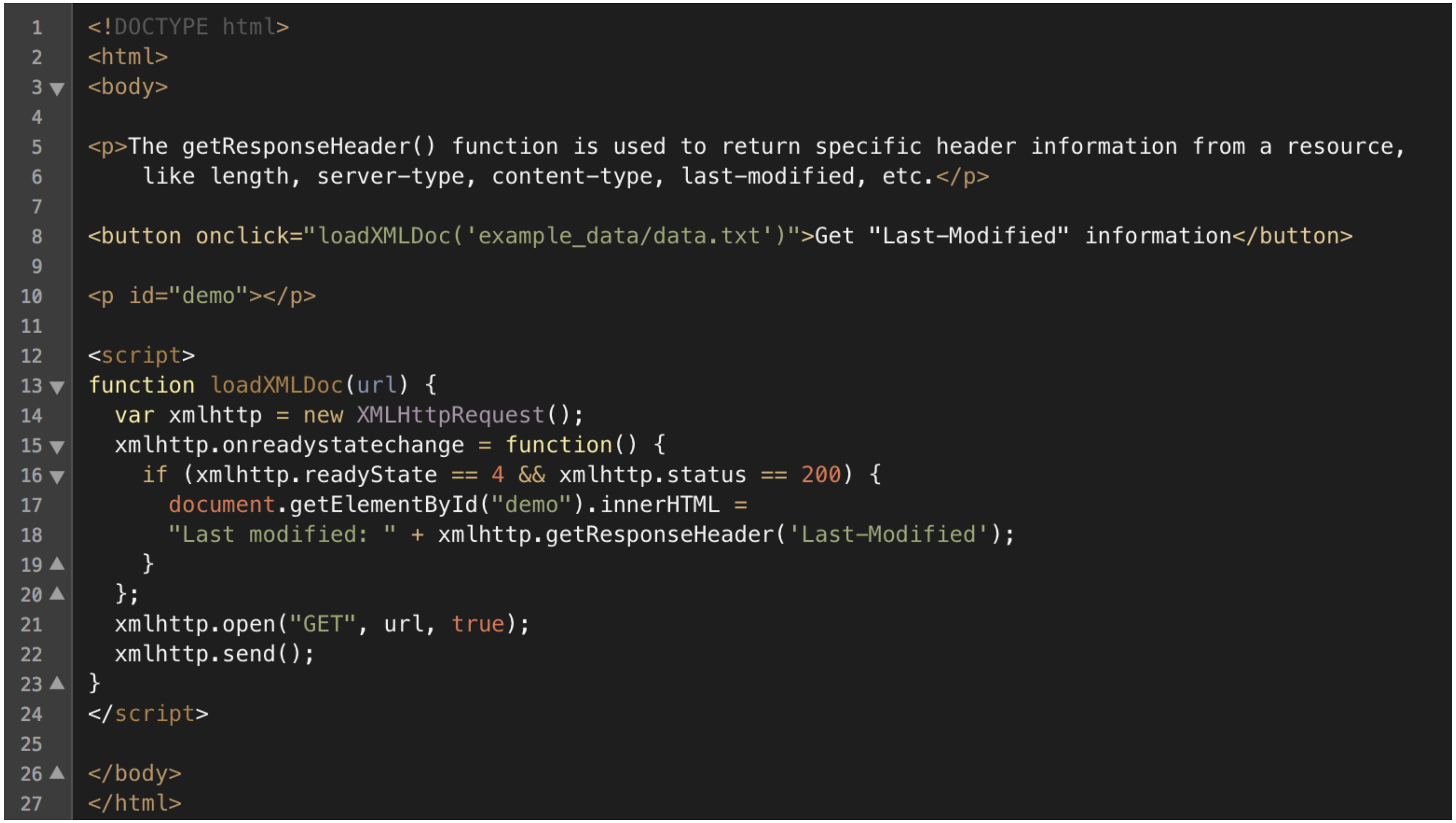
AJAX SIMPLE EXAMPLE



```
<!DOCTYPE html>
<html>
<body>

<div id="demo"><h2>Let AJAX change this text</h2></div>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      document.getElementById("demo").innerHTML = xhttp.responseText;
    }
  };
  xhttp.open("GET", "example_data/ajaxdata.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```

http://www.w3schools.com/ajax/ajax_intro.asp

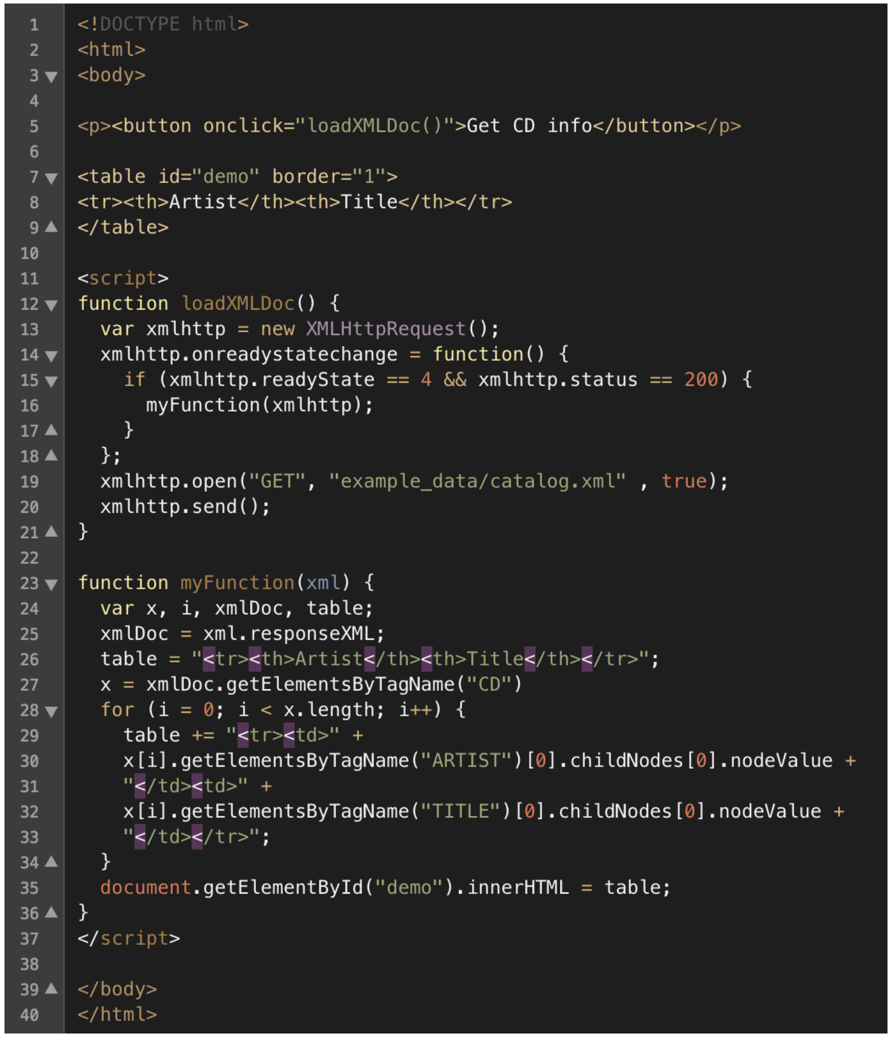
XMLHTTPREQUEST: getResponseHeader()



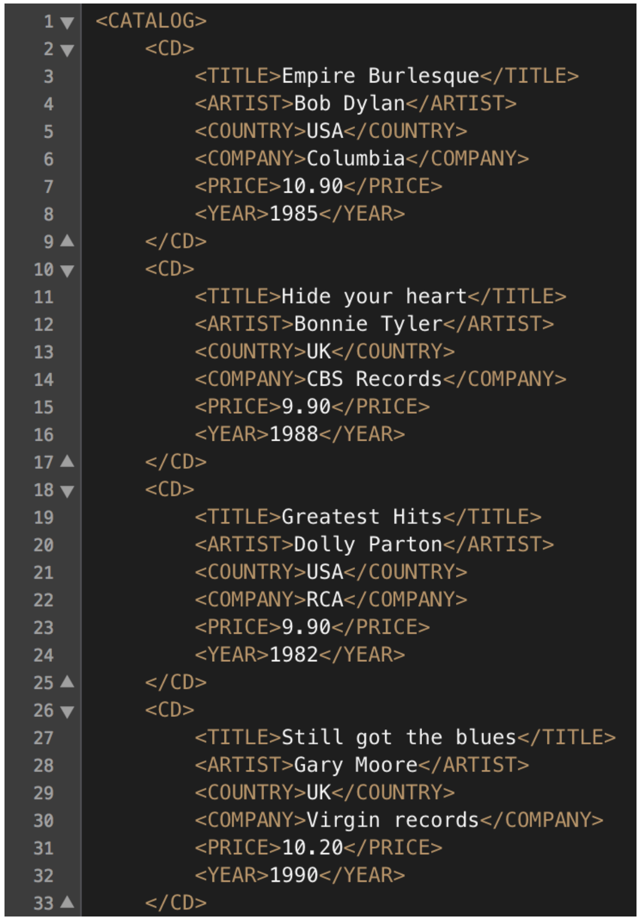
```
<!DOCTYPE html>
<html>
<body>

