Murdoch
UNIVERSITY

JSON + NodeJS JSON Parsing and Processing + NodeJS XML to JSON + JQuery Advanced

Lecture 8 (A)

Learning Objectives

- Introduction to JavaScript Object Notation (JSON)
- Look at JSON as an alternative technology for storage and transportation of data on the Web
- Look at some example code to parse and process JSON within the Node.js environment

Learning Objectives

- In the context of this unit:
  - We look at JSON as an important alternative Internet technology to use as solutions in different areas
  - A major part of developing a JSON solution is to design and implement software to deal with the information in JSON formatted documents (**JSON applications**)
  - Node.js, like JSON, uses JavaScript programming language, so there is a seamless integration

Introduction

- JSON is a lightweight, text-based open standard designed for human-readable data interchange
- JSON is used primarily to store and transmit data between a server and web applications
  - We have seen in the last three Topics that XML also provides a way of storing and transmitting such data
  - JSON provides an alternative storage format to XML, which is much more convenient to work with both at a document level and in terms of parsing and processing

Introduction

- Although originally derived from the JavaScript language, JSON is a language-independent data format
- Software for parsing and generating JSON data is readily available in a large variety of programming languages

Introduction

- The JSON format was originally specified by Douglas Crockford
- It is currently described by two competing standards
  - The ECMA-404 standard is minimal, describing only the allowed grammar syntax
  - The RFC 7159 provides some semantic and security considerations, in addition to the allowed grammar syntax

Introduction

- JSON stands for **JavaScript Object Notation**
  - It was designed for human-readable data interchange
  - It has been extended from the JavaScript scripting language
  - Its filename extension is `.json`
  - The official JSON Internet Media type is `application/json`
  - The Uniform Type Identifier is `public.json`

Introduction

- JSON can be used:
  - When writing JavaScript based applications including browser extensions and websites
  - For serializing and transmitting structured data (in JSON format) over network connections
  - Primarily to transmit data between web server applications and web clients
  - For Web Services and API's to provide public data (in JSON format)
  - With other modern programming languages

Introduction

- Characteristics of JSON:
  - It is easy to read and write
  - It is a lightweight text-based interchange format
  - It is language independent
  - It uses the JavaScript *object literal* notation to define JSON objects

Example: HTML Usage

```
<html>
  <head>
     <title>JSON Example</title>
     <script type="text/javascript" src="book.js">
     </script>
  </head>
  <body>
     <script type="text/javascript">
        displayJSON();
     </script>
  </body>
</html>
```

Example: HTML Usage

```
function displayJSON() {
   var object1 = { "bookname": "Papillon",
                   "author": "Henri Charriere" };
   document.write("<h1>JSON with JavaScript example</h1>");
   document.write("<br>");
   document.write("<h3>Bookname = "+ object1.bookname+"</h3>");
   document.write("<h3>Author = "+ object1.author+"</h3>");
   var object2 = { "bookname": "One Flew Over the Cuckoo's Nest",
                   "author": "Ken Kesey" };
   document.write("<br>");
   document.write("<h3>Bookname = "+ object2.bookname +"</h3>");
   document.write("<h3>Author = "+ object2.author+"</h3>");
   document.write("<hr />");
   document.write(object2.bookname + ", written by " +
                  object2.author + " can play with your mind!@#");
   document.write("<hr />");
}
```

Example: Browser Output

JSON with JavaScript example

Bookname = **Papillon**
Author = **Henri Charriere**

Bookname = **One Flew Over The Cuckoo's Nest**
Author = **Ken Kesey**

One Flew Over The Cuckoo's Nest, written by **Ken Kesey**, can play with your mind!@#

JSON Syntax

- JSON syntax is a subset of JavaScript syntax, and includes the following:
  - Curly braces designate objects (using the **object literal notation**)
  - Data is represented in `property:value` pairs
  - Each `property (key)` is followed by a colon (':')
  - Each `property:value` pair is separated by a comma (',') **except for the last pair**
  - Square brackets designate arrays, where elements are separated by a comma (',')

Example: JSON Object

```
{
   "book": [
   {
      "id": "01",
      "bookname": "Papillon",
      "edition": "first",
      "author": "Henri Charriere"
   },
   {
      "id": "02",
      "bookname": "One Flew Over the Cuckoo's Nest",
      "edition": "second",
      "author": "Ken Kesey"
   }]
}
```

Example: Things To Note

- The JSON object is defined within braces (i.e. curly brackets '`{''}`'), using object literal notation
- The name of the object is "`book`"
- This is immediately followed by a colon (':')
- The components of the object are defined in an array, designated by square brackets '`['']`'
- Each array element is an JSON object separated by a comma (',')
  - Again, each JSON object in the array is defined using the object literal notation

Example: Things To Note

- The properties that describe a '`book`' object are `id`, `bookname`, `edition`, and `author`
- Each property is enclosed in double quotes and is immediately by the colon (':')
- Each property has a value
  - String values are enclosed in double quote
  - Numeric, Boolean, and null values are not
- Each `property:value` pair (except for the last one) is separated by a comma (',')

JSON Data Structures

- Formally, JSON supports the following two data structures:
  1. Collection of `property:value` pairs (this Data Structure is supported by different programming languages)
  2. Ordered list of values (including array, list, vector or sequence, etc.)

JSON Data Types

Type	Description
Number	Double-precision floating-point format as in JavaScript. Octal and hexadecimal formats are not used. No NaN or Infinity is used in Number.
String	Double-quoted Unicode with backslash escaping
Boolean	true or false
Array	An ordered sequence of values
Value	Can be a string, a number, true or false, null
Object	An unordered collection of key:value pairs
Whitespace	Can be used between any pair of tokens
null	empty

JSON Data Types: Number

- The Number data type:
  - Double-precision floating-point format as in JavaScript and depends on implementation
  - Octal and hexadecimal formats are not used
  - No NaN or Infinity is used in Number

Type	Description
Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like 0.3, 0.9
Exponent	Exponent like e, e+, e-,E, E+, E-

JSON Data Types: Number

- Formal syntax:

```
var json-object = { string: number_value, ... }
```

- Example (note that because the value is a number, it is not quoted):

```
var obj = { "marks": 97 }
```

JSON Data Types: String

- The String data type:
  - A sequence of zero or more double quoted Unicode characters, with possible backslash escaping
  - Character is a single character string; i.e. a string with length 1
  - Formal syntax:

```
var json-object = { string: "string value", ... }
```

  - Example:

```
var obj = { "name" : "John" }
```

Murdoch UNIVERSITY

JSON Data Types: String Escape Characters

Type	Description
"	double quotation
\	reverse solidus
/	solidus
b	backspace
f	form feed
n	new line
r	carriage return
t	horizontal tab
u	four hexadecimal digits

JSON Data Types: Boolean

- The Boolean data type:
  - Only includes true or false values
- Formal syntax:

```
var json-object = { string: true/false, ... }
```

- Example (note boolean value not quoted):

```
var obj = { "name": "Arnold",
            "distinction": true }
```

JSON Data Types: Object

- The Object data type:
  - An unordered set of `property:value` pairs
  - These pairs are enclosed in braces (curly brackets), which means they begin with '`{`' and end with '`}`'
  - Each `property` is followed by a colon ('`:`') and the `property:value` pairs are comma separated ('`,`')
  - The `properties` must be strings and should be different from each other
  - Objects should be used when the `property` names are arbitrary strings

JSON Data Types: Object

- Formal syntax:

```
{ string : value, ... }
```

- Example:

```
{
   "id": "011A",
   "language": "JAVA",
   "price": 500
}
```

JSON Data Types: Array

- The Array data type:
  - An **ordered** collection of elements (typically objects or `property:value` pairs)
  - Arrays are designated by square brackets which means they begin with '`[`' and end with '`]`' and enclose a list of objects
  - Array elements are comma separated ('`,`')
  - Array indexing can be started at 0 or 1
  - Arrays should be used when the `properties` are sequential integers

JSON Data Types: Array

- Formal syntax:

```
[ value, ... ]
```

- Example (N.B. array values are objects):

```
{
   "book": [
      { "language": "Java", "edition": "third" },
      { "language": "C++", "edition": "fifth" },
      { "language": "Python", "edition": "second" }
   ]
}
```

JSON Data Types: Whitespace

- The Whitespace data type:
  - Can be inserted between any pair of tokens
  - Can be added to make code more readable
- Example (declaration with and without whitespace):

```
    {
    "book": [
        { "language": "Java", "edition": "third" },
        {"language":"C++","edition":"fifth"},
    ]
}
```

JSON Data Types: null

- The null data type (empty type)
- Formal syntax:

```
null
```

- Example:

```
var i = null;
if(i == null) {
   document.write("<h1>value is null</h1>");
}else{
   document.write("<h1>value is not null</h1>");
}
```

JSON Values

- A JSON value can include:
  - number (integer or floating point)
  - string
  - boolean
  - array
  - object
  - null

JSON Values

- Syntax:

```
String | Number | Object | Array |
TRUE | FALSE | NULL
```

- Example:

```
{
   "Number" : 1,
   "String" : "sachin",
   "null" : null,
   "True" : true
}
```

Creating Simple JSON Objects

- Objects can be created as an empty Object (using the object literal notation):

```
var JSONObj = {};
```

- `Property:value` pairs can be added using the dot notation '.'. Eg:

```
JSONObj.objname = "table";
```

Creating Simple JSON Objects

- OR an Object with properties (using the object literal notation):

```
var JSONObj = { "bookname": "BLACK BOOK",
                "price": 500 };
```

  - Property `bookname` with a `string` value, property `price` with a `numeric` value
  - Properties can be accessed by using dot notation '.'.

