

framework training
We love technology

Introduction to Modern Web Applications and Team- based SW

Informed Academy

📞 020 3137 3920

🐦 @FrameworkTrain

frameworktraining.co.uk

Introduction Developing Web Application and Team-based Software Development

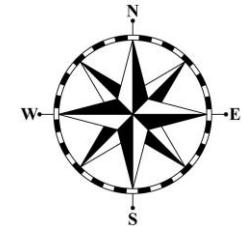


Toby Dussek

Informed Academy



framework training
business value through education

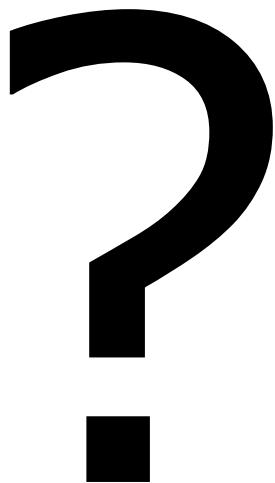


Introduction

□ Course objectives

- Learn core aspects of the HTML, CSS and JavaScript
- On this course we are looking at:
 - The basic JavaScript syntax
 - JavaScript flow of control statements (e.g. if, for statements)
 - JavaScript variables, types and expressions
 - JavaScript Functions
 - Introduction to Object Oriented Programming in JavaScript
 - JavaScript in a web page
 - AJAX and JavaScript as well as jQuery
 - JavaScript Testing and Debugging
- Teams of 2 developing project over 2 days

Questions?



How the Internet & Web Works

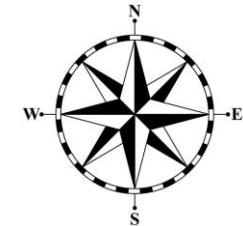


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- How the Internet Works
- How the Web Works
- How Web Applications work
- Web Application Technologies
- What is AJAX?
- Single Page Applications
 - SPA and Ajax
 - JSON
- What are Cookies
- What are Sessions
- Restful Services

It's a Web Wide World!

- The web is the biggest change in computing over the last 30 years
- Changes the nature of our systems

2020 This Is What Happens In An Internet Minute



How the Web Works



□ URL – Universal Resource Locator

- Specifies machine, port, site / page
- response is data rendered that can be rendered by a web browser e.g. Chrome, Safari, IE, Edge



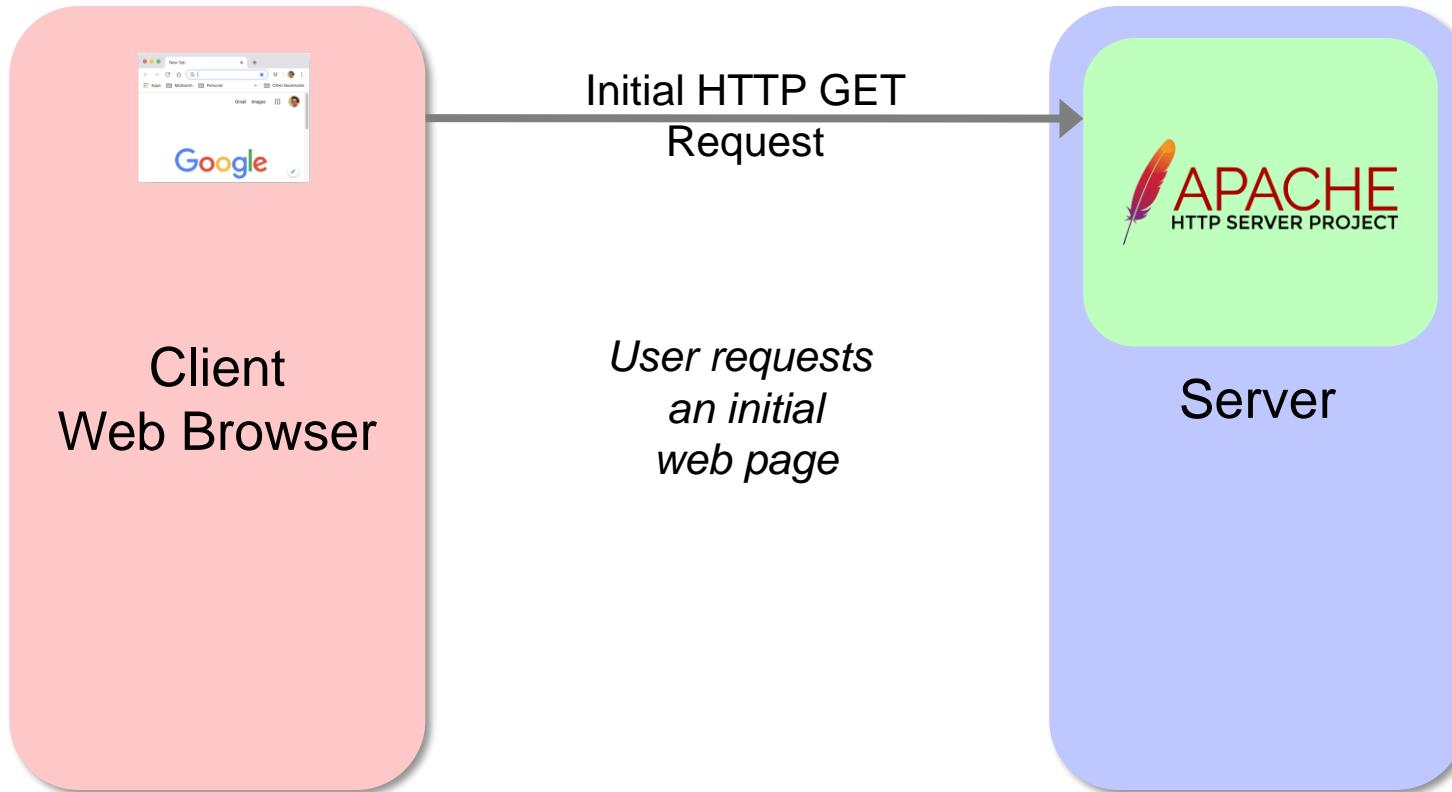
How Web Applications Work



- Web originally designed for static text at CERN
 - now widely used for web applications
 - from shopping carts, to forums, to messaging apps & games
 - all still run on this basic technology
- Today make use of range of technologies to make this happen
 - HTML – used to markup text on a page
 - CSS – to style that text
 - JavaScript – to make pages dynamic
 - HTTP Request Methods – to handle different types of operations
 - URLs – to represent different resources

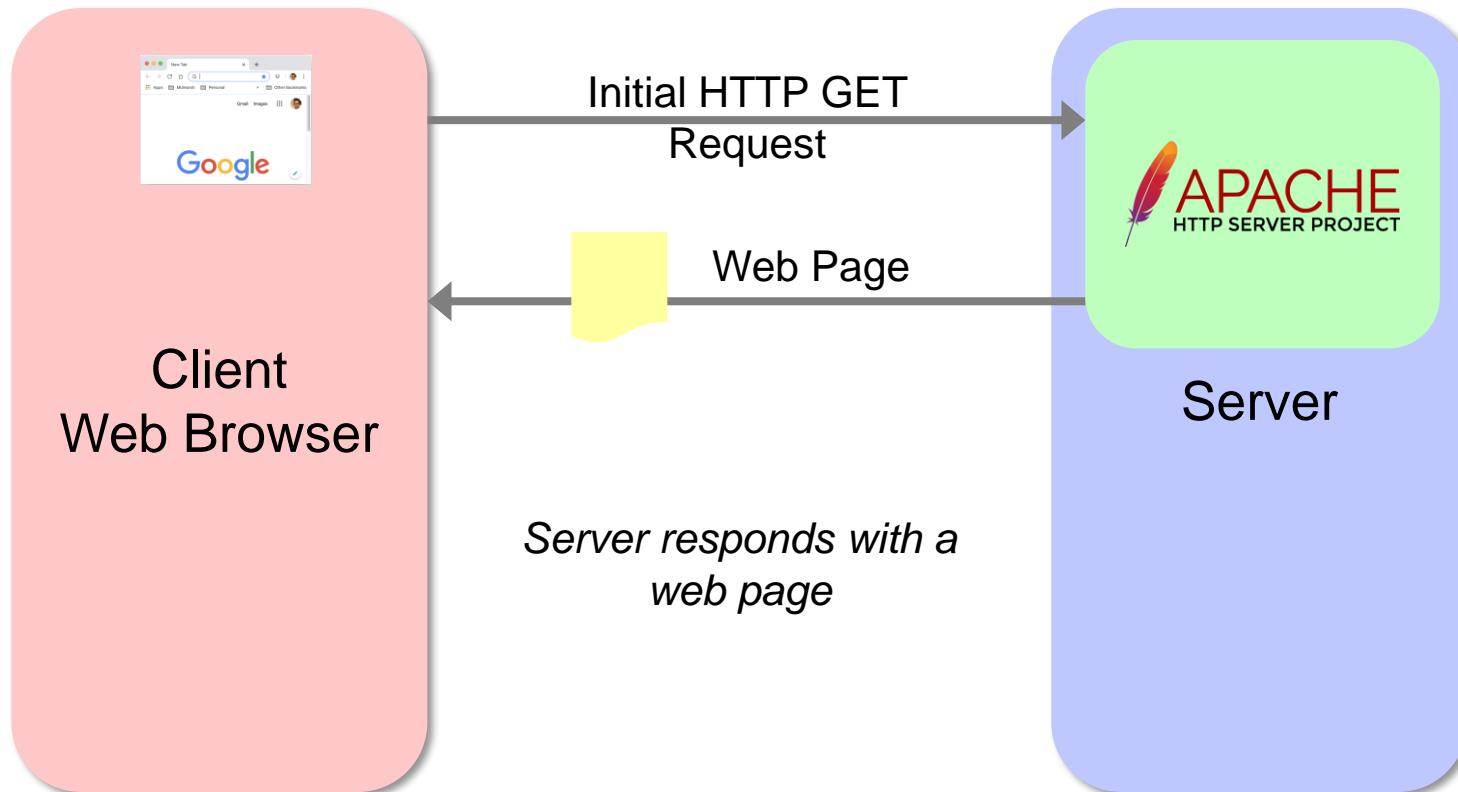


Web Application Technologies





Web Application Technologies





Web Application Technologies

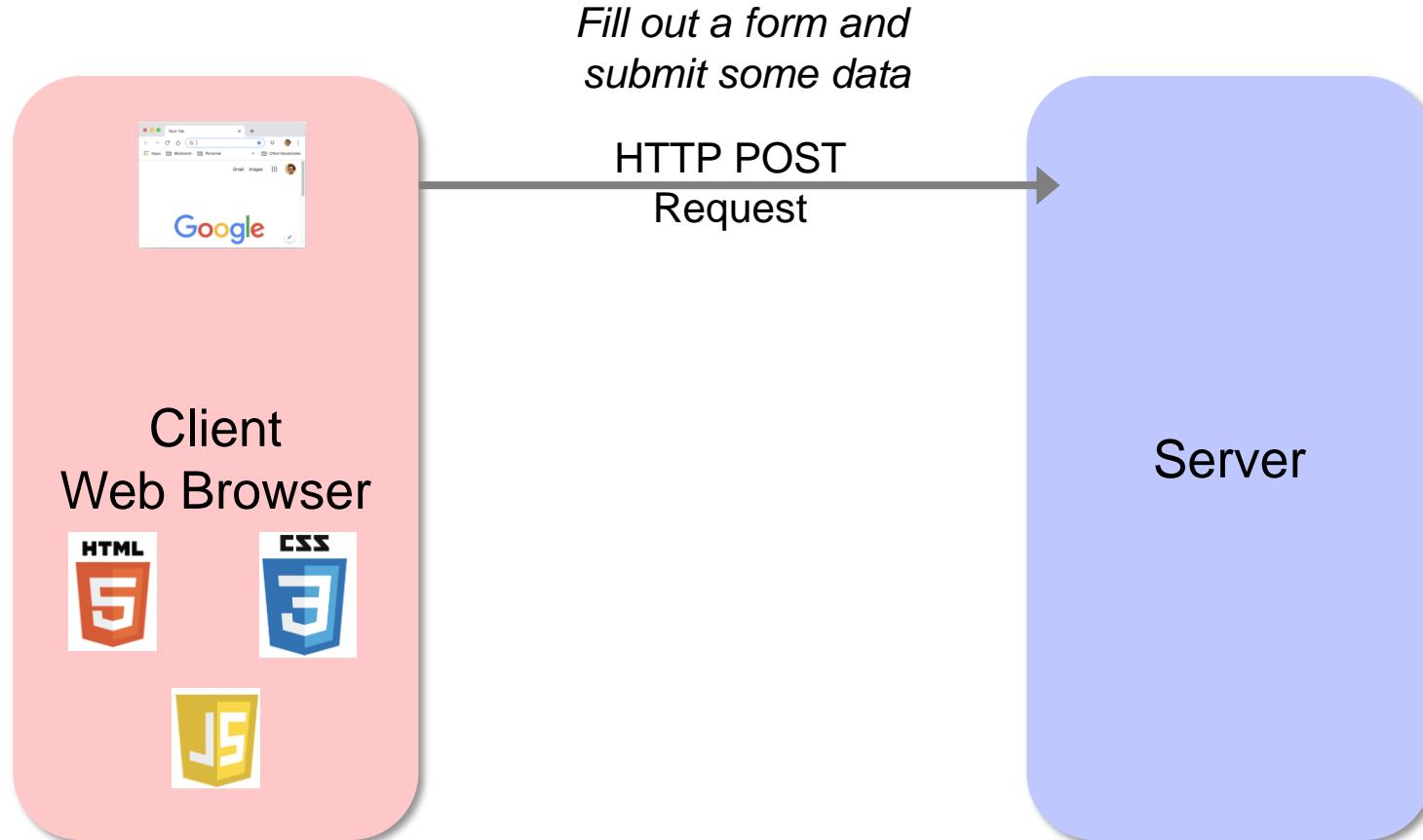


*HTML for text markup
CSS for style
JavaScript to make
page dynamic*



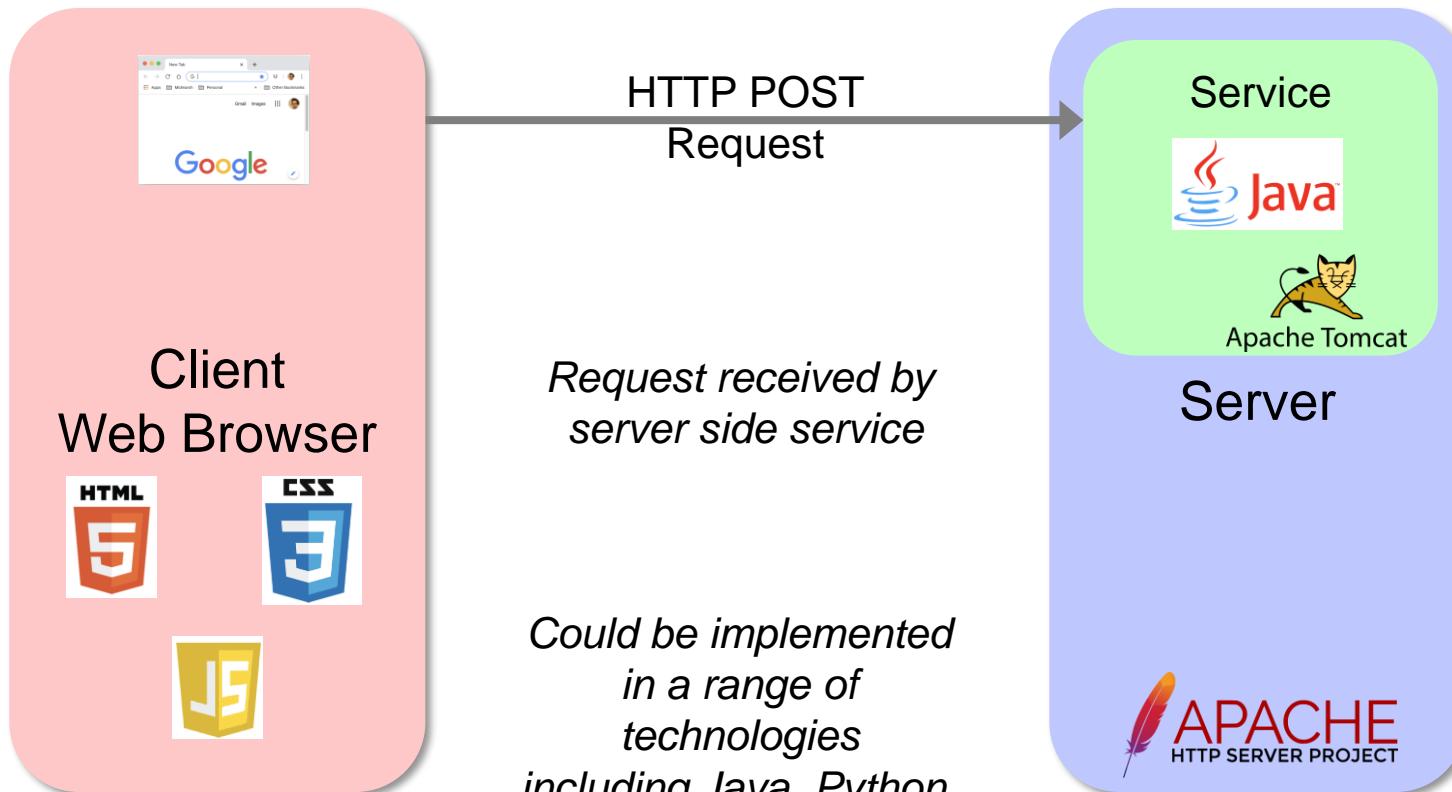


Web Application Technologies



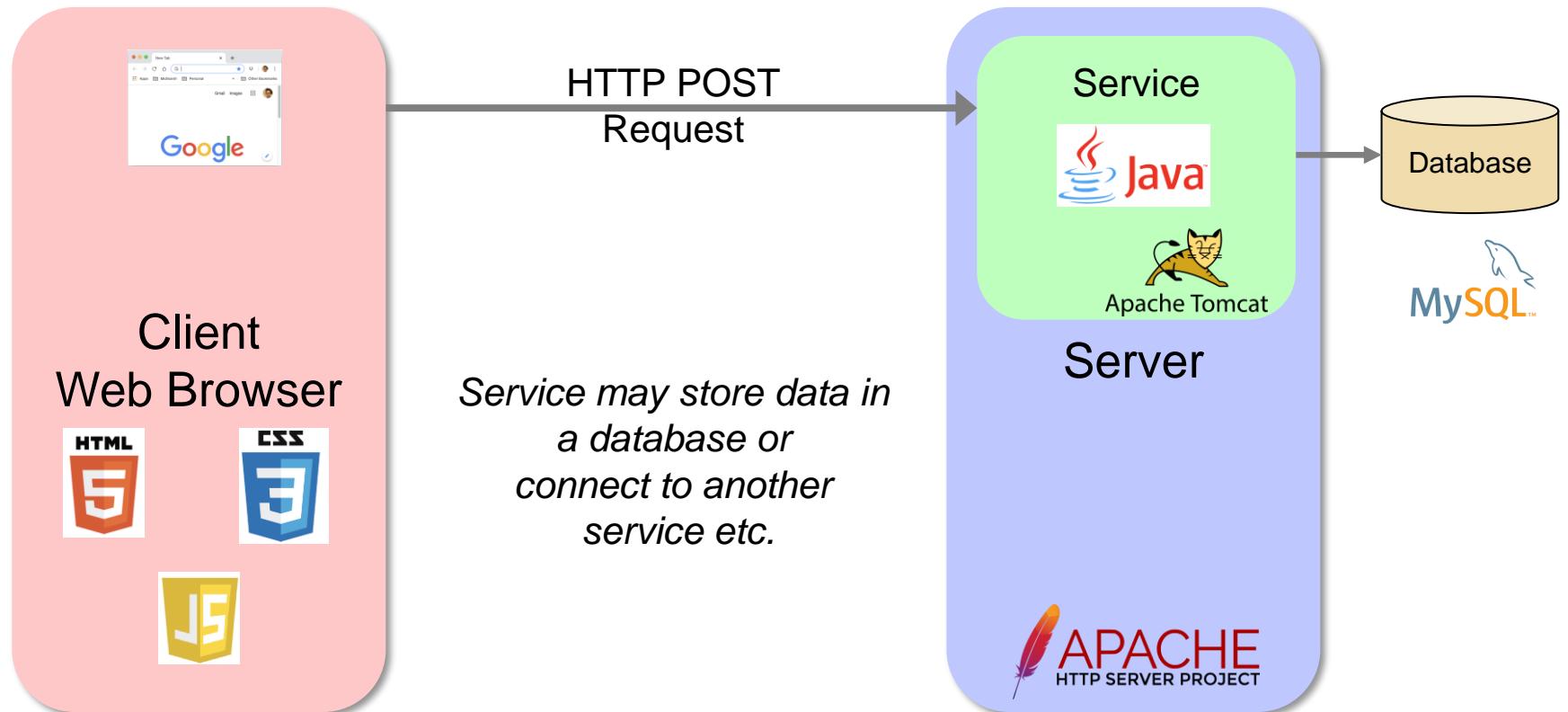


Web Application Technologies



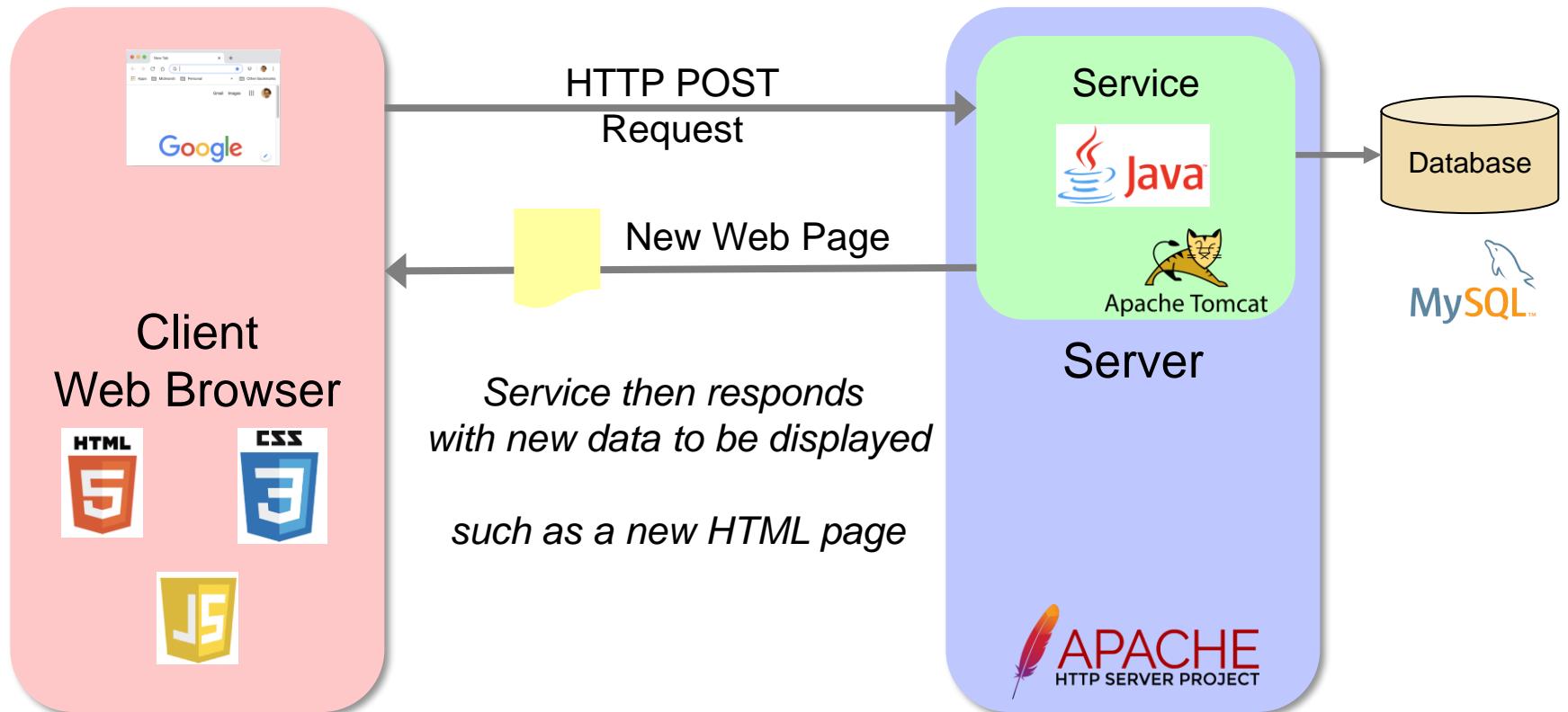


Web Application Technologies





Web Application Technologies





What is AJAX

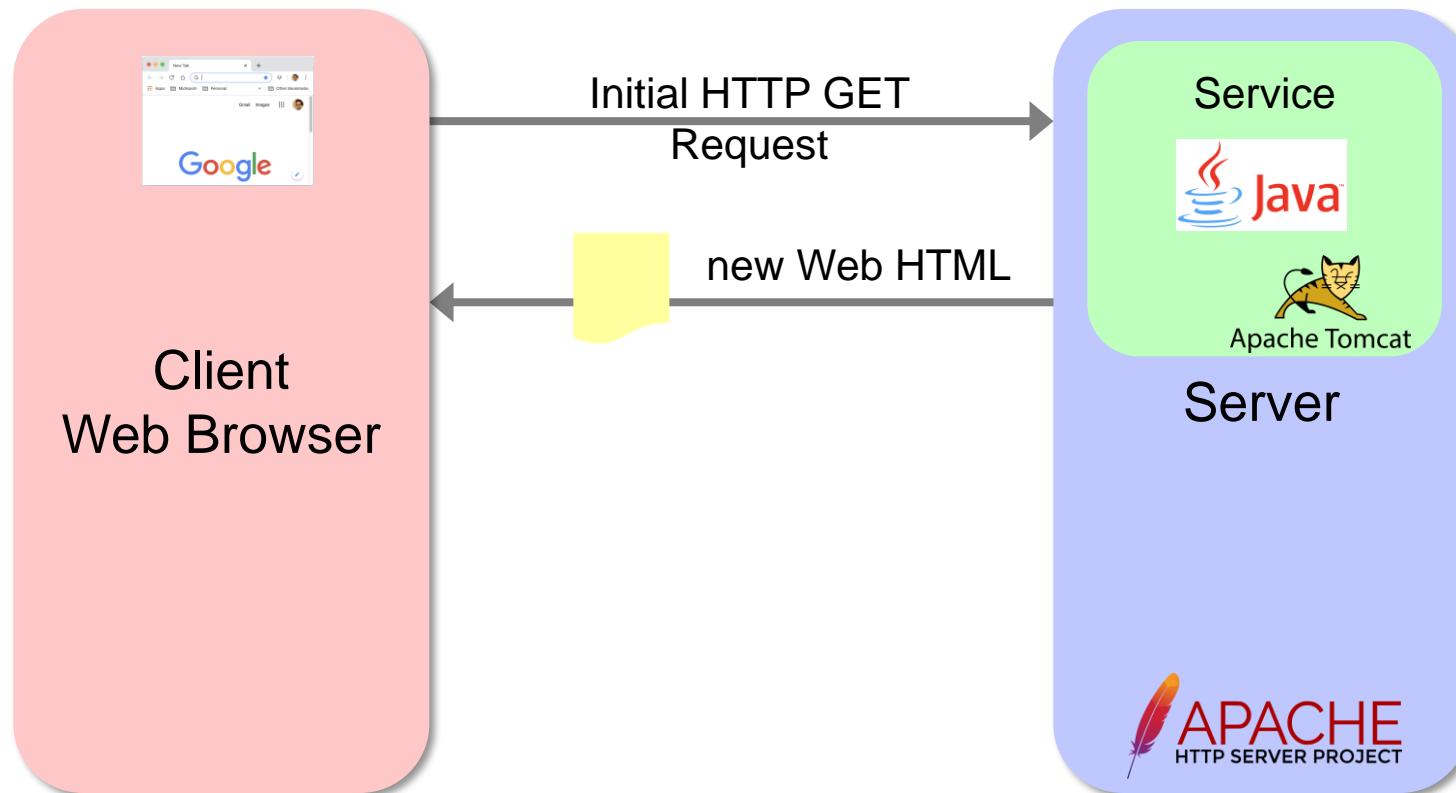
- AJAX is a design pattern, not a specific technology
 - Term coined in 2005
- AJAX is an acronym
 - Asynchronous JavaScript And Xml
 - But (Ajax is *now* a name)
 - May not always be XML, typically its JSON
 - But indicative of the type of system
- JavaScript to handle user interactions
 - JavaScript to *asynchronously* request data from a server
 - JavaScript to handle returned data
 - JavaScript used to manage how data is represented

Single Page Applications

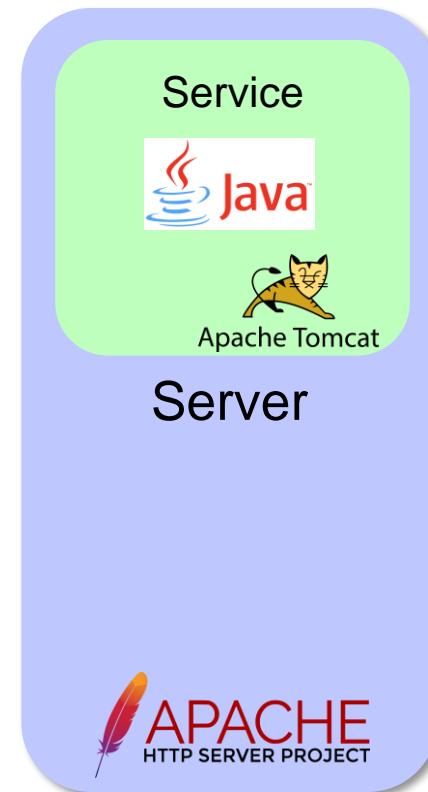


- AKA SPAs
- Ajax core to how SPAs operate
- Response to issue with multi-page sites
 - loading a whole page is slow
 - multiple HTTP requests to server
 - Browser has to draw entire page
 - but pages on site have many common aspects
 - headers, footers, menu bars
- SPAs have a single page
 - hide / display / modify the DOM dynamically