<p>The getResponseHeader() function is used to return specific header information from a resource,
    like length, server-type, content-type, last-modified, etc.</p>

<button onclick="loadXMLDoc('example_data/data.txt')">Get "Last-Modified" information</button>

<p id="demo"></p>

<script>
function loadXMLDoc(url) {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      document.getElementById("demo").innerHTML =
      "Last modified: " + xmlhttp.getResponseHeader('Last-Modified');
    }
  };
  xmlhttp.open("GET", url, true);
  xmlhttp.send();
}
</script>

</body>
</html>
```

http://www.w3schools.com/xml/dom_http.asp

XMLHTTPREQUEST: XML data



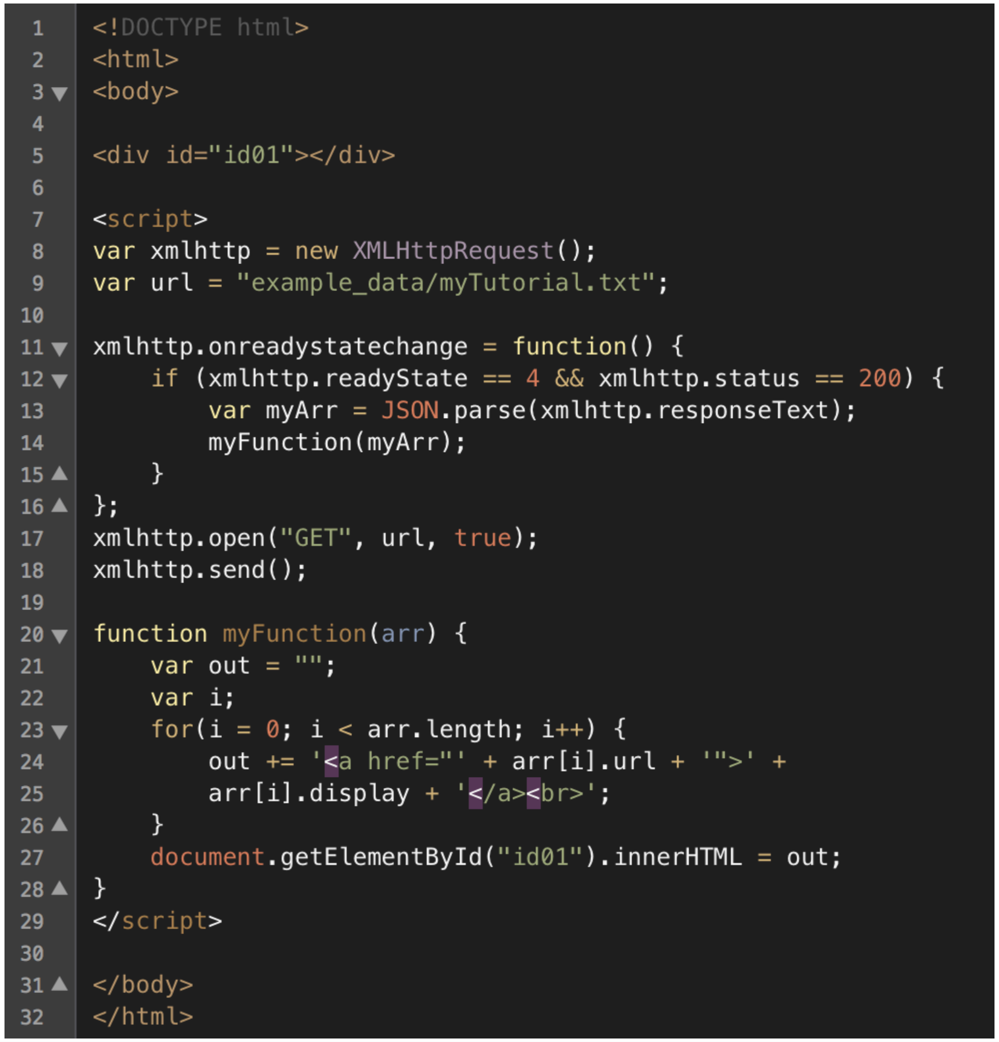
```html
<!DOCTYPE html>
<html>
<body>

<p><button onclick="loadXMLDoc()">Get CD info</button></p>

<table id="demo" border="1">
<tr><th>Artist</th><th>Title</th></tr>
</table>

<script>
function loadXMLDoc() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
      myFunction(xmlhttp);
    }
  };
  xmlhttp.open("GET", "example_data/catalog.xml" , true);
  xmlhttp.send();
}

function myFunction(xml) {
  var x, i, xmlDoc, table;
  xmlDoc = xml.responseXML;
  table = "<tr><th>Artist</th><th>Title</th></tr>";
  x = xmlDoc.getElementsByTagName("CD")
  for (i = 0; i < x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```



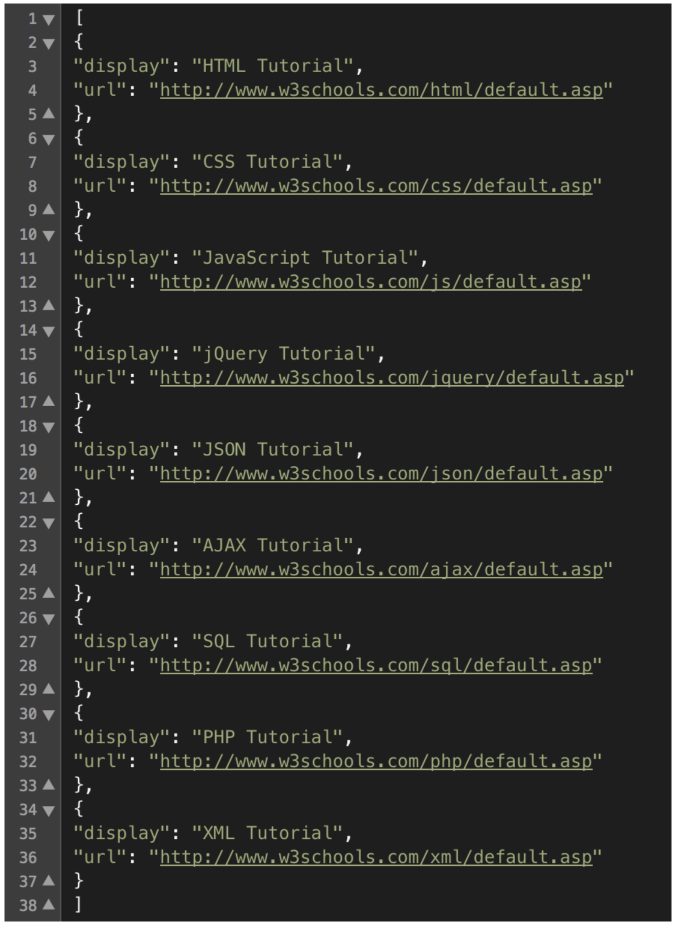
```xml
<CATALOG>
    <CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>Hide your heart</TITLE>
        <ARTIST>Bonnie Tyler</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS Records</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1988</YEAR>
    </CD>
    <CD>
        <TITLE>Greatest Hits</TITLE>
        <ARTIST>Dolly Parton</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>RCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1982</YEAR>
    </CD>
    <CD>
        <TITLE>Still got the blues</TITLE>
        <ARTIST>Gary Moore</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>Virgin records</COMPANY>
        <PRICE>10.20</PRICE>
        <YEAR>1990</YEAR>
    </CD>
```