Creating Simple JSON Objects

- OR using the Object constructor:

```
var JSONObj = new Object();
```

- `Property:value` pairs can be added using the dot notation '.'.
- Eg:

```
JSONObj.objage = 2001;
```

JSON Object Example: HTML

```
<html>
   <head>
      <title>Creating Object with JavaScript</title>
      <script language="javascript" >
         var JSONObj = { "name": "Fred","year": 2005 };
         document.write("JSON / JavaScript Example");
         document.write("<br>");
         document.write("Name=" + JSONObj.name);
         document.write("Year=" + JSONObj.year);
      </script>
   </head>
   <body>
   </body>
</html>
```

Object Example: Browser Output

JSON / JavaScript Example

Name=**Fred**

Year=**2005**

JSON Array Example: HTML

- Creating an array of objects in JavaScript using JSON:

```
<html>
   <head>
      <title>Creation of an array of objects in
            JavaScript using JSON</title>
      <script language="javascript" >
        document.writeln("JSON array object");
```

JSON Array Example: HTML

```
var books = { "Pascal" : [
   {  "Name": "Pascal Made Simple",
      "price": 700 },
   {  "Name": "Guide to Pascal",
      "price": 400 }
   ],
   "Scala": [
   {  "Name": "Scala for the Impatient",
      "price": 1000 },
   {  "Name": "Scala in Depth",
      "price": 1300 }
   ]
}
```

JSON Array Example: HTML

```
var i = 0;
document.writeln("<table border='2'<tr>");
for(i=0;i<books.Pascal.length;i++) {
   document.writeln("<td>");
   document.writeln("<table border='1'
                              width=100 >");
   document.writeln("<tr><td><b>Name</b>
                    </td><td width=50>" +
      books.Pascal[i].Name+"</td></tr>");
   document.writeln("<tr><td><b>Price</b>
                    </td><td width=50>" +
      books.Pascal[i].price+"</td></tr>");
   document.writeln("</table>");
   document.writeln("</td>");
}
```

JSON Array Example: HTML

```
    for(i=0;i<books.Scala.length;i++) {
       document.writeln("<td>");
       document.writeln("<table border='1'
                                 width=100 >");
       document.writeln("<tr><td><b>Name</b>
                        </td><td width=50>" +
          books.Scala[i].Name+"</td></tr>");
       document.writeln("<tr><td><b>Price</b>
                        </td><td width=50>" +
          books.Scala[i].price+"</td></tr>");
       document.writeln("</table>");
       document.writeln("</td>");
    }
    document.writeln("</tr></table>");
  </script>
</head><body></body></html>
```

Murdoch UNIVERSITY

Array Example: Browser Output

- JSON array object

Name	Pascal Made Simple	**Name**	Guide to Pascal	**Name**	Scala for the Impatient	**Name**	Scala in Depth
Price	700	**Price**	400	**Price**	1000	**Price**	1300

JSON in Node.js

- Node.js has emerged as a leading platform for creating fully scalable applications within the least amount of time
- Additionally, the platform is constantly upgraded to allow developers to continue delivering applications for their clients
- Unlike the case of XML, choosing JSON files for storing, retrieving, and transmitting data is a convenient option

JSON in Node.js

- The reason for this is that the JSON files are simpler to work with (using dot notation to access data and arrays to store and process multiple objects)
  - They are also less cluttered and easy-to-read
- Also, as JSON and Node.js are JavaScript based, collaborating JSON files with Node.js ensures that the information can be simply accessed for processing

Handling JSON Files in Node.js

- Step 1:
  - Create a dummy JSON file:

```
{
    "username": "xyz",
    "password": "xyz@^123",
    "email": "xyz@xyz.com",
    "uid": 1100
}
```

  - Save the above JSON file as `dummy.json`

Handling JSON Files in Node.js

- Step 2:
  - Decide whether you want the synchronous or asynchronous method for reading the JSON file
  - *Synchronous* methods of reading a JSON file basically refers to one-way execution, wherein there is a single flow which executes the JSON file line by line
  - The control flow will move to the next statement **only after the current statement has completed**

Handling JSON Files in Node.js

- Step 2:
  - *Asynchronous* flow-control will move to the next line **regardless of whether the current statement has completed**, and possibly display information about the asynchronous statement's completion
  - *Synchronous* methods may be referred to as *blocking* methods, and *asynchronous* methods may be referred to as *non-blocking* methods

Handling JSON Files in Node.js

- Step 3:
  - Depending on your selection of the method for reading the JSON file (asynchronous or synchronous), the outputs would vary
- The examples on the next few slides demonstrate how to access (parse) data stored in a JSON object
- Both asynchronous and synchronous methods are considered

JSON Example: Node.js

- Save the following code in a file `book.json`

```
{
    "book": [
    {
        "id": "01",
        "bookname": "Papillon",
        "edition": "first",
        "author": "Henri Charriere"
    },
    {
        "id": "02",
        "bookname": "One Flew Over the Cuckoo's Nest",
        "edition": "second",
        "author": "Ken Kesey"
    }]
}
```

JSON Example: Synchronous `fs`

- Create JavaScript file `testBookfss.js`:

```
// Synchronous version using fs and JSON.parse
var fs = require('fs');
var obj = JSON.parse(
   fs.readFileSync('./book.json', 'utf-8')
);
// obj contains the JavaScript object for the json file
// here, we just display it
console.log(obj);
```

JSON Example: Asynchronous `fs`

- Create JavaScript file `testBookfsa.js`:

```
var fs = require('fs');
var obj;
fs.readFile('./book.json', 'utf-8', function(error, data) {
    if (error) throw error; // skip handling for now
    obj = JSON.parse(data);
    console.log(obj);
});
// as fs.readFile is asynchronous, the following
// statement may well be executed before the parsed
// result is displayed
console.log("The next statement after asynchronous read");
```

JSON Example: Node.js

- Provided `book.json` is in the same directory as your scripts, execute using node on command line:

```
node testBookfss.js
```

- But if you run the following command:

```
node testBookfsa.js
```

You will notice that the asynchronous read hasn't finished when the next statement is executed.

JSON Example: Synchronous `jsonfile`

- Install `jsonfile` module with `npm`
- Create JavaScript file `testBookjfs.js`:

```
// Synchronous version using jsonfile
var jf = require('jsonfile');
var obj = jf.readFileSync('./book.json');
// obj contains the JavaScript object for the JSON data
// here, we just display it
console.log(obj);
```

JSON Example: Asynchronous `jsonfile`

- Create JavaScript file `testBookjfa.js`:

```
// Asynchronous version using jsonfile
var jf = require('jsonfile');
js.readFile('./book.json', function(error, obj) {
    if (error) throw error; // skip handling for now
    // obj contains the JavaScript object for the
    // JSON data - can now process it.
    // here, we just display it
    console.log(obj);
});
console.log("The next statement after asynchronous read");
```

JSON Compared With XML

- JSON and XML are both human-readable formats and language independent
- They both have support for creating, reading and decoding data in real world situations
- We can compare JSON with XML based on the following factors:

JSON Compared With XML

- Verbose
  - XML is more verbose than JSON, so it is faster for humans to read and write JSON
- Arrays Usage
  - XML is used to describe structured data which does not include arrays, whereas JSON does include arrays to help define its objects
- Parsing
  - JavaScript's `eval` method parses JSON
  - When applied to JSON, `eval` returns the described object

JSON Compared With XML

- Syntactically:
  - JSON

```
{
    "company": "Volkswagen",
    "name": "Vento",
    "price": 800000
}
```

  - XML

```
<car>
    <company>Volkswagen</company>
    <name>Vento</name>
    <price>800000</price>
</car>
```

JSON Compared With XML

- As we have seen in the lectures on XML, the development of applications to parse and process XML documents requires an advanced skill set because processing is not straight forward
- With JSON, parsing and processing is more straight forward because JSON format is more human-readable, and access to data is achieved using the dot notation and array subscripts

Parsing XML To JSON

- As JSON is the preferred format for transmitting data around the Web, yet there is still much legacy data stored in XML, there needs to be a way to retrieve XML data and simply parse the data to JSON
- There are a number of XML to JSON parsers available
  - By far the most common use SAX (eg: `xml2js`, `xml2json`, and `xmldoc`); fewer examples use DOM (eg: `xml-objtree` and `libxmljs-dom`)

Parsing XML To JSON: DOM

- Firstly, a DOM example (install library):
  - `npm install xml-objtree`

```
var ObjTree = require('xml-objtree');
var objTree = new ObjTree();
var xml = [
  '<?xml version="1.0" encoding="UTF-8"?>',
  '<book>',
      '<name>Papillion</name>',
      '<edition>first</edition>',
      '<author>Henri Charriere</author>',
  '</book>'
].join('\n');
var obj = objTree.parseXML(xml);
console.log(JSON.stringify(obj));
```

Parsing XML To JSON: SAX

- We will concentrate on the simple SAX parser
  - `npm install xml2js`

```
var fs = require('fs'), // File System Module
var xml2js = require('xml2js'); // XML2JS Module
var parser = new xml2js.Parser(); // Create parser object

// Asynchronously read and parse the file
fs.readFile('./test.xml', function(error, data){
   if (error) throw error; // skip handling for now
   parser.parseString(data, function (err, result){
      if (err) throw err;
       console.log(result);
   });
});
```