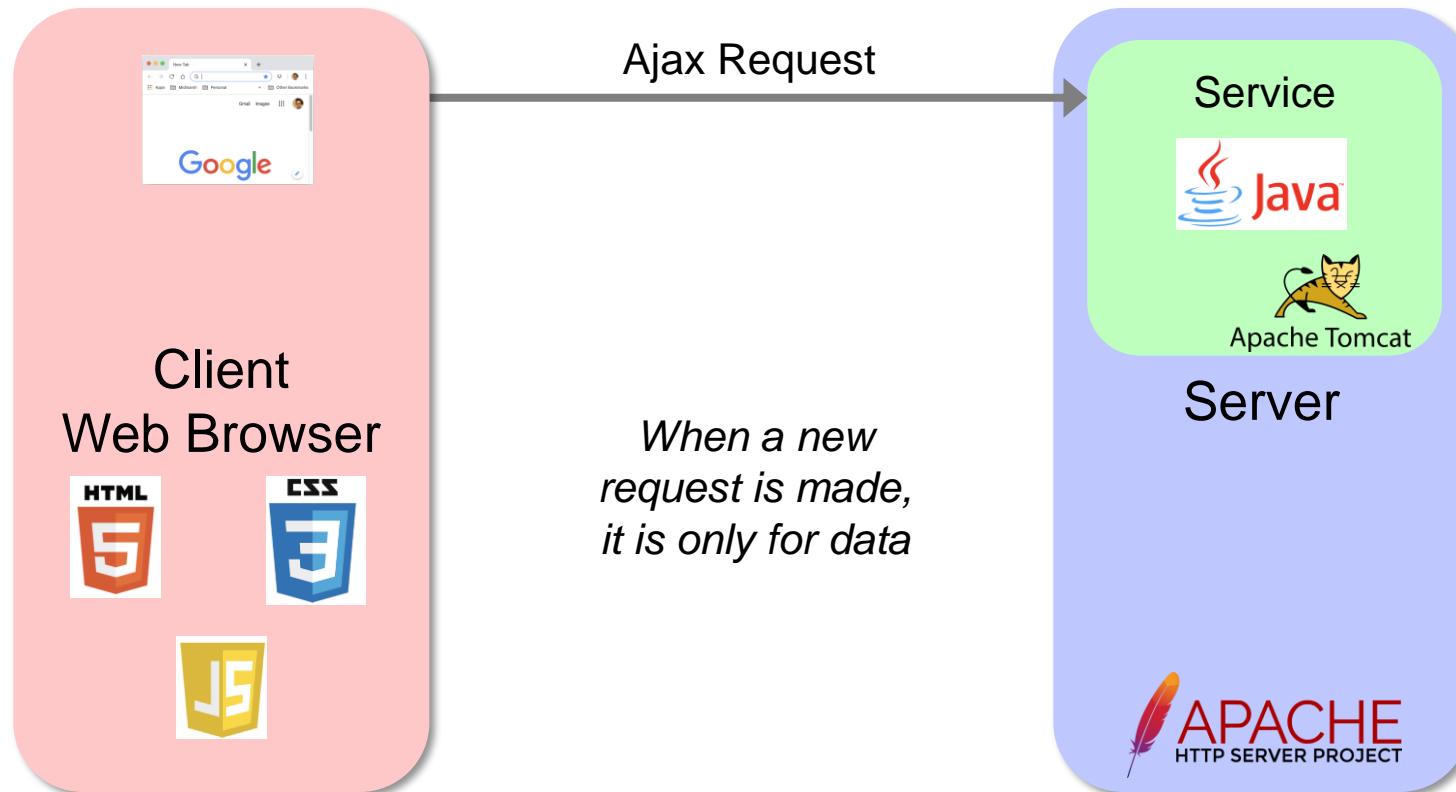
SPA and Ajax



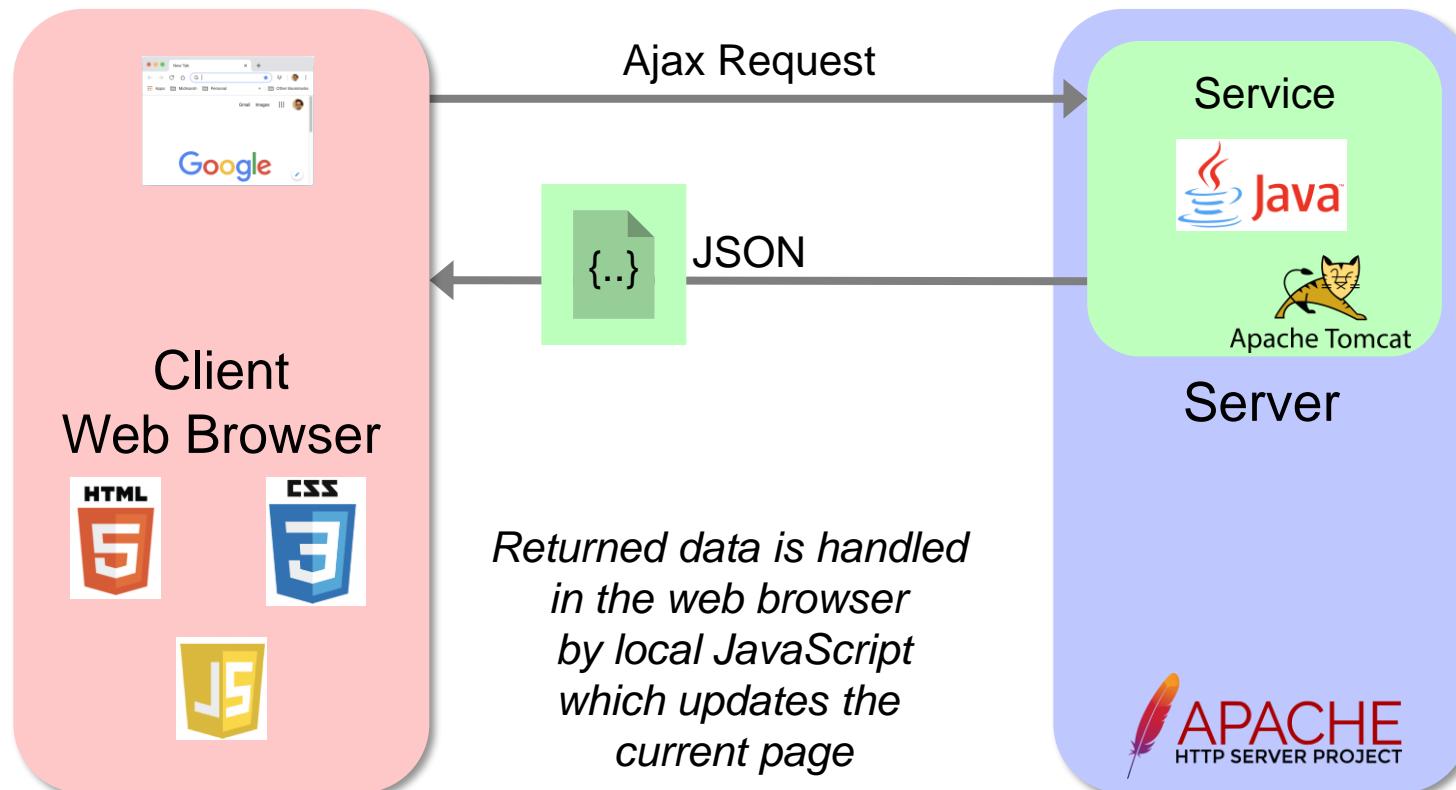
SPA and Ajax



SPA and Ajax



SPA and Ajax



HTML Introduction



Toby Dussek

Informed Academy



framework training
business value through education

Plan for Session

- What is a Mark Up Language?
- What is HTML?
- Basic HTML Page Structure
- HTML Tags
- Common Tags
- Hyperlinks
- Image Links
- Lists
- Tables
- Forms

What is a Mark Up Language?

- A mark up language is one that provides meta data about other data
 - provides information about other information
- Typically intended for human as well as machine consumption
- Lots of examples in computing
 - Tex/LaTex
 - SGML / XML / YAML
 - HTML!

What is HTML?

- Hyper Text Markup Language (HTML)
- Used to describe content in a web page
 - typically for a web browser
 - browser can then render information
 - provides structure to the information in a page
- Consists of series of
 - (typically nested) elements represented by TAGs
- HTML tags label other content as
 - headings, paragraphs, tables, forms, links
- Browser interprets the tags

HTML Versions

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

Basic HTML Page Structure

- Tags have start and end elements
 - and may have attributes
- Pages contain nested tags
 - html is the top level tag
 - typically contains a head and body element
 - head provides information to the browser on the body
 - such as the character set being used
 - the page title
 - body contains information to be rendered by the browser
 - such as headings and paragraphs of text

Basic HTML Page Structure

□ Defining a simple HTML page

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Hello World</title>
    <meta charset=UTF-8">
  </head>
  <body>
    <h1>A Heading</h1>
    <p>Well this is a paragraph of text.</p>
    <p>This is another paragraph of text - welcome world!</p>
  </body>
</html>
```

Basic HTML Page Structure

```
<html>
  <head>
    <title>Hello World</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  </head>
</html>
```

Basic HTML Page Structure

```
<html>
```

```
  <head>
```

```
    <title>Hello World</title>
```

```
    <meta charset=UTF-8" />
```

```
  </head>
```

```
<body>
```

```
  <h1>A Heading</h1>
```

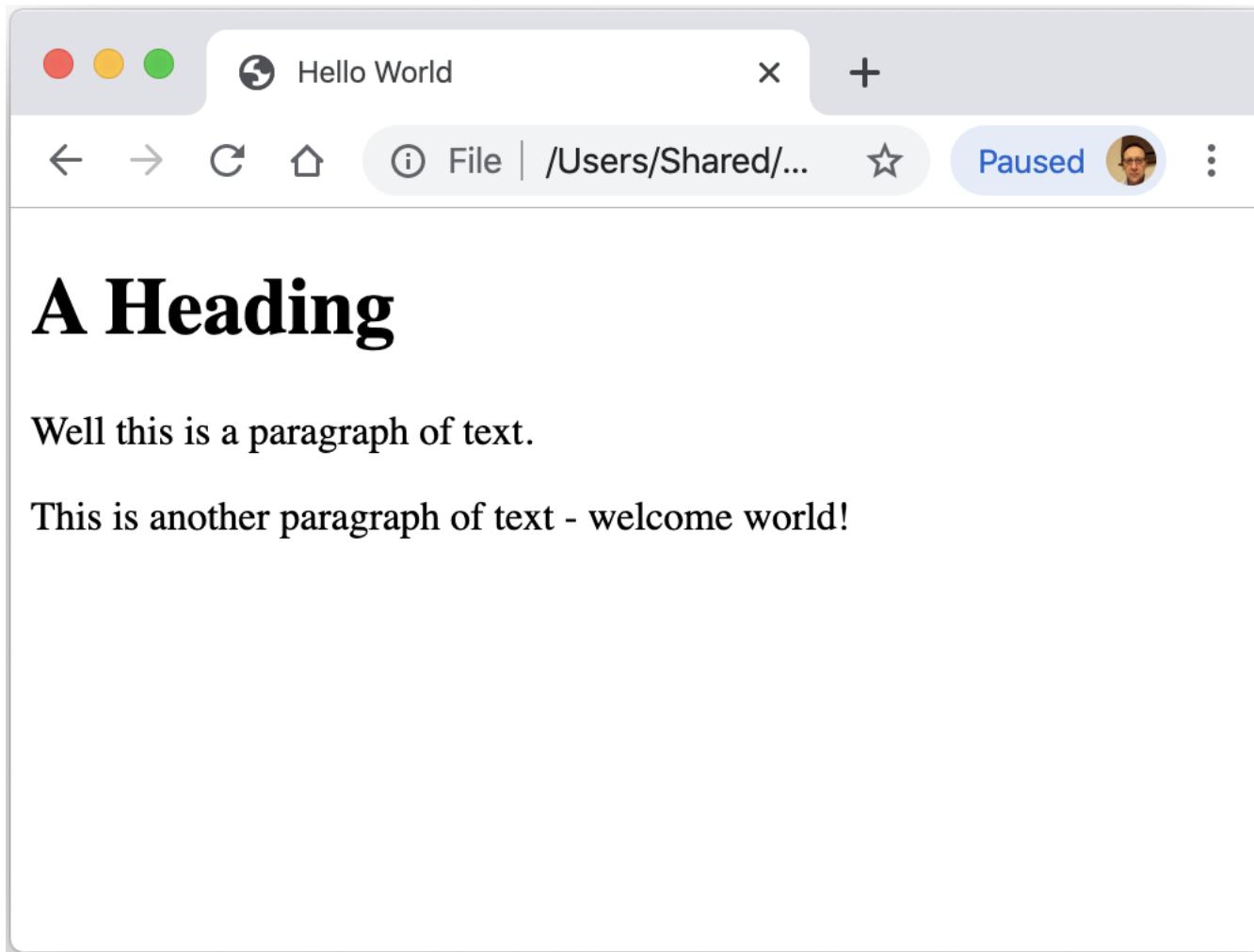
```
  <p>Well this is a paragraph of text.</p>
```

```
  <p>This is another paragraph of text - welcome world!</p>
```

```
</body>
```

```
</html>
```

Basic HTML Page Structure



HTML Tags

- <!DOCTYPE> Declaration (not a tag)
 - represents the document type
 - for HTML5 is is <!DOCTYPE html>
- Tag format (start and end tag) and not case sensitive
 - <tagname>content</tagname>
 - short hand from <tagname /> if no content e.g,

- Tags in Page
 - <html> root element of an HTML page
 - <head> element contains meta information about the document
 - <title> element specifies a title for the document
 - <body> element contains the visible page content
 - <h1> element defines a large (level 1) heading
 - The <p> element defines a paragraph

Common Tags

- Headings
 - for different levels
 - <h1></h1> <h2></h2> <h3></h3>
- Horizontal line (an empty element)
 - <hr />
- Break line
 -

- Comments in web page (ignored by browser)
 - <!-- write comment here-->
- Formatting!
 - bold; <i></i> italic; Emphasis
 - <center>text</center> - center text

Tag attributes

- A Tag can have additional attributes used with content

```
<tagname attr1="value1" attr2="value2">content</tagname>
```

- Examples

```
<input type="text" name="firstname"></input>
```

```
<form action="/query-books" method="post">  
</from>
```

```

```

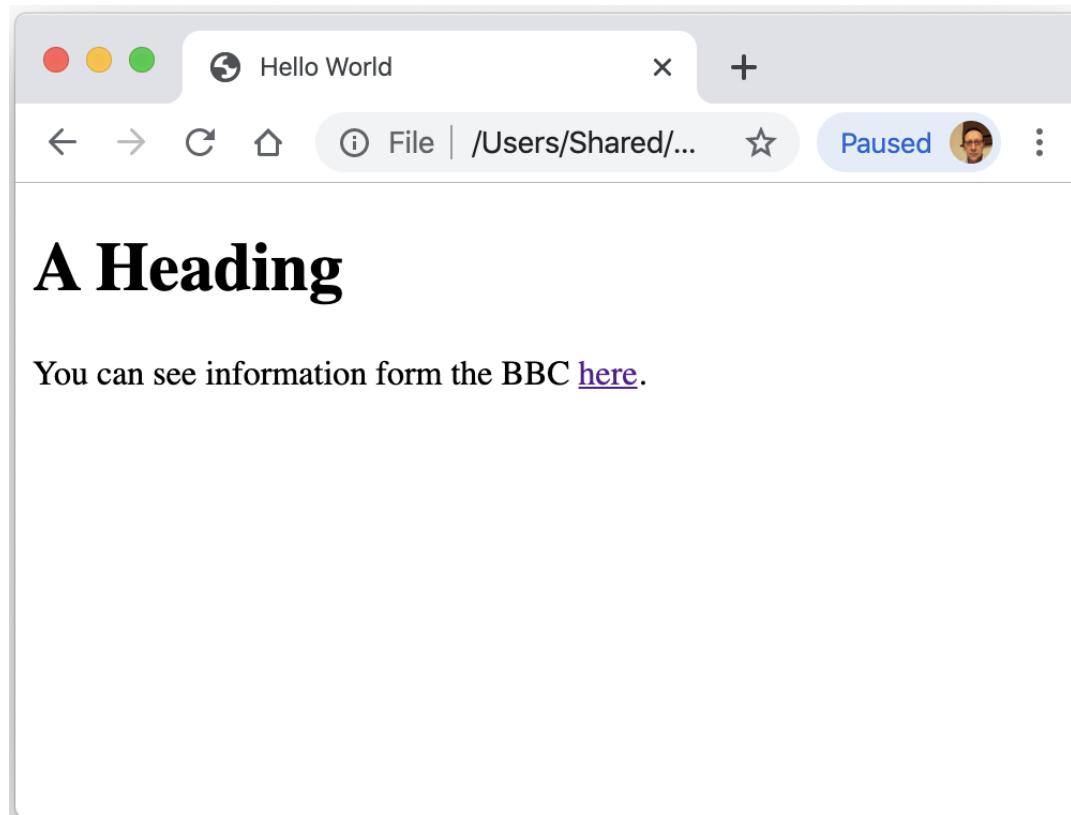
Hyperlinks

- Put the Hyper in HTML
- Link from one page to another
- Link can have text, image, or any other HTML element as content
- Hyperlinks defined by the `<a>` tag and href attribute

```
<body>
  <h1>A Heading</h1>
  <p>You can see information form the
    BBC <a href="https://www.bbc.co.uk">here</a>.</p>
</body>
```

Hyperlinks

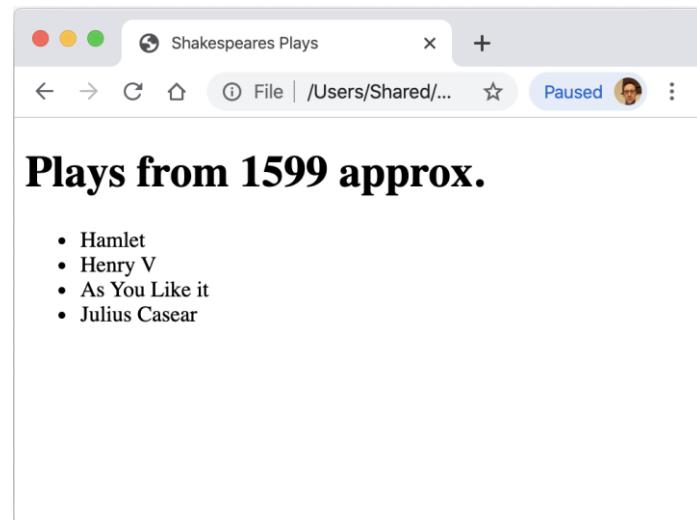
- Link is rendered using a different visual cue



Lists

- Lists of elements can be ordered or unordered
- Unordered list defined by tag
 - list elements defined by tag

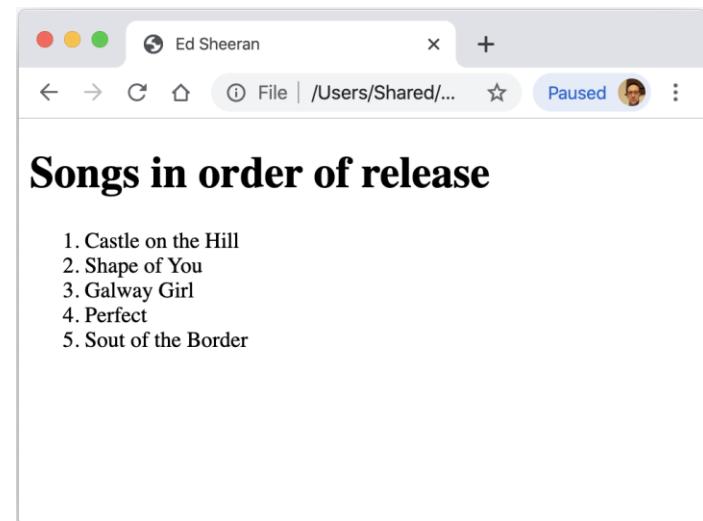
```
<body>
  <h1>Plays from 1599 approx.</h1>
  <ul>
    <li>Hamlet</li>
    <li>Henry V</li>
    <li>As You Like it</li>
    <li>Julius Casear</li>
  </ul>
</body>
```



Lists

- Ordered lists use tag
 - list elements defined by tag

```
<body>
  <h1>Songs in order of release</h1>
  <ol>
    <li>Castle on the Hill</li>
    <li>Shape of You</li>
    <li>Galway Girl</li>
    <li>Perfect</li>
    <li>South of the Border</li>
  </ol>
<body>
```

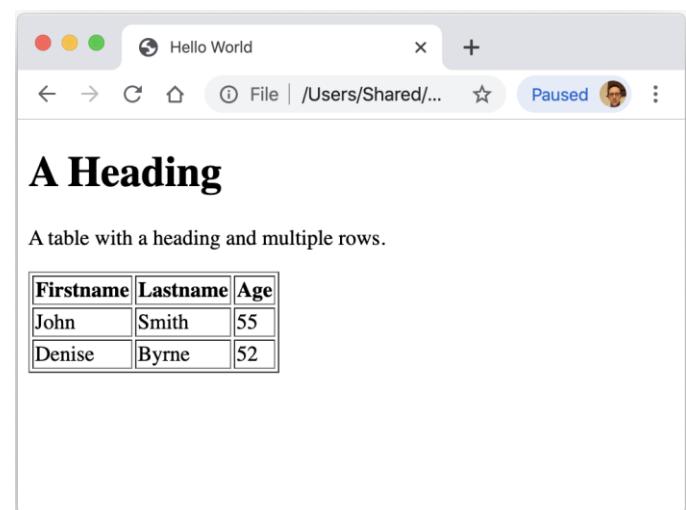


Tables

- Table has columns, rows
 - may have headings
 - may have different colours for rows
 - may or may not display borders etc.
- Defined using the <table> tag
- (Optional) table header is defined by the <th> tag
- Each row is defined by the <tr> tag
- Element cell within a row is defined by the <td> tag

Tables

```
<body>
  <h1>A Heading</h1>
  <p>A table with a heading and multiple rows.</p>
  <table border="1">
    <tr>
      <th>Firstname</th>
      <th>Lastname</th>
      <th>Age</th>
    </tr>
    <tr>
      <td>John</td>
      <td>Smith</td>
      <td>55</td>
    </tr>
    <tr>
      <td>Denise</td>
      <td>Byrne</td>
      <td>52</td>
    </tr>
  </table>
<body>
```



Tables

- Can indicate a cell / heading spans multiple columns
 - <th colspan="2">Telephone Numbers</th>
- Or multiple rows
 - <th rowspan="2">Telephone:</th>
- Can provide a caption
 - <caption>My Contacts</caption>

Forms

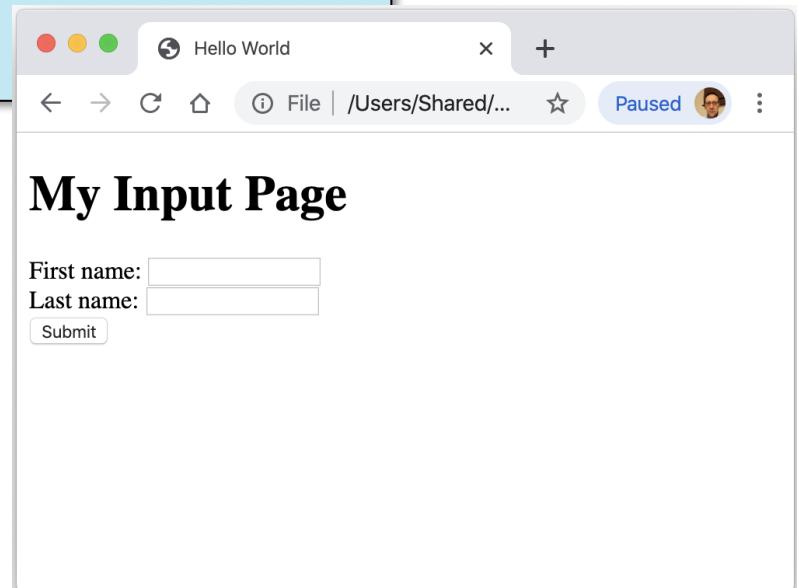
- Defined by the <form> element
 - used to collect user input
 - and submit that data (typically) to a server
- A Form contains input elements
 - <input> rendered in different ways depending upon the type

Type	Description
<input type="text">	Defines a one-line text input field
<input type="radio">	Defines a radio button (for selecting one of many choices)
<input type="submit">	Defines a submit button (for submitting the form)

- input field must have a name to be submitted

Forms

```
<body>
  <h1>My Input Page</h1>
  <form>
    First name: <input type="text" name="firstname"><br />
    Last name: <input type="text" name="lastname"><br />
    <input type="submit" value="Submit">
  </form>
<body>
```



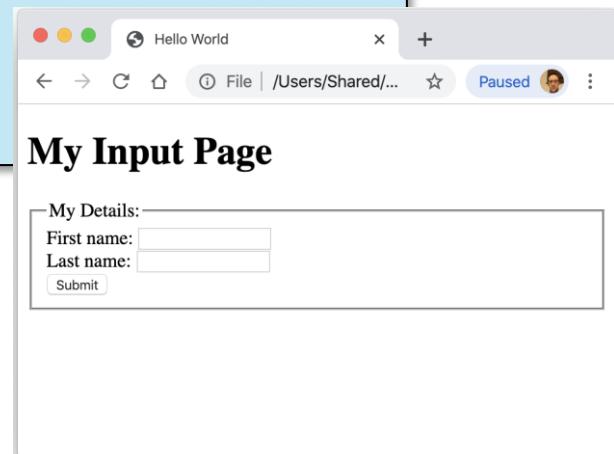
Forms

- Can provide a default value for an input
 - via value attribute
- Can indicate where data is sent via action attribute
 - <form action="/query-books">
- target attribute specifies if the submitted result will open in a new browser tab
 - <form action="/query-books" **target="_blank"**>
- Method attribute indicates which HTTP Request Method to use (default is GET)
 - <form action="/query-books" method="post">

Forms

- Can group elements using <fieldset>
 - <legend> element defines a caption for the <fieldset> element

```
<form>
  <fieldset>
    <legend>My Details:</legend>
    First name: <input type="text" name="firstname"><br />
    Last name: <input type="text" name="lastname"><br />
    <input type="submit" value="Submit">
  </fieldset>
</form>
```



HTML Images

- Incorporated into a page using the tag
 - has attributes
 - src for the location of the picture
 - alt (optional) for text to display if image is not found
 - also useful for screen readers
 - width (optional) for the width of the image
 - height (optional) for the height of the image
- Examples
 -
 -
 -

Div or *block* element

- Blocks always start a new line and may have their own style
- Defined using the <div> tag
 - <div>Hello World</div>
- Often used as a container for other HTML elements
- Often used with JavaScript to manage content
- Often used with CSS to handle styling of content

Semantic tags

- Like <div> but carry semantic meaning
 - <article>Hello World</article>
- Often used as a container for other HTML elements
- Recommended for accessibility and to benefit search engines

CSS Introduction



John Hunt

Informed Academy



framework training
business value through education

Plan for Session

- Why do we need styles?
- CSS
- CSS & Selectors
- Inline CSS
- Internal CSS
- External Style Sheets
- Example with and without Style Sheets
- Using DIV and Span with CSS
- CSS Meta Languages (LESS, Sass etc.)
- Bootstrap

Why do we need styles?

- HTML originally intended for structure
 - not presentation aspects
 - intended as a way of describing content
- HTML 3.2 blurred distinction
 - with and color attributes
- But hard to enforce standards
 - hard to have a consistent look and feel
 - need to change page content when just want to change presentation
- Separation of concerns wanted
 - content versus presentation

CSS – Cascading Style Sheets

- Describe how HTML elements should be rendered in browser
 - although potentially can be used for rendering in any media
- One style sheet can be used by many web pages
 - supporting common look and feel
- CSS can be added to HTML elements in 3 ways
 - Inline using *style* attribute of elements
 - Internal using `<style>` element in `<head>`
 - External using an external CSS file
 - this is the most common professional approach

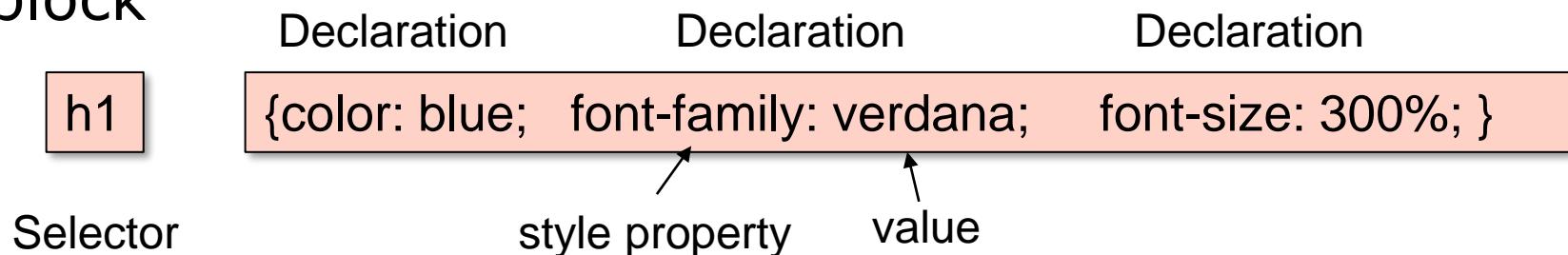


CSS Versions

- W3C Standard
 - initially proposed in 1996 (CSS 1)
 - but there was patchy adoption from browsers such as original IE
- CSS 2 defined in 1998
- CSS 2.1
 - designed to solve problems with CSS1 and CSS2
 - sorted out errors in specification / removed poorly supported features
- CSS 3 – expands and widens CSS specification

CSS & Selectors

- CSS rule-set consists of a selector and a declaration block



- selector indicates HTML element to be styled
- declaration block contains 1 or more declarations separated by ;
- a Declaration includes a CSS property name and a value, separated by a :
- Declarations always end with a ;
- can also have comments /* a comment */

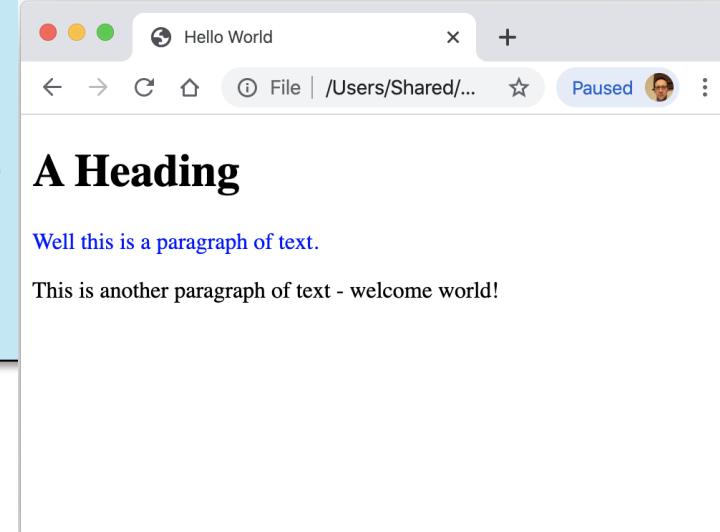
CSS Selectors

- CSS selectors can work in several ways:
 - Simple selectors (select elements based on name, id, class)
 - e.g. for an id #para1 **#para1 { color: red; }**
 - or for a paragraph with the class error
p.error { color: red; }
 - Combinator selectors
 - select elements based on a specific relationship between them
 - e.g. all <p> inside <div>
div p { color: yellow; }
 - Attribute selectors
 - select elements based on an attribute or attribute value
input[type="text"] { border: 2px solid red; }

Inline CSS

- Used to apply a style to an individual element
- Uses the `style` attribute of HTML element

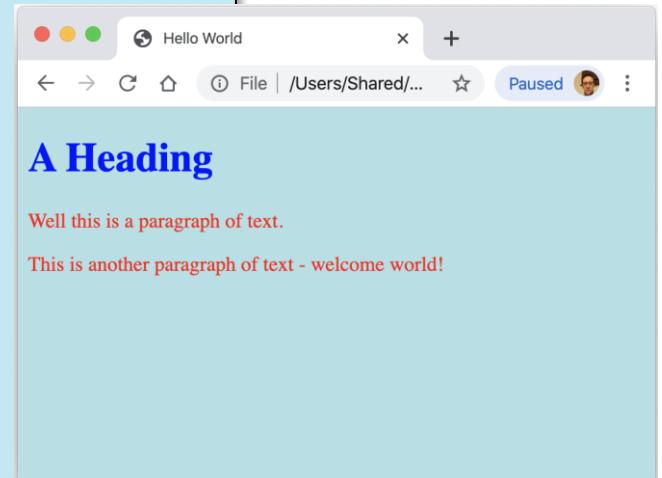
```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Hello World</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  </head>
  <body>
    <h1>A Heading</h1>
    <p style="color:blue;">Well this is a paragraph of text.</p>
    <p>This is another paragraph of text - welcome world!</p>
  </body>
</html>
```



Internal CSS

- Defines a style used throughout single HTML page
- Defined in <head> using the <style> element

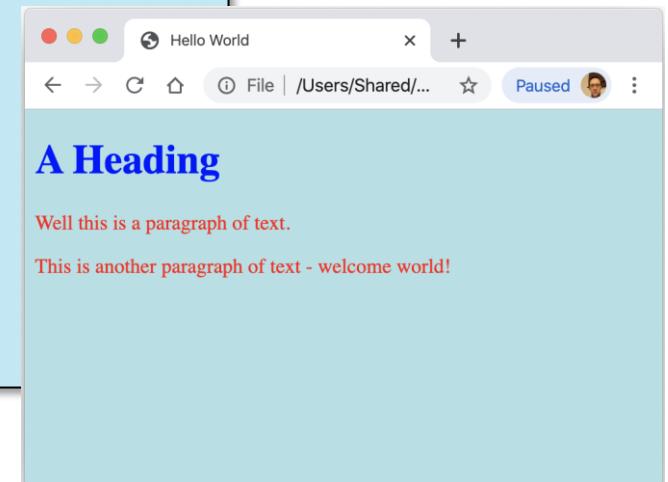
```
<!DOCTYPE HTML>
<html>
<head>
  <title>Hello World</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <style>
    body { background-color: powderblue; }
    h1 { color: blue; }
    p { color: red; }
  </style>
</head>
<body>
  <h1>A Heading</h1>
  <p>Well this is a paragraph of text.</p>
  <p>This is another paragraph of text - welcome world!</p>
</body>
</html>
```



External Style Sheets

- Defines styles for use in multiple HTML pages
- Defined via `<link>` element in `<head>` section

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Hello World</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
  <h1>A Heading</h1>
  <p>Well this is a paragraph of text.</p>
  <p>This is another paragraph of text - welcome world!</p>
</body>
</html>
```