http://www.w3schools.com/xml/dom_http.asp

XMLHTTPREQUEST: JSON data



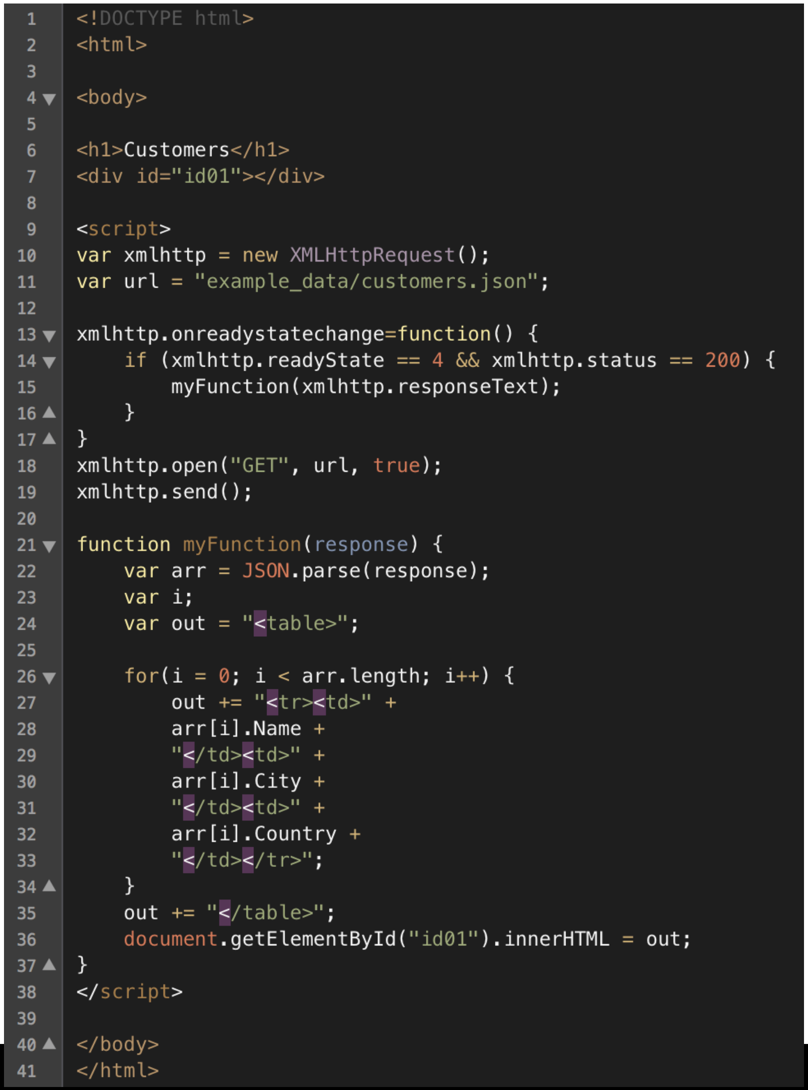
```html
<!DOCTYPE html>
<html>
<body>

<div id="id01"></div>

<script>
var xmlhttp = new XMLHttpRequest();
var url = "example_data/myTutorial.txt";

xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var myArr = JSON.parse(xmlhttp.responseText);
        myFunction(myArr);
    }
};
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(arr) {
    var out = "";
    var i;
    for(i = 0; i < arr.length; i++) {
        out += '<a href="' + arr[i].url + '">' +
        arr[i].display + '</a><br>';
    }
    document.getElementById("id01").innerHTML = out;
}
</script>

</body>
</html>
```



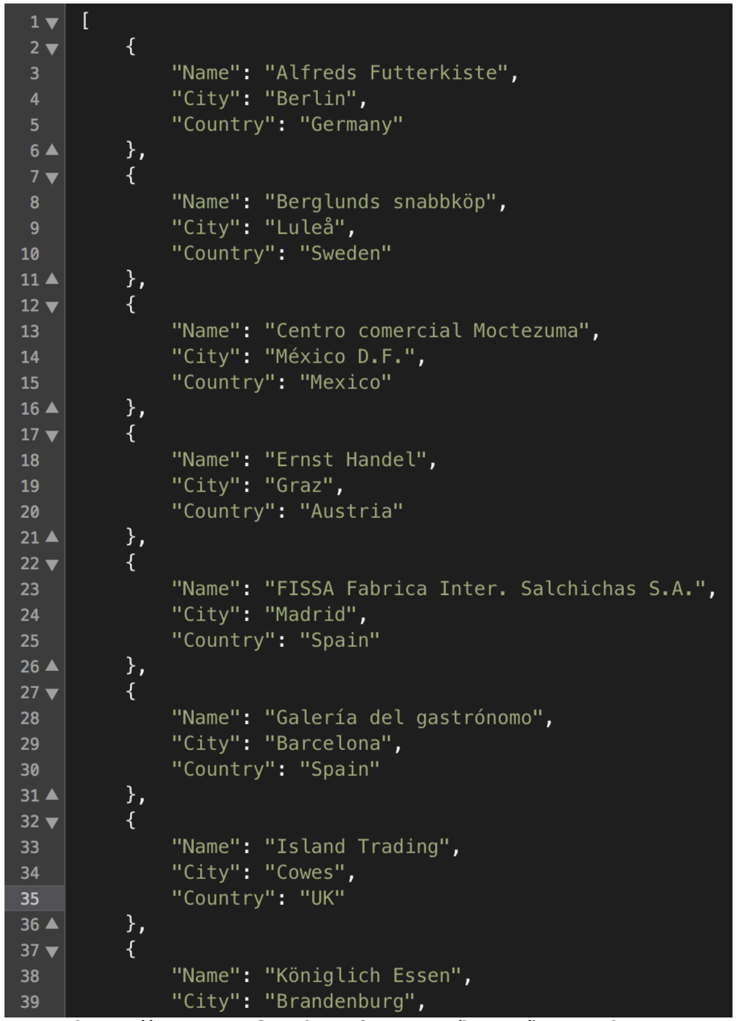
```json
[
{
"display": "HTML Tutorial",
"url": "http://www.w3schools.com/html/default.asp"
},
{
"display": "CSS Tutorial",
"url": "http://www.w3schools.com/css/default.asp"
},
{
"display": "JavaScript Tutorial",
"url": "http://www.w3schools.com/js/default.asp"
},
{
"display": "jQuery Tutorial",
"url": "http://www.w3schools.com/jquery/default.asp"
},
{
"display": "JSON Tutorial",
"url": "http://www.w3schools.com/json/default.asp"
},
{
"display": "AJAX Tutorial",
"url": "http://www.w3schools.com/ajax/default.asp"
},
{
"display": "SQL Tutorial",
"url": "http://www.w3schools.com/sql/default.asp"
},
{
"display": "PHP Tutorial",
"url": "http://www.w3schools.com/php/default.asp"
},
{
"display": "XML Tutorial",
"url": "http://www.w3schools.com/xml/default.asp"
}
]
```