Parsing XML To JSON: SAX

- Another example:

```
var fs = require('fs'), // File System Module
var xml2js = require('xml2js'); // XML2JS Module
var inputFile = "books.xml"; // XML file
var parser = new xml2js.Parser(); // Create parser object

// Synchronously read
var xmlData = fs.readFileSync(inputFile, 'utf8');

// Parse the file
parser.parseString(xmlData, function(error, result){
   if (error) throw error;
   var s = JSON.stringify(result);
   console.log( "Result" + "\n", s, "\n" );
});
```

Parsing XML To JSON: SAX

- When XML elements contain attributes, the previous code version creates an attribute object named `$` on the parent, then assigns the values as children of the attribute object
- To place attributes as direct children of the parent object (without the intervening `$` object), we can set the `mergeAttrs` option to `true` when declaring the parser:

```
var parser = new xml2js.Parser(
                { mergeAttrs: true }
);
```

Parsing XML To JSON: SAX

- The module `xml-to-json-promise` is a promise-supported wrapper around the `xml2js` library
  - This module makes it easy to convert XML data and files to the JSON format
  - To install:
    - `npm install xml-to-json-promise`

Parsing XML To JSON: SAX

- To convert raw XML **data** to JSON, we can use the function:

```
xmlDataToJSON(xml, [options])
```

- The function requires the `xml` (as a string), and the optional `options` are the `xml2js` options you want to use when parsing to JSON
- The function returns a promise with the `json` data

Parsing XML To JSON: SAX

```
convert = require('xml-to-json-promise');

// convert an xml file to json
convert.xmlFileToJSON('xmlfile.xml').then(json => {
   console.log(json);
});

// convert raw xml data to json
convert.xmlDataToJSON('<book>Help</book>').then(json => {
   console.log(json);
});
```

Parsing XML To JSON: SAVE TO JSON FILE

- Here is an example for saving your XML **file** to a JSON file using `xml-to-json-promise`

```
var convert = require('xml-to-json-promise');
var fs = require('fs');

convert.xmlFileToJSON('xmlfile.xml').then( json => {
   fs.writeFile('file.json', JSON.stringify(json),
                     error => {
      if (error) { throw error };
      console.log('file saved!');
   });
});
```

JSON With AJAX

- AJAX is Asynchronous JavaScript and XML
- It is used on the client side as a group of inter-related web development techniques in order to create asynchronous Web applications
- Using the AJAX model, Web applications can asynchronously send and retrieve data from a server without interfering with the display or behaviour of an existing page

JSON With AJAX

- Many developers use JSON to pass AJAX updates between client and server
- A note in relation to terminology:
  - AJAX = Asynchronous JavaScript and XML
  - AJAJ = Asynchronous JavaScript and JSON
  - Because AJAX was well established prior to the adoption of AJAJ, and AJAJ (as an acronym) is a bit awkward to pronounce, the term AJAX is commonly used to refer to AJAJ

JSON With AJAX

- Websites updating live sports scores can be considered examples of AJAX usage
- If these scores have to be updated on the website, then they must be stored on the server so that the webpage can retrieve the score when it is required
- This is where we can more easily make use of JSON formatted data, rather than XML formatted data

JSON With AJAX

- Any data that is updated using AJAX can be stored using JSON format on a web server
- AJAX is used so JavaScript can retrieve JSON files when necessary, parse them, and then perform one of the following two tasks:
  - Store parsed values in variables for processing before displaying them on the webpage
  - Directly assign data to DOM elements in the webpage, so that it gets displayed on the website

Example: JSON With AJAX

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
   if (this.readyState == 4 && this.status == 200){
      var myObj=JSON.parse(this.responseText);
      document.getElementById("demo").innerHTML
                                    = myObj.name;
   }
};
xmlhttp.open("GET", "json_demo.txt", true);
xmlhttp.send();
```

JSON With AJAX

- There is example code on `ceto` (in this weeks tutorial) in the file: `jsonajax.html`
- It provides another demonstration of JSON with AJAX
- You should attempt to understand the code:
  - As you have done AJAX in the pre-requisite unit ICT286, you should be able to get a feel for it
  - Hint: loading the function `loadJSON()` is used asynchronously to upload JSON data
  - It has a link to a data file `data.json` on the tutorialspoint website

Research Future

- Note: we have only given a brief coverage of the fundamentals of JSON in this lecture
- We will cover JSON Schema in the next set of slides
- Recall from the Unit Objectives in week 1, that this unit will require you to research independently to discover how to utilize technologies that we cover briefly: JSON processing is one such instance (in particular, parsing XML to JSON can be handy!)

References

- JavaScript JSON Cookbook, Ray Rischpater. Packt publishing.
- Beginning JSON, Ben Smith. Apress.
- JSON: JavaScript Object Notation, Tutorials Point
  - http://www.tutorialspoint.com/
- JSON QUICK GUIDE, Tutorials Point
  - http://www.tutorialspoint.com/json/json_quick_guide.htm

JSON Schema

Lecture 8 (B)

Learning Objectives

- Fundamentals of JavaScript Schema
- Look at how JSON Schema can be used to validate the structure and data types of JSON documents

JSON Schema

- JSON Schema is:
  - A specification for the JSON-based format for defining structure of JSON data
  - Written in JSON
  - Not a computer program, but data in a declarative format for “describing the structure of other data”
  - Written under Internet Engineering Task Force (IETF). Draft 2020-12 was published on 1 February 2021.

JSON Schema

- JSON Schema:
  - Describes your existing JSON data format
  - Provides clear, human-readable, and machine-readable documentation
  - Provides complete structural validation for
    - Automated testing
    - Validating client-submitted data

JSON Schema Validation Libraries

- There are several validators currently available for different programming languages
- Currently the most complete and compliant JSON Schema validator available is JSV

JSON Schema Validation Libraries

Language	Libraries
JavaScript	JSV; json-schema; schema.js; Orderly (BSD); Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0).
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
C	WJElement (LGPLv3)

JSON Schema Keywords

- The table on the next slide outlines the "keywords" that are available for validation of a JSON file based on a schema
  - When you see an example, you will notice that the keywords are in fact object properties
- A complete list of keywords, which can be used in defining JSON schema, is available at:
  - http://json-schema.org

JSON Schema Keywords

Keywords	Description
$schema	The $schema keyword states that this schema is written according to the draft v4 specification.
title	You will use this to g ive a title to your schema
description	A little description of the schema
type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.
minimum	This is the constraint to be put on the value and represents minimum acceptable value.
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
maximum	This is the constraint to be put on the value and represents maximum acceptable value.
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid ag ainst "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	The leng th of a string instance is defined as the maximum number of its characters.
minLength	The leng th of a string instance is defined as the minimum number of its characters.
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

JSON Schema Example

- A basic JSON Schema defining a product catalogue description:

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "title": "Product",
   "description": "Product catalogue",
   "type": "object",
   "properties": {
      "id": {
         "description": "Product identifier",
         "type": "integer"
      },
```

JSON Schema Example

```
    "name": {
      "description": "Name of the product",
      "type": "string"
    },
    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
  },
  "required": ["id", "name", "price"]
}
```

Validating With Our Schema

- This JSON can be validated via our schema:

```
[
      {
         "id": 2,
         "name": "An ice sculpture",
         "price": 12.50
      },
      {
         "id": 3,
         "name": "A blue mouse",
         "price": 25.50
      }
]
```

`type` Keyword

- The `type` keyword is the most common in a JSON Schema, and is used to restrict values to a specific data type
- The `type` keyword is used as follows:

```
{ "type": "string" }
```

- Correct value: “short”
- Incorrect value: 2 or { "name": "Kevin" } or true

`$schema` Keyword

- The `$schema` keyword is used to declare that a JSON fragment is actually a piece of JSON Schema
- It also declares which version of the JSON Schema standard that the schema was written to comply with

`$schema` Keyword

- It is strongly recommended that all JSON Schema have a `$schema` entry, which must be at the root of the schema
- The `$schema` keyword is used as follows:

```
"$schema": http://json-schema.org/schema#
```

- The statement above declares that your schema was written to comply with the latest version of the JSON Schema standard

`$schema` Keyword

- You can declare that your schema was written to comply with a specific version of the JSON Schema standard, by using one of the following pre-defined values:
  - JSON Schema written to comply with the current version of the specification
    - http://json-schema.org/schema#

`$schema` Keyword

- JSON Schema written to comply with this version (i.e. the schema being described in the current document)
  - http://json-schema.org/draft-07/schema#

`$schema` Keyword

- If you have extended the JSON Schema language to include **your own custom keywords** for validating values, you can use a custom URI for `$schema`
- However, it **must not** be one of the pre-defined examples seen above

Combining Schema

- Combining schema may be as simple as allowing a value to be validated against multiple criteria at the same time OR combining schemas from multiple files or JSON trees
- JSON Schema includes four keywords for combining schema:
  - `allOf`
  - `anyOf`
  - `one of`
  - `not`

Combining Schema: `allOf`

- `allOf` must be valid against **all** of the sub-schema (analogous to AND; the schema is true if and only if **all** sub-schema are true)

```
{
   "allOf": [
      { "type": "string" },
      { "maxLength": 5 }
   ]
}
```

- Correct value: “short”
- Incorrect value: “too long”