External Style Sheets

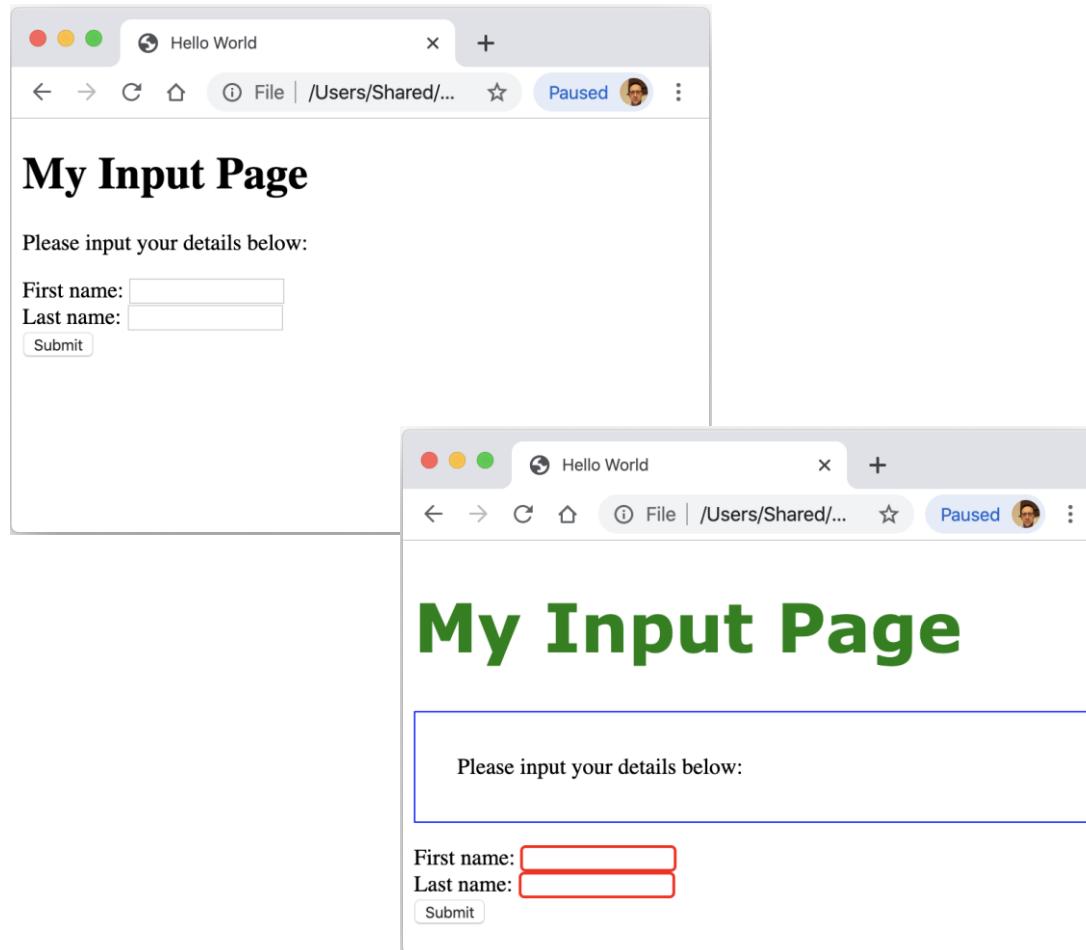
- ❑ Separate (external) CSS file e.g. styles.css

```
body {  
    background-color: powderblue;  
}  
  
h1 {  
    color: blue;  
}  
  
p {  
    color: red;  
}
```

- ❑ Can specify any number of style attributes

```
...  
  
h1 {  
    color: blue;  
    font-family: verdana;  
    font-size: 300%;  
}  
  
...
```

Example with and without Style Sheets



```
p {  
  border: 1px solid blue;  
  padding: 30px;  
}  
h1 {  
  color: green;  
  font-family: verdana;  
  font-size: 300%;  
}  
tr:nth-child(even) {  
  background: white;  
}  
tr:nth-child(odd) {  
  background: rgb(80, 78, 78);  
}  
input[type="text"] {  
  border: 2px solid red;  
  border-radius: 4px;  
}
```

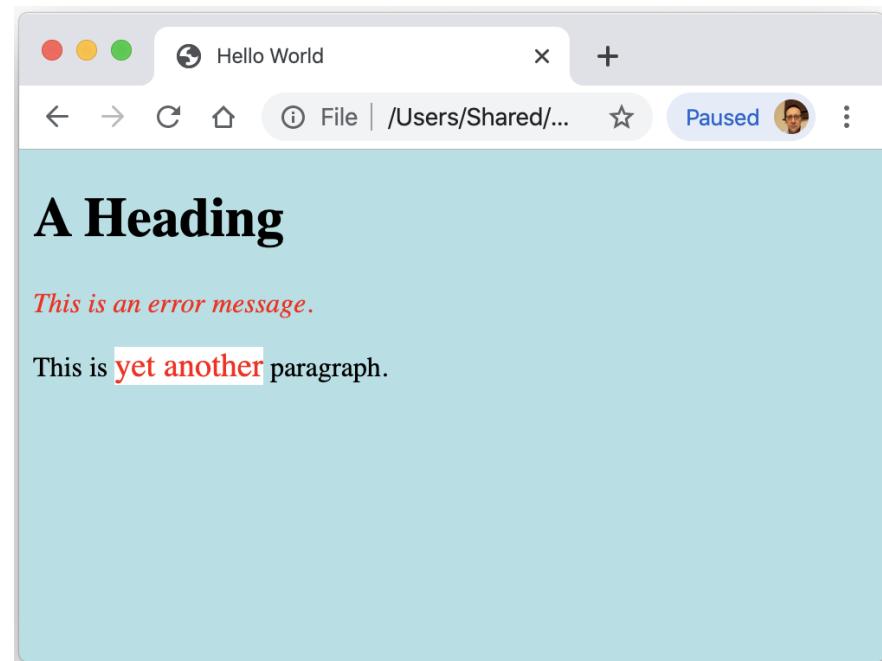
Using DIV and Span with CSS

- <div> defines section (division) of a block
- defines an inline section
- Both can be used with styles to format content
- Both can have an id and a class attribute
 - makes it easy to apply styles
- Widely used to handle styling as well as content grouping

Using DIV and Span with CSS

```
<h1>A Heading</h1>
<div class='error'>This is an error message.</div>
<p>This is <span class='warn'>yet another</span> paragraph.</p>
```

```
body {
  background-color: powderblue;
}
.error {
  color: red;
  font-style: italic;
}
.warn {
  background: white;
  color: red;
  font-size: larger;
}
```



CSS Challenges

- CSS can be very verbose
 - can end up repeating a lot of style information
 - as not possible to nest selectors / styles
- Can be quite brittle
 - one change can break a whole site
- Vendors provide prefixes to offer a namespace for their styles
 - these can be verbose and easy to break
 - each vendor does their own thing

CSS Meta Languages

- Meta Languages can help simplify CSS
 - provide a richer syntax for describing styles
 - pre-processed into plain CSS

- Several widely used options
 - Less – Leaner Style Sheets
 - Sass – Syntactically Awesome Style Sheets



```
@pale-green-color: #4D926F;  
#header {  
    color: @pale-green-color;  
}  
h2 {  
    color: @pale-green-color;  
}
```



```
#header {  
    color: #4D926F;  
}  
h2 {  
    color: #4D926F;  
}
```



Bootstrap

Bootstrap

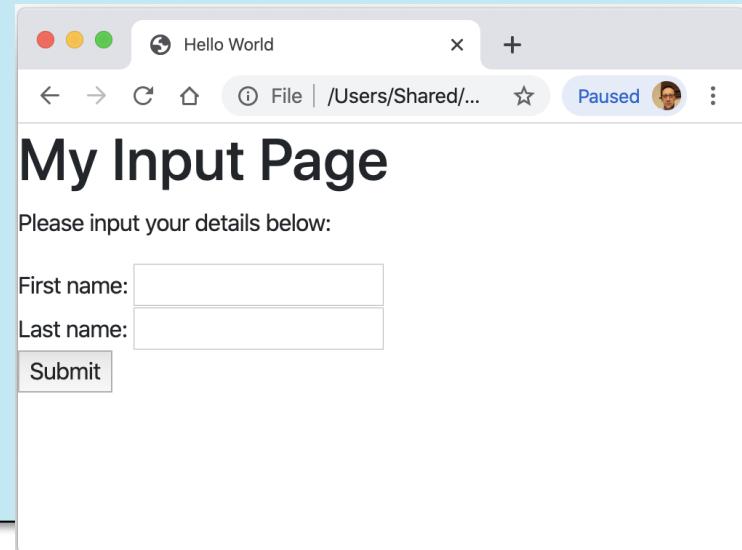
- Widely used framework for styling web sites
 - also supports mobile device browsing
 - use BootstrapCDN to add to site
 - CDN = Content Delivery Network
- Some features require JavaScript and jQuery
 - but they are hidden from you other than adding to web page
- Requires HTML5
- See <https://getbootstrap.com/>



Bootstrap

Bootstrap

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Hello World</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <!-- Bootstrap CSS -->
  <link rel="stylesheet"
    href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
    crossorigin="anonymous">
</head>
<body>
  <h1>My Input Page</h1>
  <p>Please input your details below:</p>
  <form>
    First name: <input type="text" name="firstname"><br>
    Last name: <input type="text" name="lastname"><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```



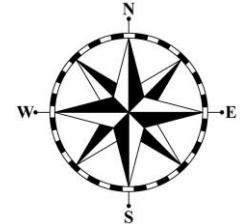
Vagrant: Management for Virtual Development Environments



Toby Dussek

Informed Academy





Why Vagrant?

- Different Projects Need Different Environments
 - Linux, Windows, MacOS
 - Different versions of MacOS 14 v MacOS 15 Catalina
- Virtual Machines
- Installing and running Vagrant
 - Synced Vagrant Folders
- Serve Web Pages with Apache
 - Installing Apache and Accessing Apache
 - Serving up Pages
- Shutting Down Vagrant

Different Projects Need Diff't Envs

- Sometimes need
 - Specific versions of Languages
 - Or specific operating system versions
 - Or specific tools
 - Or specific environment settings
- Managing these on a Development project can be difficult
- Want to avoid *it only works on one machine* issues
- Want to standardise environments

Virtual Machines

- Virtual Machines (VMs) are one way to handle these
 - such as VirtualBox, VMWare etc
- But can be time consuming to maintain
- Can be set up in different ways by different people
- Vagrant is a VM Environment Manager
 - allows configuration and deployment of VMs
 - enabling reproducible / portable work environments





To Install Vagrant

- Need a VM provider
 - e.g. VirtualBox (free)
 - see <https://www.virtualbox.org/wiki/Downloads>
 - install this first
- Need Vagrant itself
 - see <https://www.vagrantup.com/downloads.html>
- In both cases ensure that you choose the correct version for your platform

Working from the Command Line

- Many developer oriented tools
 - are available from the command line
 - sometimes also available via some GUI
 - e.g. git and SourceTree etc.
- Open a Command / Terminal window
- Create a directory to work in
 - e.g. > mkdir coursework
- Move into that directory
 - e.g. > cd coursework





Starting Vagrant

- Vagrant provides a repository of *Boxes*
 - environments that can be installed such as Ubuntu Linux
 - can initialise a new VM using one of these Boxes
 - > vagrant init hashicorp/bionic64
 - creates a VirtualBox VM running Ubuntu 12.04 LTS 64-bit
 - creates a Vagrantfile in the current working directory
 - can now start the VM
 - > vagrant up
 - You now have a VM running on your computer
 - Can access it using ssh (ssh must be in the path)

```
> vagrant ssh
```



Starting Vagrant

```
Johns-iMac:devenvs jeh$ vagrant ssh
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-58-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Thu Dec  5 11:52:51 UTC 2019

System load:  0.05          Processes:            91
Usage of /:   2.5% of 61.80GB  Users logged in:    0
Memory usage: 11%          IP address for eth0: 10.0.2.15
Swap usage:   0%

0 packages can be updated.
0 updates are security updates.

vagrant@vagrant:~$
```



Synced Folders

- By default Vagrant shares
 - the current directory containing the Vagrantfile
 - to the /vagrant directory of the VM

```
vagrant@vagrant:~$ ls -la /vagrant
total 8
drwxr-xr-x  1 vagrant vagrant  128 Dec  5 11:50 .
drwxr-xr-x 24 root    root     4096 Dec  5 11:51 ..
drwxr-xr-x  1 vagrant vagrant  128 Dec  5 11:50 .vagrant
-rw-r--r--  1 vagrant vagrant 3025 Dec  5 11:48 Vagrantfile
```

- Note the default directory when you SSH in is
 - /home/vagrant
 - (not /vagrant)
- Can use this to share files between host and VM

Apache HTTP Server



- Aka Apache
- Free open-source, cross platform web server
- Typically run on a Linux box
 - but can be run on Windows and Mac
- Can be used to serve up web pages
 - based on HTTP Requests
- Probably most widely used web server
- Allegedly named
 - because it was a patched version of an older server
 - i.e. it was 'a Patchy' server



Installing Apache

- Need to create a script to setup Apache
 - Create a bootstrap.sh file in the same directory as Vagrantfile
 - can use whatever editor you want on the host system e.g. Notepad++

```
#!/usr/bin/env bash

apt-get update -y -qq
apt-get install -y apache2

APACHE_CONFIG=/etc/apache2/apache2.conf
VIRTUAL_HOST=localhost
DOCUMENT_ROOT=/var/www/html

echo -e "-- Adding ServerName to Apache config\n"
grep -q "ServerName ${VIRTUAL_HOST}" "${APACHE_CONFIG}" || echo
"ServerName ${VIRTUAL_HOST}" >> "${APACHE_CONFIG}"

echo -e "-- Allowing Apache override to all\n"
sed -i "s/AllowOverride None/AllowOverride All/g" ${APACHE_CONFIG}
```



```
echo -e "--- Updating vhost file\n"
cat > /etc/apache2/sites-enabled/000-default.conf <<EOF
<VirtualHost *:80>
    ServerName ${VIRTUAL_HOST}
    DocumentRoot ${DOCUMENT_ROOT}

    <Directory ${DOCUMENT_ROOT}>
        Options Indexes FollowSymlinks
        AllowOverride All
        Order allow,deny
        Allow from all
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/${VIRTUAL_HOST}-error.log
    CustomLog ${APACHE_LOG_DIR}/${VIRTUAL_HOST}-access.log combined
</VirtualHost>
EOF

echo -e "--- Restarting Apache web server\n"
service apache2 restart

if ! [ -L /var/www ]; then
    rm -rf /var/www
    ln -fs /vagrant /var/www
fi
```



Installing Apache

- Next modify the *Vagrantfile* to run this shell script

- add the lines

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.network "private_network", ip: "192.168.50.10"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, guest: 80, host: 8080
end
```

- last line linking port on the host machine to a port on the VM
- Now refresh Vagrant, either use
 - `vagrant up` – if your VM is not running
 - `vagrant reload --provision` – if your VM is still running



Accessing Apache

- On the VM use apache2 -v

```
vagrant@vagrant:~$ apache2 -v
Server version: Apache/2.4.29 (Ubuntu)
Server built:  2020-08-12T21:33:25
vagrant@vagrant:~$
```

- Can now test Apache using
 - http://192.168.50.10

The screenshot shows a web browser window with the following details:

- Address Bar:** Shows the URL `Not Secure | 192.168.50.10`.
- Content Area:** Displays the text "Not Found" in large, bold, black font, followed by the message "The requested URL was not found on this server."
- Page Footer:** Shows the server information: `Apache/2.4.29 (Ubuntu) Server at 192.168.50.10 Port 80`.



Accessing Apache

- Create a directory called html

- in the current working directory
 - i.e. the one with the Vagrant file

```
> mkdir html
```

- Add an index.html file to html directory containing

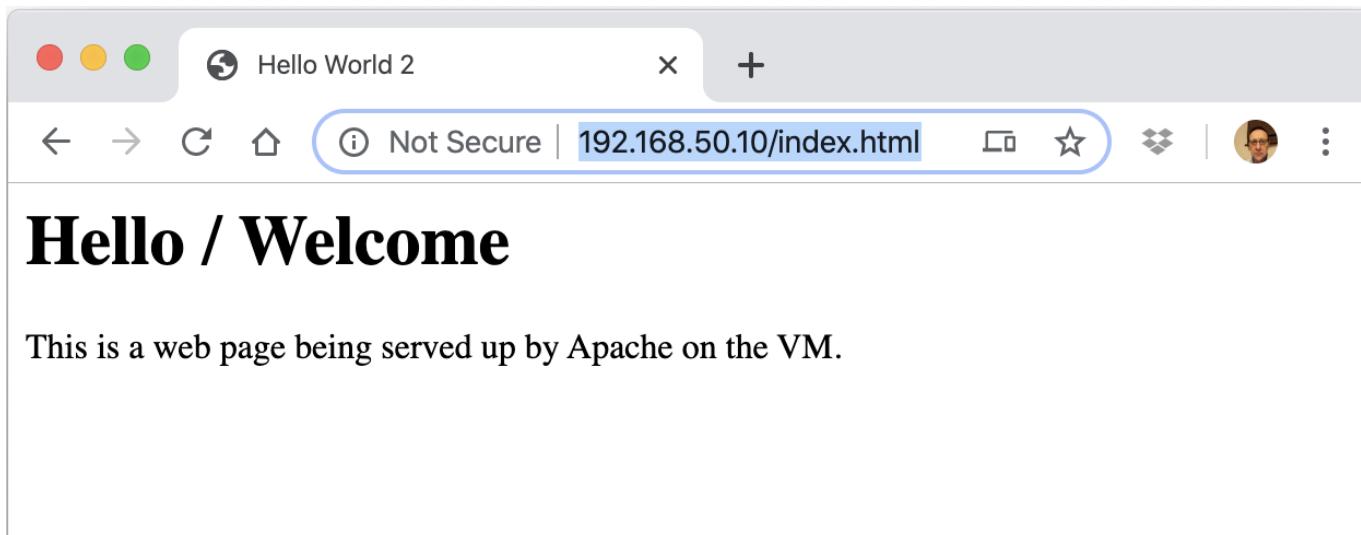
```
<html>
  <head><title>Hello World 2</title></head>
  <body>
    <h1>Hello / Welcome</h1>
    <p>This is a web page being served up by
      Apache on the VM.</p>
  </body>
</html>
```



Accessing Apache

- Now access via url

- <http://192.168.50.10/index.html> or
- <http://localhost:8080/index.html>





Shutting Down Vagrant

- Logging out of the VM does not shut it down
 - it is still running in the background
- The following commands are available
 - `vagrant suspend` – suspends the current VM in its current state
 - `vagrant halt` - shuts down the VM can be restarted but in clean state
 - `vagrant destroy` - removes the VM
 - `vagrant up` - starts / restarts the VM

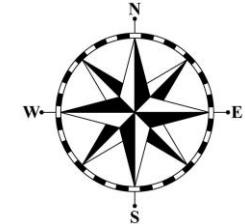
Writing Tests with Selenium



Toby Dussek

Informed Academy





Plan for Session

- Intro to Selenium
- Using JUnit to drive Selenium
- Setting up a Project
- Selenium Configuration
- Writing a JUnit Selenium test
- Reviewing the Results
- Some additional Selenium Features



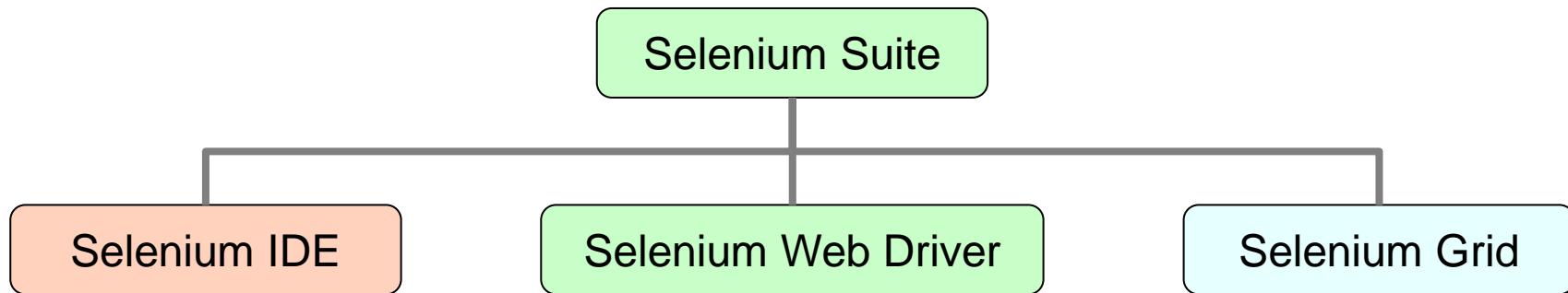
Selenium

- Free open source automated testing tool suite for Web Applications / sites
- Supports
 - wide range of browsers using plugin drivers
 - can be used from multiple programming languages including Java, JavaScript, C#, Python etc
- Very widely used
 - also supports CI/CD usage
- See
 - <https://www.selenium.dev/>



Selenium Components

- Selenium is a suite of
 - open source, testing automation tools
 - (now a de facto standard)



- supports a wide range of languages inc. JavaScript, Java, Python, C#, PHP etc.



Selenium Operation

□ How Selenium works



- Test sends requests to Selenium for actions to be run, data to be returned etc.
- Selenium sends request onto Browser
- Browser performs action
- Browser sends data back to Selenium
- Selenium sends data back to test
- Test can verify response

Using JUnit to drive Selenium

- JUnit very widely used within Java community
 - ideally want to write tests in familiar language
 - can use JUnit to drive Selenium
- To do this need to
 - Add libraries to support Selenium in Java language
 - can use Maven POM for this
 - Configure Selenium
 - handy to have a config class so can reuse
 - Load an appropriate browser driver
 - for example ChromeDriver
 - Write tests in JUnit

Setting up Project

- Need to add Selenium as a Maven dependency

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
</dependency>
```

- Make sure version matches your project
- Plus JUnit dependencies

```
<dependencies>
...
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>${junit-platform-runner-version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit-version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>${junit-version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>${selenium-java-version}</version>
    <scope>test</scope>
</dependency>
</dependencies>
```

```
<properties>
    <junit-version>5.7.0</junit-version>
    <junit-platform-runner-version>1.7.0</junit-platform-
runner-version>
    <selenium-java-version>3.141.59</selenium-java-
version>
</properties>
```

Selenium Configuration

- Selenium WebDriver
 - uses a browser to display web page
 - run tests defined
- Different Web Browsers need different Drivers
 - used by the Selenium WebDriver
- For example to use Chrome need ChromeDriver
 - needs to be installed and configured
 - <https://sites.google.com/a/chromium.org/chromedriver/>
 - download version for your platform

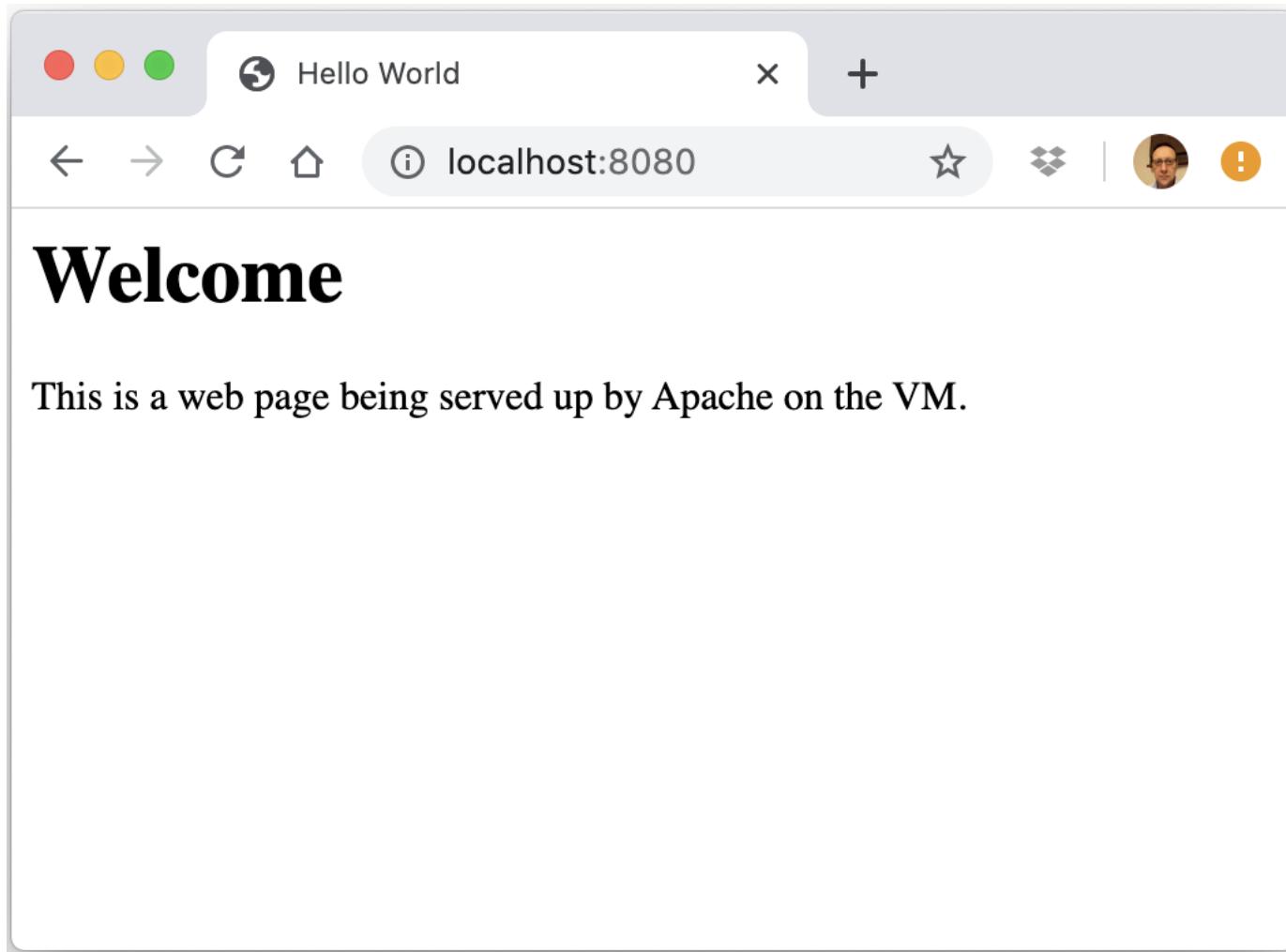
```
public class SeleniumConfig {  
    private WebDriver driver;  
  
    public SeleniumConfig() {  
        driver = new ChromeDriver();  
        driver.manage().deleteAllCookies();  
        driver.manage().timeouts().pageLoadTimeout(40, TimeUnit.SECONDS);  
    }  
  
    static {  
        System.setProperty("webdriver.chrome.driver", "chromedriver");  
    }  
  
    public void close() {  
        driver.close();  
    }  
    public WebDriver getDriver() {  
        return driver;  
    }  
}
```

Selenium Configuration

```
public class SeleniumJUnitTest {  
    private static SeleniumConfig config;  
  
    @BeforeAll  
    public static void setUp() {  
        config = new SeleniumConfig();  
        config.getDriver().get("http://localhost:8080/index.html");  
    }  
  
    @AfterAll  
    public static void tearDown() { config.close(); }  
  
    @Test  
    public void check_page_title_is_Hello_World() {  
        String actualTitle = config.getDriver().getTitle();  
        assertEquals("Hello World", actualTitle);  
    }  
  
    @Test  
    public void check_heading_is_Welcome() {  
        String heading =  
            config.getDriver().findElement(By.className("heading")).getText();  
        assertEquals("Welcome", heading);  
    }  
}
```