http://www.w3schools.com/json/json_http.asp

XMLHTTPREQUEST: JSON data – fields



```
<!DOCTYPE html>
<html>

<body>

<h1>Customers</h1>
<div id="id01"></div>

<script>
var xmlhttp = new XMLHttpRequest();
var url = "example_data/customers.json";

xmlhttp.onreadystatechange=function() {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        myFunction(xmlhttp.responseText);
    }
}
xmlhttp.open("GET", url, true);
xmlhttp.send();

function myFunction(response) {
    var arr = JSON.parse(response);
    var i;
    var out = "<table>";

    for(i = 0; i < arr.length; i++) {
        out += "<tr><td>" +
        arr[i].Name +
        "</td><td>" +
        arr[i].City +
        "</td><td>" +
        arr[i].Country +
        "</td></tr>";
    }
    out += "</table>";
    document.getElementById("id01").innerHTML = out;
}
</script>

</body>
</html>
```



```
[
    {
        "Name": "Alfreds Futterkiste",
        "City": "Berlin",
        "Country": "Germany"
    },
    {
        "Name": "Berglunds snabbköp",
        "City": "Luleå",
        "Country": "Sweden"
    },
    {
        "Name": "Centro comercial Moctezuma",
        "City": "México D.F.",
        "Country": "Mexico"
    },
    {
        "Name": "Ernst Handel",
        "City": "Graz",
        "Country": "Austria"
    },
    {
        "Name": "FISSA Fabrica Inter. Salchichas S.A.",
        "City": "Madrid",
        "Country": "Spain"
    },
    {
        "Name": "Galería del gastrónomo",
        "City": "Barcelona",
        "Country": "Spain"
    },
    {
        "Name": "Island Trading",
        "City": "Cowes",
        "Country": "UK"
    },
    {
        "Name": "Königlich Essen",
        "City": "Brandenburg",
```

http://www.w3schools.com/json/json_http.asp

MALMÖ UNIVERSITY

FORMS AND FORM FIELDS

- Forms are used as a way to submit information provided by the user over HTTP.
- Their elements are part of the DOM
- DOM elements that represent form fields support a number of properties and events that are not present on other elements.
- These make it possible to inspect and control such input fields with JavaScript programs and do things such as adding functionality to a traditional form or using forms and fields as building blocks in a JavaScript application.

FIELDS

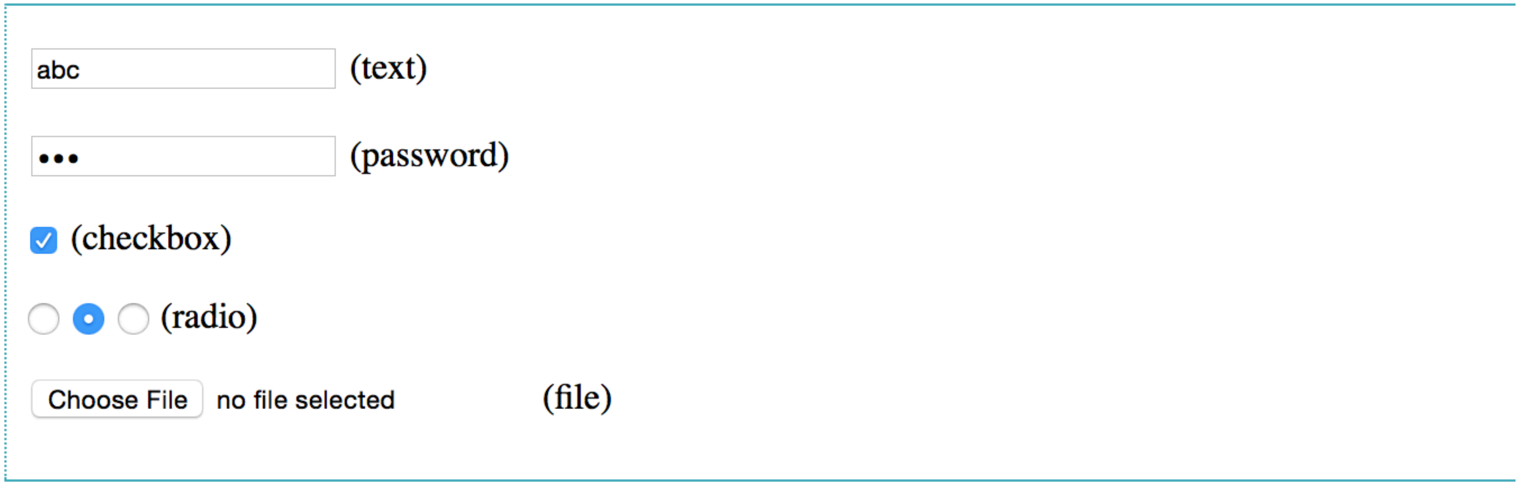
- A web form consists of any number of input fields grouped in a `<form>` tag.

- A lot of field types use the `<input>` tag. This tag's type attribute is used to select the field's style. These are some commonly used `<input>` types:

• `text`	A single-line text field
• `password`	Same as text but hides the text that is typed
• `checkbox`	An on/off switch
• `radio`	(Part of) a multiple-choice field
• `file`	Allows the user to choose a file from their computer

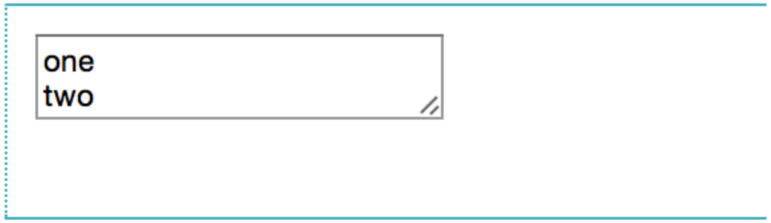
EXAMPLE

```html
<p><input type="text" value="abc"> (text)</p>
<p><input type="password" value="abc"> (password)</p>
<p><input type="checkbox" checked> (checkbox)</p>
<p><input type="radio" value="A" name="choice">
   <input type="radio" value="B" name="choice" checked>
   <input type="radio" value="C" name="choice"> (radio)</p>
<p><input type="file"> (file)</p>
```



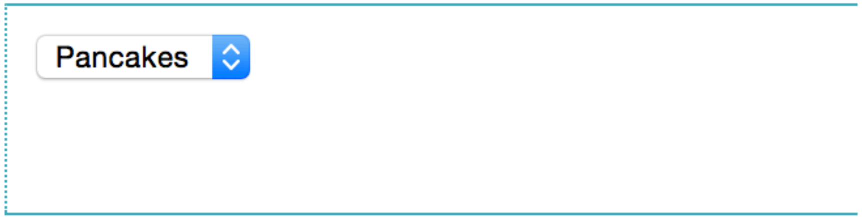
EXAMPLE (CONT'D)

```html
<textarea>
one
two
three
</textarea>
```



Multiline text fields

```html
<select>
   <option>Pancakes</option>
   <option>Pudding</option>
   <option>Ice cream</option>
</select>
```



`<select>` tag is used to create a field that allows the user to select from a number of predefined options

FOCUS

- Unlike most elements in an HTML document, form fields can get *keyboard focus*. When clicked—or activated in some other way—they become the currently active element, the main recipient of keyboard input.

```
<input type="text">
<script>
  document.querySelector("input").focus();
  console.log(document.activeElement.tagName);
  // → INPUT
  document.querySelector("input").blur();
  console.log(document.activeElement.tagName);
  // → BODY
</script>
```

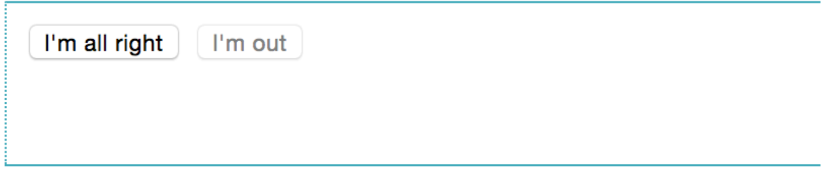
but HTML also provides the autofocus attribute:

```
<input type="text" autofocus>
```