Murdoch UNIVERSITY

Combining Schema: `allOf`

- In the previous example, the first sub-schema requires a string and the second sub-schema requires that the string be a maximum of 5 characters in length
- As long as a value validates against **both** of these sub-schema, it is considered valid against the combined schema

Combining Schema: `allOf`

- Be careful not to create schema that are logically impossible:

```
{
   "allOf": [
      { "type": "string" },
      { "type": "number" }
   ]
}
```

- This schema will not validate against any value, since a value can not be both a string and a number at the same time

Combining Schema: `anyOf`

- `anyOf` can be valid against **any** of the sub-schema (analogous to OR; the schema is true if one or more sub-schema are true)

```
{
   "anyOf": [
      { "type": "string", "maxLength": 5 },
      { "type": "number", "minimum": 0 }
   ]
}
```

- Correct values: “short”, “hi”, 4, or 5
- Incorrect values: “too long”, -1

Murdoch UNIVERSITY

Combining Schema: `anyOf`

- In the previous example, the first sub-schema allows a string with maximum length 5 and the second sub-schema allows a number with a minimum value of 0
- As long as a value validates against **either** of these sub-schema, it is considered valid against the combined schema

Combining Schema: `anyOf`

```
{
   "anyOf": [
      { "type": "string" },
      { "type": "number" }
   ]
}
```

- Correct value: “short one”, 45
- Incorrect value: { "name": "Kevin" }
- This schema will validate against any string or any number, as either can be valid
- It will not validate against an object

Murdoch UNIVERSITY

Combining Schema: `oneOf`

- `oneOf` must be valid against **exactly one** of the sub-schema

```
{
   "oneOf": [
      { "type": "number", "multipleOf": 5 },
      { "type": "number", "multipleOf": 3 }
   ]
}
```

- correct values: 10, 9
- Incorrect value: 2 (not valid with either); 15 or 90 (valid with both)

Combining Schema: `not`

- `not` must **not** be valid against the given schema

```
{
   "not": { "type": "string" }
}
```

- Correct values: { "key": "value" } or 15
- Incorrect value: “a string”

Combining Schema: `not`

- `not` does not strictly combine schema, but it belongs here because it modifies the effect of schema in some way
- The `not` keyword declares that a instance validates if it **does not** validate against the given sub-schema

Combining Schema

- All of these keywords (except `not`) must be set to an array, where each item is a sub-schema
- It is important to note that the schema listed in an `allOf`, `anyOf`, or `oneOf` array know nothing of one another

Regular Expressions

- The `pattern` and `Pattern Properties` keywords use regular expressions to express constraints
- The regular expression syntax used is from JavaScript (specifically ECMA 262)
- However, that complete syntax is not widely supported, therefore it is recommended that you stick to the subset of that syntax described on the next slide

Regular Expressions

Syntax	Description
^	Matches only at the beginning of the string.
$	Matches only at the end of the string.
(...)	Group a series of regular expressions into a single regular expression.
|	Matches either the regular expression preceding or following the | symbol.
[abc]	Matches any of the characters inside the square brackets.
[a-z]	Matches the range of characters.
[^abc]	Matches any character not listed.
[^a-z]	Matches any character outside of the range.
+	Matches one or more repetitions of the preceding regular expression.
*	Matches zero or more repetitions of the preceding regular expression.
?	Matches zero or one repetitions of the preceding regular expression.
+?, *?, ??	The *, +, and ? qualifiers are all greedy; they match as much text as possible. Sometimes this behavior is not desired and you want to match as few characters as possible.
{x}	Match exactly x occurrences of the preceding regular expression.
{x,y}	Match at least x and at most y occurrences of the preceding regular expression.
{x,}	Match x occurrences or more of the preceding regular expression.
{x}?, {x,y}?, {x,}?	Lazy versions of the above expressions.

Murdoch UNIVERSITY

Structuring A Complex Schema

- When writing computer programs of even moderate complexity, it is advisable to structure the design by making it modular
  - This provides re-usable functionality, reduces duplication, and makes the program more portable for use by a wider audience
- In JSON Schema, for any but the most trivial schema, it is really useful to structure the schema into parts that can be re-used in a number of places

Structuring A Complex Schema

- Object to re-use:

```
{
   "address": {
      "type": "object",
      "properties": {
         "street_address": { "type": "string" },
         "city":           { "type": "string" },
         "state":          { "type": "string" }
      },
      "required": ["street_address", "city", "state"]
   }
}
```

Structuring A Complex Schema

- Because we want to be able to re-use our schema, it is typical (but not required) to put it in a parent schema under a `property` called `definitions`:

Structuring A Complex Schema

```
{
   "definitions": {
      "address": {
         "type": "object",
         "properties": {
            "street_address": { "type": "string" },
            "city":           { "type": "string" },
            "state":          { "type": "string" }
         },
         "required": ["street_address","city","state"]
      }
   }
}
```

Structuring A Complex Schema

- The schema can then be referred to from elsewhere using the `$ref` keyword
- The value of `$ref` is a string in a format called JSON Pointer
  - i.e. `$ref` is logically replaced with the object that it points to
- To refer to our schema, we would include:

```
{ "$ref": "#/definitions/address" }
```

Structuring A Complex Schema

- The hash symbol (`#`) refers to the current document, and the forward slash (`/`) separates `properties`, allowing traversal of the properties in the document
- In our example `"#/definitions/address"` means:
  1. go to the root of the document
  2. find the value of property "`definitions`"
  3. within that object, find the value of the property "`address`"

Murdoch UNIVERSITY

Structuring A Complex Schema

- `$ref` can be a relative or absolute URI, so if you prefer to include your definitions in separate files, you can also do that
- For example, below we load the address schema from another file `definitions.json` which resides in the same directory as the current one :

```
{ "$ref": "definitions.json#/address" }
```

Structuring A Complex Schema

- To create our `address` schema which allows a customer to create a valid JSON document:

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "definitions": {
      "address": {
         "type": "object",
         "properties": {
            "street_address": { "type": "string" },
            "city":           { "type": "string" },
            "state":          { "type": "string" }
         },
         "required": ["street_address", "city", "state"]
      }
   },
```

Structuring A Complex Schema

```
    "type": "object",
    "properties": {
       "bill_address": { "$ref": "#/definitions/address" },
       "ship_address": { "$ref": "#/definitions/address" }
    }
}
```

Structuring A Complex Schema

- The JSON object below would be valid according to our schema:

```
{
   "ship_address": {
      "street_address": "16 Pennsylvania Avenue NW",
      "city": "Washington",
      "state": "DC"
   },
   "bill_address": {
      "street_address": "1st Street SE",
      "city": "Washington",
      "state": "DC"
   }
}
```

JSON Schema: `id` Property

- The `id` property serves two purposes:
  1. It declares a unique identifier for the schema
  2. It declares a base URL, against which `$ref` URLs can be resolved
- It is best practice that `id` is a URL, preferably in a domain that you control
  - For example, if you own the **`foo.bar`** domain, and you had a schema for addresses, you may set its `id` as follows:

```
"id": "http://foo.bar/schemas/address.json"
```

JSON Schema: `id` Property

- This provides a unique identifier for the schema, as well as, in most cases, indicating where it may be downloaded
- But be aware of the second purpose of the `id` property: "to declare a base URL for relative `$ref` URLs elsewhere in the file"

JSON Schema: `id` Property

- For example, if you had:

```
{ "$ref": "person.json" }
```

in the same file, a JSON schema validation library would fetch `person.json` from `http://foo.bar/schemas/person.json` even if `address.json` was loaded from the local filesystem

References

- Understanding JSON Schema, Release 1.0, Michael Droettboom. Space Telescope Science Institute.


Murdoch
UNIVERSITY
jQuery
Basics
Lecture 8 (C)

Lecture Objective/Outline

- Relevance to assessments:
  - jQuery can provide very convenient usage client-side, which may be beneficial for the assignment
- Why introduce jQuery for this unit?
- Introduction to the jQuery
  - jQuery basic language features
  - jQuery functions
- How to get up to speed with jQuery

Introduction

- jQuery:
  - Is a client-side JavaScript library released in 2006 by John Resig
  - Takes common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code
  - Also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation

Introduction

- jQuery library brings together the following set of functionalities:
  - HTML/DOM navigation and manipulation
  - CSS manipulation
  - Event handling methods
  - HTML effects and animations
  - Developing with AJAX
  - Other utilities

jQuery Usage

- The two simplest ways to start using jQuery on your web site are:
  1. Download the jQuery library from jQuery.com
     - This will require downloading the latest updates when they are released

  OR

  2. Include jQuery from a Content Delivery Network (CDN), such as Google
     - The latter is the recommended usage, as the latest updates are always readily available

jQuery Download

- The following steps are required for the download option:
  1. Download the latest jQuery from https://jquery.com/download/
     - The jQuery library is a single JavaScript file called `jQuery-(version).js`
     - Place the downloaded file in the same directory as the pages where you wish to use it
     - You reference it with the HTML `<script>` tag (the `<script>` tag should be placed inside the `<head>` section of your html page)

jQuery Download

2. Link to the downloaded .js file in the `script` tag with the `src` attribute