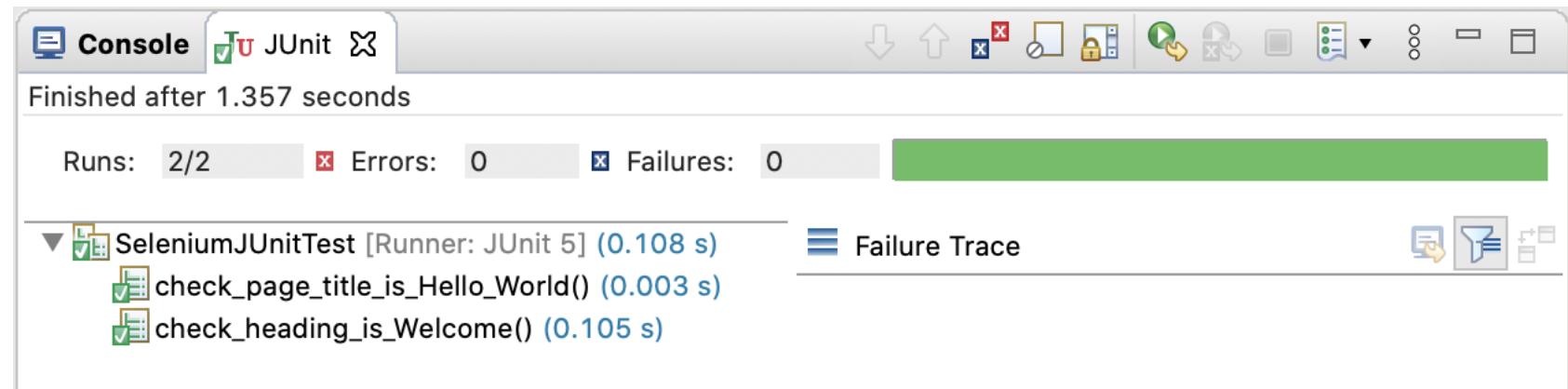
JUnit Selenium Test

Web Page to Test



Result of Running Tests

- Tests run Chrome via Selenium



JUnit Selenium Additional Features

- Can also perform a button click

```
driver.findElement(By.cssSelector("input[type='button']")).click();
```

- Alternatively you can use

```
driver.focus("name=Valuation"); //name of button  
driver.click("Valuation"); //pass that name to click
```

- Can select from a drop down list

```
Select select =new Select(driver.findElement(By.id("country-dropdown")));  
select.selectByVisibleText("UK");
```

```
Select select =new Select(driver.findElement(By.id("country-dropdown")));  
select.selectByValue("United Kingdom");
```

JavaScript Introduction

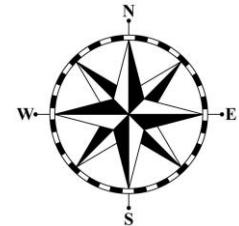


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- JavaScript Background
- JavaScript Versions
- Browser Support
- JavaScript Programming Language
- JavaScript Execution Model
- Running JavaScript Programs
- Visual Studio Code



JavaScript Background

- The JavaScript programming language
 - general purpose programming language
 - originates in 1990s; created Netscape
 - as a scripting language for use in their Netscape Navigator
 - originally called LiveScript
 - but renamed to JavaScript for Netscape Navigator 2.0 beta 3 Dec 1995
 - possibly to exploit the popularity of Java
 - but there is no connection between Java and JavaScript something which has caused a lot of confusion over the years
 - aim was to make the web more interactive / more executable
 - also released a version for server-side scripting in Dec 1995
 - Microsoft's IIS supported server-side JavaScript from 1996
 - now Node.js makes server-side JavaScript very popular



JavaScript Versions

- ECMAScript Language specification standard June 1997
 - Nov 1996 Netscape submitted JavaScript to ECMA International to create standard
 - ActionScript and Jscript became other implementations of ECMAScript
- ECMAScript 2 June 1997
- ECMAScript 3 December 1999
 - now acts as base line for modern JavaScript
- ECMAScript 4 – never released but influenced later versions
- ECMAScript 5 December 2009
 - (previously known as ECMAScript 3.1 during development)
- ECMAScript 5.1 June 2011
- **ES 6 ECMAScript 2015 June 2015 / ES 7 ECMAScript 2016 June 2016 / ES 8 ECMAScript 2017 June 2017 / ES ECMAScript 2018**

Browser Support

□ ES 6 (ECMAScript 2015)

Browser	Version	Date
Chrome	51	May 2016
Firefox	54	Jun 2017
Edge	14	Aug 2016
Safari	10	Sep 2016
Opera	38	Jun 2016



□ ES 7 (ECMAScript 2016)

Browser	Version	Date
Chrome	68	May 2018
Opera	55	Aug 2018

ES7

JavaScript Programming Language

- A Hybrid Dynamically Typed language

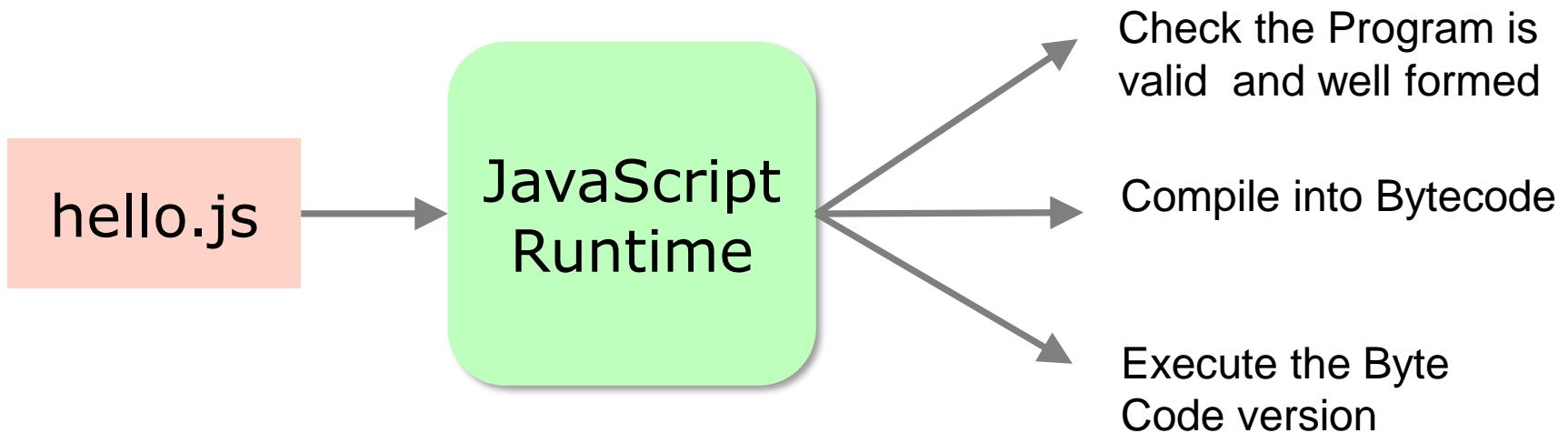
- Allows for Procedural Programming
- Supports Object Oriented Programming
 - prototype based pre ES 6
- Facilities Functional Programming
- Support for Event driven programming
- Has dynamic typing
- Extensively used within web pages
- Key element of the Ajax approach
 - it is the j in Ajax
- Increasingly used on the Server c.f. Node.js





JavaScript Execution Model

- Not a precompiled language
 - execution can be handled in many ways



- Simplified process illustrative of modern runtimes
 - not JavaScript in the browser is single threaded
 - each page can run one thread of JavaScript at a time

Running JavaScript Programs

- Several ways to run JavaScript programs
 - Placed in a web page and run
 - Run from the command line using Node.js
 - Run from within an IDE that supports JavaScript
 - Run from within an IDE using underlying Node.js

JavaScript IDEs

▀ Numerous JavaScript Editors / IDEs available

- ▀ Visual Studio Code

- Free IDE from Microsoft



- ▀ Sublime Text

- free light weight editor



- ▀ Atom

- another free light weight editor



- ▀ Eclipse with JavaScript Dev Tools

- full blow IDE – widely used in Java space



- ▀ WebStorm

- full blown IDE from JetBrains who provide numerous IDEs for many languages



WebStorm

- ▀ Microsoft Visual Studio

- heavier weight IDE environment for the .Net world





Visual Studio Code

- A JavaScript aware IDE
- Plugins extend JavaScript support including
 - Prettier. Category: formatter.
 - ESLint. Category: linter. ...
 - Debugger for Chrome
 - Node.js extensions
 - see <https://code.visualstudio.com/docs/nodejs/extensions>
- Run JavaScript in Visual Studio Code
 - use Code Runner Extension
 - see <https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner>



Visual Studio Code

- Hello World in JavaScript
 - uses console to log a string as output

The screenshot shows the Visual Studio Code interface with a dark theme. On the left is the Explorer sidebar, which lists a workspace named 'javascript-intro'. Inside this workspace, there's a folder '01-helloworld' containing a file 'simplehelloworld.js'. This file is currently selected and shown in the main editor area. The code in the editor is:

```
JS simplehelloworld.js ×
01-helloworld > JS simplehelloworld.js
1   console.log('Hello World!')
```

Below the editor, the Output panel shows the execution results:

```
PROBLEMS OUTPUT ...
[Running] node "/Users/Shared/workspaces/visualstudiocode/javascript-intro/01-helloworld/simplehelloworld.js"
Hello World!

[Done] exited with code=0 in 0.081 seconds
```

At the bottom of the screen, the status bar displays:

Ln 1, Col 28 Spaces: 4 UTF-8 LF JavaScript Prettier ☺ 🔔

JavaScript Coding Standards

- Several commonly used reference sites
- Mozilla JavaScript guidelines
 - https://developer.mozilla.org/en-US/docs/MDN/Contribute/Guidelines/Code_guidelines/JavaScript
- Google JavaScript Style Guide
 - <https://google.github.io/styleguide/jsguide.html>
- w3schools.com
 - https://www.w3schools.com/js/js_conventions.asp

JavaScript Variables and Types

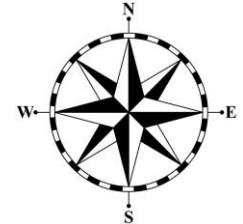


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- Variables and Types
- ES6 const
- Comments
- Operators
 - numeric, increment, assignment, logical, precedence
 - Comparison operators
- Strings
- String Operations

Variables

- Containers for storing data values
 - Can assign values to variables
 - Can change value held by variable over its lifetime
- Variable names must be unique
 - otherwise values can be overwritten
- Variable names can be
 - short such as x, y, I, j
 - descriptive such as total, numberOfStudents, age, person
 - must begin with a letter, \$ and _
 - are case sensitive thus age and AGE are different variables
 - can contain letters, digits, _ and \$
- Cannot be reserved words such as if or for

JavaScript Variables

- Can be declared using the **var** keyword

```
var count = 1;
```

- declares a *variable* called `count` which has the initial value 1
- note can omit **var** but then the value is a global
- using `var` defines a scope – which for the moment will also be global – but better style

- Can reuse `count` by assigning another value to it

```
count = count + 1;
```

- takes the current value of `count` and adds one to it
- the result is then stored back into `count`

JavaScript Variables

□ Examples of declaring various variables

```
// This is a comment
```

```
var count = 1;          // Variable called count storing a numeric integer value
var total = 42.6;       // Variable holding a numeric decimal number
var name = 'Jasmine';  // Variable storing a string
var flag = true;        // Variable holding a boolean value true
var age;                // a declared variable but without an assigned value
```

```
console.log('count:', count);
console.log('total:', total);
console.log('name:', name);
console.log('flag:', flag);
console.log('age:', age);
```

```
count: 1
total: 42.6
name: Jasmine
flag: true
age: undefined
age now is: 55
```

```
age = 55;              // assigning a value to the variable age
console.log('age now is:', age);
```

Let to define variable

- Since JavaScript ES 6 can use `let` to declare a variable

```
let count = 1;          // Variable called count storing a numeric integer value
let total = 42.6;       // Variable holding a numeric decimal number
let name = 'Jasmine';  // Variable storing a string
let flag = true;        // Variable holding a boolean value true
let age;                // a declared variable but without an assigned value
```

- Main difference is that at the top level, `let`, unlike `var`, does not create a property on the *global* object
- Best practice now considers `let` the default
 - only use `var` when you want to add to the global object

JavaScript Types

- JavaScript is not an un-typed language
- JavaScript is a dynamically typed language
 - the type of a variable depends on what it holds
 - at runtime if you try and do something with a variable
 - the result may depend on the types involved
 - e.g. + does different things depending on the types

JavaScript Types

- Each data value has a type

```
console.log('typeof count:', typeof count );
console.log('typeof total:', typeof total );
console.log('typeof name:', typeof name );
console.log('typeof flag:', typeof flag );
```

typeof count: number
typeof total: number
typeof name: string
typeof flag: boolean

- But type is dynamic; it can change
 - from number to string etc.

```
let myvar;           // Now myvar is undefined
myvar = 5;          // Now myvar is a Number
myvar = "John";     // Now myvar is a String
```

JavaScript Types

- **number**
 - represent integers and floating point numbers
 - e.g. 1, 4, 4.5, 1.33333
- **string**
 - represent string literal types
 - e.g. 'John', 'Paul', 'George', 'Ringo'
- **boolean**
 - represent true or false
- **Object**
 - represent data with (potentially) multiple members
 - {firstName = "John", lastName="Smith"}

undefined, null

- A variable that is declared but does not have a value assigned to it is undefined
- Can also assign undefined to a variable
 - removes any previous value
- Can also define null to a variable
 - which represents nothingness rather than being undefined
- Two values are logically equal
 - so can use either if need to remove a value
 - use the most semantically meaningful

undefined, null

```
// null represents nothingness - as opposed to undefined
let project = null;
console.log('project = null: ', project);

// can also set to undefined
project = undefined;
console.log('project = null: ', undefined);

// Are logical equal
console.log(null == undefined)
```

project = null: null
project = null:
undefined
true

JavaScript const

- Introduced in ES 6
- Indicates that a container is a constant
 - its value cannot be changed
 - useful for values that should not be changed
 - avoids accidental modification
 - *may allow runtime to perform optimizations*
 - *but that's not the main benefit*

```
const MAX_NUMBER = 100;  
// MAX_NUMBER = 200; // Can't reassign a value once it is set
```

JavaScript Comments

- Comments are for the programmer
 - to help understand the code
 - generally considered good style
- Two types of comment in JavaScript
 - Single line comments
 - indicated by a // - comment follows to end of line
 - // This is a Comment
 - let x = 10; // this is an assignment
 - Multi line comments
 - indicated by /* */

```
/*
This is a multi line comment
it can stretch over
as many lines as
you need
*/
```

Strings

- Are ordered sequences of characters
 - that are indexed from zero and have a length
- Can be declared using single or double quotes

```
let name1 = "John";
console.log('name1:', name1);
let name2 = 'John';
console.log('name2:', name2);
```

name1: John
name2: John

- Declaration option
 - allows single or double quotes to be embedded in a string

```
let longString = "They said 'Hello'";
console.log('LongString:', longString)
longString = 'They said "Hello"';
console.log('LongString:', longString)
```

LongString: They said 'Hello'
LongString: They said "Hello"

Numerical Operators

- Note modulus / remainder operator %
- Also note Exponential operator **

```
let x = 10, y =2;

console.log('x=', x, ', y=', y);
console.log('x + y:', x + y);
console.log('x * y:', x * y);
console.log('x / y:', x / y);
console.log('x % 3:', x % 3);
console.log('x - y:', x - y);
```

```
// Exponentiation operator - raises 1st to the power of the 2nd
console.log('x ** 2:', x ** 2);
```

```
// Can combine operators
var result = (x + y) * 10;
console.log('result = (x + y) * 10:', result);
```

```
x= 10 , y= 2
x + y: 12
x * y: 20
x / y: 5
x % 3: 1
x - y: 8
x ** 2: 100
result = (x + y) * 10: 120
```

Increment Operators

- Both pre and post increment
 - note the difference

```
console.log('x=', x, ', y=', y);

console.log('++x', ++x); // Pre Increment
console.log('x++', x++); // Post Increment

console.log('--x', --x); // Pre Decrement
console.log('x--', x--); // Post Decrement

console.log('x=', x, ', y=', y);
```

x= 10 , y= 2
++x 11
x++ 11
--x 11
x-- 11
x= 10 , y= 2

Assignment

- The assignment operator is used to assign a value to a variable
 - e.g. let x = 5;
 - e.g. x = x + 1;
 - e.g. x = 12;
- Actually an expression as x = 5; returns a value
 - e.g. 5

```
console.log('x = 5:', x = 5);
```

```
x = 5: 5
```

Assignment Operators

- Combine arithmetic operator with assignment
 - e.g. `x += 2;` is equal to
 - `x = x + 2;`

```
console.log('x =', x  
console.log('x += 2:', x += 2)  
console.log('x -= 2:', x -= 2)  
console.log('x *= 2:', x *= 2)  
console.log('x /= 2:', x /= 2)
```

```
x = 10  
x += 2: 12  
x -= 2: 10  
x *= 2: 20  
x /= 2: 10
```

String Operations

- String manipulation very common in JavaScript
- Many String operations available
 - not many are invoked using the dot notation on the actual string
 - they do not effect the original string but create a new one
 - e.g. 'John'.toUpperCase(); // returns John as JOHN

```
let name = 'Phoebe Davies';
console.log('name.length:', name.length);

// Index of methods
console.log("name.indexOf('Davies'):", name.indexOf('Davies'));
console.log('name.lastIndexOf("Davies"):', name.lastIndexOf("Davies"));
console.log("name.indexOf('Smith'):", name.indexOf('Smith'));
```

```
name.length: 13
name.indexOf('Davies'): 7
name.lastIndexOf("Davies"): 7
name.indexOf('Smith'): -1
```

String Operations

```
let name = 'Phoebe Davies';
// Strings are indexed from Zero
console.log('name.slice(2, 6):', name.slice(3, 6));
// If omit second index takes to end of string
console.log('name.slice(2):', name.slice(2));
// Can also count from end of String
console.log('name.slice(-2):', name.slice(-2));

// Also have Substring - like slice but can't take negative numbers
console.log('name.substring(2, 6):', name.substring(2, 6));
console.log('name.substring(2):', name.substring(2));

// And substr() - like slice but 2nd param indicates length of string
console.log('name.substr(2, 6):', name.substr(2, 6));
console.log('name.substr(2):', name.substr(2));
console.log('name.substr(-2):', name.substr(-2)); // starts at end

// Extracting individual characters
console.log('name.charAt(3):', name.charAt(3));
```

```
name.slice(2, 6): ebe
name.slice(2): oebe Davies
name.slice(-2): es
name.substring(2, 6): oebe
name.substring(2): oebe Davies
name.substr(2, 6): oebe D
name.substr(2): oebe Davies
name.substr(-2): es
name.charAt(3): e
```

String Operations

```
// Replacing String content
console.log('name.replace("Davies", "Smith"):',  
           name.replace("Davies", "Smith"));
// Above returns a new string - doesn't affect original string
console.log('name:', name);
// Converting to Upper or Lower Case
console.log('name.toUpperCase():', name.toUpperCase());
console.log('name.toLowerCase():', name.toLowerCase());
// Convert a number to a string
var x = 123;
var y = 456;
console.log('x.toString() + y.toString():',  
           x.toString() + y.toString());
// Trim operator - removes white space
var location = '    London    ';
console.log('location:', location);
console.log('location.trim():', location.trim());
```

name.replace("Davies", "Smith"): Phoebe
Smith
name: Phoebe Davies
name.toUpperCase(): PHOEBE DAVIES
name.toLowerCase(): phoebe davies
x.toString() + y.toString(): 123456
location: London
location.trim(): London

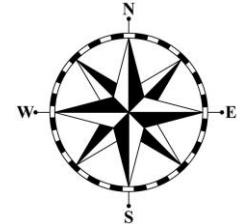
Flow of Control in JavaScript



Toby Dussek

john.hunt10@gmail.com





Plan for Session

- Logical operators
- if statement
 - if
 - if .. else
 - if .. else if .. else
- switch statement
- while loops
- do loops
- for loops

Logical Operators

- All return true or false
 - based on some logical test
 - e.g. is x greater than y – $x > y$
- Not different types of equality
 - `==` indicates logical equality of value
 - `===` indicates equal value and type
- Thus `10 == '10'` will return true as JavaScript converts string to an int for you
- But `10 === '10'` will return false as they are different types

Logical Operators

```
let x = 10, y = 20;  
console.log('x == y:', x == y);  
console.log('x == 10:', x == 10);  
// JavaScript converts the string to a number  
console.log("x == '10':", x == '10');  
console.log("x === 10:", x === 10);  
console.log("x === '10':", x === '10');  
console.log("x != y:", x != y);  
console.log("x !== y:", x !== y);  
console.log("x > y:", x > y);  
console.log("x >= y:", x >= y);  
console.log("x < y:", x < y);  
console.log("x <= y:", x <= y);  
// Ternary operator  
console.log("x > y ? 'greater' : 'lessthan':",  
          x > y ? 'greater' : 'lessthan');
```

x == y: false
x == 10: true
x == '10': true
x === 10: true
x === '10': false
x != y: true
x !== y: true
x > y: false
x >= y: false
x < y: true
x <= y: true
x > y ? 'greater' : 'lessthan':
lessthan

Logical Operators

- Also logical AND (`&&`), OR (`||`) and NOT (`!`)
 - Note: bitwise versions are single characters

```
let flag1 = true, flag2 = false;  
  
console.log('flag1 && flag2:', flag1 && flag2);  
console.log('flag1 || flag2:', flag1 || flag2);  
console.log('! flag1:', ! flag1)
```

```
// Be careful of bitwise operators  
// Bitwise AND really 0101 & 0001 => 0001  
console.log('5 & 1:', 5 & 1);
```

```
flag1 && flag2: false  
flag1 || flag2: true  
! flag1: false  
5 & 1: 1
```

if statement

□ Basic Structure

```
if (<condition-evaluating-to-boolean>)
    statement
```

- Note curly brackets indicate a block of statements

□ Examples

```
let age = 17;
console.log('age:', age);

// Simple if statement
if (age < 18)
    console.log('Under 18');

// Better style - with curly brackets
if (age < 18) {
    console.log('Under 18');
}
```

age: 17
Under 18
Under 18

Using else and else if

- Optionally can have else

```
if (age < 18) {  
    console.log('Under 18');  
} else {  
    console.log('18 or over');  
}
```

- Also additional conditional tests

```
if (age < 13) {  
    console.log('Child');  
} else if (age < 20) {  
    console.log('Teenager');  
} else {  
    console.log('Over 20');  
}
```

Nesting if statements

- Can nest one if statement inside another

```
let snowing = true;  
temp = -1;  
console.log('-----');  
if (temp < 0) {  
    console.log('It is freezong');  
    if (snowing) {  
        console.log('Put on Boots');  
    }  
    console.log('Time for Hot Chocolate');  
}  
console.log('-----');
```

It is freezong
Put on Boots
Time for Hot Chocolate

switch statement

- Selects one option from many

- useful when there are many alternatives to select from

- tests cases to find a match
- Note use of break to escape a block – will drop through if not included
- default only runs if no case matches

```
switch (age) {  
    case 0:  
        console.log('Baby');  
        break;  
    case 17:  
        console.log('Can Drive');  
        break;  
    case 18:  
        console.log('Can drink');  
    case 21:  
    case 40:  
    case 60:  
        console.log('Its a big birthday');  
    default:  
        console.log('Some other age');  
}
```

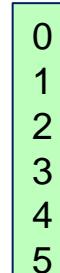
While loops

- Basic form

```
while (<test-condition-is-true>)  
  statement or statements
```

- An example

```
let i = 0;  
while (i < 6) {  
  console.log(i);  
  i++;  
}
```



0
1
2
3
4
5

- Test happens at start of loop

Do loops

- Basic form

```
while (<test-condition-is-true>)  
    statement or statements
```

- An example

```
let i = 0;  
do {  
    console.log(i);  
    i++;  
} while (i < 10);
```

```
0  
1  
2  
3  
4  
5
```

- Note test happens at end of loop

For Loop

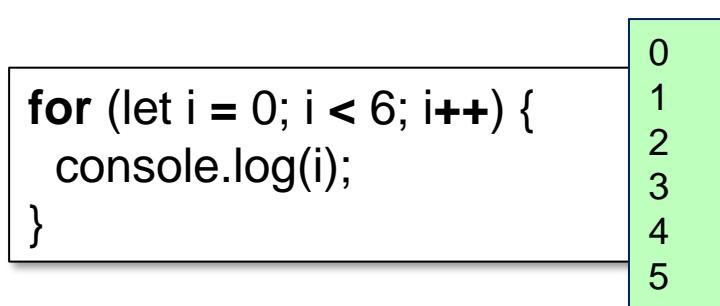
□ Basic from of the for loop

```
for (statement1; statement2; statement3) {  
    statements to be executed  
}
```

- Note meaning of each section
- statement1 - (optional) executed at start of loop
 - usually used to initialise the loop variable
- statement2 – defines condition for executing the code block
 - must be true to continue looping
- statement3 – is executed each time round the loop
 - usually used to change the value of the loop variable

For Loop

□ Basic For loop



□ The keyword let was introduced in ES6

- scopes the variable i to just the loop
- older JavaScript used var which introduced a new variable that still existed after the loop
- best practice will use let unless variable is needed after the loop

Nesting Loops

- Can nest any loop inside another
 - e.g. while inside do, do inside while, for inside while etc.

```
for (let i = 0; i < 6; i++) {  
    for (let j = 0; j < 6; j++) {  
        console.log("i * j =", i * j);  
    }  
}
```

```
i * j = 0  
...  
i * j = 3  
i * j = 6  
i * j = 9  
...  
i * j = 15  
i * j = 20  
i * j = 25
```

Functions in JavaScript



Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- Defining Functions
 - example function definitions
 - functions with parameters
 - return values
 - function expressions
 - function alias
- Higher Order Functions
- Functions can return functions
- Arrow Functions
- Variable Scope
- Block Scope
- Math functions

Defining Functions

□ Basic Syntax

```
function fname(parameter list) {  
    statement  
    statement(s)  
}
```

- all (named) functions are defined using the *keyword* function
- can have *named* and *anonymous* functions
- parameter list is optional
- curly bracket marks end of *function header* before *function body*
- one or more statements form function body
- function may or may not return a value

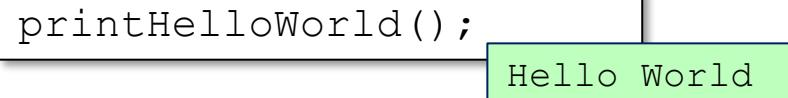
Example Function definitions

- Simplest function – no params no return value

```
function printHelloWorld() {  
    console.log('Hello World');  
}
```

- Can execute via

```
printHelloWorld();
```



- don't forget the brackets
- if forget the brackets then don't invoke the function
- just reference the function object

```
console.log('printHelloWorld:', printHelloWorld);
```

```
printHelloWorld: function printHelloWorld() {  
    console.log('Hello World');  
}
```

Functions with parameters

- Functions can take parameters
 - used within the body of the function

```
function printMessage(message) {  
    console.log(message);  
}
```

- Execute by providing values for parameters

```
printMessage("Hello JavaScript World");  
printMessage("Welcome");  
printMessage("Ola");
```

Hello JavaScript World
Welcome
Ola

- Can have multiple parameters

```
function printInformation(info1, info2, info3) {  
    console.log(info1, info2, info3);  
}  
  
printInformation('John', 'was', 'here');
```

John was here

Functions can return values

- Can return a value using `return` statement

```
function addTen(i) {  
    return i + 10;  
}
```

- Execute by providing argument value

```
console.log('addTen(5):', addTen(4));  
  
// # Store result from addTen in a variable  
var result = addTen(5);  
console.log('result = addTen(5):', result);  
  
// Use the result returned from square in a conditional expression  
if (addTen(3) < 15) {  
    console.log('Still less than 15');  
}
```

```
addTen(5): 14  
result = addTen(5): 15  
Still less than 15
```

Function Expressions

- Can define anonymous functions
 - can assign this to variable

```
var sum = function(x, y) {  
    return x + y;  
}
```

- can use sum as if it was a function

```
console.log('sum(4, 5):', sum(4, 5));  
result = sum(3, 4);
```

```
console.log(result);
```

sum(4, 5): 9
7

Function Alias

- Can create an alias for a function

- Given

```
function square(i) {  
    return i * i;  
}
```

- Can call the function `square`
- or assign it to a variable – note no brackets

```
console.log('square(5) : ', square(5));  
  
var doit = square;  
console.log('doit(5) : ', doit(5));
```

square(5): 25
doit(5): 25

Higher Order Functions

- Are functions that
 - either take one or more functions as a parameter
 - return a function as a result
 - or do both
- The processor function is a higher order function

```
function processor(func, x, y) {  
    return func(x, y);  
}
```
- Can pass it an anonymous function that will be *run*

```
var result = processor(function(a, b) { return a + b }, 4, 5);  
console.log('result:', result);
```

result: 9

Higher Order Functions

- Can use apply with any function
 - that meets the two-parameter protocol
 - note do not need to be anonymous

```
function multByTwo(num1, num2) {  
    return num1 * num2;  
}
```

```
function multByFive(num1, num2) {  
    return num1 + num2 * 5  
}
```

```
function addOne(num1, num2) {  
    return num1 + num2 + 1  
}
```

12
35
6

```
console.log(processor(multByTwo, 3, 4))  
console.log(processor(multByFive, 5, 6))  
console.log(processor(addOne, 3, 2))
```

Functions can return functions

- Function can return another function
 - typically acts as a factory for functions
 - note below parameter x is closed in the function multiplier

```
function createMultiplier(x) {  
    function multiplier(y) {  
        return x * y;  
    }  
    return multiplier;  
}
```

```
var func1 = createMultiplier(10);  
console.log(func1); // Does not invoke the function  
console.log("func1(5):", func1(5));
```

```
var doubleit = createMultiplier(2);  
console.log('doubleit(5):', doubleit(5));
```

```
var trebleit = createMultiplier(3);  
console.log('trebleit(5):', trebleit(5));
```

[Function:
multiplier]
func1(5): 50
doubleit(5): 10
trebleit(5): 15

Arrow Functions

- Introduced in ES6
 - as short hand form of an anonymous function
 - aka fat arrow
- Basic Structure

```
(argument list) => {  
    statement or statements  
}
```

- Arrow function with no parameters

```
var func = () => {  
    console.log('Hello World');  
}  
  
func();
```

Hello World

Arrow Functions

- Have a short hand form

```
var func2 = () => console.log('Hello World');
```

- Return a value by default

```
var hello = () => "Hello World!";
console.log('hello():', hello());
```

Hello World!