DISABLED FIELDS

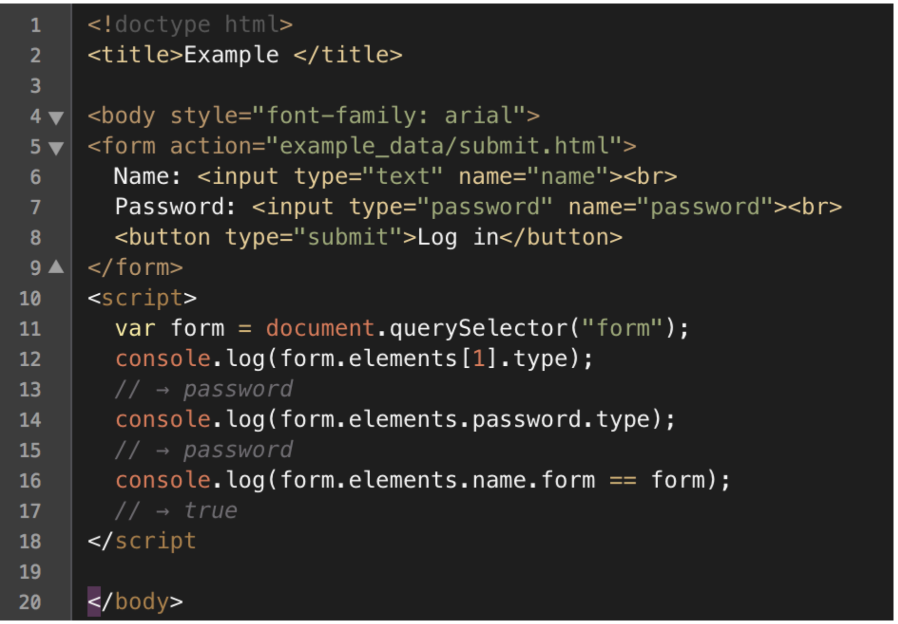
- All form fields can be *disabled* through their disabled attribute, which also exists as a property on the element’s DOM object.

```html
<button>I'm all right</button>
<button disabled>I'm out</button>
```



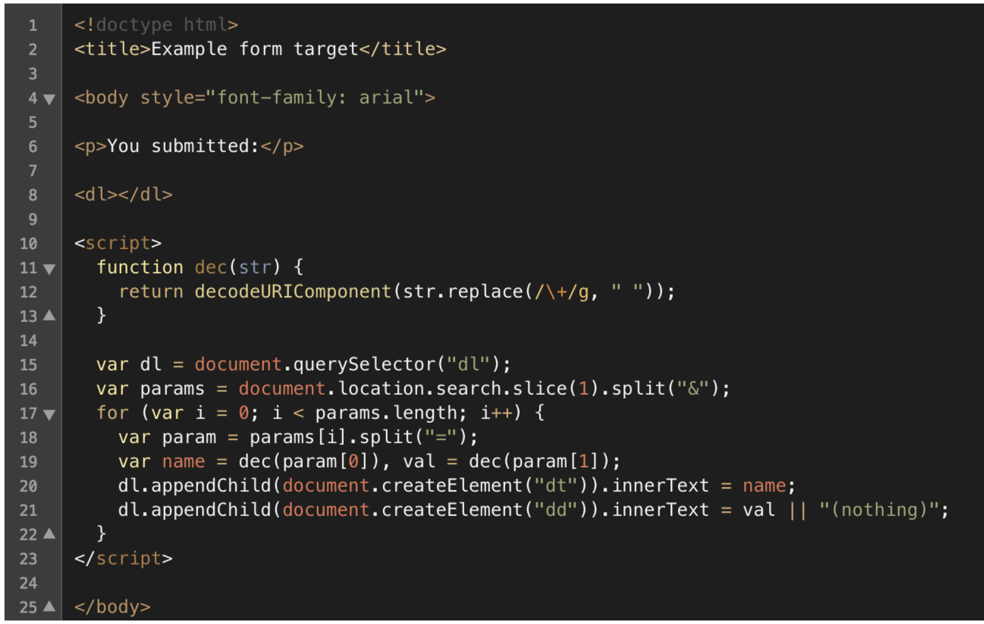
THE FORM AS A WHOLE

- When a field is contained in a <form> element, its DOM element will have a property form linking back to the form's DOM element.
- The <form> element, in turn, has a property called elements that contains an array-like collection of the fields inside it.



```html
<!doctype html>
<title>Example </title>

<body style="font-family: arial">
<form action="example_data/submit.html">
  Name: <input type="text" name="name"><br>
  Password: <input type="password" name="password"><br>
  <button type="submit">Log in</button>
</form>
<script>
  var form = document.querySelector("form");
  console.log(form.elements[1].type);
  // → password
  console.log(form.elements.password.type);
  // → password
  console.log(form.elements.name.form == form);
  // → true
</script

</body>
```

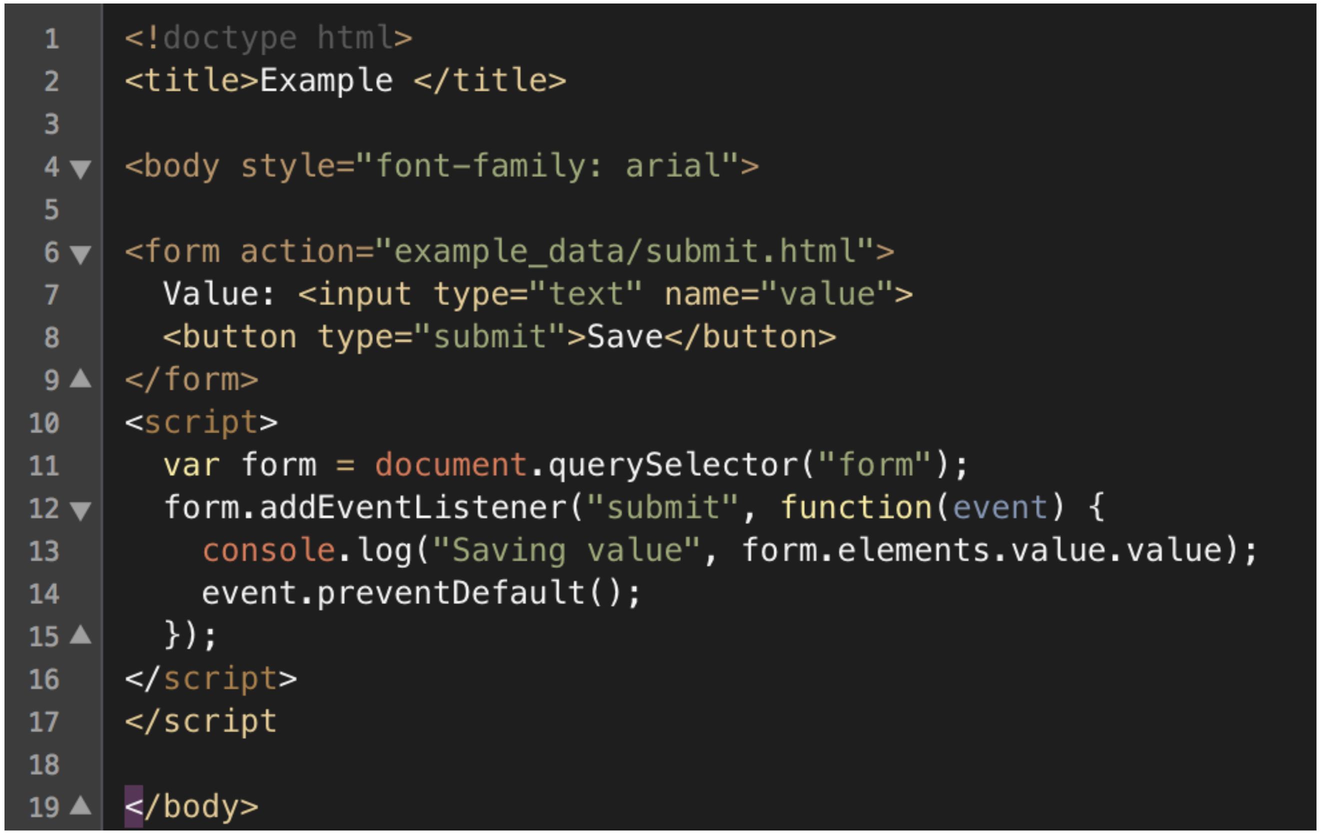
Form example



```html
<!doctype html>
<title>Example form target</title>

<body style="font-family: arial">

<p>You submitted:</p>

<dl></dl>

<script>
  function dec(str) {
    return decodeURIComponent(str.replace(/\+/g, " "));
  }

  var dl = document.querySelector("dl");
  var params = document.location.search.slice(1).split("&");
  for (var i = 0; i < params.length; i++) {
    var param = params[i].split("=");
    var name = dec(param[0]), val = dec(param[1]);
    dl.appendChild(document.createElement("dt")).innerText = name;
    dl.appendChild(document.createElement("dd")).innerText = val || "(nothing)";
  }
</script>

</body>
```