```
<script type="text/javascript" src="jquery-3.2.1.js">
</script>
```

   - Note: the above `script` tag adds the function **`jQuery()`** to the global namespace
   - The most common usage of the function `jQuery()` is its alias **`$()`**
   - In fact, the function `jQuery()` is rarely used or seen

3. Write your code

jQuery Content Delivery Network

- If you don't want to download and host jQuery yourself, the simplest and recommended method is to include a link to it from a Content Delivery Network (CDN), such as Google
  - The reference to the `<script>` tag should be inside the `<head>` tags

```
<head>
  <script src="https://ajax.googleapis.com/
    ajax/libs/jquery/3.2.1/jquery.min.js">
  </script>
</head>
```

jQuery: Concept

- The central concept behind jQuery is:
  - “find something, do something”
  - For example, select DOM elements from an HTML document and then do something with them using jQuery methods

jQuery: Example

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <script src="…"> <!-- include src CDN address -->
      </script>
   </head>
   <body>
      <!-- the following jQuery statement will change -->
      <a href=""></a>
      <!-- to: <a href="http://www.jquery.com">jQuery</a> -->
      <script>
         jQuery('a').text('jQuery').
                     attr('href'. 'http://www.jquery.com');
      </script>
   </body>
</html>
```

Murdoch UNIVERSITY

jQuery: Example

- The jQuery `text` method inserts the text "jQuery" between the anchor tags
- The jQuery `attr` method sets the `href` attribute to the jQuery Web site
- In order to run the code, save it to a HTML file and insert the CDN url below into the `src` attribute in the first `script` tag in the head section (i.e. replace the …)
  - https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js

jQuery: Another Example

- We can also use jQuery to create new DOM elements and then do something with these
- In the example on the next slide, we create the anchor element (which is not an element of the original DOM)
- Then append it to the DOM
  - This is done with the jQuery `appendTo` method

jQuery: Another Example

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <script src="…"> <!-- include src CDN address -->
      </script>
   </head>
   <body>
      <!- Note: no anchor tag -->
      <script>
         jQuery('<a>jQuery</a>').
                attr('href', 'http://www.jquery.com').
                appendTo('body');
      </script>
   </body>
</html>
```

jQuery APIs

- jQuery Application Programmer Interfaces:
  - jQuery core
  - Selectors
  - Attributes
  - Traversal
  - Manipulation
  - CSS
  - Events
  - Effects
  - Ajax
  - Utilities
  - jQuery User Interface

jQuery API: Core

- jQuery core functions:
  - $(expression, [context])
    - Eg: `$('input: radio', document.form[0]);`
  - $(html)
    - Eg: `$('<div id="load">Loading… </div>');`
  - $(elements)
    - Eg: `$(document.body).css('background', 'red');`
  - $(callback)
    - Eg: `$(function() { alert("test"); } )`

jQuery API: Selectors

- Selectors:
  - jQuery supports nearly all CSS selectors from CSS 1 through 3
  - Always use the jQuery alias **`$()`**, no matter which type of selector you use
- Types of Selectors:
  - Basic – element, #id, class, .classA.classB
    - Eg: `$('p'),$('#id'),$('.class'),$('.classA.classB')`
  - Hierarchy – ancestor, descendent, parent > child, prev + next
    - Eg: `$('form input'),$('#main > *'),$('label + input')`

jQuery API: Selectors

- Form

Selectors	Matched Elements
:input	input, select, textarea and button elements
:text, :radio, :checkbox, :image, :submit, :reset, :password, :file	input element and attribute that is equal to the specified selectors
:button	button element, input element with type "button"

jQuery API: Selectors

- Basic filters
  - :first, :last, :not(selector), :even, :odd, :eq(index), :gt(index), :lt(index), :header, :animated
- Attribute filters
  - [attribute], [attribute!=value], [attribute^=value], [attribute$=value], [attribute*=value], [filter1][filter2]
  - Select elements having specified attribute, where:
    - ^= value begins exactly with a given string
    - != does not contain given value
    - $= value ends exactly with a given string
    - *= contains a given substring

jQuery API: Attributes

- Attributes:
  - attr:
    - attr(name), attr(properties), attr(key,value), removeAttr(name)
  - class:
    - addClass(class), removeClass(class), toggleClass(class)
  - html: html(), html(value)
  - text: text(), text(value)
  - value: val(), val(value)

jQuery API: Events

- Events:
  - Page load: ready(fn)
  - Event handling:
    - bind(type, fn), unbind(type, fn), trigger(event)
  - Event helpers:
    - click(), click(fn), mousedown(fn), mouseout(fn), …

jQuery API: Effects

- Effects:
  - Basics:
    - show(), show(speed), hide(), toggle(), toggle(speed)
  - Fading:
    - fadeIn(speed), fadeOut(speed), fadeTo(speed, opacity)

jQuery API: Ajax

- Ajax Request:
  - $.ajax(options); options is a set of `key:value` pairs
  - $.get(url, [data], [callback], [type])
  - $.post(url, [data], [callback], [type])
  - $.getJSON(url, [data], [callback], [type])
  - $.getScript(url, [callback])

jQuery Syntax

- As mentioned earlier, the jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s)
- Basic syntax is:

  `$(selector).action()`

  - The `$()` alias is used to define/access the `jQuery()` method
  - A `selector` is used to "query (or find)" HTML elements
  - A jQuery `action()` method is used to be perform some task on the "selected" element(s)

jQuery: Example

```
// hides all <p> elements
$("p").hide()
// hides the element with id="test"
$("#test").hide()
```

- Refer jQuerySelectors.xls for a complete list of jQuery selectors

jQuery: `ready` Event

- It is good practice to wait for the DOM to be fully "loaded and ready" before working on it
- Code that manipulates the DOM can run in a handler for this event
- This handler is the DOM `ready` event

```
// DOM Ready Event
$(document).ready( function() {
   // methods go here...
});
```

jQuery: `ready` Event

- This handler prevents any jQuery code from running before the DOM is finished loading
- The event is typically placed in the **head section**, before the body of the document
- Below is a variant of this method call, that is much more succinct:

```
// DOM Ready Event
$( function() {
   // methods go here...
});
```

jQuery: `ready` Event Examples

```
// hides all <p> elements when button is clicked
$( function() {  // Document Ready Event
    $("button").click( function() {
        $("p").hide();
    });
});

// hides element with id="test" when button is clicked
$( function() {  // Document Ready Event
    $("button").click( function() {
        $("#test").hide();
    });
});
```

jQuery: `ready` Event Examples

```
// hides element with class="test" when button clicked
$( function() {  // Document Ready Event
    $("button").click( function() {
        $(".test").hide();
    });
});
```

- You can attach as many `ready` events to the document as you like
- They are executed in the order they are added

jQuery: `load` Event

- `ready` will execute once the DOM is loaded, but before the window loads
- So we do not have to wait for the `window.onload` to manipulate the DOM
- However, sometimes we might want to wait for `window.onload` event, so that code is executed once the entire page (including all assets) is completely loaded

jQuery: `load` Event

- This is done by attaching a `load` event handler to the `window` object
- The `load` method can invoke a function once the `window` is completely loaded

```
// Window Load Event
$(window).load( function() {
   // methods go here...
});
```

jQuery: `ready` Event Re-visited

- Most jQuery usage will attempt to manipulate the DOM, which is why we need the `ready` event
- OR do we??
- Nowadays (with modern browsers) this is not entirely necessary
  - We can simply place our jQuery code before the closing body tag i.e. `</body>`
  - This ensures that the DOM is completely loaded as the document will be parsed from top to bottom

jQuery: `ready` Event Re-visited

```
<!DOCTYPE html>
<html lang="en">
   <head><title>Ready Event</title></head>
   <body>
      <p>This demonstrates ready without the method</p>
      <script src="https://ajax.googleapis.com/
                  ajax/libs/jquery/3.2.1/jquery.min.js">
      </script>
      <script>alert($("p").text());</script>
   </body>
</html>
```

jQuery: External Files

- Like JavaScript, you can put your jQuery functions in a separate `.js` file
- This is encouraged, **as is modular design**
- To use jQuery functions from a separate file, use the `src` attribute to refer to the `.js` file:

```
<head>
   <script src="https://ajax.googleapis.com/
                ajax/libs/jquery/3.2.1/jquery.min.js">
   </script>
   <script src="my_jQuery_functions.js"></script>
</head>
```

jQuery Event Methods

- Most DOM events have an equivalent jQuery method
- Refer to jQueryEvents.xls for a list of jQuery methods that correspond to DOM events
- For example, to assign a click event to all paragraphs on a page, you can do the following:

```
$("p").click();
```

jQuery Event Methods

- However, we must assign an appropriate action for when an event is triggered
- This is done by passing a function to the event:

```
$("p").click( function() {
    // action (event handler) goes here!
});
```

jQuery Event Methods

- Eg: When a click event fires on a <p> element, we can hide the current <p> element:

```
$("p").click( function() {
    $(this).hide();
});
```

jQuery Event Methods

- Eg: The 1st function below is executed when the mouse enters the HTML element `("id=#p1")`, and the 2nd function is executed when the mouse leaves that HTML element:

```
$("#p1").hover( function() {
    alert("You entered p1!");
  }, // note: comma separating the functions
  function() {
    alert("Bye! You now leave p1!");
});
```

The `on()` Method

- The `on()` method attaches one or more event handlers for the selected elements
- Eg: Attach a click event to a `<p>` element:

```
$("p").on("click", function() {
    $(this).hide();
});
```

The `on()` Method

- Eg: Attach multiple event handlers to a `<p>` element:

```
$("p").on({
    mouseenter: function(){
      $(this).css("background-color", "gray");
    },
    mouseleave: function(){
      $(this).css("background-color", "blue");
    },
    click: function(){
      $(this).css("background-color", "red");
    }
});
```

jQuery Effects

- You can apply various effects to HTML elements with the following methods:
  - Hide, Show, Toggle (between Hide and Show), Slide, Fade, and Animate (allows you to manipulate ALL CSS properties), Stop (effect)
- You should research the exact syntax required for these methods

jQuery Callback Functions

- JavaScript statements are executed line by line
- However, with effects such as those just mentioned, the next line of code can be run even though the effect is **not** finished
- This can create errors!!

jQuery Callback Functions

- To prevent these types of errors, you can create a callback function, which is executed after the current effect is 100% finished
- Typical syntax:

```
$(selector).effect(speed, callback);
```

jQuery Callback Functions

- To demonstrate how an error may occur, the example below has *no callback parameter*, so the alert box will be displayed before the hide effect is completed:

```
$("button").click( function() {
   $("p").hide(1000);
   alert("Paragraph is now hidden");
});
```

jQuery Callback Functions

- To demonstrate how to prevent the error, the example below *has a callback parameter* to the `hide` method that will be executed only after the hide effect is completed:

```
$("button").click( function() {
   $("p").hide("slow", function() {
     alert("Paragraph is now hidden");
   });
});
```

jQuery: Chaining

- jQuery allows us to run multiple jQuery methods (on the same element) within a single statement
- This is achieved by *chaining* together actions/methods

jQuery: Chaining

- To chain an action, you simply append the action to the previous action using the 'dot' notation
- The following example chains together the `css()`, `slideUp()`, and `slideDown()` methods
- The "p1" element first changes to red, then it slides up, and then it slides down

```
$("#p1").css("color","red").slideUp(2000).slideDown(2000);
```

jQuery: Chaining

- When chaining, the line of code could become quite long
- jQuery is not very strict on layout
  - You can format it like you want, including line breaks and indentations
- Eg:

```
$("#p1").css("color", "red")
        .slideUp(2000)
        .slideDown(2000);
```

jQuery: DOM Manipulation

- jQuery comes with many DOM related methods that make it easy to access and manipulate DOM elements and attributes
- To **get content**, there are three useful jQuery methods for DOM manipulation:
  - `text()` – Returns the text content of selected elements
  - `html()` – Returns the content of selected elements (including HTML markup)
  - `val()` – Returns the value of form fields

jQuery: DOM Manipulation

- The following examples demonstrate how to get content from `id #btn1` and `#test` with the jQuery `text()` and `html()` methods:

```
$("#btn1").click( function() {
   alert("Text: " + $("#test").text());
});

$("#btn2").click( function() {
   alert("HTML: " + $("#test").html());
});
```

jQuery: DOM Manipulation

- The following example demonstrates how to get the value of a form input field (`id #btn1` and `#test`) with the jQuery `val()` method:

```
$("#btn1").click( function() {
   alert("Value: " + $("#test").val());
});
```

jQuery: DOM Manipulation

- The same three useful jQuery methods for DOM manipulation exist to **set content**:
  - `text()` – Sets/modifies the text content of selected elements
  - `html()` – Sets/modifies the content of selected elements (including HTML markup)
  - `val()` – Sets/modifies the value of form fields

jQuery: DOM Manipulation

- The following example demonstrates how to set content with the jQuery `text()` and `html()` methods:

```
$("#btn1").click( function() {
   $("#test1").text("Hello World"));
});

$("#btn2").click( function() {
   $("#test2").html("<b>Hello World</b>"));
});
```

jQuery: DOM Manipulation

- The following example demonstrates how to set the value of a form input field with the jQuery `val()` method:

```
$("#btn1").click( function() {
   $("#test3").val("Minnie Mouse"));
});
```

jQuery: DOM Manipulation

- The jQuery `attr()` method can be used to get attribute values
- The following example demonstrates how to get the value of the `href` attribute in a link:

```
$("button").click( function() {
   alert($("#w3s").attr("href"));
});
```

- For a complete list of all jQuery HTML methods, refer to jQuery-html-css-methods.xls

jQuery: DOM Manipulation

- The jQuery `attr()` method can also be used to set/modify attribute values
- The following example demonstrates how to set the value of the `href` attribute to a link:

```
$("button").click( function() {
   alert($("#w3s").attr("href",
     "https://www.w3schools.com/jquery/"));
});
```

jQuery: DOM Manipulation

- The `attr()` method also allows setting multiple attributes at the same time
- The following demonstrates how to set the `href` and `title` attributes at the same time:

```
$("button").click( function() {
  $("#w3s").attr( { "href" :
    "https://www.w3schools.com/jquery/",
      "title" : "W3Schools jQuery Tutorial"
  });
});
```

jQuery: DOM Manipulation

- All of the three jQuery methods: `text()`, `html()`, and `val()`, also come with a callback function option
- The callback function has two parameters:
  1. The index of the current element in the list of elements selected, and
  2. The original (old) value
- You then return the string you wish to use as the new value from the function

jQuery: DOM Manipulation

- The following two examples demonstrate `text()` and `html()` with a callback function:

```
$("#btn1").click( function() {
   $("#test1").text( function(i, origText){
      return "Old text: " + origText +
             " New text: Hello world!
             (index: " + i + ")";
   });
});
```

jQuery: DOM Manipulation

```
$("#btn2").click( function() {
   $("#test2").html( function(i, origText) {
        return "Old html: " + origText +
               " New html: Hello <b>world!</b>
               (index: " + i + ")";
   });
});
```

jQuery: DOM Manipulation

- The jQuery method `attr()`, also comes with a callback function
- The callback function has two parameters:
  1. The index of the current element in the list of elements selected, and
  2. The original (old) attribute value
- You then return the string you wish to use as the new attribute value from the function

jQuery: DOM Manipulation

- The following example demonstrates `attr()` with a callback function:

```
$("button").click( function() {
   $("#w3s").attr("href", function(i, origAttr)
   {
      return origAttr + "/jquery/";
   });
});
```

jQuery: Add Elements/Contents

- The following jQuery methods are used to add *new HTML content*:
  - `append()` – Inserts content at the end of the selected elements
  - `prepend()` – Inserts content at the beginning of the selected elements
  - `after()` – Inserts content after the selected elements
  - `before()` – Inserts content before the selected elements

jQuery: Add Elements/Contents

- The `append()` and `prepend()` methods can also be used to add *new HTML elements*
  - They can take an infinite number of new elements as parameters
  - The new elements can be generated with text/HTML with jQuery, or with JavaScript code and DOM elements
  - `append()` adds the new elements and text to the end of the page body
  - `prepend()` adds to the beginning of the page body

jQuery: Add Elements/Contents

- Eg: create 3 new elements (eg: paragraphs):

```
function appendText() {
   // Create element with HTML
   var txt1 = "<p>Text1.</p>";
   // Create with jQuery
   var txt2 = $("<p></p>").text("Text2.");
   // Create with DOM
   var txt3 = document.createElement("p");
   txt3.innerHTML = "Text3.";
   // Append the 3 new elements
   $("body").append(txt1, txt2, txt3);
}
```

jQuery: Remove Elements/Contents

- To remove elements and content, there are mainly two jQuery methods:
  - `remove()` – Removes the selected element **AND** its child elements
  - `empty()` – Removes the child elements **FROM** the selected element(s)

jQuery: Remove Elements/Contents

- jQuery `remove()` Method

```
$("#div1").remove();
```

  - N.B. `#div1` AND any contents are gone!
- jQuery `empty()` Method

```
$("#div1").empty();
```

  - N.B. `#div1` still exists; however, the contents of `#div1` are gone
- Refer w3schools for examples

jQuery: Manipulating CSS

- jQuery has several methods for CSS manipulation:
  - `addClass()` – Adds one or more classes to the selected elements
  - `removeClass()` – Removes one or more classes from the selected elements
  - `toggleClass()` – Toggles between adding/removing classes from the selected elements
  - `css()` – Sets or returns the style attribute
  - Refer w3schools for examples

jQuery: Manipulating CSS

- To return the value of a specified CSS property, use the following syntax:

```
css("propertyname");
```

- To set a specified CSS property, use the following syntax:

```
css("propertyname","value");
```

- To set multiple CSS properties, use the following syntax:

```
css({"propertyname":"value",...})
```

jQuery: Dimensions

- jQuery has several important methods for working with dimensions:
  - `width()` – sets or returns the width of an element
  - `height()` – sets/returns the height of an element
  - `innerWidth()`
  - `innerHeight()`
  - `outerWidth()`
  - `outerHeight()`
  - Refer w3schools for examples

jQuery: Traversing

- jQuery traversing methods are used to "find" (or select) HTML elements based on their relation to other elements
- Starting with one selection and moving through that selection until you reach the elements you desire
- Think of DOM as composed of elements in a hierarchical tree structure

jQuery: Traversing

- You can easily move up (ancestors), down (descendants) and sideways (siblings) within the tree structure, starting from the selected (current) element
  - An ancestor is a parent, grandparent, great-grandparent, and so on
  - A descendant is a child, grandchild, great-grandchild, and so on
  - Sibling share the same parent

jQuery: Traversing

- Three useful jQuery methods for traversing up the DOM tree are:
  - `parent()` – returns the direct parent element of the selected element; i.e. a single step up the tree
  - `parents()` – returns all ancestor elements of the selected element; i.e. all the way up to the root element of the document tree
  - `parentsUntil()` – returns all ancestor elements between the selected element and a given argument

jQuery: Traversing

- Two useful jQuery methods for traversing down the DOM tree are:
  - `children()` – returns all direct children of each occurrence of the selected element; i.e. a single step down the tree
    - You can filter the search with an optional parameter
  - `find()` – returns all descendant elements of the selected element all the way down to the last descendant
- Refer w3schools for other methods/examples

jQuery: Traversing

- There are many useful jQuery methods for traversing sideways within the DOM tree:
  - `siblings()` – returns all sibling elements of the selected element
    - You can filter the search with an optional parameter
  - `next()`, `nextAll()`, `nextUntil()`
  - `prev()`, `prevAll()`, `prevUntil()`
- Refer w3schools for examples related to traversing using sibling methods

jQuery: Filter Methods

- The most basic filtering methods are:
  - `first()`, `last()` and `eq()`
  - These allow you to select a specific element based on its position in a group of elements
- Other filtering methods:
  - `filter()` and `not()`
  - These allow you to select elements that match, or do not match, a certain criteria

jQuery: Filter Methods

- The `first()` method returns the first element of the specified elements
- The `last()` method returns the last element of the specified elements
- The `eq()` method returns an element with a specific index number of the selected elements
  - The index numbers start at 0

jQuery: Filter Methods

- The `filter()` method lets you specify a criteria
  - Elements that do not match the criteria are removed from the selection, and those that match will be returned
- The `not()` method returns all elements that do not match the criteria
  - Logically, this specifies the opposite of `filter()`

jQuery

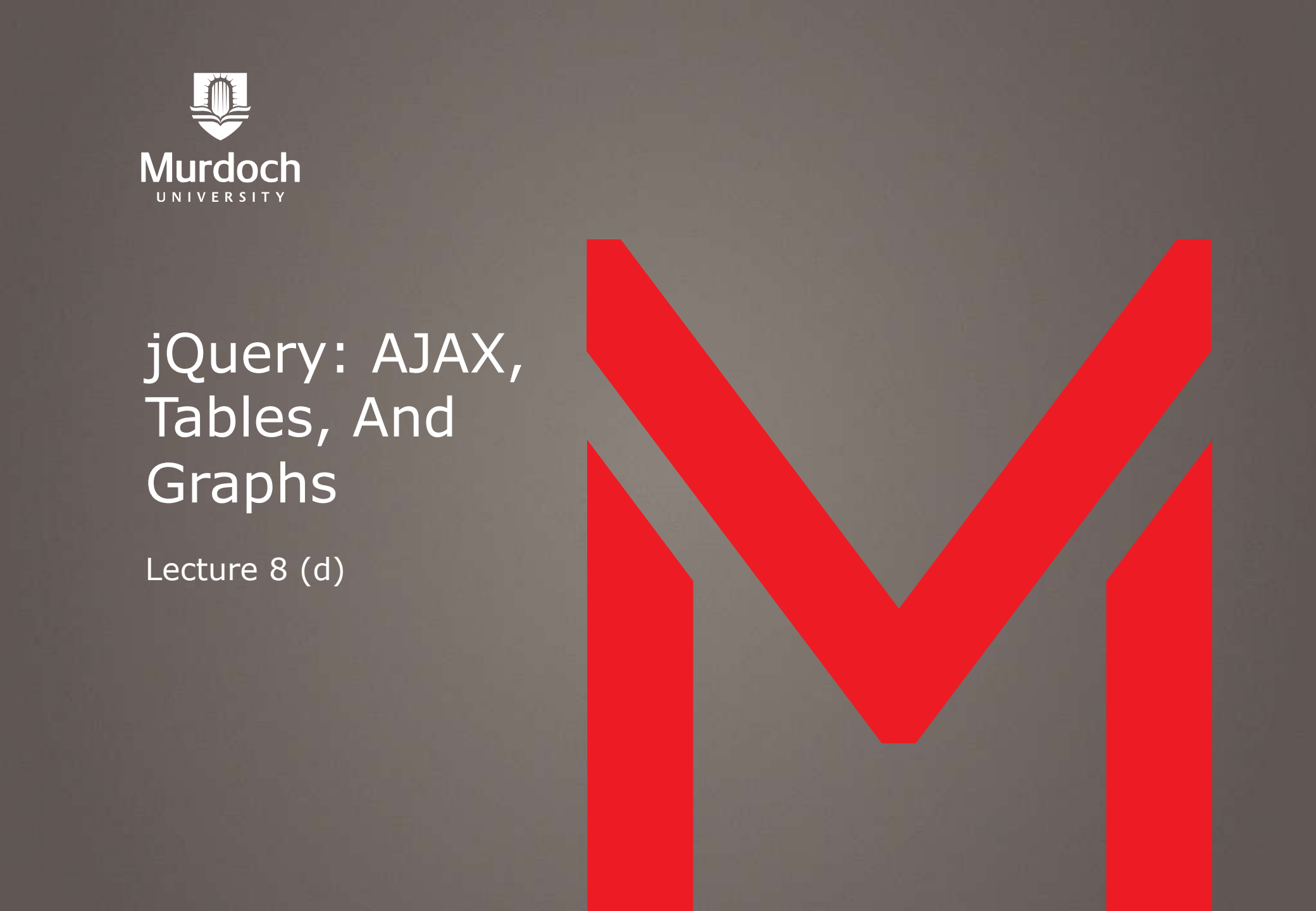
- It must be emphasized that usage of jQuery is **not** compulsory in this unit
  - It is presented in this lecture for your edification
  - There are no exercises set for jQuery in the tutorial for this topic
  - However, you encouraged to learn and use jQuery for your second assignment; you will find it easier to develop an application using jQuery client-side
- IMPORTANTLY, there will be no questions on jQuery in the final examination

Further Reading

- This lecture does NOT cover the jQuery comprehensively
- You should utilize any of the materials suggested in the next slide
- Visit the jQuery homepage for useful materials, and visit one of the online tutorials suggested

References

- w3schools tutorial online:
  - https://www.w3schools.com/Jquery/default.asp
- jQuery Succinctly, 2012. Lindley, C.
  - Available for free from www.syncfusion.com
- Learning jQuery, Fourth Edition 2013. Jonathan Chaffer and Karl Swedberg
- jQuery in Action, Third Edition 2015. Bear Bibeault, Yehuda Katz, and Aurelio De Rosa



jQuery: AJAX, Tables, And Graphs

Lecture 8 (d)

Lecture Objectives

- Relevance to unit objectives:
  - Learning objective 2: Writing software
- Relevance to assessments:
  - jQuery can provide very convenient client-side usage
  - It could prove most beneficial in your second assignment for AJAX usage and the required tabular and graphical presentation of output

jQuery: AJAX

- As ICT286 is a pre-requisite unit for this unit, you should be conversant with, and know how to use AJAX, in client/server communication
- If you are vague about the details (in particular the usage), you should review the lecture material and lab work from last semester
  - AJAX usage is required for the second assignment

jQuery: AJAX

- jQuery provides several methods to facilitate AJAX functionality
- With the jQuery AJAX methods, you can request text, HTML, XML, or JSON from a remote servers using both HTTP GET and POST methods
- You can load the external data directly into the selected HTML elements of your web page!

jQuery: AJAX

- The jQuery `ajax()` function is the lowest level of abstraction available for `XMLHttpRequests`
  - It provides greater flexibility and functionality than the other available AJAX functions
  - In fact, the AJAX functions listed below leverage the `ajax()` function:
    - `load()`
    - `get()`
    - `post()`
    - `getJSON()`
    - `getScript()`

jQuery: AJAX

- These other functions, which can be considered shortcuts of the `ajax()` function, are handy for individual tasks that do not require the full features of `ajax()`
- When you do require the full features and customizations that jQuery offers for AJAX, you should use `ajax()`
- N.B. `ajax()` and `load()` both use the GET HTTP method by default

jQuery: `ajax()` Method

- An AJAX Request Example:

```
$.ajax(
  {
     url:  "process.php",
     type: "POST",
     data: "class=6470&name=Tim",
     success: function(msg){
        alert("Data:" + msg);
     }
  }
);
```

jQuery: AJAX `load()` Method

- The `load()` method loads data from a server and puts the returned data into the selected HTML element
- Syntax:

```
$(selector).load(URL, data, callback);
```

  - The required URL parameter specifies the URL you wish to load
  - The optional callback parameter is the name of a function to be executed after the `load()` method is completed

jQuery: AJAX `load()` Example

- Given the contents of a file `"demo_test.txt"` below:

```
<h2>jQuery and AJAX is FUN!!!</h2>

<p id="p1">Some text in a paragraph.</p>
```

- The following code loads the content of the file `demo_test.txt` into a specific `<div>` element:

```
$("#div1").load("demo_test.txt");
```

jQuery: AJAX `load()` Example

- It is also possible to add a jQuery selector to the URL parameter
- The following example loads the content of the element with `id="p1"`, inside the file `demo_test.txt`, into a specific `<div>` element:

```
$("#div1").load("demo_test.txt #p1");
```

jQuery: AJAX `load()` Method

- The optional callback parameter specifies a callback function to run when the `load()` method is completed
- The callback function can have the following parameters:
  - `responseTxt` - contains the resulting content if the AJAX call succeeds
  - `statusTxt` - contains the status of the AJAX call
  - `xhr` - contains the `XMLHttpRequest` object

jQuery: AJAX `load()` Method

- The following example displays an alert box after the `load()` method completes
- If the `load()` method has succeeded, it displays:

  `"External content loaded successfully!"`
- If it fails, it displays an error message with the `status` and `statusTxt`

jQuery: AJAX `load()` Method