- Can take parameters

```
var printMessage = (message) => {
    console.log(message);
}
printMessage('Hello JavaScript world');
```

Hello JavaScript world

```
// Short hand form
printMessage = message => console.log(message);
```

Variable Scope

- Functions can access global variables

```
var role = 'Writer';
function printRole() {
    // accesses a global function
    console.log('role inside function:', role);
}
```

- Can define variables scoped to just function

```
function printIt() {
    var ageLimit = 21; // variable that is local to a function
    console.log('ageLimit inside function:', ageLimit);
}
printIt();

// can't access ageLimit here - it is not defined
// console.log(ageLimit);
```

Block Scope

□ Introduced in ES6

- previously function vars visible anywhere in function
- use let to scope a variable to just a block of code

```
function myNewFunction() {  
    // flag not available here  
    console.log('In myNewFunction');  
    // Define a block of code  
    {  
        let flag = true  
        console.log('flag:', flag);  
    }  
    // flag not available here  
    // console.log('flag:', flag);  
}
```

□ Use let in preference to var whenever possible

Math functions

□ Lots of functions provided by Math

```
console.log(Math.round(4.7));      // prints 5
console.log(Math.round(4.4));      // prints 4
console.log(Math.ceil(4.4));       // prints 5
console.log(Math.floor(4.7));      // prints 4

console.log(Math.pow(8, 2));        // prints 64
console.log(Math.sqrt(64));         // prints 8
console.log(Math.abs(-4.7));        // prints 4.7

console.log(Math.min(0, 150, 30, 20, -8, -200)); // prints -200
console.log(Math.max(0, 150, 30, 20, -8, -200)); // prints 150

console.log(Math.random());         // prints a random number between 0 and 1
```

Classes And Objects in JavaScript

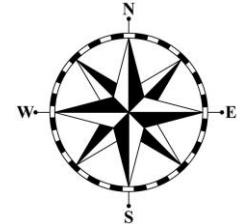


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

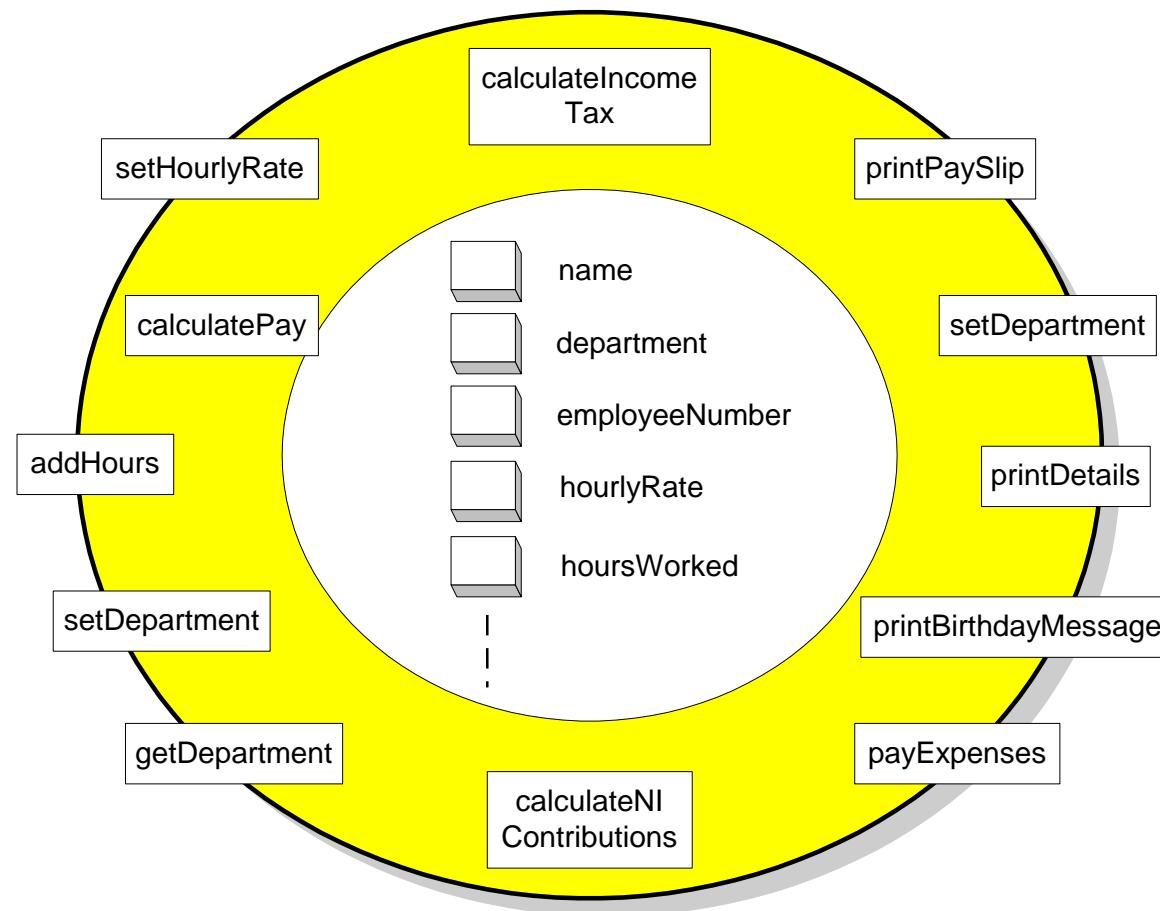
- OOP
- Objects in JavaScript
- Classes in ES6
- Static Behaviour
- The this reference

What is an Object?

- Objects fundamental building blocks
 - in object oriented programming
- Originated in Simula in 1960s
- Combination of data and code
 - Example employee object
 - Behaviour as methods
 - Characteristics as variables / attributes
- Objects can be
 - Physical, conceptual, transient, persistent



Employee Object



Objects in JavaScript

- JavaScript objects can have multiple attributes
 - act as containers for named values
 - aka attributes or properties
- An object can be assigned to a variable as a whole

```
var car = {brand: 'Porsche', model: '911', colour: 'Red'};  
console.log('car:', car);
```

```
car: { brand: 'Porsche', model: '911', colour: 'Red' }
```

Objects in JavaScript

- Layout is often important with objects
 - to make them easier to read / understand

```
var person = {  
    name: 'John',  
    age: 55  
}  
console.log('person:', person)
```

```
person: { name: 'John', age: 55 }
```

- note code layout does not effect how person object is printed

Objects in JavaScript

- Can access individual properties
 - using either '.' (dot) or index notations

```
// Access object properties
console.log('person.name is person.age:', person.name, 'is',
            person.age);

// alternatively can use [] index notation
console.log("person['name'] is person['age']:", person['name'],
            'is', person['age']);
```

person.name is person.age: John is 55
person.name is person.age: John is 55

- dot notation is more common
- but index notation allows for string selection of properties

Objects in JavaScript

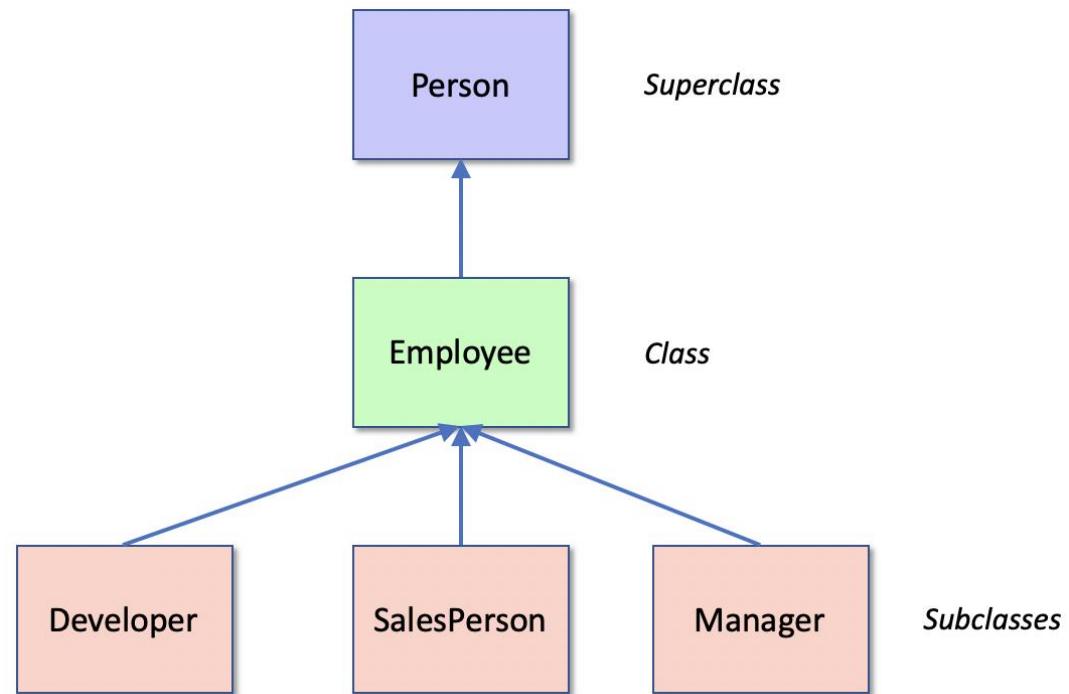
- Objects can have behaviour
 - functions that can be called on the object
 - have a special variable *this* allows self reference to object

```
var person = {  
    name: 'John',  
    age: 55,  
    birthday: function() {  
        this.age = this.age + 1;  
        console.log('Happy Birthday, you are now', this.age);  
    }  
}  
  
person.birthday();
```

Happy Birthday, you are now 56

OO Terminology

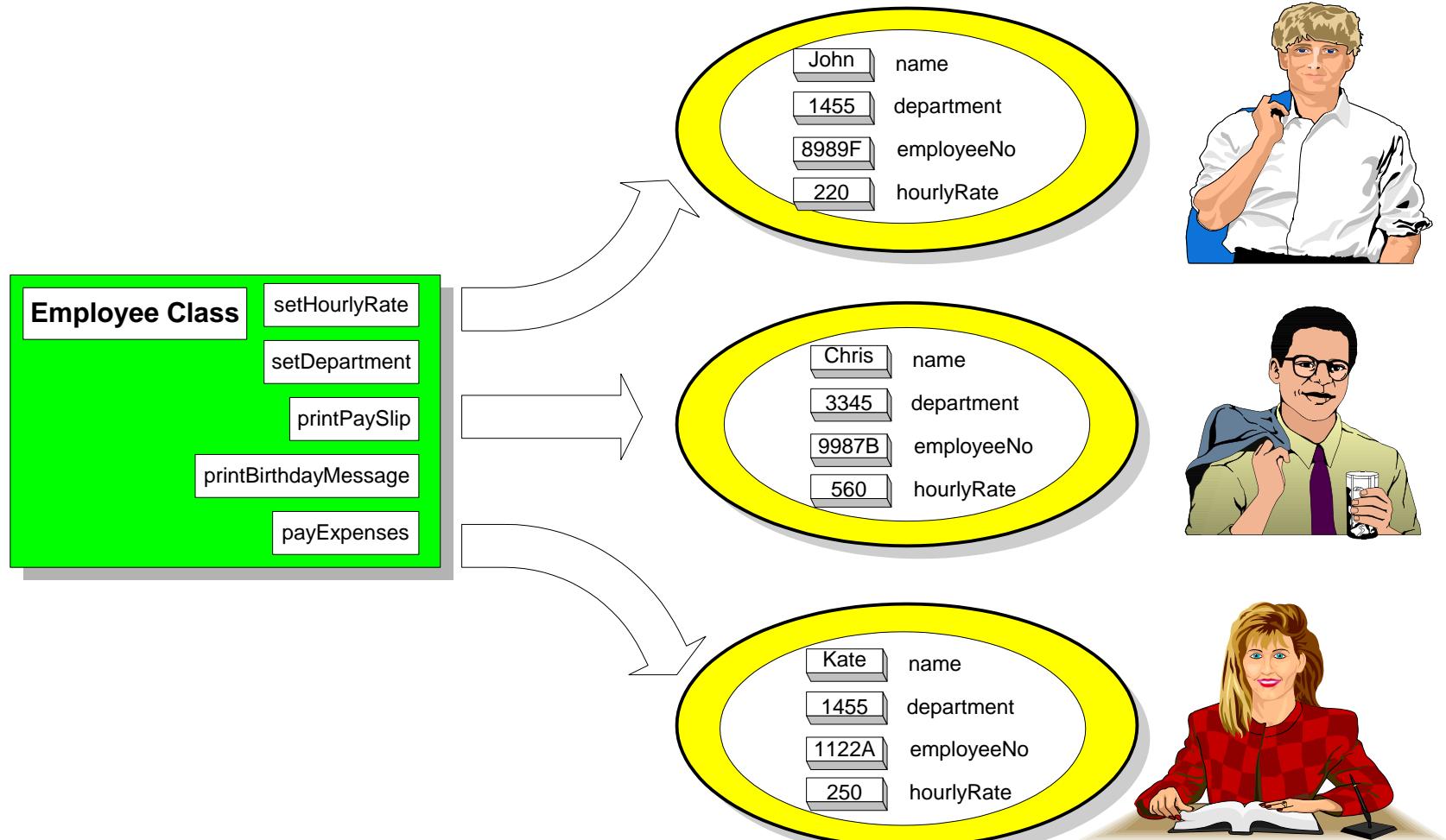
- Class
- Subclass
- Superclass
- Instance / Object
- Instance variable
- Method



What is a Class?

- Treat classes as templates for objects
- Objects are examples of classes
- Single class can have multiple objects
- Code held by class
- Variables held by objects
- In OO languages classes also define types

Class Object Relationship



Creating User Defined classes

- Introduced in ES6
- Class definition has the format

```
class NameOfClass {  
    constructor() {}  
    attributes  
    methods  
}
```

- Class Person

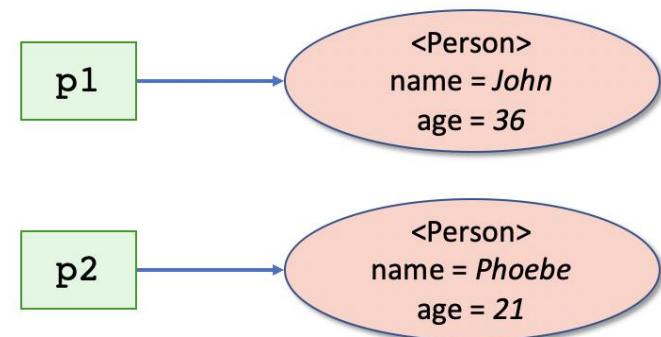
```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```

- name follows Camel Case conventions with no spaces (or _)
- defines two attributes name and age
- also defines constructor used to initialise instances

Using Classes

- Can create a new instance of a class
 - using the keyword new
 - must provide any parameters required by the constructor

```
var p1 = new Person("John", 36);
var p2 = new Person("Phoebe", 21);
```



- creates two instances of the class `Person`
- one with the name *John* and the age 25
- and one with the name *Phoebe* and the age 22
- each object has its own area in memory

Using Classes

- Can access properties name and age of instances

```
console.log(p1);
console.log(p1.name, 'is', p1.age);

console.log(p2);
console.log(p2.name, 'is', p2.age);
```

```
Person { name: 'John', age: 25 }
John is 25
Person { name: 'Denise', age: 22 }
Denise is 22
```

- Note instances have a default way of being printed
 - includes the class name at the start followed by the properties and their values

Printing Objects

- Can override default behaviour using `toString()`
 - but this is only used by a template or when used explicitly

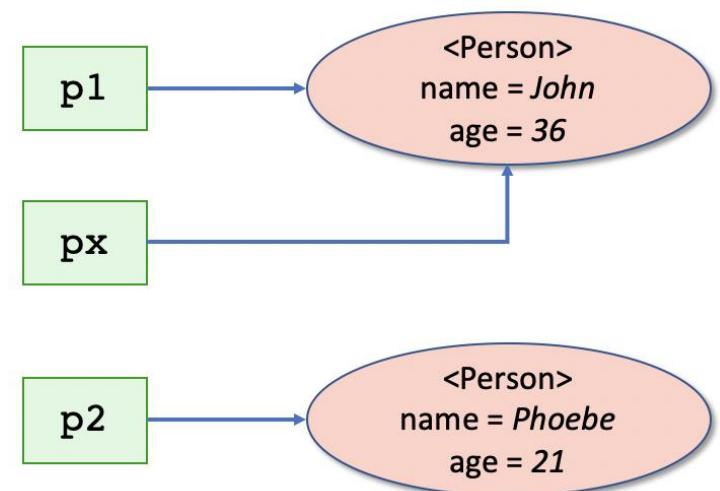
```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
    toString() {  
        return "Person(" + this.name + "," + this.age + ")";  
    }  
}  
  
var p1 = new Person("John", 36);  
var p2 = new Person("Phoebe", 21);  
  
console.log('p1:', p1);  
console.log('p1.toString():', p1.toString());  
console.log(` ${p1} `);  
  
console.log('P2:', p2);  
console.log('p2.toString():', p2.toString());  
console.log(` ${p2} `);
```

p1: Person { name: 'John', age: 36 }
p1.toString(): Person(John,36)
` \${p1} `: Person(John,36)
P2: Person { name: 'Phoebe', age: 21 }
p2.toString(): Person(Phoebe,21)
` \${p2} `: Person(Phoebe,21)

Be Careful with Assignment

- Assignment copies address of object

```
var p1 = new Person("John", 36);
var p2 = new Person("Phoebe", 21);
var px = p1;
```



- May not be obvious when logged

```
console.log(p1);
console.log(px);
```

John is 36
John is 36

Adding Behaviour

□ Add some behaviour

- Method names follow variable naming conventions

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    birthday() {  
        this.age = this.age + 1;  
        console.log("Happy Birthday you are now", this.age);  
    }  
  
    toString() {  
        return "Person(" + this.name + "," + this.age + ")";  
    }  
}
```

```
var p1 = new Person("John", 36);  
  
p1.birthday();
```

Happy Birthday you are now 37

Adding Behaviour

- Methods can take parameters and / or return values

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
    // ....  
  
    getFormattedName() {  
        return "'" + this.name.toUpperCase() + "'";  
    }  
  
    isTeenager() {  
        if ((this.age > 12) && (this.age<20)) {  
            return true;  
        } else {  
            return false;  
        }  
    } }
```

```
console.log(p1.isTeenager());  
var formatedName = p1.getFormattedName();  
console.log(formatedName);
```

false
'JOHN'

Setters and Getters

- Can also define getters and setters
 - allows some protection for properties
 - can verify that values are valid
 - defined via the get and set keywords
 - often underlying property pre-fixed with '_'

Setters and Getters

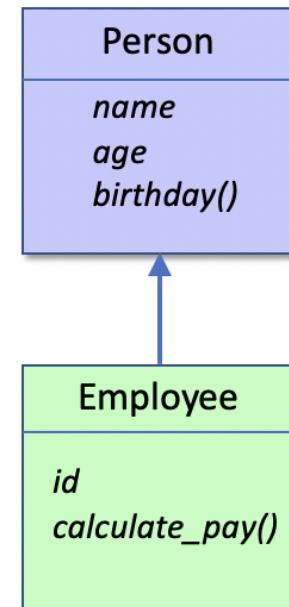
```
class Person {  
    constructor(name, age) {  
        this._name = name;  
        this.age = age;  
    }  
    // Define setters and getters for property  
    get name() {  
        console.log("In getter for name");  
        return this._name;  
    }  
    set name(n) {  
        console.log("In setter for name with ", n);  
        this._name = n;  
    } // ...  
}
```

```
person1 = new Person("John", 55);  
person1.birthday();  
console.log(person1.name);  
person1.name = 'Bob';  
console.log(person1.name);
```

In getter for name
John
In setter for name with Bob
In getter for name
Bob

What is Inheritance?

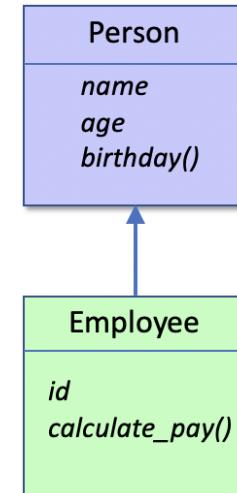
- Allows one class to inherit features defined in another class
- Provides a form of reuse
- Allows data and behaviour to be defined in one place
 - and thus maintained in one place
- Person defines
 - name, age attributes and birthday()
- Employee
 - *inherits* the definitions of name, age and birthday()
 - and adds id property and calculate_pay() method



Declaring Inheritance

- Given the following classes

```
class Person {  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
    birthday() {  
        this.age = this.age + 1;  
        console.log("Happy Birthday you are now",  
            this.age);  
    }  
}
```



```
class Employee extends Person {  
    constructor(name, age, id, rate) {  
        super(name, age); // super must be first statement in constructor  
        this.id = id;  
        this.rate = rate;  
    }  
    calculatePay(hours) {  
        return this.rate * hours;  
    }  
}
```

JavaScript Arrays

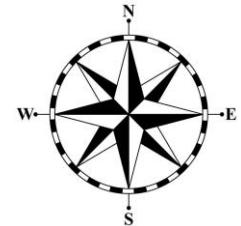


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- What is an Array?
- Creating an Array
- Avoid using new with Arrays
- Accessing Elements in Arrays
- Changing an Element
- Arrays of Mixed Types & Length
- Arrays are objects (but not the same)
- Array Iteration
- Array Methods

What is an Array?

- A type of object that can hold an ordered sequence of values
- An array can hold any number of values
- Values can be of any type
 - numbers, strings, booleans, objects, instances of classes etc.
- Arrays can be treated individual or by accessing values
- Values have an index
 - starting at zero
 - but do not have a *name*

Creating an Array

- Arrays are created using '[]'

- with each value separated by a ','

```
[item1, item2, item3, item4];
```

- called an array literal
 - for example

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];
console.log('brands:', brands);
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
```

- creates an array literal of 4 elements
 - all of the type string
 - numbered 0, 1, 2 and 3
 - the array is of length 4

Avoid using new with Arrays

- Long hand from for creating an array uses the key word new
- And the class Array

```
let fruit = new Array('Banna', 'Apple', 'Raspberry');  
console.log('fruit:', fruit);
```

- Does the same thing as the array literal
 - but generally considered poor style
 - as longer, more verbose, easier to mis read
 - also execution speed may not be as good

Accessing Elements in Arrays

- Array elements accessed using index number
 - remember indexed from zero
 - used with []

```
console.log('brands[0]:', brands[0]);  
console.log('brands[1]:', brands[1]);  
console.log('brands[2]:', brands[2]);  
console.log('brands[3]:', brands[3]);
```

```
brands[0]: BMW  
brands[1]: Apple  
brands[2]: Microsoft  
brands[3]: BT
```

Changing an Element

- Can change an element of an array using [] and index with assignment

```
console.log('Modify array element with index 0');
brands[0] = 'Ford';
console.log('brands[0]:', brands[0]);
```

Modify array element with index 0
brands[0]: Ford

- Can delete elements using keyword delete and array index

```
console.log('Deleting array elements');
console.log('brands:', brands);
delete brands[1];
console.log('brands:', brands);
```

Deleting array elements
brands: ['Ford', 'Apple', 'Microsoft', 'BT']
brands: ['Ford', <1 empty item>, 'Microsoft', 'BT']

Arrays of Mixed Types & Length

- Can have arrays of different types

```
// Mixed type arrays
let data = ['John', 55, 23.55, true];
console.log('data:', data);
```

```
data: [ 'John', 55, 23.55, true ]
```

- Arrays have a length

```
console.log('brands.length:', brands.length);
```

```
brands.length: 4
```

Array Iteration

- Several approaches to array iteration
 - traditional for loop
 - for-in
 - for-of
 - forEach()
 - map()
 - filter()

Array Iteration for loop

□ Traditional (old school) approach

- Use a for loop with I looping from 0 to *length* of array
- very common in C-like languages

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];

for (let i = 0; i < brands.length; i++) {
    console.log(brands[i]);
}
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
BMW
Apple
Microsoft
BT
```

- long winded and potentially error prone
- must remember to be < (less than) array.length
- and to increment i

Array Iteration for-in loop

- Newer form of for loop

- uses `in` keyword which loops through each index

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];

for (let i in brands) {
    console.log(brands[i]);
}
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
BMW
Apple
Microsoft
BT
```

- but still need to use index to access elements

Array Iteration for-of loop

- ❑ Newer from that loops through each value in the array in turn

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];

for (let brand of brands) {
    console.log("brand:", brand);
}
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
BMW
Apple
Microsoft
BT
```

- ❑ Good modern JavaScript style

Array forEach() method

- Modern functional programming style of iteration
- Supply a function that is applied to each value in turn

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];

brands.forEach(e => console.log(e.toUpperCase()));
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
BMW
APPLE
MICROSOFT
BT
```

Array map() method

- Like forEach() but collects results up into a new array of the same size as the original

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];

let lowerCaseBrands = brands.map(e => e.toLowerCase());
console.log("lowerCaseBrands:", lowerCaseBrands);
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
lowerCaseBrands: [ 'bmw', 'apple', 'microsoft', 'bt' ]
```

Array filter() method

- Used to select only those values that pass the filter function

```
let brands = ["BMW", 'Apple', 'Microsoft', 'BT'];

let longNameBrands = brands.filter(e => e.length > 5);
console.log("longNameBrands:", longNameBrands);
```

```
brands: [ 'BMW', 'Apple', 'Microsoft', 'BT' ]
longNameBrands: [ 'Microsoft' ]
```

Array Methods

□ Many additional methods

```
let songs = ['Set Fire to the Rain', 'Guilty', 'Havana']
console.log('songs:', songs);
```

```
// Convert an array to a string
let label1 = songs.toString();
console.log('label1:', label1);
```

```
// join elements in an array using link string
let label2 = songs.join('; ');
console.log('label2:', label2);
```

```
// Merging Concatenating Arrays
let edSongs = ['Shape of You', 'Perfect',
              'Castle on the Hill'];
let newSongs = songs.concat(edSongs);
console.log('newSongs:', newSongs);
```

```
songs: [ 'Set Fire to the
Rain', 'Guilty', 'Havana' ]
label1: Set Fire to the
Rain,Guilty,Havana
label2: Set Fire to the Rain;
Guilty; Havana
newSongs: [ 'Set Fire to the
Rain',
            'Guilty',
            'Havana',
            'Shape of You',
            'Perfect',
            'Castle on the Hill' ]
```

Array Methods

```
// Pop removes the last element from an array
console.log('songs:', songs);
console.log('songs.pop()');

let song = songs.pop();
console.log('song:', song);
console.log('songs:', songs);
// Push adds a new element to the end of the array
console.log("songs.push(Livin' La Vida Loca)");
songs.push("Livin' La Vida Loca");
console.log('songs:', songs);
// Shift removes the first element in the array
console.log("songs.shift()");
let song1 = songs.shift();
console.log('song1:', song1);
// Unshift adds an element to the start of array
console.log('songs:', songs);
console.log("songs.unshift('Space Oddity')");
let newLength = songs.unshift('Space Oddity');
console.log('songs:', songs);
console.log('newLength:', newLength);
```

```
songs: [ 'Set Fire to the
songs.pop()
song: Havana
songs: [ 'Set Fire to the
Rain', 'Guilty' ]
songs.push(Livin' La Vida
Loca)
songs: [ 'Set Fire to the
Rain', 'Guilty', '\'Livin\''
La Vida Loca' ]
songs.shift()
song1: Set Fire to the Rain
songs: [ 'Guilty', '\'Livin\''
La Vida Loca' ]
songs.unshift('Space Oddity')
songs: [ 'Space Oddity',
'Guilty', '\'Livin\' La Vida
Loca' ]
newLength: 3
```

Destructuring

Assignment feature added in ES6

- can unpack values from arrays, strings or properties from objects, into distinct variables

```
const brands = ["BMW", 'Apple', 'Microsoft', 'BT'];
console.log(brands);
console.log('-----');

const [b1, b2, b3, b4] = brands;
console.log(b1);
console.log(b2);
console.log(b3);
console.log(b4);
console.log('-----');

let [a1, a2] = brands;
console.log(a1);
console.log(a2);
console.log('-----');

let [x1, , , x2] = brands;
console.log(x1);
console.log(x2);
console.log('-----');

let [head, ...tail] = brands
console.log(head);
console.log(tail);
console.log('-----');

let x, y;
[x, y] = brands;
console.log(x);
console.log(xy);
```

```
[ 'BMW', 'Apple', 'Microsoft',
  'BT' ]
-----
BMW
Apple
Microsoft
BT
-----
BMW
Apple
-----
BMW
BT
-----
BMW
[ 'Apple', 'Microsoft', 'BT' ]
-----
BMW
Apple
```

Dstructuring

□ Strings

```
const msg = "Hello";
let [a,b,...rest] = msg;
console.log(a);
console.log(b);
console.log(rest);
```

H
e
['l', 'l', 'o']

□ Objects

```
const p1 = {age: 42, name: "John"};
const {age, name} = p1;
```

```
console.log(age);
console.log(name);
```

```
const {age: foo, name: bar} = p1;
console.log(foo);
console.log(bar);
```

```
let a, b;
// parentheses required
// when using object
// without a declaration.
({a, b} = {a: 1, b: 2});
console.log(a);
console.log(b);
```

```
console.log(b);
console.log(rest);
```

42
John
42
John
1
2

Error Handling in JavaScript

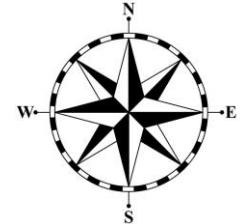


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- Exception Handling
- Exception Handling Terminology
- Local Handling
- Exceptions and Errors in JavaScript
- Exception Handling Example

Exception Handling

- Based on C++ / similar to Java & Python
- Exception *raised* by instantiating an exception class
- Then thrown from where it is generated
- And Caught where it is handled
- Exception handling blocks manage how exceptions handled
 - `try {} catch {} finally {}` blocks
 - `throw` statement creates custom errors

Exception Handling Terminology

- Common terminology used in JavaScript as in other languages

Term	Description
Exception / Error	An error which is generated at runtime.
Raising an exception	Generating a new exception.
Throwing an exception	Triggering a generated exception.
Handling an exception	Processing code that deals with the error.
Handler	The code that deals with the error (referred to as the catch block).

Local Handling

- ❑ Uses the try {} catch {} blocks

```
try {  
    function dosomething() {  
        raise an error  
    }  
} catch (ex) {  
    // ...  
}
```

- ❑ except catches exception
 - runs block when exception occurs
- ❑ If no exception occurs catch block is not executed

Exceptions and Errors in JavaScript

- Can be any type
 - can be numbers, strings, booleans, objects
- May be instances of specific error classes
- For example ReferenceError
 - used when a variable is accessed but has not been declared
 - includes a stack trace of how code was called

```
ReferenceError: count is not defined
    at Object.<anonymous> (/Users/Shared/workspaces/visualstudiocode/javascript-intro/08-errors/try-catch-syntax-errors.js:3:15)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
```

Exception Handling Example

- ❑ runcalc can throw the ZeroDivisionError

```
try {
    runcalc(6);
} catch(ex) {
    console.log('oops');
}
```

- ❑ Enters try block
- ❑ If all ok runs runcalc and jumps to line after last }
- ❑ If runcalc throws exception,
 - jump to catch block and
 - runs code in catch block

Exception Handling Example

- Can define a finally block

