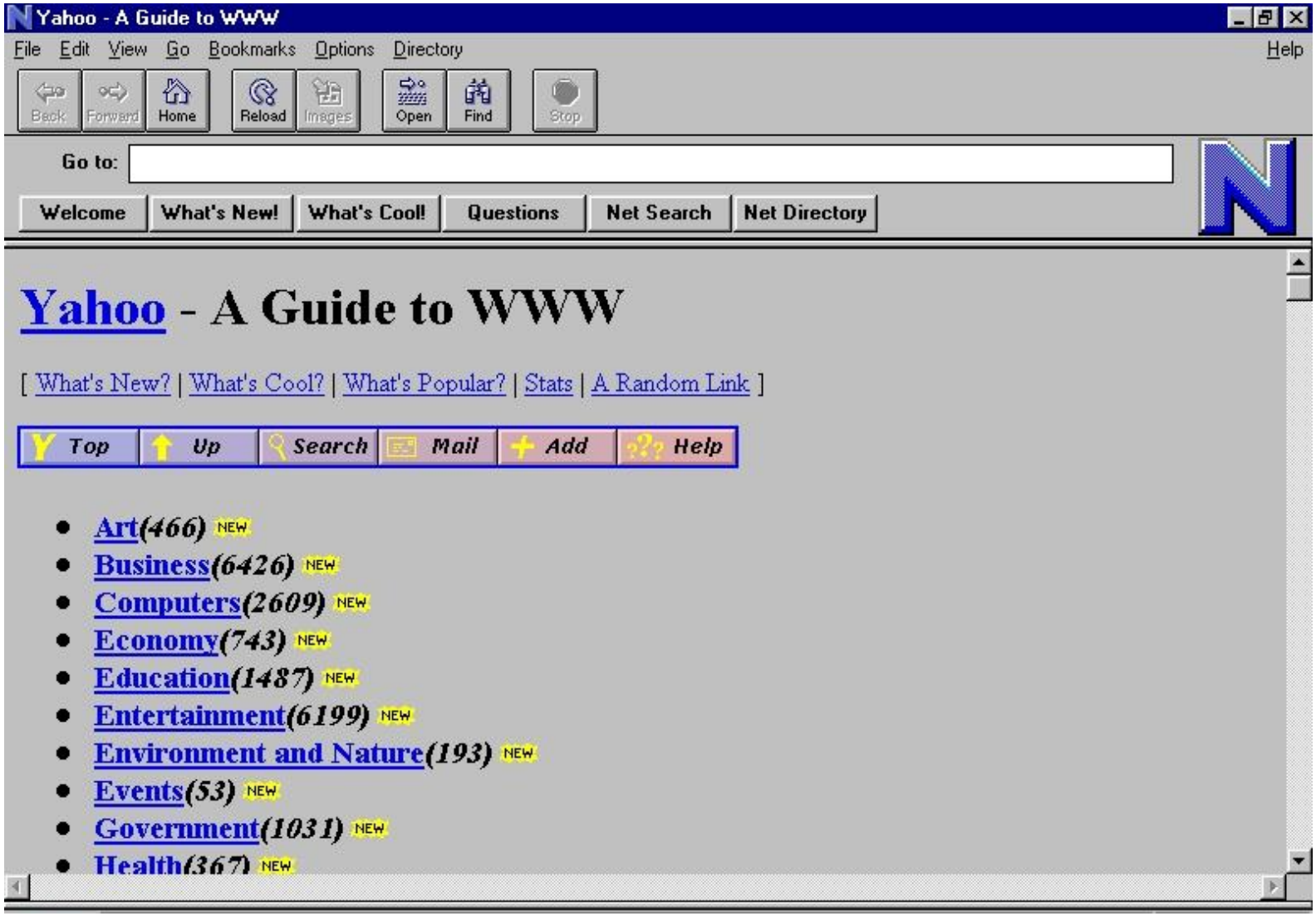
Mobile Application Development (COMP2008)

Lecture 7: Web Development

Updated: 18th October, 2018

Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

Outline



(http://www.appcessories.co.uk/where-are-alta-vista-geocities-friendster-hotornot-now/.)