```
$("button").click( function() {
   $("#div1").load("demo_test.txt",
      function(responseTxt, statusTxt, xhr) {
         if(statusTxt == "success")
            alert("External content loaded
                                successfully!");
         if(statusTxt == "error")
            alert("Error: " + xhr.status + ": "
                                + xhr.statusTxt);
   });
});
```

Murdoch UNIVERSITY

jQuery: AJAX `get()` and `post()` Methods

- The `$.get()` method requests data from the server using the HTTP GET request
- Syntax:

```
$.get(URL, callback);
```

  - The required URL parameter specifies the URL of the resource you wish to request
  - The optional callback parameter is the function to be executed if the request succeeds

jQuery: AJAX `get()` and `post()` Methods

- The `$.post()` method requests data from the server using the HTTP POST request
- Syntax:

```
$.post(URL, data, callback);
```

  - The required URL parameter specifies the URL of the resource you wish to request
  - The optional data parameter specifies some data to be send along with the POST request
  - The optional callback parameter is the function to be executed if the request succeeds

jQuery: AJAX `get()` and `post()` Methods

- In the following example, the first parameter of `$.get()` is the URL we wish to request
- The second parameter is a callback function
  - The first callback parameter holds the returned content of the page we requested
  - The second callback parameter holds the status of the request
- N.B. the fictional script `demo_get.php` performs some hyperthetical processing server-side

jQuery: AJAX `get()` and `post()` Methods

```
$("button").click( function() {
   $.get("demo_get.php", function(data, status){
      // display returned data and status
      alert("Data: "+data+" Status: "+status);
   });
});
```

jQuery: AJAX `get()` and `post()` Methods

- In the following example, the first parameter of `$.post()` is the URL we wish to request
- The second parameter is some data to send along with the request (a JSON object)
- The third parameter is a callback function
  - The first callback parameter holds the returned content of the page we requested
  - The second callback parameter holds the status of the request

jQuery: AJAX
`get()` and `post()` Methods

```
$("button").click( function() {
   $.post("demo_post.php",
      {
          name: "Luke Skywalker",
          city: "Spacecity"
      },
      function(data, status) {
          alert("Data: "+data+"Status: "+status);
      });
});
```

jQuery API: AJAX

- AJAX Events:
- `ajaxComplete(callback), ajaxStart(callback), ajaxStop(callback), ajaxSend(callback), ajaxError(callback), ajaxSuccess(callback)`
- Eg:

```
$('div id="loading">Loading...</div>')
    .insertBefore("#images")
    .ajaxStart( function() {
      $(this).show();
    }).ajaxStop( function() {
      $(this).hide();
    }
);
```

jQuery Tables

- jQuery's DOM manipulation and iteration utility methods facilitate manipulation of table cells without having to worry too much about tags
- The following material introduces some basic table operations by way of an example

jQuery Tables

- The next slide shows a function that accepts a container element and multi-dimensional array
- The outer array contains the rows and an inner one holds the columns
- All of the styling is done using CSS generated by the free online CSSTableGenerator (http://csstablegenerator.com/)
  - You can of course use your own CSS file, and **will** be required to do so for the 2nd assignment

jQuery Tables

- This function should be put into a JavaScript:

```
function makeTable(container, data) {
   var table =
      $("<table/>").addClass('CSSTableGenerator');
      $.each(data, function(rowIndex, r) {
         var row = $("<tr/>");
         $.each(r, function(colIndex, c) {
            row.append($("<t"+(rowIndex ==
                  0 ?  "h" : "d")+"/>").text(c));
         });
         table.append(row);
      });
      return container.append(table);
}
```

jQuery Tables

- The `$.each()` function is an easy way to iterate through an array's elements
- The outer one deals with the rows, the inner/nested one creates the column elements
- The `row.append()` call uses `rowIndex` to use table headers (`<th>`) for the first row, instead of the regular cells (`<tr>`)
- Only the "h" and "d" letters set them apart

jQuery Tables

- Embed the previous function in a HTML file
  - `makeTable()` should be called in the `ready` event, so that the DOM has fully loaded
- This table contains 4 rows containing 3 cities

```
$(document).ready(function() {
   var data = [["City 1", "City 2", "City 3"],
               ["New York", "LA", "Seattle"],
               ["Paris", "Milan", "Rome"],
               ["Pittsburg", "Wichita", "Boise"]]
   var cityTable= makeTable($(document.body),data);
});
```

Table Operations: Append Row

- There is no one *method* to append a row to the end of a table because of all the different possible layout options
- If we disregard the rare (and dubious) use of nested tables, we can use the `last` attribute to the `<tr>` tag in the table and the following code to do the job

Table Operations: Append Row

```
function appendTableColumn(table, rowData) {
   var lastRow =
      $('<tr/>').appendTo(table.find('tr:last'));
      $.each(rowData, function(colIndex, c) {
         lastRow.append($('<td/>').text(c));
   });
   return table.append(lastRow);
}
// Usage: add the following line below the call to
//        the function makeTable on slide 25
appendTableColumn(cityTable, ["Calgary", "Ottawa",
                              "Yellowknife"]);
```

Table Operations: Retrieving Contents

- The same logic that worked in creating a table can be used to retrieve contents of table cells
  - In fact, there really isn't a whole lot of difference between the two
- The only caveat is that the `find()` function has to search for both `TH` and `TD` cell types
- `find()` supports multiple selectors, but they have to be supplied via one string argument and separated by commas

Table Operations: Retrieving Contents

```
function getTableData(table) {
    var data = [];
    table.find('tr').each(function(rowIndex,r){
        var cols = [];
        $(this).find('th,td')
                .each(function(colIndex, c){
            cols.push(c.textContent);
        });
        data.push(cols);
    });
    return data;
}
```

Table Operations

- When it comes to working with HTML tables and data on the client-side, JavaScript/jQuery combined provides a convenient approach (as we just seen)
- However, loading table data that has been returned from a server (i.e. via AJAX) requires a more powerful approach
- For that, it is recommended that you utilize the DataTables jQuery plug-in
  - https://datatables.net/

DataTables Plug-in for jQuery

- DataTables is a plug-in for the jQuery JavaScript library
- It is a highly flexible tool, based upon the foundations of progressive enhancement, and will add advanced interaction controls to any HTML table
- You should reference the manual for installation and usage available at:
  - https://datatables.net/manual

DataTables Plug-in for jQuery

- Getting started with DataTables is as simple as including two files in your web-site, the CSS styling and the DataTables script itself
- These two files are available on the DataTables Content Delivery Network (CDN):
  - //cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css
  - //cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js

DataTables Plug-in for jQuery

- There are many libraries that can be used for presentation of table data, but DataTables is a commonly used library that has good support and numerous examples
- There is a DataTables youtube tutorial in the reference list at the end of these slides, to get your started

DataTables Plug-in for jQuery

- It is therefore recommended that you use the DataTables library for your 2nd assignment
- You may of course choose to use another library or approach, but you will need to investigate any library usage for the task required

jQuery Tables: Final Word

- The second assignment requires you to present data (returned from a server) in a table format
- You should spend time now to learn to work with the DataTables plug-in (or another library), that will allow jQuery to easily and nicely display table data
- This will mean investigating and trying out the way to achieve this task, in your own time

jQuery Tables: Graphics

- The second assignment also requires you to present data (returned from a server) in a **line** graph
- You should spend time now to learn to work with a JavaScript or a jQuery plug-in that allows you to easily and nicely display a **line** graph
- This will mean investigating and trying out the ways to achieve this task, in your own time

jQuery Tables: Graphics

- Some references are provided at the end of these slides, to help get you started
- There is a tutorial on Canvas.js, which is a commonly used and well supported library (with numerous examples)
- It is therefore recommended that you use the Canvas.js library for your 2nd assignment
- You may of course choose to use another, but you will need to investigate any library usage for the task required

References (Tables)

- Working with Tables Using jQuery
  - https://www.htmlgoodies.com/beyond/css/working_w_tables_using_jquery.html
- DataTables Table plug-in for jQuery
  - https://datatables.net/
- Example:
  - https://datatables.net/examples/data_sources/js_array
- jQuery Datatables Plugin Tutorial for Beginners (youtube video):
  - https://www.youtube.com/watch?v=sRjWHPv7JLk

References (Graphics)

- Canvas.js library for **line** graph:
  - https://canvasjs.com/jquery-charts/
  - https://canvasjs.com/docs/charts/integration/jquery/chart-types/jquery-line-chart/
  - https://canvasjs.com/docs/charts/basics-of-creating-html5-chart/updating-chart-options/
  - https://canvasjs.com/javascript-charts/dynamic-live-line-chart/
- Tutorial on Creating Charts | CanvasJS JavaScript Charts
  - https://canvasjs.com/docs/charts/basics-of-creating-html5-chart/

References (Graphics)

- jQuery **line** graph using Canvas:
  - https://web.archive.org/web/20130407101311/http://www.worldwidewhat.net/2011/06/draw-a-line-graph-using-html5-canvas/
- Plotly.js (JavaScript graphing library)
  - https://plot.ly/javascript/
  - https://plot.ly/javascript/line-charts/