```
try {
    runcalc(6);
} catch(ex) {
    console.log('oops');
} finally {
    console.log("Always runs");
}
```

- Finally block is run whether there is an error not
- is the last block of code run for the try-catch-finally group

JavaScript in Web Pages

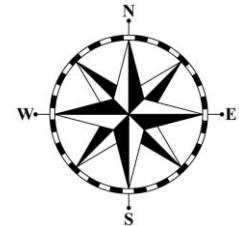


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- Browser Based JavaScript
- JavaScript in a Web Page
- JavaScript in the Body
- JavaScript in the Head
- JavaScript in External Files
- Advantages of External Files
- Dialogs in JavaScript

Browser Based JavaScript

- JavaScript can be executed within a web browser
- Different browsers (and versions) support different versions of JavaScript
 - this can cause problems for developers who want to run their applications in as wide a range of browsers as possible
 - most of this course has focussed on ES6 (ECMAScript 2015)
 - as it has wide spread support amongst newer browsers

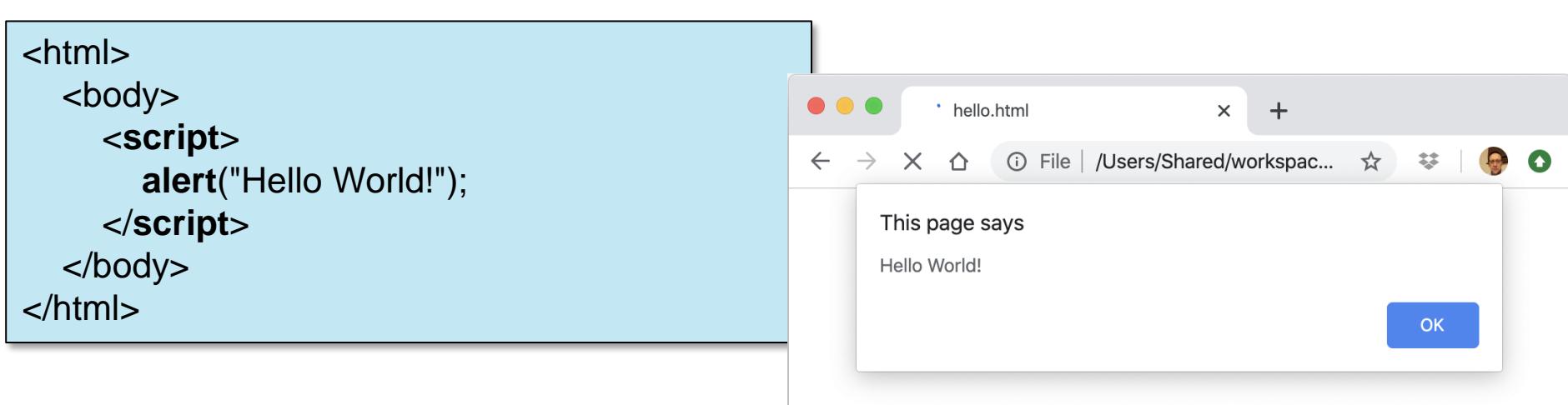
Browser	Version	Date
Chrome	51	May 2016
Firefox	54	Jun 2017
Edge	14	Aug 2016
Safari	10	Sep 2016

JavaScript in a Web Page

- JavaScript can be defined in 3 locations
 - In the body of a web page
 - In the head of a web page
 - In an external file
- JavaScript can run in
 - either the body or the head
- JavaScript defined between <script> </script> tags
 - note older JavaScript examples may use
 - <script type="text/javascript">
 - The type attribute is no longer required
 - JavaScript is the default scripting language in HTML

JavaScript in the Body

- Can place <script></script> tags in body of page
- Code will be executed when page is loaded
 - Placing scripts at the bottom of the <body> element improves the display speed,
 - because script interpretation increases the time taken to render the page



The image shows a split-screen view. On the left, a code editor displays the following HTML code:

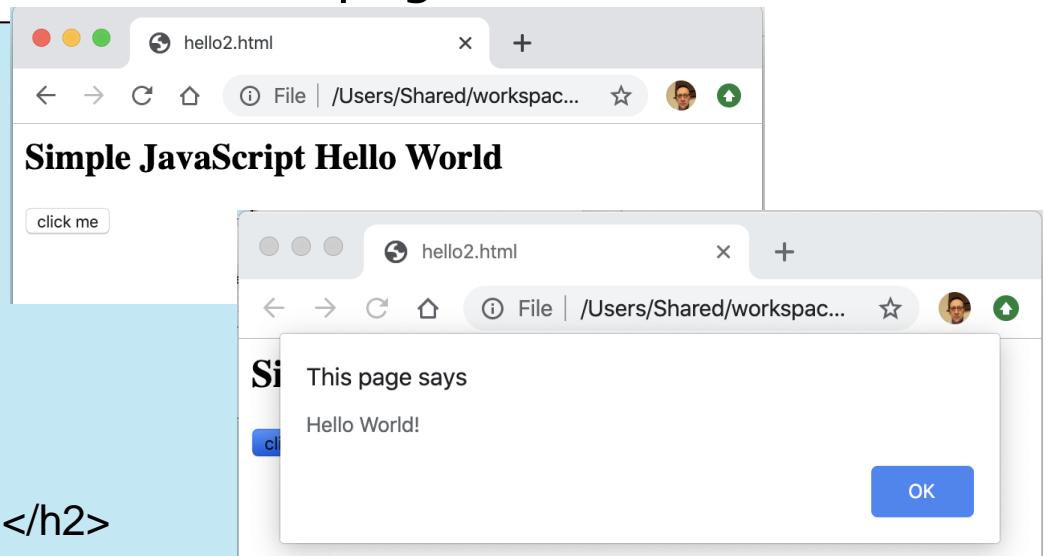
```
<html>
  <body>
    <script>
      alert("Hello World!");
    </script>
  </body>
</html>
```

On the right, a web browser window titled "hello.html" shows a modal dialog box with the text "This page says" followed by "Hello World!" and an "OK" button.

JavaScript in the Head

- JavaScript can be placed in the head
 - loaded early so important if needed in page

```
<html>
  <head>
    <script>
      function hello() {
        alert("Hello World!");
      }
    </script>
  </head>
  <body>
    <h2>Simple JavaScript Hello World</h2>
    <form>
      <input type="button" value="click me" onclick="hello()" />
    </form>
  </body>
</html>
```



JavaScript External File

- Common for scripts to be placed in an external file
 - external file has .js extension
 - often placed in a js or javascript directory
 - script is referenced in HTML page using <script src='file.js'>

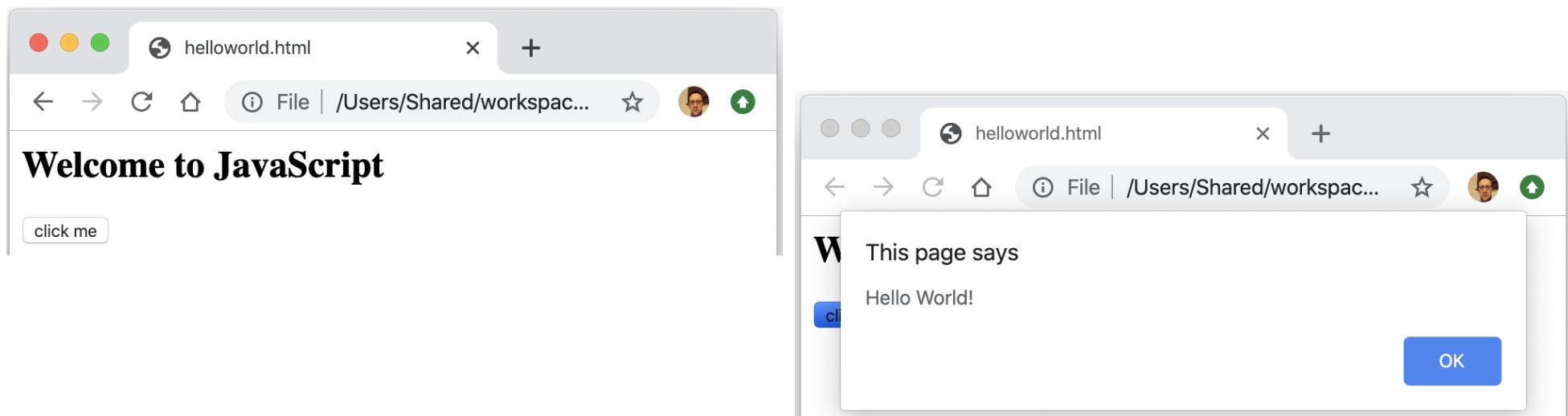
```
<html>
<head>
    <script src="helloworld.js"></script>
</head>
<body>
    <h2>Welcome to JavaScript</h2>
    <form>
        <input type="button" value="click me" onclick="hello()" />
    </form>
</body>
</html>
```

JavaScript External File

- External file is helloworld.js

```
function hello() {  
    alert("Hello World!");  
}
```

- Behaviour the same



Advantages of External Files

- Advantages include:

- separates HTML and code
- makes HTML and JavaScript easier to read / maintain
- Cached JavaScript files can speed up page loads
- external file can be reused in multiple pages
- multiple files can be used in a page

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

- can reference files via file URLs

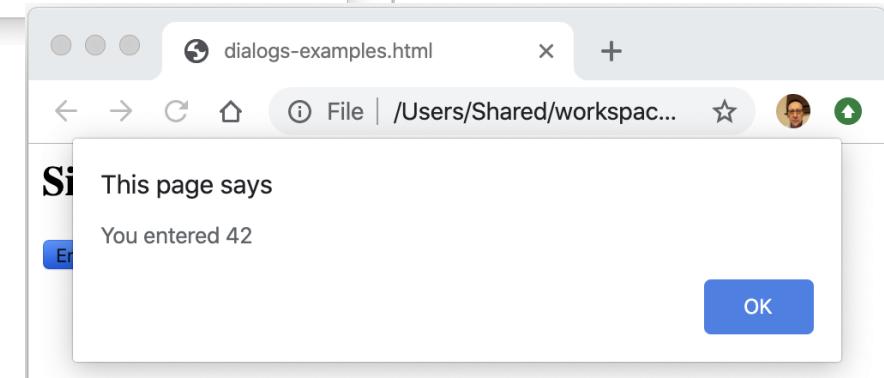
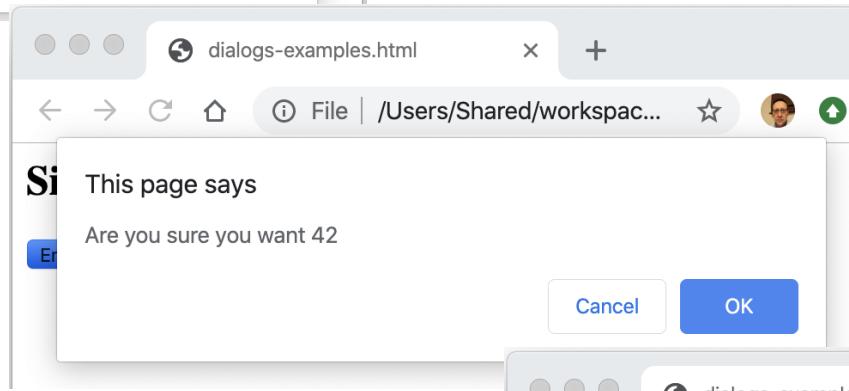
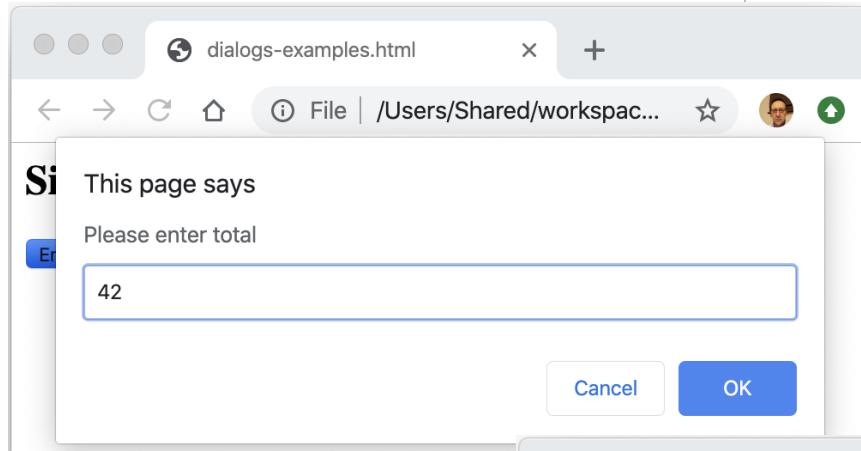
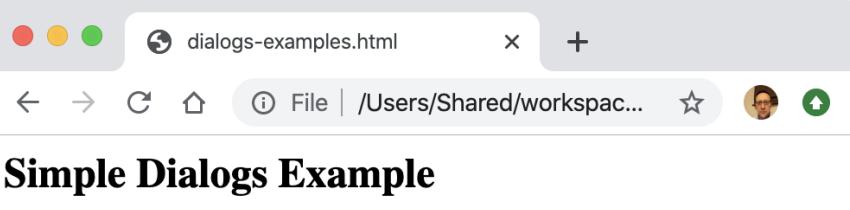
```
<script src="https://www.w3schools.com/js/myScript1.js"></script>
```

Dialogs in JavaScript

- There are several commonly used dialogs available in JavaScript
- alert – provide notifications, warnings etc.
 - do not return a value
- prompts – used to obtain input data from the user
 - return the value entered
- confirm – used to confirm actions
 - return true or false

Dialogs in JavaScript

```
<html>
<head>
    <script>
        function showDialogs() {
            console.log('in showDialogs');
            var total = prompt("Please enter total");
            var ok = confirm('Are you sure you want ' + total);
            console.log('ok:', ok);
            if (ok) {
                alert('You entered ' + total);
            }
        }
    </script>
</head>
<body>
    <h2>Simple Dialogs Example</h2>
    <form>
        <input type="button" value="Enter" onclick="showDialogs()" />
    </form>
</body>
</html>
```



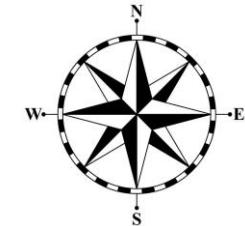
Dialogs in JavaScript

JavaScript and the HTML DOM



Toby Dussek





Plan for Session

- What is a DOM (Document Object Model)?
- DOM Tree Structure
- What is the HTML DOM?
- HTML DOM for a web page
- DOM Programming Interface
- Accessing Elements by Id
- Accessing Elements by Tag
- Accessing Form Data
- Querying by CSS Selectors
- Changing the style of an Element
- HTML DOM Collections and HTML DOM NodeList

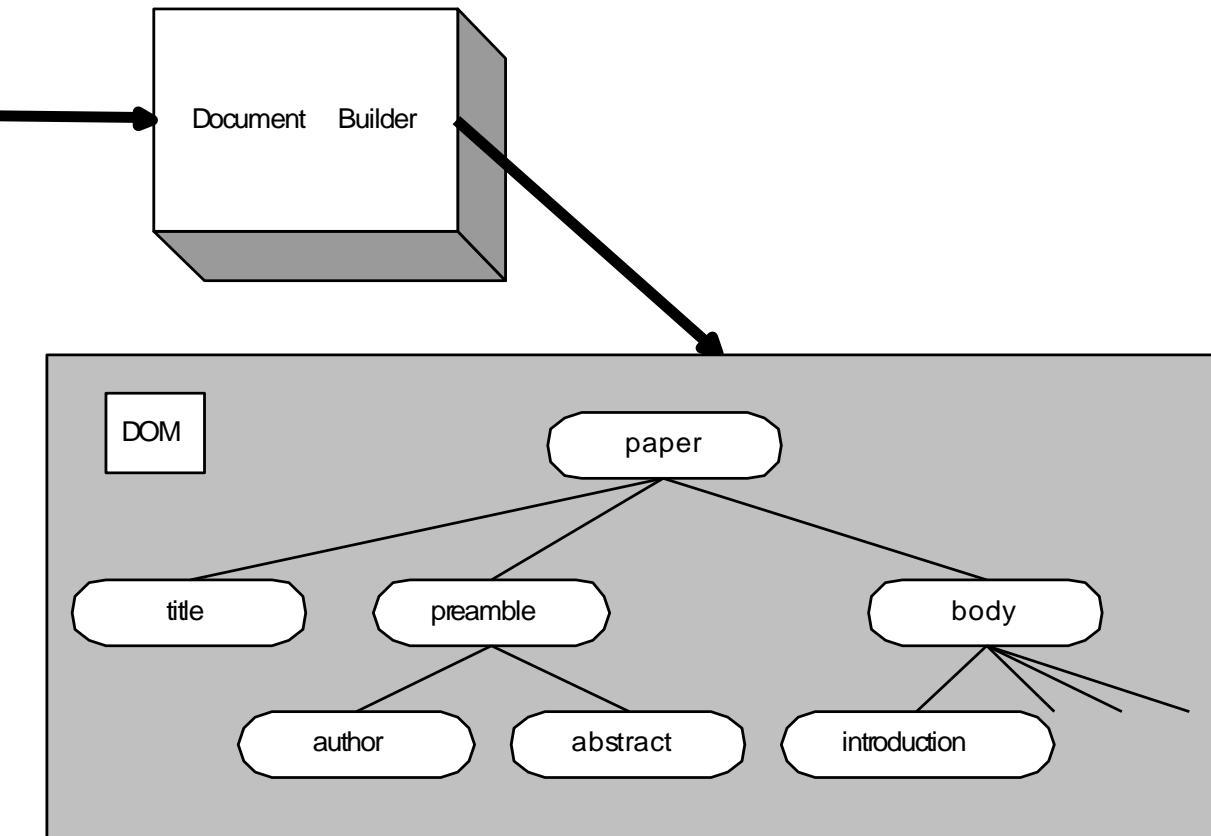
Document Object Model

□ DOM - Document Object Model

- Standard object model
- In-memory tree manipulation of nodes
 - e.g. HTML, XML
- W3C DOM Standard
 - Core DOM - standard model for all document types
 - XML DOM - standard model for XML documents
 - HTML DOM - standard model for HTML documents
- Defines interfaces in IDL
- Standard API for many languages inc. JavaScript

DOM Tree structure

```
<paper>
  <title>XML and Java: a marriage made
  in heaven
  </title>
  <preamble>
    <author>John Hunt</author>
  <abstract>What is the ...</abstract>
  </preamble>
  <body>
    <introduction>This paper..
    </introduction>
    ...
    <body>
  </paper>
```



What is HTML DOM?

- Document Object model of the Web Page
- Built by the browser when a web page is loaded
- It is a tree of HTML element nodes
 - HTML elements represented by objects in tree
- JavaScript can access the DOM
 - change HTML elements, add, remove elements
 - change HTML element attributes
 - change CSS styles in the page
- Key to dynamic web pages
- Key to Ajax based web applications

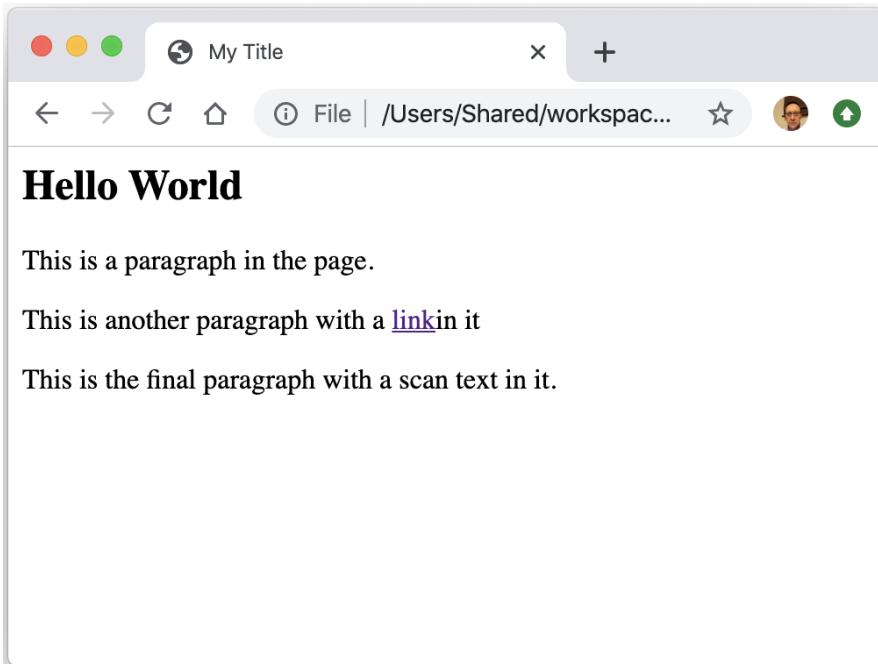
HTML DOM for a web page

- Given a simple web page

```
<html>
  <head>
    <title>My Title</title>
  </head>
  <body>
    <h2>Hello World</h2>
    <p>This is a paragraph in the page.</p>
    <p>This is another paragraph with a <a href="https://www.bbc.co.uk">link</a>in it</p>
    <p>This is the final paragraph with a <span id="scan1">scan text</span>in it.</p>
  </body>
</html>
```

HTML DOM for a web page

□ Given a simple web page



<https://software.hixie.ch/utilities/js/live-dom-viewer/>

```
HTML
└ HEAD
   └ #text:
      └ TITLE
         └ #text: My Title
   └ #text:
   └ #text:
   └ BODY
      └ #text:
         └ H2
            └ #text: Hello World
         └ #text:
         └ P
            └ #text: This is a paragraph in the page.
         └ #text:
         └ P
            └ #text: This is another paragraph with a
            └ A href="https://www.bbc.co.uk"
               └ #text: link
            └ #text: in it
         └ #text:
         └ P
            └ #text: This is the final paragraph with a
            └ SCAN id="scan1"
               └ #text: scan text
            └ #text: in it.
   └ #text:
```

DOM Programming Interface

- Provides standard for how JavaScript can get, change, add or delete html elements and attributes
- Has a tree of HTML objects aka nodes
- Objects provide access to properties on objects
 - value that you can get or set
 - (like changing the content of an HTML element)
- Objects provide methods to
 - find, modify, create HTML elements

Accessing Elements By Id

- Can find an element by its (unique) id
 - `document.getElementById("<id>")`

```
<html>
  <head>
    <title>DOM Example 1A</title>
    <script>
      function hello() {
        document.getElementById("demo").innerHTML = "Hello World!";
      }
    </script>
  </head>
  <body>
    <h2>JavaScript DOM Manipulation Example</h2>
    <p id="demo">This is place holder</p>
    <form>
      <input type="button" value="say hello" onclick="hello()" />
    </form>
  </body>
</html>
```

Accessing Elements By Id

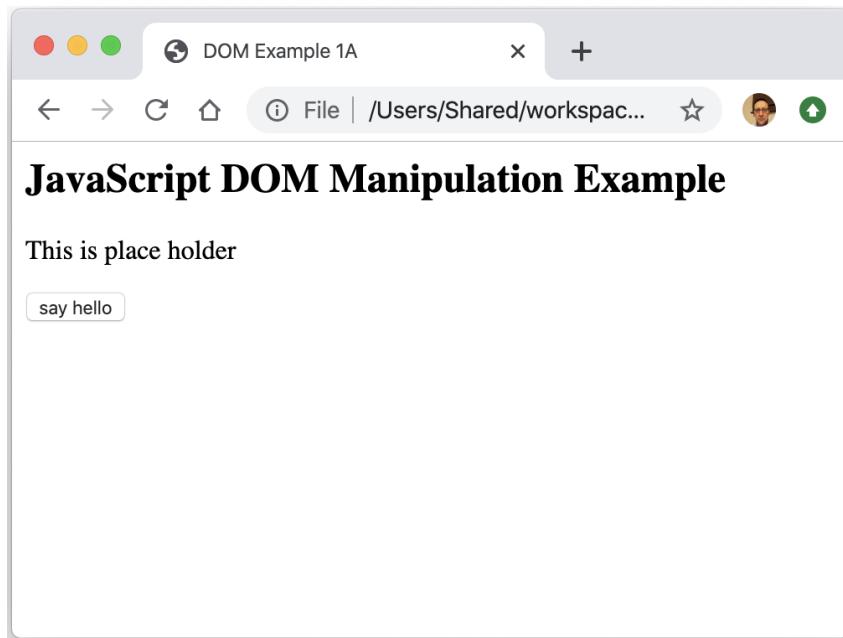
□ The getElementById Method

- common way to access an HTML element
- in example getElementById method used id="demo" to find the element

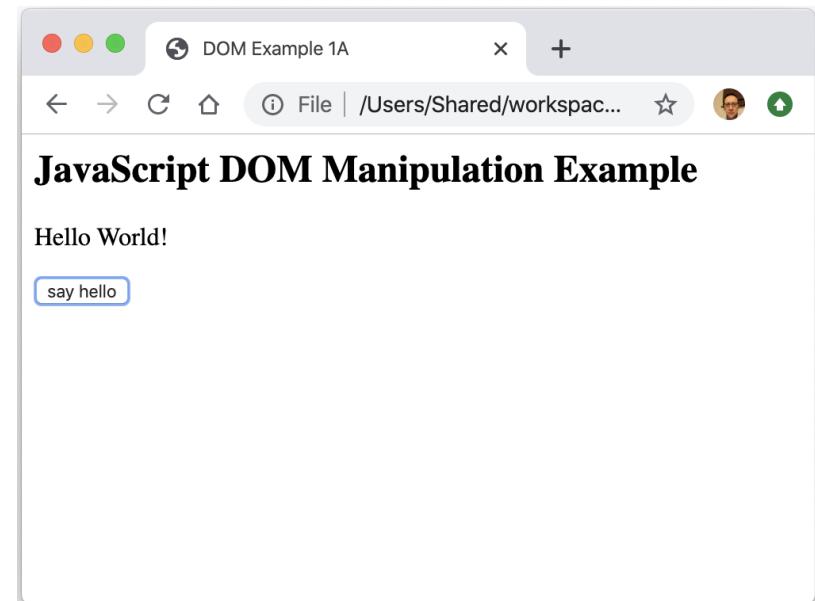
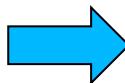
□ The innerHTML Property

- easiest way to get the content of an element
- can be used to replace the content of HTML elements
- can be used to get or change any HTML element
 - including <html> and <body>

Accessing Elements By Id



Click Button

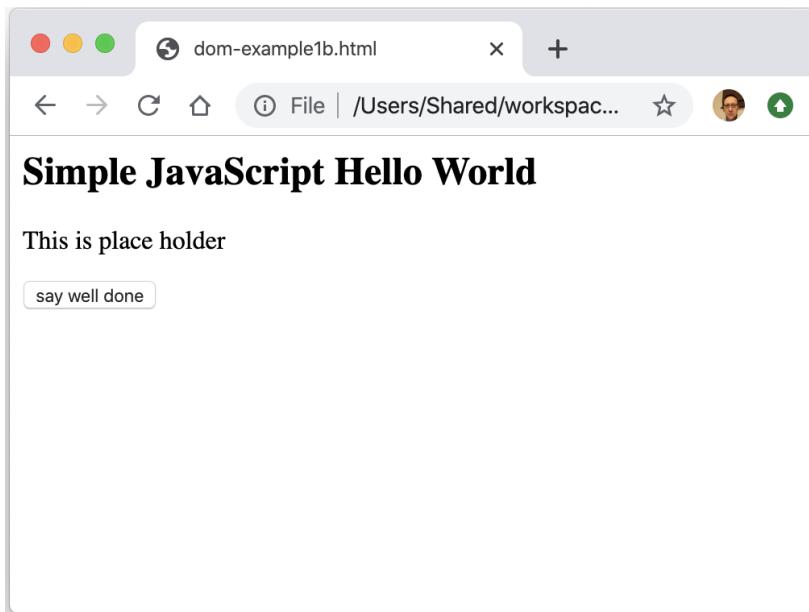


Accessing Elements by Tag

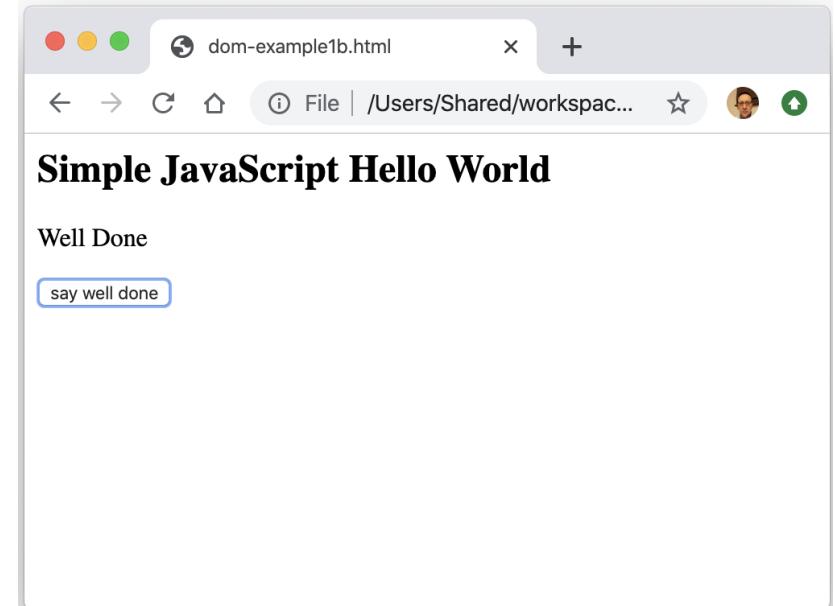
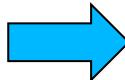
- Can also access collection of nodes by Tag type
 - note have to access the zero element of array like container

```
<html>
  <head>
    <script>
      function wellDone() {
        document.getElementsByTagName("p")[0].innerHTML = "Well Done";
      }
    </script>
  </head>
  <body>
    <h2>Simple JavaScript Hello World</h2>
    <p>This is place holder</p>
    <form>
      <input type="button" value="say well done" onclick="wellDone()" />
    </form>
  </body>
</html>
```

Accessing Elements by Tag



Click Button



Accessing elements by Name

- ❑ Ids are unique; names are not
- ❑ `getElementsByName` returns a collection of nodes
 - here we are looping through the nodes returned by name

```
<html>
<head>
<script type="text/javascript">
function enter() {
    var options = document.getElementsByName("option");
    alert('Options: ' + options.length);
    for (let option of options) {
        console.log('option: ', option);
        if (option.checked) {
            var value = option.value;
            console.log('value:', value);
            if (value == 'tea') {
                alert('They want Tea');
            }
        }
    }
}
</script>
</head>
<body>
<h2>Simple JavaScript Hello World</h2>
<p id="demo">This is place holder</p>
<form name="drinks">
    Tea: <input type="radio" name="option" value='tea' /> <br />
    Coffee: <input type="radio" name="option" value='coffee' /> <br />
    Water: <input type="radio" name="option" value='water' /> <br />
    <input type="button" value="Enter" onclick="enter()" />
</form>
</body>
</html>
```

Accessing Form data

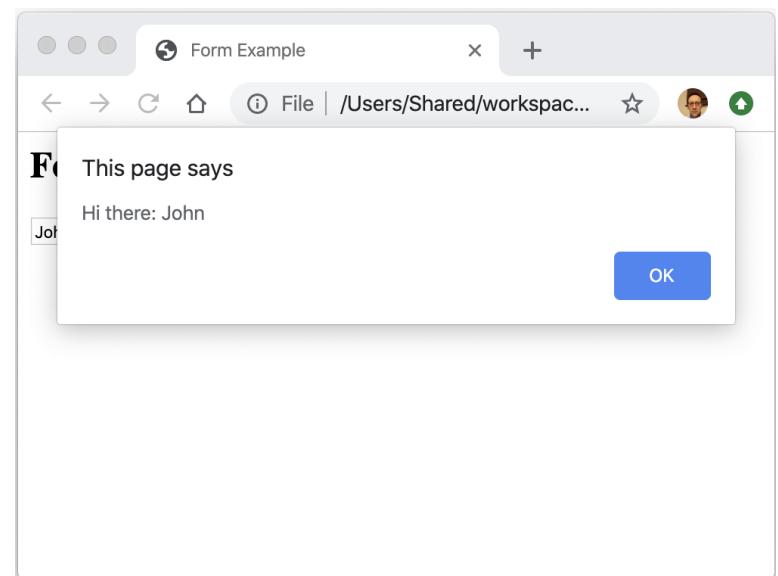
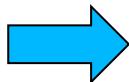
- Can also access named form data
 - note *form* has a name and *input* has a name