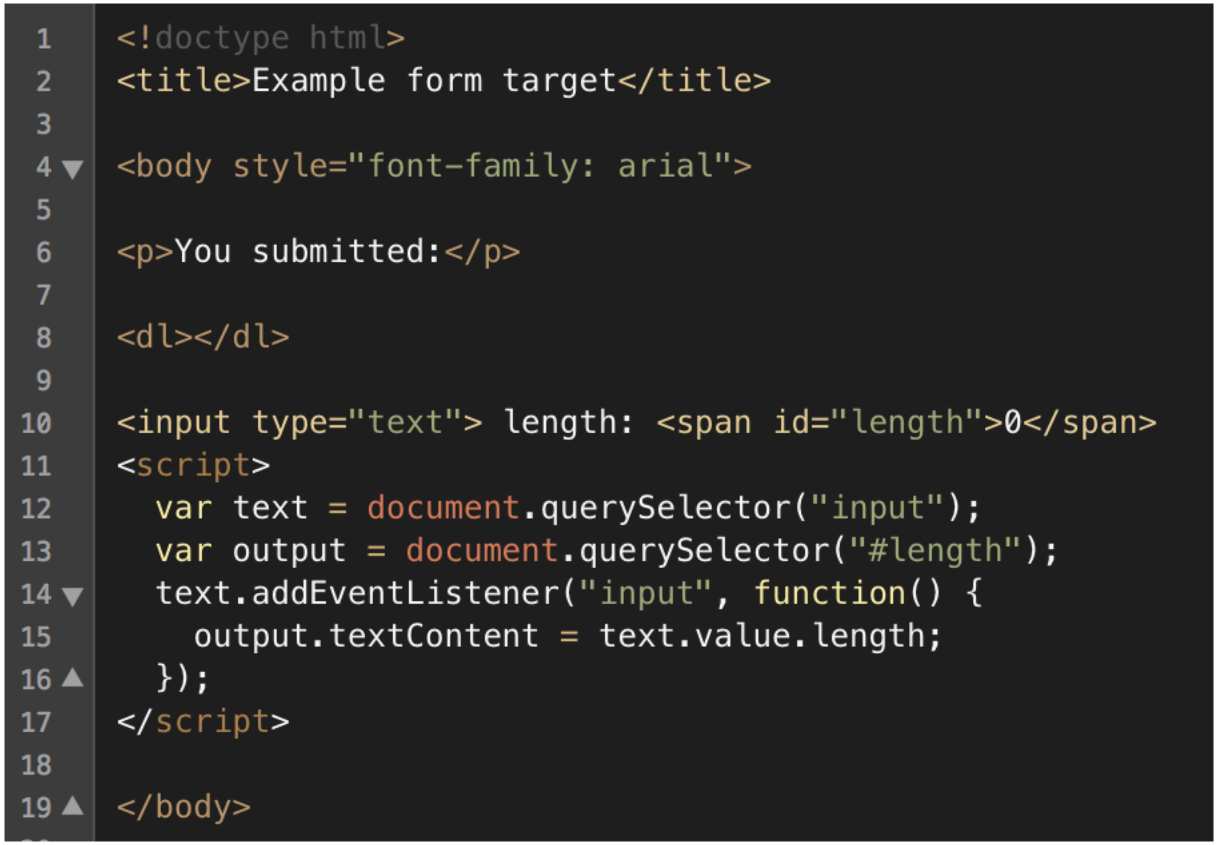
Submit example

THE FORM AS A WHOLE (CONT'D)



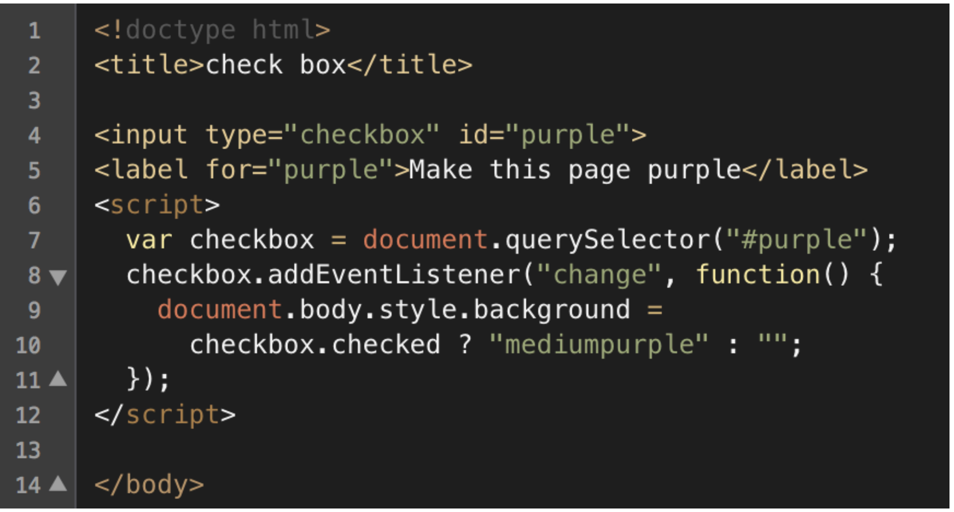
```
<!doctype html>
<title>Example </title>

<body style="font-family: arial">

<form action="example_data/submit.html">
  Value: <input type="text" name="value">
  <button type="submit">Save</button>
</form>
<script>
  var form = document.querySelector("form");
  form.addEventListener("submit", function(event) {
    console.log("Saving value", form.elements.value.value);
    event.preventDefault();
  });
</script>
</script

</body>
```

THE FORM AS A WHOLE (TEXT FIELD)

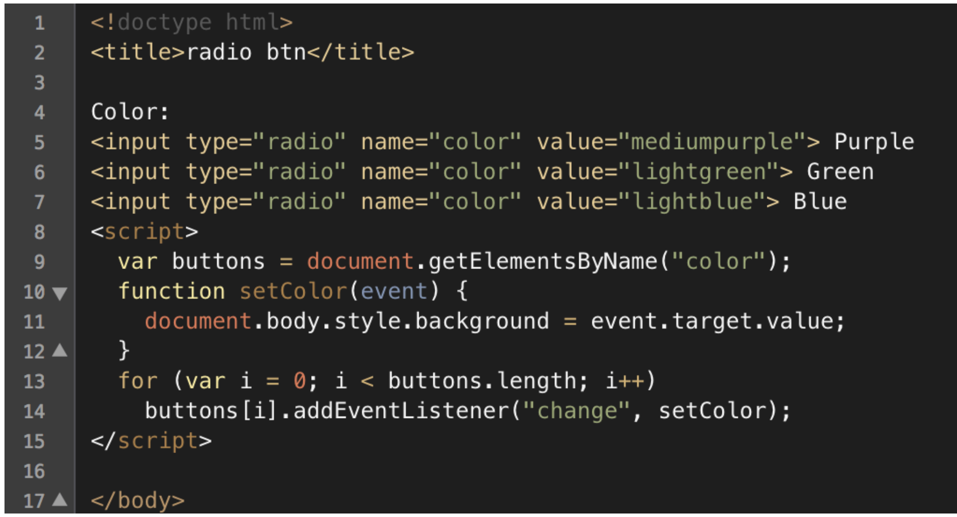


```html
<!doctype html>
<title>Example form target</title>

<body style="font-family: arial">

<p>You submitted:</p>

<dl></dl>

<input type="text"> length: <span id="length">0</span>
<script>
  var text = document.querySelector("input");
  var output = document.querySelector("#length");
  text.addEventListener("input", function() {
    output.textContent = text.value.length;
  });
</script>

</body>
```