Scope of this Lecture

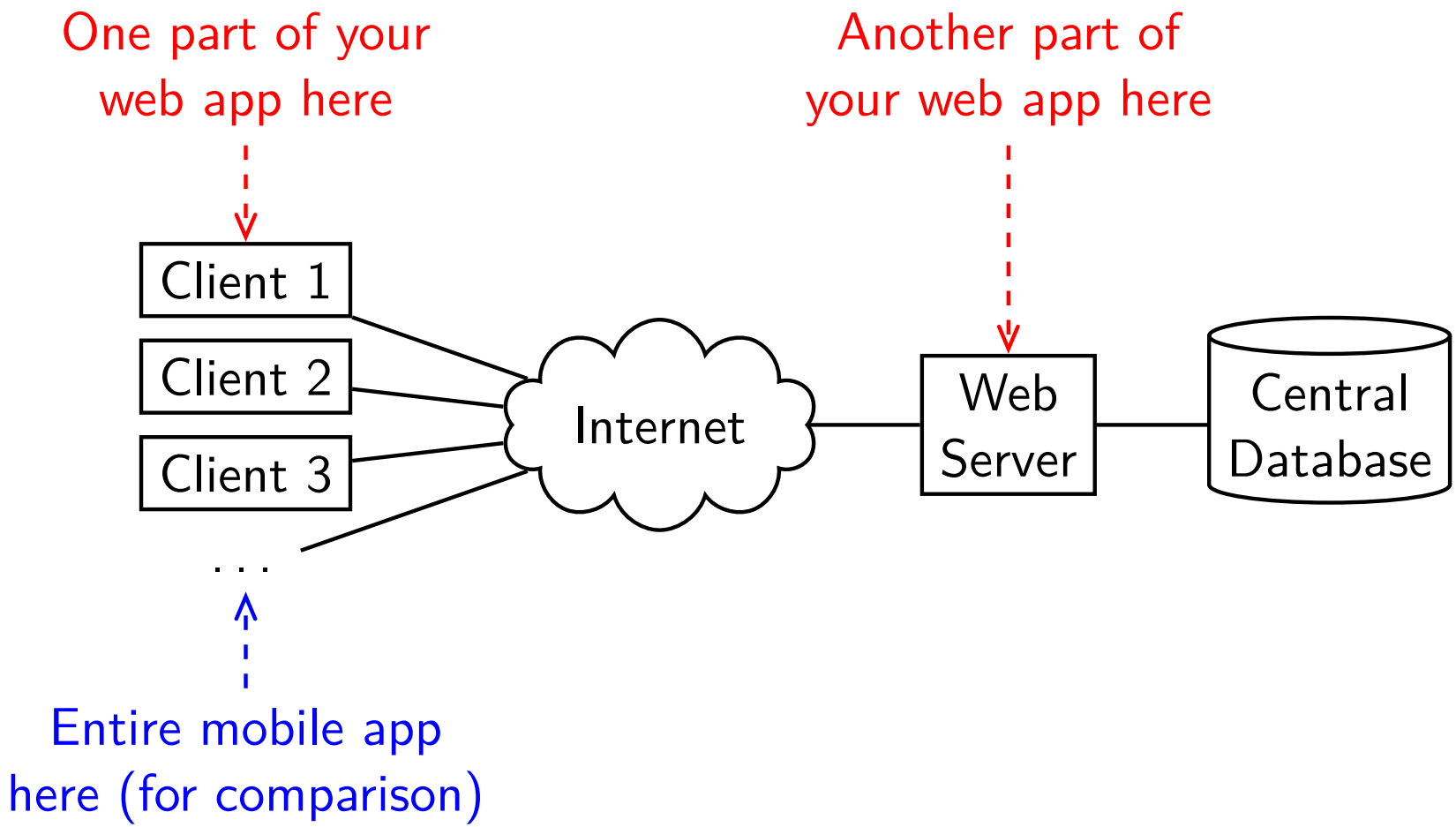
- Web development could easily be *several entire units*.
- We only have time to cover a relatively small part of it.
- But which part(s) *should* be covered?
  - We will doubtless have debates about this.
- I've chosen to try to cover some of the lower-level building blocks of web development.
  - HTML, CSS, JavaScript, jQuery.
- There are certainly also higher-level frameworks that fit over the top.
  - You tend to use the high-level stuff in practice, but they *change a lot*.

Web Development

- Web apps can often fulfil the same requirements as mobile apps.
- Can be a quicker, cheaper option.
  - *One* well-written web-app will run on practically any device.
- Web apps definitely require a server, and internet access.
  - But then many mobile apps do too anyway.
- Of course, the user experience is not quite the same.
  - The user will have to bookmark your web app, or remember the URL.
  - Web apps are slower – there's an extra level of language interpretation.

Web App Architecture

- Central challenge in web development: your app is slit into two parts.
- Some things happen on the client, and some on the server.



Web App Architecture

The sequence of events is basically this:

1. The user enters the URL of your web site (app) into their browser.
2. The web app starts up on the server (and reads its database if needed).
3. The web app/server sends back a response, consisting of:
   - An HTML document and CSS "stylesheet(s)";
   - JavaScript code.
4. The JavaScript code is the *client* part, and now executes inside the user's web browser.
5. The client periodically communicates with the server as needed.
   - A bit like a mobile app communicating with a web service.
   - . . . Except here the client and server are two separated parts of the same app.