```
<html>
<head>
    <title>Form Example</title>
    <script type="text/javascript">
        function enter() {
            var name = document.userform.username.value;
            alert('Hi there: ' + name);
        }
    </script>
</head>
<body>
    <h2>Form Data Example</h2>
    <form name="userform">
        <input type="text" name="username" />
        <input type="button" value="Enter" onclick="enter()" />
    </form>
</body></html>
```

Accessing Form data

A screenshot of a web browser window titled "Form Example". The address bar shows the path "/Users/Shared/workspace...". The main content area has a heading "Form Data Example" and contains a single-line text input field followed by a button labeled "Enter".

Enter a Name &
Click Button



Querying by CSS Selectors

- Can also find elements based on CSS Selectors

```
elementList = parentNode.querySelectorAll(selector);
```

- returns array like structure of all elements matching selector

```
<html>
<head>
  <script>
    function enter() {
      var h3NodeList = document.querySelectorAll("h3");
      console.log(h3NodeList);
    }
  </script>
</head>
<body>
  <h2>A H2 Level heading</h2>
  <p>Some text</p>
  <h3>Subheading 1</h3>
  <p>Some more text.</p>
  <h3>Subheading 2</h3>
  <p>Even more text.</p><button type="button" onclick="enter()" />Click Me!</button>
</body>
</html>
```

Changing the style of an element

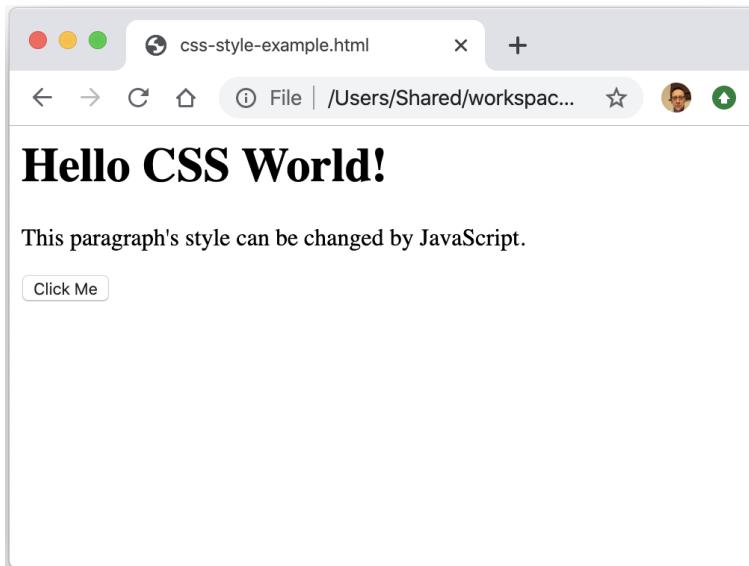
- To change the style of an HTML element, use this syntax:

```
parentNode.getElementById(id).style.property = new style
```

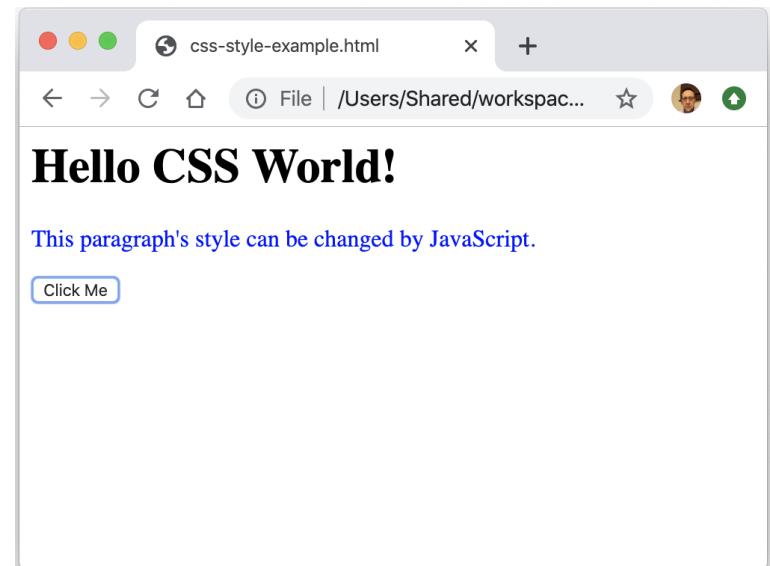
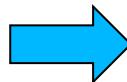
- For example

```
<html>
<head>
  <script>
    function changeStyle() {
      document.getElementById("p1").style.color = "blue";
    }
  </script>
</head>
<body>
  <h1>Hello CSS World!</h1>
  <p id="p1">This paragraph's style can be changed by JavaScript.</p>
  <form><input type="button" value="Click Me" onclick="changeStyle()" /></form>
</body>
</html>
```

Changing an elements style



Click Button



HTML DOM Collections

- Are array like collections of nodes / elements
 - but they are not arrays
- Hold elements in document order
- Can iterate over them
 - can tell you their length
- Has own methods
 - `HTMLCollection.item(index)` Returns node at the given index.
 - `HTMLCollection.namedItem(name)` Returns the node whose ID or, as a fallback, name matches

HTML DOM NodeList

- Very similar to HTMLCollection
 - but returned by some methods (particularly on older browsers)
 - e.g. most browsers return nodeList from querySelectorAll()
- Differences
 - HTMLCollection items can be accessed by their name, id, or index number.
 - NodeList items can only be accessed by their index number.
 - Only the NodeList object can contain attribute nodes and text nodes.

jQuery Introduction

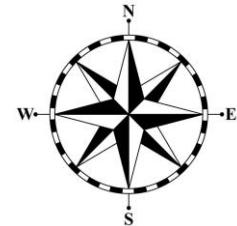


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- What is jQuery?
- Obtaining jQuery
- Working with jQuery
- The `$()` Factory function
- jQuery function chaining
- Selectors
- Useful links
- Some jQuery Books

What does jQuery do?



- JavaScript library to simplify
 - DOM tree traversal and manipulation
 - Modify the appearance of Web Pages
 - Alter the content of a document
- Respond to user interaction
 - via event handling
- Animate changes being made to a document
- Retrieve information from a server
 - Without refreshing a page (AJAX style)
- Simplify common JavaScript tasks

History of jQuery



- Initial Development Phase
 - Started by John Resig in Aug 2005
- 1.0 August 2006
 - 1.1 – 1.12 (2007 – 2016)
- 2.0 April 2013
 - 2.1 – 2.2 (2014-2016)
- 3.0 June 2016
 - 3.1 (2016) – 3.4 (April 2019)
- 3.5.1 Current version
 - released (April 2020)
- Plus many plugins

Popularity of jQuery



- 2015 jQuery used in 62.7% of top 1 million websites
 - and 17% of all Internet websites
- 2017 jQuery used on 69.2% of top 1 million websites
- 2018 jQuery used on 78% of top 1 million websites
- 2019 jQuery used on 80% of top 1 million websites
 - and 74.1% of the top 10 million
- Feb 2020 jQuery is used by 74.4% of top 10 million websites

Distribution of jQuery

- Typically distributed as a single JavaScript file
 - defines all its interfaces, including DOM, Events, and Ajax functions
 - can be a local file within a web site
 - or via a content delivery network (CDN) hosted by MaxCDN

```
<script src="https://code.jquery.com/jquery-3.5.1.js"></script>
```
 - or can use minified version

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
```
- Plus optional (inc. 3rd party) plugins
 - as separate files

Adding jQuery code

- Add a function which runs once the pages document model is loaded

```
$document).ready( function() {  
    //Do some stuff to the DOM elements on this page  
});
```

- We will change the colour of the heading when clicked

```
$( "h1" ).click( function() {  
    $(this).addClass("blue").fadeOut("slow");  
});
```

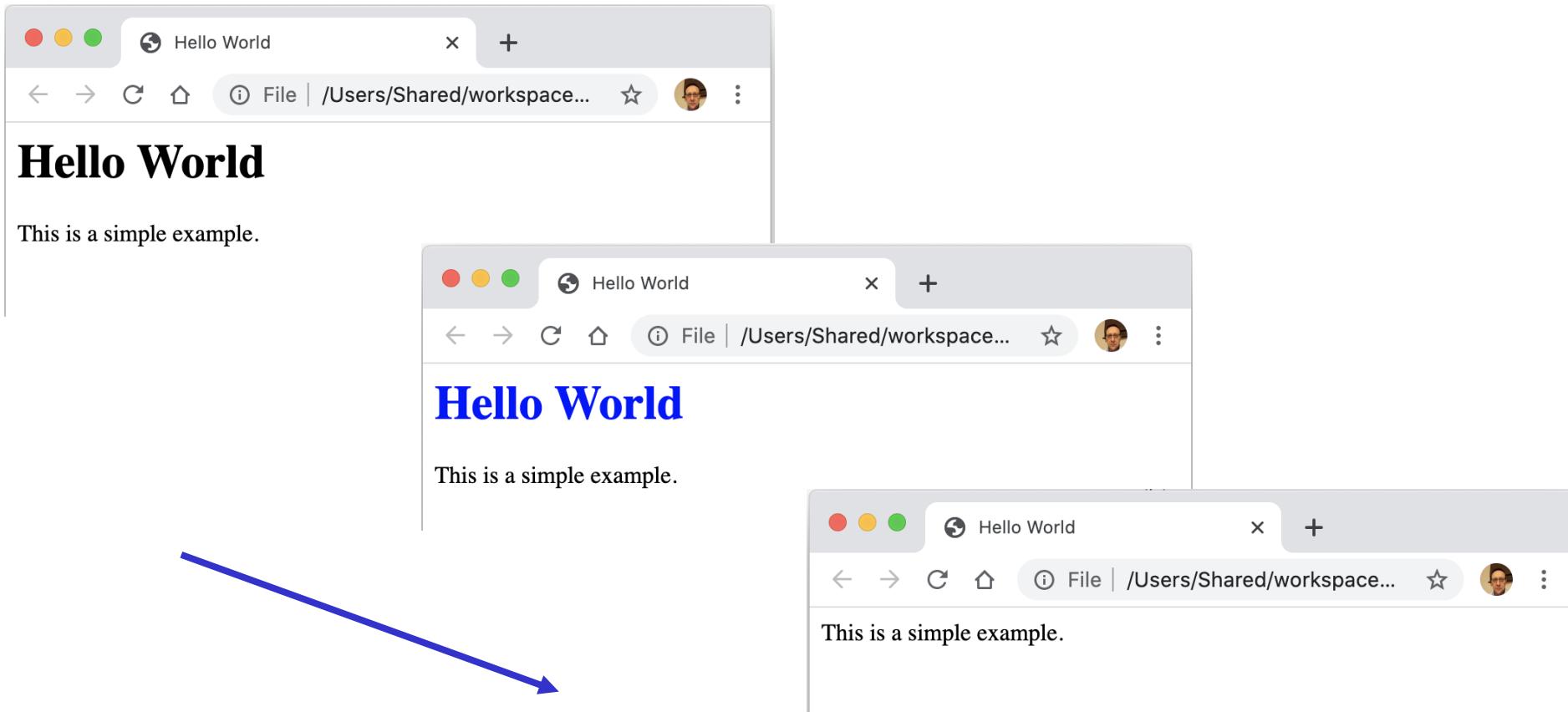
- For a style

```
<style type="text/css">  
    .blue { color: blue; }  
</style>
```

The Web page

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
    <style type="text/css">
      .blue { color: blue; }
    </style>
    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>
    <script>
      $(document).ready(function() {
        console.log("Start up");
        $("h1").click(function(){
          console.log("Clicked on H1");
          $(this).addClass("blue").fadeOut("slow");
        });
      });
    </script>
  </head>
  <body>
    <h1>Hello World</h1>
    This is a simple example.
  </body>
</html>
```

In Chrome



The `$()` Factory function

- Selecting part of document is fundamental operation
- A JQuery object is a wrapper for a selected group of DOM nodes
- `$()` function is a factory method that creates JQuery objects
 - Actually an alias for jQuery
 - Represented by both `$` and `jQuery`
- `$` is an overload symbol in Javascript frameworks
 - To overcome this problem use
 - `jQuery.noConflict()`

The \$() function cont'd

- `$("dt")` returns a JQuery object (set)
 - containing all the **dt** elements in the document
 - Note set of elements returned by a function is always wrapped in a jQuery object
- Can apply functions to all members of a set
 - Avoids unnecessary iteration
 - `$("dt").hide()`
 - `.addClass()` method changes DOM nodes by adding 'class' attribute
 - `$("dt").addClass("emphasize")` will change all occurrences of <dt> to <dt class="emphasize">
- See also `.removeClass()`

Short Hand Document Ready function

- We have been using
 - `$(document).ready(function () {...});`
- This calls the ready method on a jQuery object constructed from the document
- There is a shorthand form which defaults to the current document and the ready method
 - `$(function() { ...});`

Other Notes on Application

- `.click(...)` – binding function to click event on all objects in set
- `function() {...}` – function to run when click happens
- `$(this)` – this equals element selected (`$(...)` makes it a jQuery object)
- `.fadeOut()` – prebuilt jQuery animation effect

Also note implicit iteration over jQuery set

jQuery Chaining

- Most functions return a jQuery object
 - Thus the result of 1 function can be used with another function
 - Can give better performance as only 1 selector
 - But can be harder to understand

`$(selector).func1(...).func2(...).funcN(...);`

\$ jQuery Object

selector Selector syntax, many different selectors allowed

func Chainable, most functions return a jQuery object

(...) Function parameters

jQuery Chaining Example

- JQuery uses chaining as follows

```
$('a:contains("John")')
    .parent()
    .addClass("emphasize")
```

- Add emphasize to the parent elements of all links containing the String “John”
- jQuery Animation effects
 - .show() / hide()
 - Show / Hide the matched elements
 - .hide(speed) or .show(speed)
 - Where speed is ‘slow’, ‘normal’ or ‘fast’

jQuery Chaining Example

- Hide then slow reveal the heading

```
<!DOCTYPE HTML>
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>
    <script >
      $(document).ready(function() {
        $("h1").click(function(){
          $(this)
            .hide()
            .show("slow")
        });
      });
    </script>
  </head>
  <body>
    <h1>Welcome</h1>
    <p>This is a plain piece of text.</p>
  </body>
</html>
```

Iteration Functions

- A number of iterator style functions
- Each - apply a function to each set element

```
$objList
.each(function(i) {
    requestQueue.addToListOfRequests(objList[i]);
});
```

- Map - apply a function to an arbitrary set of data

```
$(function () {
    let arr = [ "a", "b", "c", "d", "e" ]
    arr = jQuery.map(arr, function(n, i){
        return (n.toUpperCase() + i);
    });
    $("div").text(arr.join(", "));
});
```

Attribute Selectors

- Select all h2 elements
 - `$('h2')`
- Can select an element via one of its attributes
 - E.g. a link's title or an image's alt
- Syntax is element[" attribute "]
 - `$('img[alt]')` – select all images with an attribute alt
- Attribute selectors can use a wildcard syntax
 - Beginning "^"
 - Ending "\$"
 - Arbitrary position *
 - !negated value

Some attribute selector examples

- `$('input[type=text]')` – select all input elements with the type attr set to type
- `$('div[title^=my]')` – select all div elements where the title attr starts with “my”
- `$('a[href$=.pdf]')` – select all anchors where the href attr ends with .pdf
- `$('a[href*=google]')` – select all anchors which contain google anywhere in their href

Adding to our jQuery script

- Include an attribute selector
 - a[href]
 - And a while card - ^ starting with

```
$(document).ready(function() {  
    $('h2').addClass("title");  
    $('#mymailto').addClass('mailto');  
});
```

Custom Selectors

- jQuery adds custom selectors to CSS style selectors
- Custom selectors help obtain items from a *jQuery object set*
- Custom selectors start with a colon (:)
 - E.g. \$('div.horizontal:eq(1)')
 - Select the **second** element of the set
 - CSS selectors are **1** based!

Some jQuery Custom Selectors

p:first	first paragraph
li:last	last list item
a:nth(3)	fourth link
a:nth-child(odd even nth)	Selects all odd or even or nth children of the parent (note nth-child is 1-based)
a:eq(3)	fourth link
p:even <i>or</i> p:odd	every other paragraph
a:gt(3) <i>or</i> a:lt(4)	every link after the 4th <i>or</i> up to the fourth
a:contains(<i>string</i>)	links that contain the word indicated by <i>string</i>
ul:has(li)	select ul elements that have li

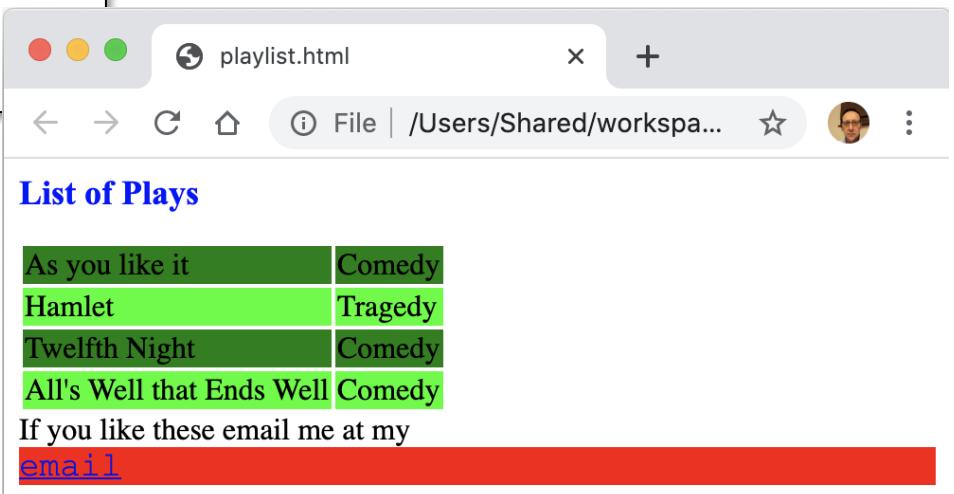
```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="css/style.css" type="text/css" media="screen"></link>
    <script src="https://code.jquery.com/jquery-3.5.1.js"></script>
    <script src="js/playlist.js"></script>
  </head>
  <body>
    <h2>List of Plays</h2>
    <table>
      <tr>
        <td>As you like it</td> <td>Comedy</td>
      </tr>
      <tr>
        <td>Hamlet</td> <td>Tragedy</td>
      </tr>
      <tr>
        <td>Twelfth Night</td> <td>Comedy</td>
      </tr>
      <tr>
        <td>All's Well that Ends Well</td> <td>Comedy</td>
      </tr>
    </table>
    If you like these email me at my
    <div id="mailto"><a href="mailto:popeye@town.co.uk">email</a></div>
  </body>
</html>
```

Adding styles for odd and even rows

□ Add jQuery code

- Make use of the :odd and :even custom selectors

```
$(document).ready(function() {  
    console.log("In set up");  
    $('h2').addClass("title");  
    $('tr:odd').addClass("odd");  
    $('tr:even').addClass("even");  
    $('#mymailto').addClass('mailto');  
});
```



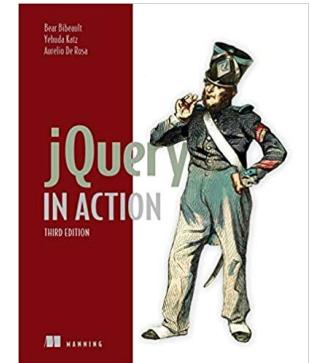
Useful jQuery links

- Project website
 - <http://www.jquery.com>
- API Documentation
 - <https://api.jquery.com/>
- Learning Center
 - <https://learn.jquery.com/> – jQuery tutorial blog
 - <https://learn.jquery.com/style-guide/> style guide
- jQuery user interface
 - <https://jqueryui.com/>
 - <https://learn.jquery.com/jquery-ui/>
- Tutorials
 - <https://www.w3schools.com/jquery/>
- jQuery Plugins
 - <https://plugins.jquery.com/>

Recommended jQuery Books

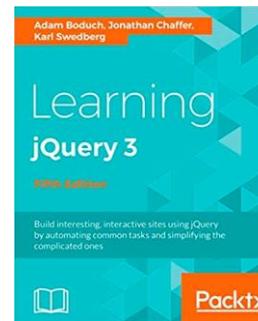
□ jQuery in Action

- B. Bibeault, Y. Katz A De Rosa
- Manning Publications, 9781617292071, 2015



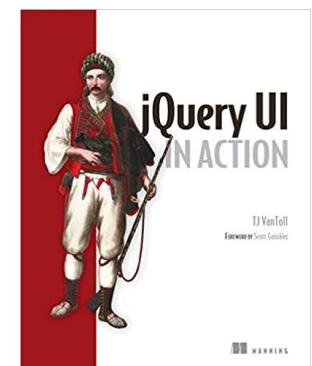
□ Learning jQuery 3

- A. Boduch, J. Chaffer & K. Swdberg
- Packt Publishing Limited 2017



□ jQuery UI in Action

- T. J. Van Toll,
- Manning Publications, 1617291935, 2014



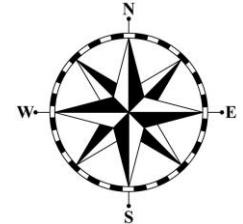
Ajax Style Web Applications



Toby Dussek

Informed Academy





Plan for Session

- jQuery and Ajax
- jQuery ajax function and shorthand forms
- Bookshop AJAX application example

jQuery and Ajax



- Pure AJAX involves several technologies working together
 - Need to know and understand these to make it all work
 - But at the core AJAX is a means for loading data from a server without a page refresh
 - jQuery provides a framework that simplifies and standardises this
- Various jQuery methods
 - all based on the `$.ajax` function
 - allows for different request operations
 - success and failure callbacks etc

jQuery ajax function

- Provides ability to invoke server side HTTP methods
 - providing in URL, any body content, plus call back functions

```
$.ajax({  
    type: "POST",  
    url: url,  
    data: data,  
    success: success_function,  
    failure: error_function,  
    contentType: "application/json; charset=utf-8",  
    dataType: "json"  
});
```

- can be used with GET, POST, PUT and DELETE
- jQuery also provide convenience methods
 - `$.post(url, data, success_function)`

jQuery Shorthand AJAX methods

□ **jQuery.get()**

- Load data from the server using a HTTP GET request.

□ **jQuery.getJSON()**

- Load JSON-encoded data from server using GET request

□ **jQuery.getScript()**

- Load JavaScript file from server using GET HTTP request, then execute it.

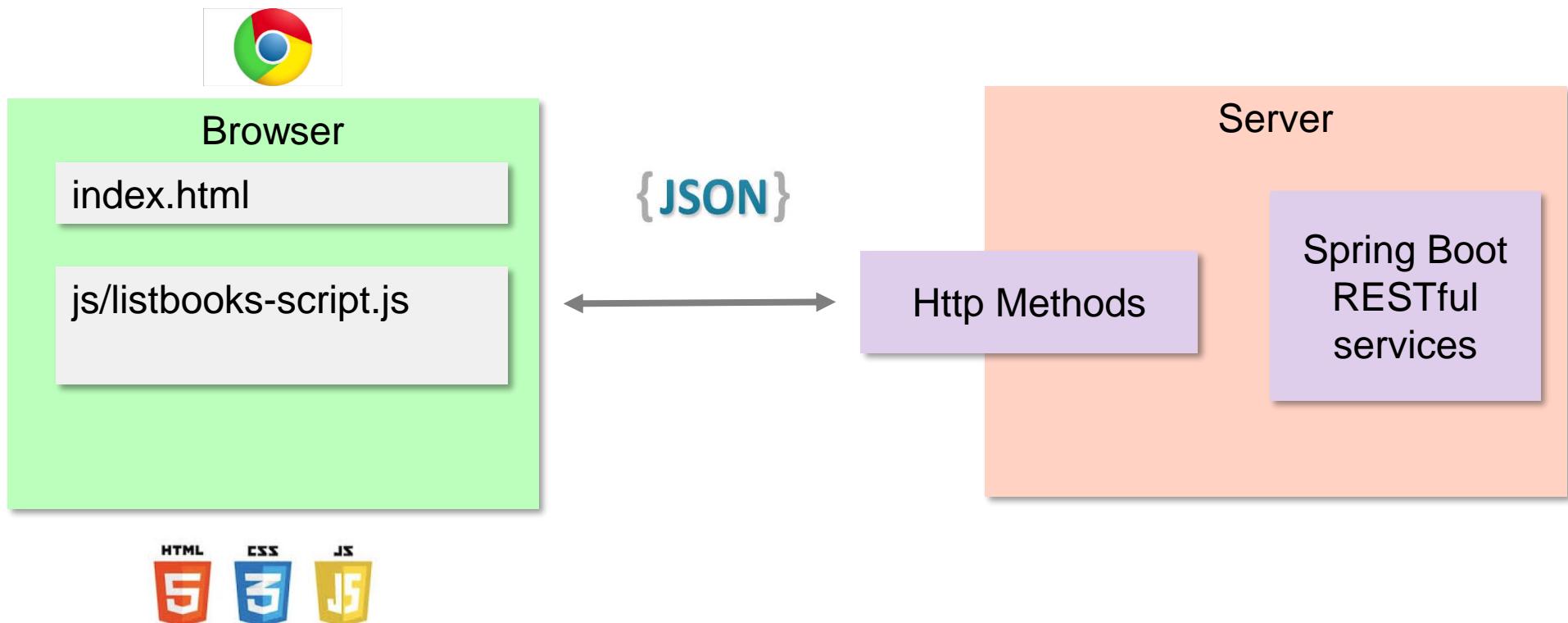
□ **jQuery.post()**

- submit data to the server using a HTTP POST request.

□ <https://api.jquery.com/category/ajax/shorthand-methods/>

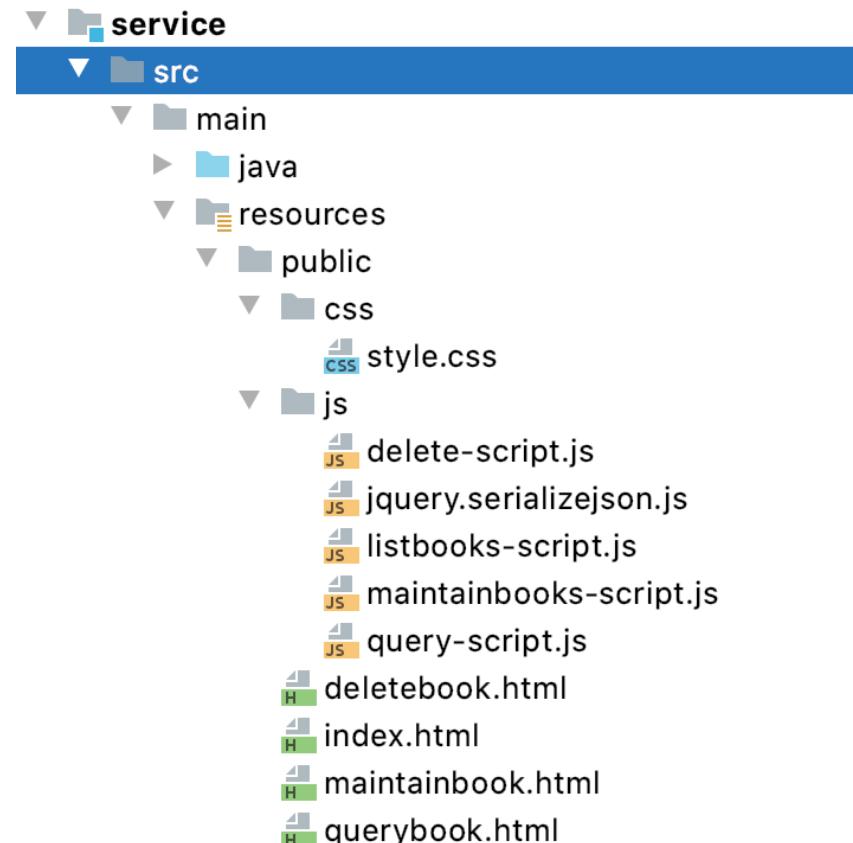
A simple jQuery Ajax style example

- ❑ Illustrates structure of running application



Using jQuery with Spring Boot Services

- Spring Boot picks up web resources
 - place under resources/public
 - or resources/static
 - will automatically be picked up and served by Tomcat
 - usual to organise css and JavaScript files in sub directories



Accessing booklist services

- In web page have a button and a div