CHECKBOXES AND RADIO BUTTONS

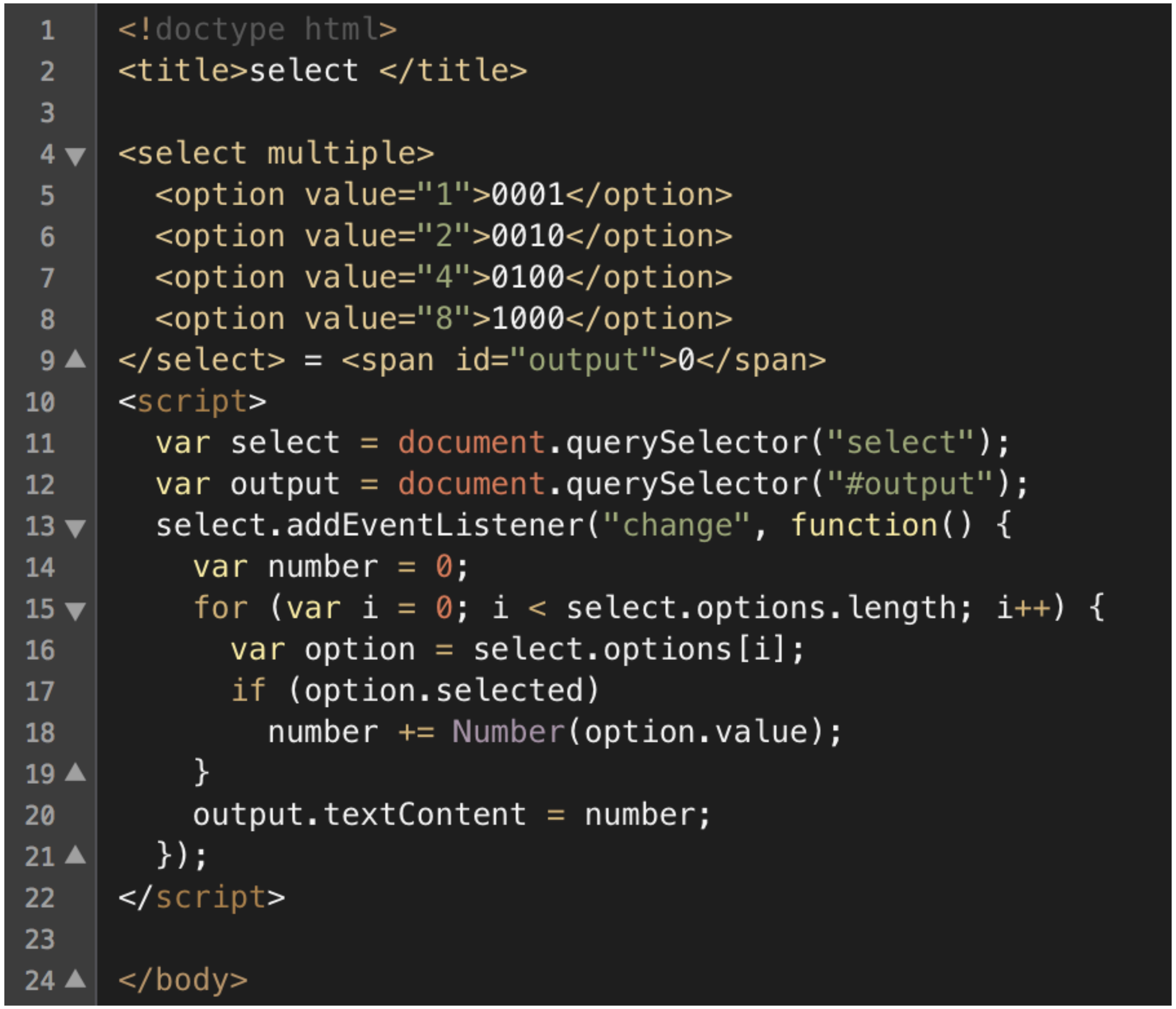


```
<!doctype html>
<title>check box</title>

<input type="checkbox" id="purple">
<label for="purple">Make this page purple</label>
<script>
  var checkbox = document.querySelector("#purple");
  checkbox.addEventListener("change", function() {
    document.body.style.background =
      checkbox.checked ? "mediumpurple" : "";
  });
</script>

</body>
```



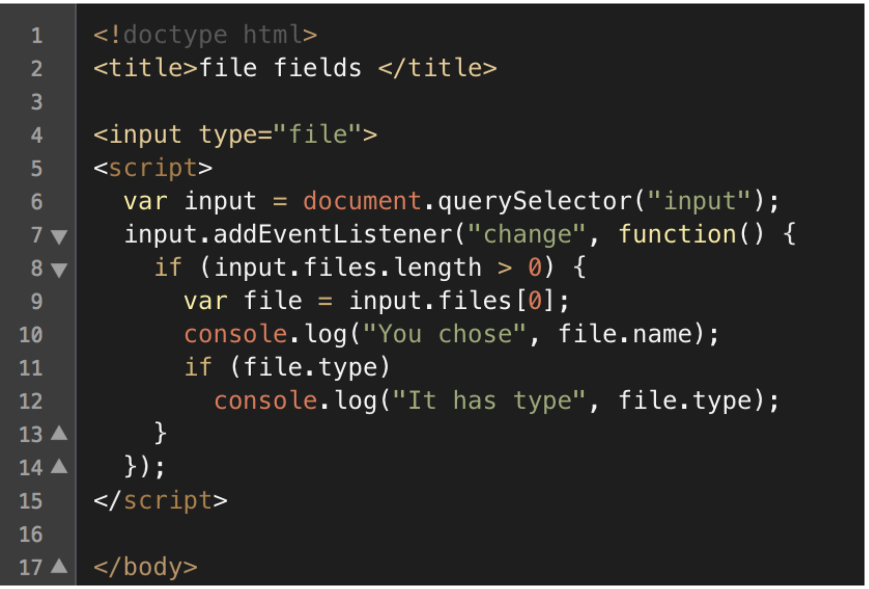
```
<!doctype html>
<title>radio btn</title>

Color:
<input type="radio" name="color" value="mediumpurple"> Purple
<input type="radio" name="color" value="lightgreen"> Green
<input type="radio" name="color" value="lightblue"> Blue
<script>
  var buttons = document.getElementsByName("color");
  function setColor(event) {
    document.body.style.background = event.target.value;
  }
  for (var i = 0; i < buttons.length; i++)
    buttons[i].addEventListener("change", setColor);
</script>

</body>
```