```
<p>Click on the following button to list all books.</p>
<form>
  <input type="submit" id="show" value="Show"/>
</form>
<div id="booklist"></div>
```

- In JavaScript file

- register a function with the button
- should ensure nothing else processes the event

- `event.preventDefault();`

- Call the RESTful service

- <http://localhost:8080/bookshop/list>

- using the GET method request via `$.get(url, callback)`

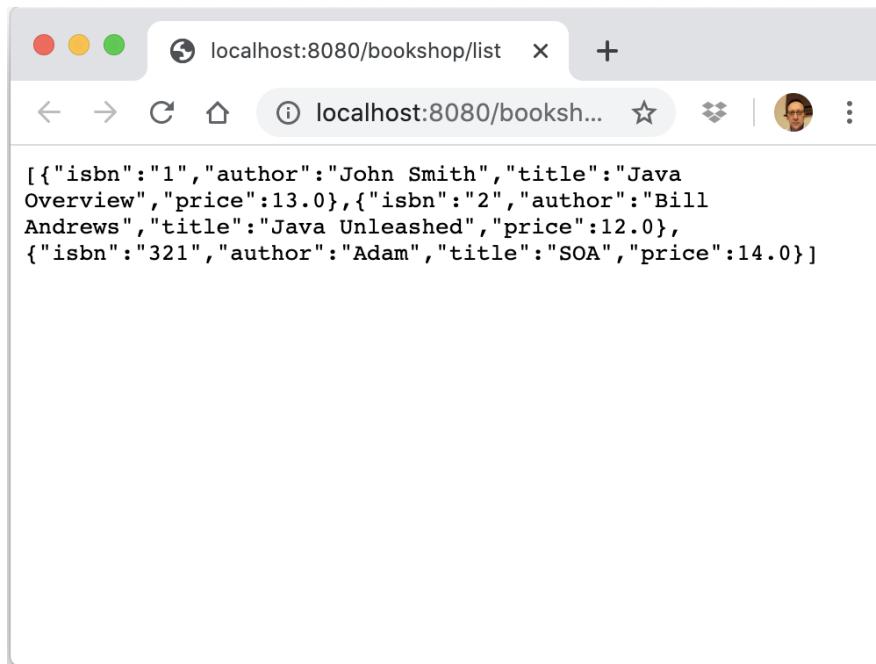
- Process the JSON data returned

Accessing booklist services

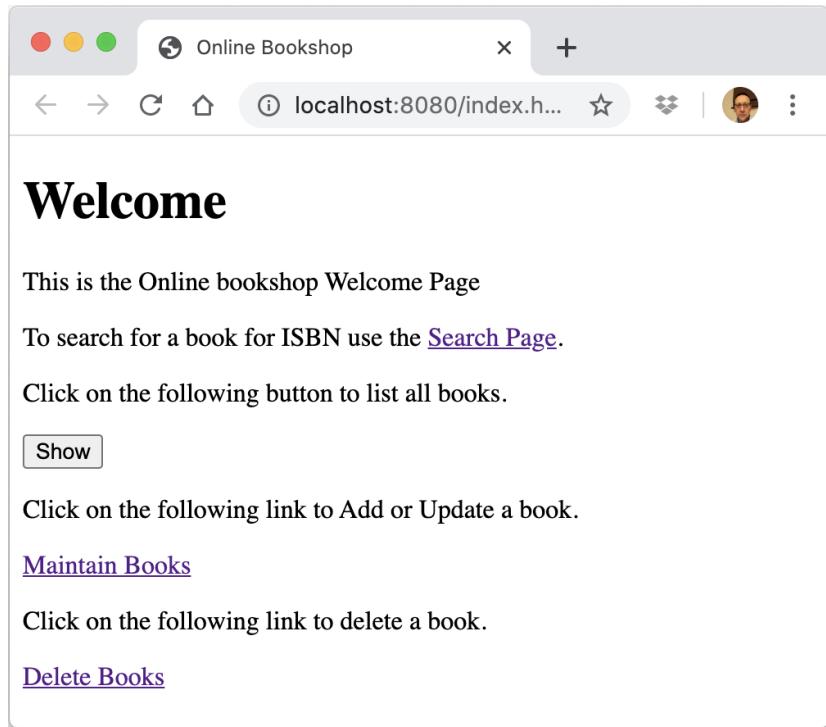
```
$("#show").click(function() {
    event.preventDefault();
    $.get("http://localhost:8080/bookshop/list", function(books) {
        $("#booklist").empty();
        let html = "<div class='book'>";
        $.each(books, function(i, book) {
            console.log(book);
            html += "<h3 class='title'>" + book.title + "</h3>";
            html += "<ul>";
            html += "<li>Author: " + book.author + "</li>";
            html += "<li>price: " + book.price + "</li>";
            html += "<li>isbn: " + book.isbn + "</li>";
            html += "</ul>";
        });
        html += "</div>";
        $("#booklist").append($(html));
    });
});
```

Invoking the server side directly

- Can be done from browser as using HTTP GET
 - returns a JSON document



The result



This screenshot shows a web browser window titled "Online Bookshop" at the URL "localhost:8080/index.h...". The page content is as follows:

Welcome

This is the Online bookshop Welcome Page

To search for a book for ISBN use the [Search Page](#).

Click on the following button to list all books.

Show

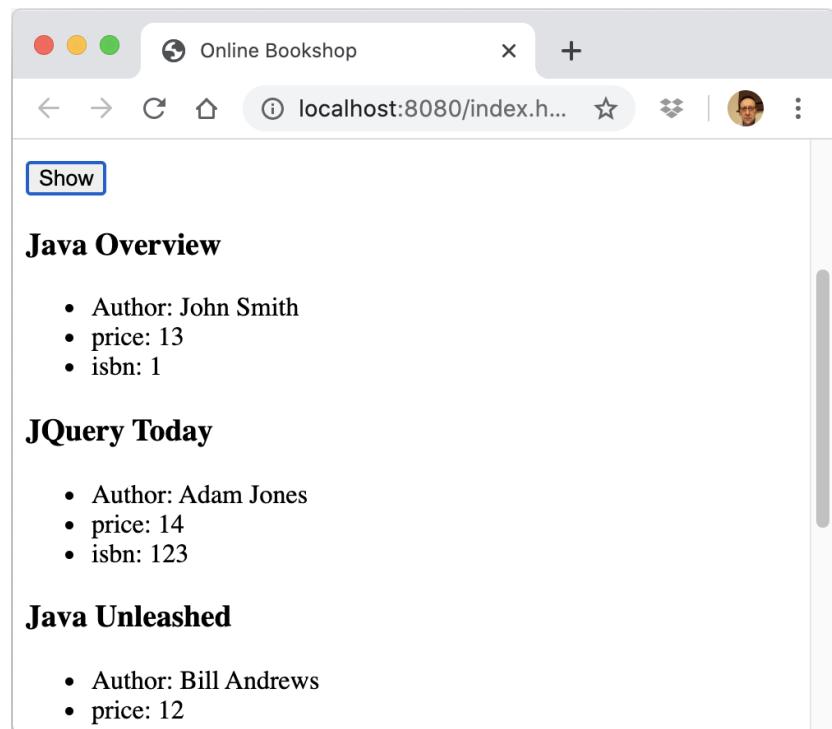
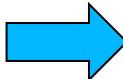
Click on the following link to Add or Update a book.

[Maintain Books](#)

Click on the following link to delete a book.

[Delete Books](#)

Click Button



This screenshot shows a web browser window titled "Online Bookshop" at the URL "localhost:8080/index.h...". The page content is as follows:

Java Overview

- Author: John Smith
- price: 13
- isbn: 1

JQuery Today

- Author: Adam Jones
- price: 14
- isbn: 123

Java Unleashed

- Author: Bill Andrews
- price: 12

Sending Data to the Service

- Need to
 - obtain data in web page form
 - needs to be in JSON format
 - can use a jquery.serialization plugin to handle
 - as we have numbers in our data
 - define a success function
 - (optionally) an error function
 - indicate we are sending JSON
 - indicate we want JSON back

```
$ (document) .ready(function() {
    console.log('Setting up maintain form');
    $("#maintain-form") .submit(function(event) {
        event.preventDefault();

        let url = "http://localhost:8080/bookshop";
let obj = $(this).serializeJSON();
let data = JSON.stringify(obj);

        $.ajax({
            type: "POST",
            url: url,
            data: data,
            success: function(result) {
                alert('Books Updated');
            },
            failure: functionerrMsg) {
                alert(errMsg);
            },
            contentType: "application/json; charset=utf-8", // data sent
            dataType: "json" // type of data returned
        });
    });
});
```

Web Page

- Load jQuery, CSS, jquery plugin and own file

```
<form id="maintain-form" action="">
    ISBN <input type="text" name="isbn:number" id="isbn" type="number"/> <br />
    Title <input type="text" name="title" id="title"/> <br />
    Author <input type="text" name="author" id="author"/> <br />
    Price <input type="text" name="price:number" id="price"/> <br />
    <p>
        <button type="submit" name="add" value="add">Update</button>
    </p>
</form>
```

The Web App

Book Maintenance

Please enter book details to Add / Update a Book

ISBN

Title

Author

Price

Show

Show

Java Overview

- Author: John Smith
- price: 13
- isbn: 1

JQuery Today

- Author: Adam Jones
- price: 14
- isbn: 123

Java Unleashed

- Author: Bill Andrews
- price: 12
- isbn: 2

AJAX observer functions

- Useful to know when processing starts and stops
 - E.g. to notify the user
- E.g. Can attach .ajaxStart() and .ajaxStop()
methods within an ajax operation

```
$("#loading").ajaxStart(function() {  
    $(this).show();  
}).ajaxStop(function() {  
    $(this).hide();  
});
```

Range of Functions

- ajaxComplete(callback)
- ajaxError(callback)
- ajaxSend(callback)
- ajaxStart(callback)
- ajaxStop(callback)
- ajaxSuccess(callback)
- Can chain so that stop runs after start
 - `$(...).ajaxStart(function() {...}).ajaxStop(function() {...});`

Testing JavaScript

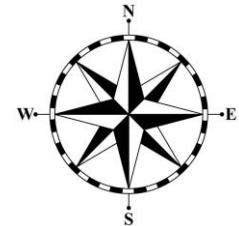


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- Testing frameworks for JavaScript
- Jasmine
- Jasmine Set Up
- Working with Jasmine
- Running Jasmine
- Working with Jasmine
- Test Fixtures in Jasmine
- Nested describe functions
- Test a DOM oriented function

Testing Frameworks for JavaScript

- Several popular test frameworks available

- Jasmine  **Jasmine**

- widely used with JavaScript, TypeScript and Angular

- Jest from Facebook



- aims to be an easy to use framework gaining in popularity

- Mocha (and Chai)



- typically used with Node.js
 - see <https://mochajs.org/>

- Cucumber another BDD test framework

- typically used for acceptance testing
 - see <https://cucumber.io/>



Jasmine



- Supports Behaviour-Driven Development style Tests
 - can be used for unit tests also supports BDD style tests
 - see <https://jasmine.github.io/>
- Why use Jasmine?
 - Jasmine does not depend on any other JavaScript framework.
 - Jasmine does not require any DOM.
 - Jasmine is an open-source framework
 - can be used in the browser or in Node.js
 - can download standalone library from web site



Jasmine Set Up

- We will first run tests within a browser
- Need to create a web page which
 - references the Jasmine framework, your code and tests

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="shortcut icon" type="image/png" href="jasmine/lib/jasmine-3.5.0/jasmine_favicon.png">
<link rel="stylesheet" type="text/css" href="jasmine/lib/jasmine-3.5.0/jasmine.css">

<script type="text/javascript" src="jasmine/lib/jasmine-3.5.0/jasmine.js"></script>
<script type="text/javascript" src="jasmine/lib/jasmine-3.5.0/jasmine-html.js"></script>
<script type="text/javascript" src="jasmine/lib/jasmine-3.5.0/boot.js"></script>
<script type="text/javascript" src="..js/myscript.js"></script>
<script type="text/javascript" src="test-myscript.js"></script>
</head>
<body></body>
</html>
```



Working with Jasmine

□ Core Jasmine Concepts

- specs, short for (test) specifications part of terminology of BDD
- defined by **it** function – takes a description and a function to run

□ The *describe* block / clause

- essentially a logical grouping of tests
- can be nested to provide related groupings

□ Simplest Jasmine test

```
describe('calculator', function () {
  it('1 + 1 should equal 2', function() {
    expect(1 + 1).toEqual(2);
  });
});
```



Running Jasmine

- Using just the simple test can generate a report

A screenshot of a web browser window titled "mytest.html". The page displays the Jasmine test runner interface. At the top, it says "Jasmine 3.5.0". Below that, there's a green bar indicating "1 spec, 0 failures, randomized with seed 43715" and "finished in 0.005s". Underneath, a section labeled "calculator" shows a single test: "• 1 + 1 should equal 2".

A screenshot of a web browser window titled "mytest.html". The page displays the Jasmine test runner interface. At the top, it says "Jasmine 3.5.0" with a red 'X' icon. Below that, there's a red bar indicating "1 spec, 1 failure, randomized with seed 45174" and "finished in 0.004s". Underneath, a section labeled "calculator > 1 + 1 should equal 2" shows a failure message: "Expected 2 to be 1." followed by an error stack trace: "Error: Expected 2 to be 1. at <Jasmine> at UserContext.<anonymous> (file:///Users/Shared/workspaces/visualstudiocode/java... at <Jasmine>".



Working with Jasmine

□ Matchers

- toBe check is the same thing / toEqual checks for equality
- toBeNull tests if an expression is null
- toContain tests if a value is included in a collection
- toBeLessThan / toBeGreaterThan
- can be negated with .not

□ Matches can be used with expect

- expect(true).toBe(true);
- expect(false).not.toBe(true);
- expect(1).toEqual(1);
- expect('foo').toEqual('foo');
- expect('foo').nottoEqual('bar');



Test Fixtures

- Also have setup and tear down fixtures
- Can run before all tests in a describe block
 - `beforeAll(function() {});`
 - `afterAll(function() {});`
- Can run before each test in a describe block
 - `beforeEach(function() {...});`
 - `afterEach(function() { ...});`
- Can have any number; run in order defined



Test Fixtures

```
describe('calculator', function () {
  beforeEach(function() {
    console.log('beforeEach');
  });
  afterEach(function() {
    console.log('afterEach');
  );
  beforeEach(function() {
    console.log('beforeAll');
  );
  afterEach(function() {
    console.log('afterAll');
  );

  it('1 + 1 should equal 2', function() {
    expect(1 + 1).toEqual(2);
  );
}

it('2 - 1 should equal 1', function() {
  expect(2 - 1).toEqual(1);
}
);
});
```



Test Fixtures

mytest.html

Jasmine 3.5.0

Options

2 specs, 0 failures, randomized with seed 52219 finished in 0.005s

calculator

- 2 - 1 should equal 1
- 1 + 1 should equal 2

beforeAll simple-test2.js:9
beforeEach simple-test2.js:3
afterEach simple-test2.js:6
beforeEach simple-test2.js:3
afterEach simple-test2.js:6
afterAll simple-test2.js:12

Console What's New



Nested describe functions

```
describe("Testing the functions in myscript", function() {  
  describe("Testing divideBy(x, y) function", function() {  
    it("should return the result (2) of dividing 4 by 2", function() {  
      expect(divideBy(4, 2)).toBe(2);  
    });  
    it("should return the result (1.33333) of dividing 4 by 3", function() {  
      expect(divideBy(4, 3)).toBe(1.333333333333333);  
    });  
    it("should return the result (POSITIVE_INFINITY) of dividing 4 by 0", function() {  
      expect(divideBy(4, 0)).toBe(Number.POSITIVE_INFINITY);  
    });  
  });  
  
  describe("testing the isNumeric(num) function", function() {  
    it('should return true for "4"', function() {  
      expect(isNumeric("4")).toBe(true);  
    });  
    it('should return false for "A"', function() {  
      expect(isNumeric("A")).toBe(false);  
    });  
  });  
});
```



Nested describe functions

A screenshot of a web browser window titled "mytest.html". The browser interface includes standard controls like back, forward, and search. The main content area shows a Jasmine test run. At the top left is the Jasmine logo and version "3.5.0". To the right is an "Options" button. Below this, a green bar displays the test summary: "5 specs, 0 failures, randomized with seed 17452" and "finished in 0.011s". The test results are listed below:

```
Testing the functions in myscript
  testing the isNumeric(num) function
    • should return true for "4"
    • should return false for "A"

  Testing divideBy(x, y) function
    • should return the result (2) of dividing 4 by 2
    • should return the result (1.33333) of dividing 4 by 3
    • should return the result (POSITIVE_INFINITY) of dividing 4 by 0
```

Running from the Command Line

- Can also run jasmine from the command line
- Can use Karma
- Can also use Jasmine plus Node.js (simpler)
 - need node.js installed
 - need to have Jasmine node.js dependency installed
 - e.g. npm install -g jasmine
 - can now run tests using
 - jasmine mytest.js

Running from the Command Line

- Need to export functions / classes for node testing

```
// Check to see if running under Node.js for testing
if (typeof process !== "undefined" && process.title === "node") {
  module.exports = { enter, divideBy: divideBy, isNumeric: isNumeric };
}
```

- Import into test script
 - where they will now be prefixed by calc

```
const calc = require('../js/myscript.js');

describe("testing the isNumeric(num) function", function() {
  it('should return true for "4"', function() {
    expect(calc.isNumeric("4")).toBe(true);
  });
  it('should return false for "A"', function() {
    expect(calc.isNumeric("A")).toBe(false);
  })
});
```

Running from the Command Line

- To run Jasmine from the command line

```
jasmine test-divide.js
```

- Output generated
 - each dot is a passed test

```
Randomized with seed 15822
Started
.....
5 specs, 0 failures
Finished in 0.008 seconds
Randomized with seed 15822 (jasmine --random=true --seed=15822)
```

Testing a DOM oriented Function

- Can also test JavaScript that references the DOM
 - but may need to set up test structure
- We want to test the following code

```
function enter() {  
    var x = document.myform.xoperand.value;  
    var y = document.myform.yoperand.value;  
    var result = divideBy(x, y);  
    document.getElementById("result").innerHTML = result;  
}
```

- it reads data from a form
- performs a calculation
- and updates the DOM

```

describe("testing enter() function", function() {
  it("should access x and y from form and set result to 2", function() {
    var body = document.getElementsByTagName("body")[0];
    const myform = document.createElement("form");
    myform.name = "myform";
    body.appendChild(myform);
    const inputx = document.createElement("input");
    inputx.name = "xoperand";
    inputx.value = "4";
    myform.appendChild(inputx);
    const inputy = document.createElement("input");
    inputy.name = "yoperand";
    inputy.value = "2";
    myform.appendChild(inputy);
    const resultElement = document.createElement("div");
    resultElement.id = "result";
    body.appendChild(resultElement);
    enter();
    expect(document.getElementById("result").innerHTML).toBe('2');
  });
});
}
);

```

Running from Command Line

- Need to mock out when running from command line via jasmine command
- Can use jsdom
 - need to install that via node
 - e.g. node install -g jsdom

```
const jsdom = require("jsdom");
const dom = new jsdom.JSDOM('<html><head></head><body></body></html>');
const document = dom.window.document;
```

Debugging JavaScript

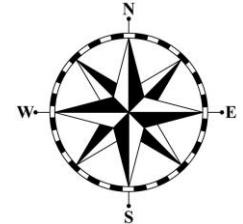


Toby Dussek

Informed Academy



framework training
business value through education



Plan for Session

- Issues with JavaScript Programs
- JavaScript Debuggers
- The Console
- Breakpoints
- Manual Breakpoints
- Watch Variables
- Dealing with Minified Code

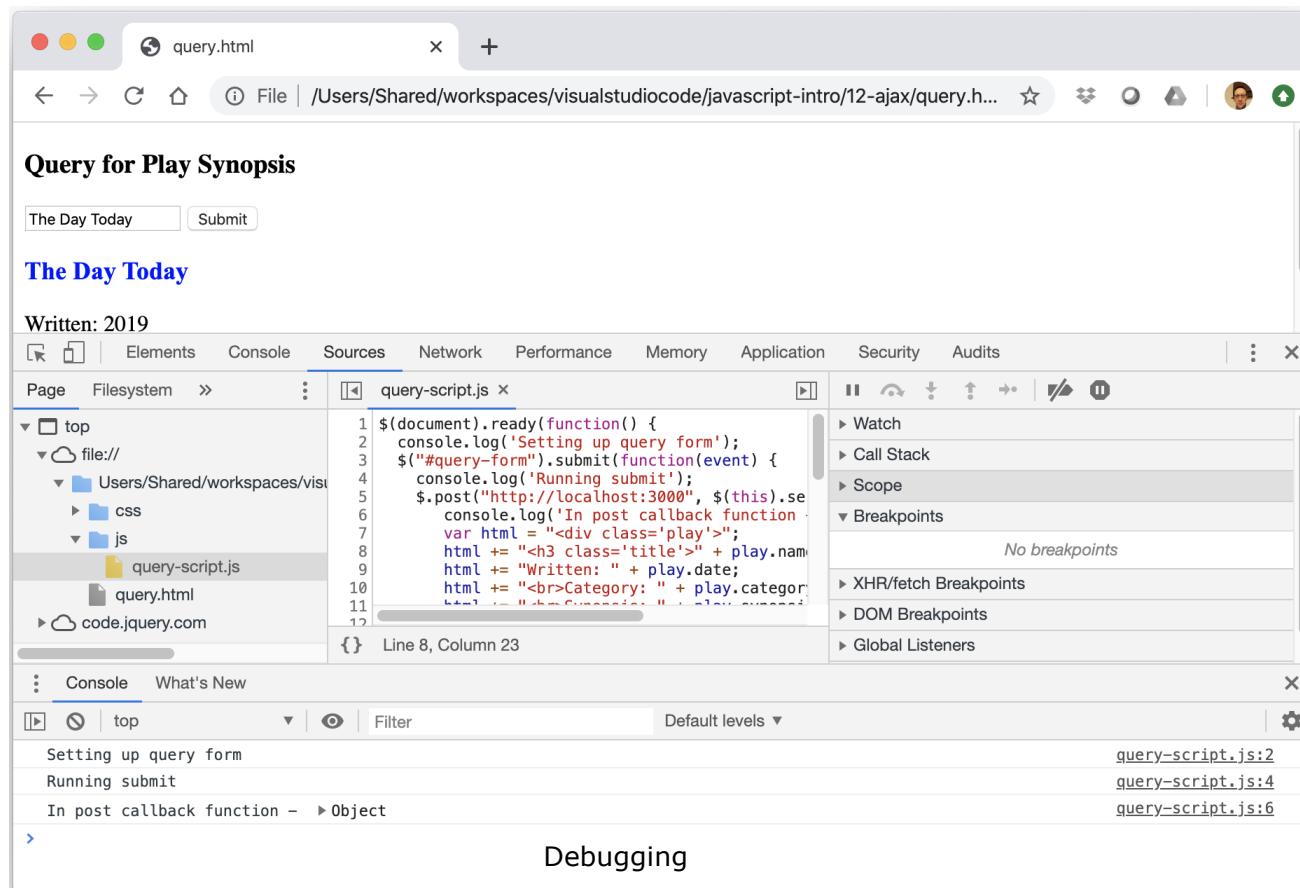
Issues with JavaScript Programs

- JavaScript programs have errors
 - syntax errors
 - logical errors
- Other issues caused by browser dependent implementations
 - IE acts differently to Chrome
 - Chrome acts differently to Firefox
 - Firefox acts differently to ...
- Often when there are errors nothing happens
 - no errors or indications where to search for errors



JavaScript Debuggers

- Chrome Developer Tools are recommended
 - activated by pressing F12



The Console

- ❑ First place to look when there is an error
- ❑ Useful for debugging syntax errors

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. On the left, the file tree shows 'query.html' is currently selected. In the main pane, a portion of 'query-script.js' is displayed:

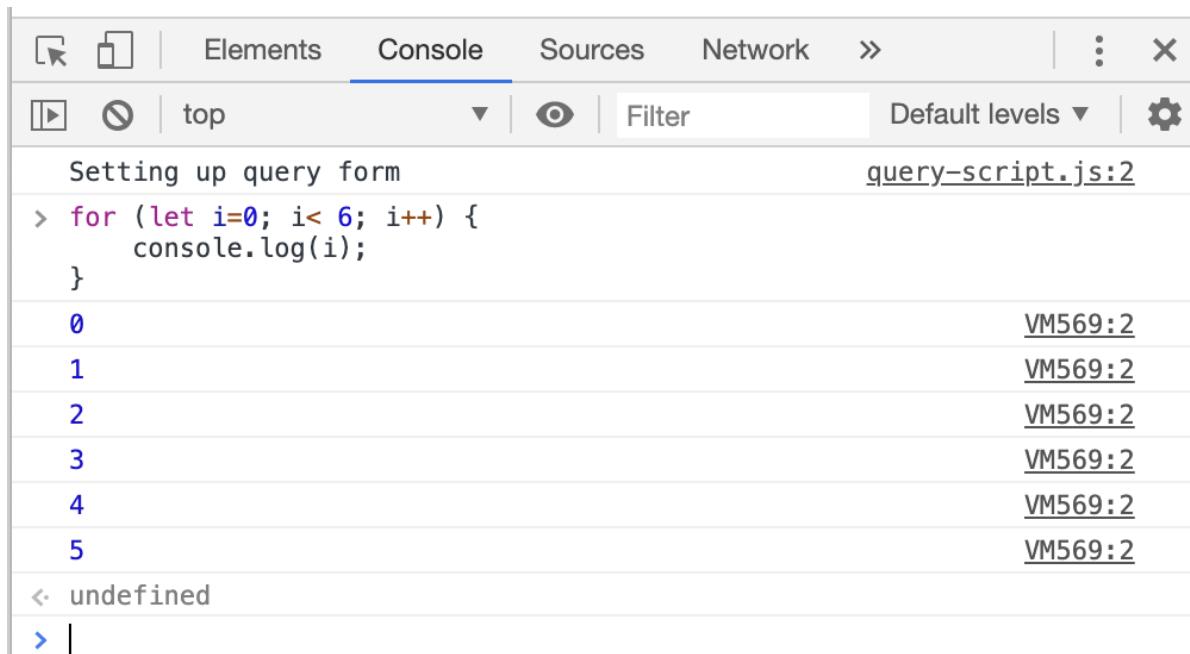
```
1 $(document).ready(function() {
2   console.log('Setting up query form'); ×
3   $("#query-form").submit(function(event) {
4     console.log('Running submit');
5     $.post("http://localhost:3000", $(this).se
6     console.log('In post callback function
7     var html = "<div class='play'>";
8     html += "<h3 class='title'>" + play.name;
9     html += "Written: " + play.date;
10    html += "<br>Category: " + play.category;
11    html += "<br>Synopsis: " + play.synopsis;
12  })
```

The code editor highlights the closing brace of the first function at line 11, column 23, with a red squiggle under it. A tooltip 'Uncaught SyntaxError: Invalid or unexpected token' points to this location. At the bottom of the DevTools window, the 'Console' tab is active, showing the same error message:

✖ Uncaught SyntaxError: Invalid or unexpected token

The Console

- The console can also be used as a REPL
 - write code and see it get evaluated immediately



A screenshot of the Chrome DevTools interface, specifically the 'Console' tab. The tab bar above shows 'Elements', 'Console' (which is highlighted in blue), 'Sources', and 'Network'. Below the tab bar, there are buttons for 'Run' and 'Stop', a dropdown menu set to 'top', a 'Filter' input field, and a 'Default levels' dropdown. The main area displays a log of console statements. At the top, it says 'Setting up query form' and 'query-script.js:2'. Below that, a for loop is executed:

```
> for (let i=0; i< 6; i++) {
    console.log(i);
}
```

The output shows the values 0 through 5, each preceded by 'VM569:2' (representing the VM context). At the bottom, there are navigation arrows and a prompt '> |'.

Breakpoints

- Very effective approach to debugging logical errors
- Pause execution of code and investigate
 - variables
 - call stack
- Chrome development tools support
 - manual breakpoints
 - conditional breakpoints
- Can add `debugger` keyword to program
 - acts like a breakpoint

Manual Breakpoints

- Can be set by clicking on the line number under Sources tab
- Can be removed by clicking the line number again

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. On the left, the file tree shows a project structure with 'file://', 'Users/Shared/workspaces/visual-studio-code', 'css', 'js', and 'query.html'. 'query.html' is currently selected. On the right, the code editor displays 'query-script.js' with the following content:

```
1 $(document).ready(function() {  
2     console.log('Setting up query form');  
3     $("#query-form").submit(function(event) {  
4         console.log('Running submit');  
5         $.post("http://localhost:3000", $(this).serialize(), function(response) {  
6             console.log('In post callback function - ', play);  
7             var html = "<div class='play'>";  
8             html += "<h3 class='title'>" + play.name + "</h3>";  
9             html += "Written: " + play.date;  
10            html += "<br>Category: " + play.category;  
11            html += "<br>Supervision: " + play.supervision;  
12        });  
13    });  
14});
```

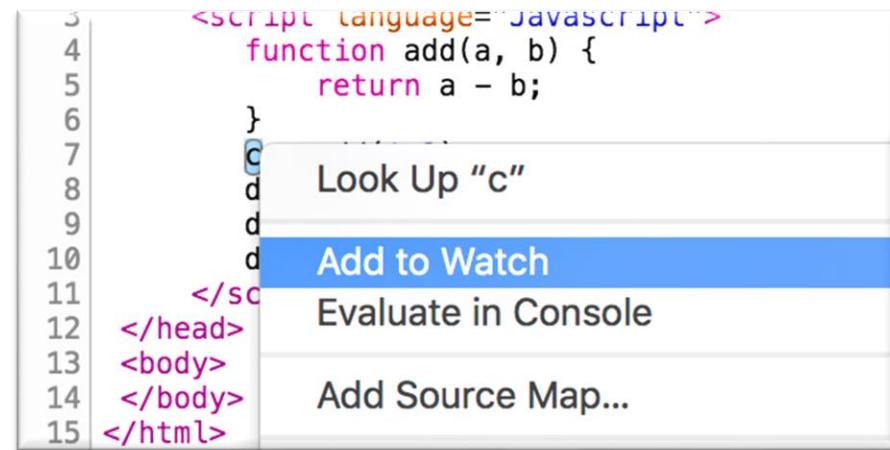
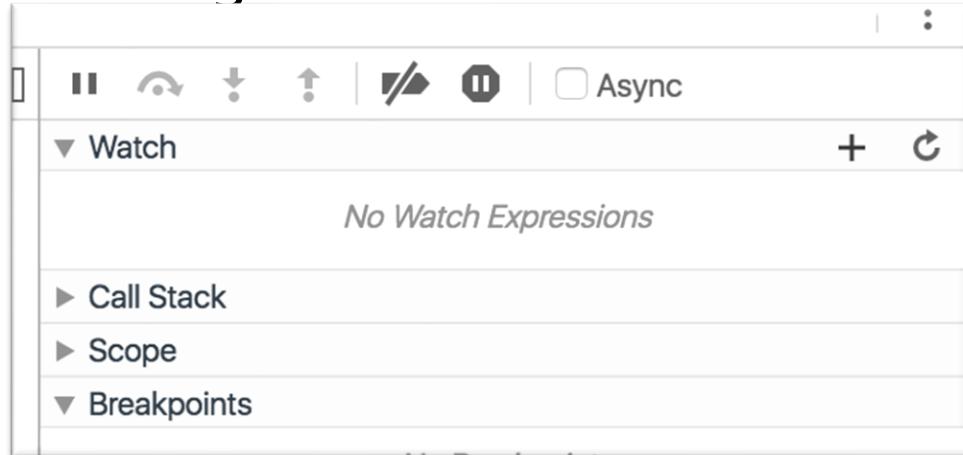
A blue arrow points to the line number 4, indicating where a manual breakpoint has been set. The status bar at the bottom shows 'Line 8, Column 23'.

Watch Variables

- Useful for debugging logic errors
 - is a variable undefined?
 - what value has a variable got
 - how is the variable changing?
- Much better approach than logging to console
 - does not change the program which can introduce other errors
- Not to be confused with breakpoints
 - only monitors what variable values are
 - program execution continues as normal

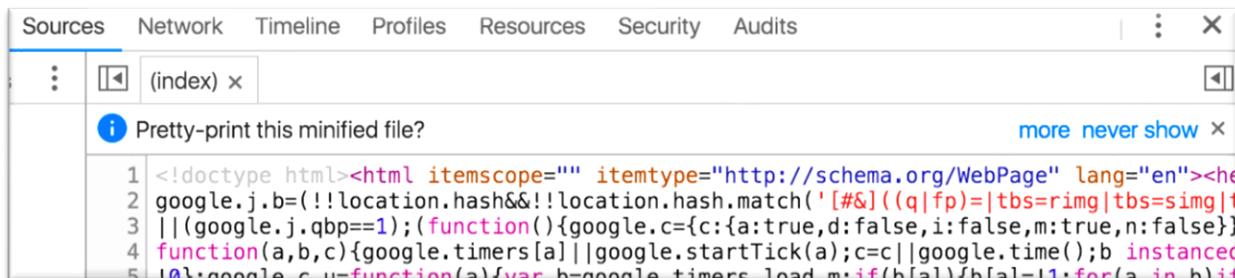
Watch Variables

- Adding watch variables



Dealing with Minified Sources

- JavaScript source code is often minified
 - a form of code obfuscation to reduce bandwidth
- Minified sources are hard to debug
 - hard to read; white space is removed
 - meaningless (short) variable names
- Google Developer Tools detects minified sources
 - provides option to “pretty-print” to help readability



The screenshot shows the 'Sources' tab of the Google Developer Tools. A file named '(index)' is selected. A tooltip message 'Pretty-print this minified file?' is displayed above the code area. The code itself is heavily minified, appearing as a single column of compressed text.

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><he  
google.j.b=(!location.hash&&!location.hash.match('[#]<(q|fp)=|tbs=rimg|tbs=simg|t  
||(google.j.qbp==1);(function(){google.c={c:a:true,d:false,i:false,m:true,n:false}  
function(a,b,c){google.timers[a]||google.startTick(a);c=c||google.time();b instanced  
10};google.c.u=function(a){var b=google.timers.load.m;if(h[a])h[a]=11;for(a in h)if
```