SELECT FIELDS

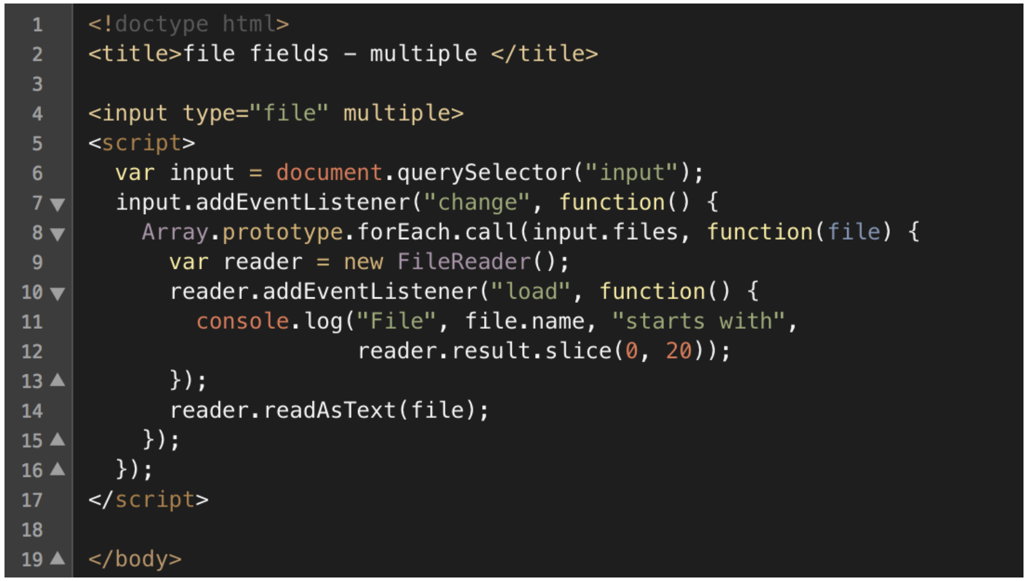


```
<!doctype html>
<title>select </title>

<select multiple>
  <option value="1">0001</option>
  <option value="2">0010</option>
  <option value="4">0100</option>
  <option value="8">1000</option>
</select> = <span id="output">0</span>
<script>
  var select = document.querySelector("select");
  var output = document.querySelector("#output");
  select.addEventListener("change", function() {
    var number = 0;
    for (var i = 0; i < select.options.length; i++) {
      var option = select.options[i];
      if (option.selected)
        number += Number(option.value);
    }
    output.textContent = number;
  });
</script>

</body>
```

FILE FIELDS



```html
<!doctype html>
<title>file fields </title>

<input type="file">
<script>
  var input = document.querySelector("input");
  input.addEventListener("change", function() {
    if (input.files.length > 0) {
      var file = input.files[0];
      console.log("You chose", file.name);
      if (file.type)
        console.log("It has type", file.type);
    }
  });
</script>

</body>
```



```html
<!doctype html>
<title>file fields - multiple </title>

<input type="file" multiple>
<script>
  var input = document.querySelector("input");
  input.addEventListener("change", function() {
    Array.prototype.forEach.call(input.files, function(file) {
      var reader = new FileReader();
      reader.addEventListener("load", function() {
        console.log("File", file.name, "starts with",
                    reader.result.slice(0, 20));
      });
      reader.readAsText(file);
    });
  });
</script>

</body>
```

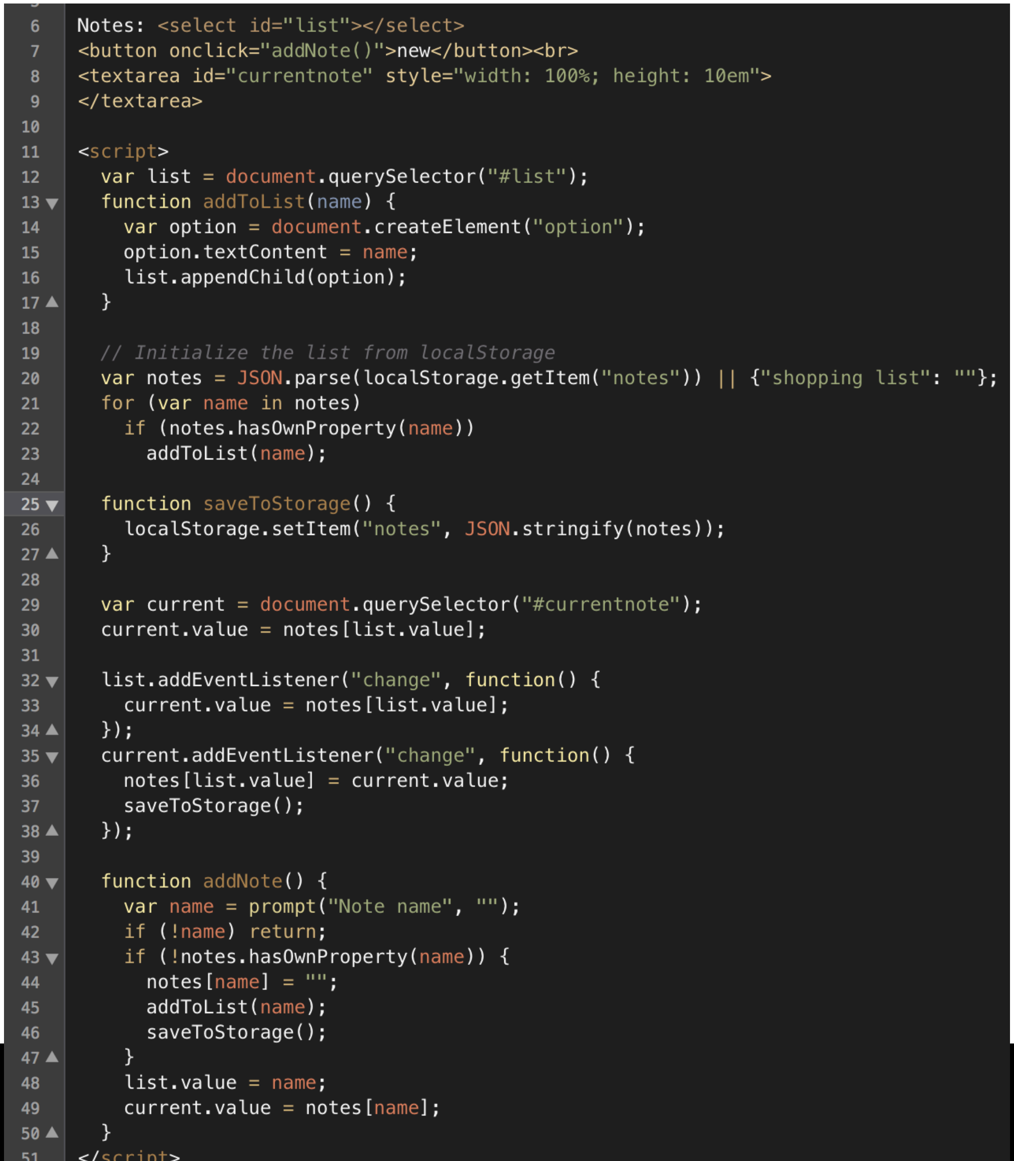
STORING DATA CLIENT-SIDE

You can store string data in a way that survives page reloads by putting it in the `localStorage` object

A value in localStorage sticks around until it is overwritten, it is removed with `removeItem`, or the user clears their local data.

```
localStorage.setItem("username", "marijn");
console.log(localStorage.getItem("username"));
// → marijn
localStorage.removeItem("username");
```

STORING DATA CLIENT-SIDE (CONT'D)



```
Notes: <select id="list"></select>
<button onclick="addNote()">new</button><br>
<textarea id="currentnote" style="width: 100%; height: 10em">
</textarea>

<script>
  var list = document.querySelector("#list");
  function addToList(name) {
    var option = document.createElement("option");
    option.textContent = name;
    list.appendChild(option);
  }

  // Initialize the list from localStorage
  var notes = JSON.parse(localStorage.getItem("notes")) || {"shopping list": ""};
  for (var name in notes)
    if (notes.hasOwnProperty(name))
      addToList(name);

  function saveToStorage() {
    localStorage.setItem("notes", JSON.stringify(notes));
  }

  var current = document.querySelector("#currentnote");
  current.value = notes[list.value];

  list.addEventListener("change", function() {
    current.value = notes[list.value];
  });
  current.addEventListener("change", function() {
    notes[list.value] = current.value;
    saveToStorage();
  });

  function addNote() {
    var name = prompt("Note name", "");
    if (!name) return;
    if (!notes.hasOwnProperty(name)) {
      notes[name] = "";
      addToList(name);
      saveToStorage();
    }
    list.value = name;
    current.value = notes[name];
  }
</script>
```

MALMÖ UNIVERSITY

REFLECTION

- HTTP
- AJAX
- XMLHttpRequest
- FORMS

THANK YOU

QUESTIONS?

Literature:

Haverbeke, M. (2014). *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press.