HTML (HyperText Markup Language)

- Looks very much like XML, but technically different.
  - Both HTML and XML are derived from Standard Generalized Markup Language (SGML).
- Originally designed to be a document format.
  - And still is, really.
  - Contains elements for formatting paragraphs, lists, tables, etc.
  - Ebook formats (epub) do *now* what HTML was originally designed to do.
- Now used as the basis for web apps' user interfaces.

HTML Document Syntax

```
<!DOCTYPE html>
<html>
    <head>
        <title>My HTML Document</title>
        ...
    </head>
    <body>
        ...
    </body>
</html>
```

- <head>...</head> – metadata, including links to CSS and JavaScript.
- <body>...</body> – structure and contents.

Playing at Home

- Most browsers support these features:
  - Press Ctrl-U to see the HTML source for the current webpage.
  - Press F12 to open a more interactive developer-tools window.

HTML Tags and Elements

- <body> contains plain text intermixed with other tags. e.g.

```
<div>
    <p>Some paragraph text with an
       <em>emphasised</em> word.</p>
    <p>And <span>another</span> paragraph.</p>
</div>
```

- Tags define a hierarchy – a tree of tags and text.
- Stylistic meaning:
  - Lots of tags have specific stylistic meaning; e.g. <p> = paragraph, = emphasis (typically an italic font).
  - However, and don't "look" like anything, by default.
- Boxes and inline text:
  - Some elements are "boxes" of content (e.g. <p> and).
  - Some elements rendered as free-flowing text (e.g. and).

HTML Classes and IDs

- Any element in the body can have:
  - A unique ID value.

```
<div id="toolbar">...</div>
```

  - Any number of (space-separated) "class" values
    - Really a kind of *tag*, unrelated to OO classes.

```
<div class="section">...</div>
<p class="section highlighted">...</p>
```

  - Both, or neither.
- These are important for:
  - Defining styles – how the document is displayed.
  - Letting your code access parts of the HTML document.
    - To query what's displayed, or change it.

Forms

- HTML was always (more or less) designed to be interactive.
  - (It was never really a "pure" document format.)
- Most user input is done via `<input>` elements.
- These must (technically) be within `<form>...</form>`.
  - Forms can make the web browser perform an HTTP GET/POST request directly – without any code.
  - *But* this creates a major usability headache.
  - Forms a reload of the entire page (code and data) from the server!
  - We now typically avoid that...
- ...And use our own JavaScript code to examine the `<input>` elements.

<input>

- Two important attributes: `type="..."` and `value="..."`.
- There are numerous types: `button`, `text`, `number`, `email`, `date`, `password`, `checkbox`, `radio`, etc.
- You can specify an initial value.

```
<form>
    <p>Your email address:
       <input id="eml"
              type="email"
              value="you@example.com" /></p>
    <p>Your password:
       <input id="pw" type="password" /></p>
</form>
```

Events

- We also need to know about button presses (and sometimes other events).
- We can attach an `onclick="..."` attribute to any element:

```
<form>
    ...
    <input type="button"
           value="Do Something"
           onclick="doSomething()" />
</form>
```

- The value (in quotes) is a snippet of JavaScript code.
  - Typically a function call.

DOM (Document Object Model)

- HTML documents are represented in memory as the "DOM".
  - Basically a tree of objects.
  - Each pair of tags (e.g. ...) becomes an object in the tree.
    - It's child nodes/objects are those between its open and close tags, including any plain text.
  - The root element is called document.
- Find elements by ID (unique) or by tag name (multiple):

```
let element = document.getElementById("toolbar");
```

```
let sectionArray =
    document.getElementsByTagName("toolbar");
```

- You can navigate between parent, child and sibling nodes, but this is cumbersome.

DOM Objects

- DOM objects are *initially* based on HTML, but at runtime they have state.
- We can get and set the ID, class, and other attributes:

```
let element = document.getElementById("toolbar");
...
console.log(element.id);
console.log(element.className); // Not just 'class'
console.log(element.getAttribute("foobar"));
element.id = "new_id";
element.className = "new_class";
element.setAttribute("foobar", "new_value");
```

DOM Objects

- User input from form elements is likewise accessible:

```
let checkbox = document.getElementById("mycheckbox");
if(checkbox.checked) {...}
```

```
let textfield = document.getElementById("mytextfield");
console.log(textfield.value);
```

Cascading Style Sheets (CSS)

- While HTML describes document structure, CSS describes what it looks like on the screen.
- CSS "code" normally goes in a .css file, linked to from the HTML document:

```
<html>
    <head>
        <link rel="stylesheet"
              type="text/css" href="my_style.css"/>
        ...
    </head>
    <body>...</body>
</html>
```

- The web browser then asks the server to give it `my_style.css`.
- (You can also embed CSS directly inside the HTML.)

CSS Syntax

- CSS files consist of several *rules*.
- Each rule has:
  - A *selector*, an expression that identifies part(s) of the HTML/DOM.
  - One or more *declarations*, which specify values for various properties – fonts, colour, size, etc.

```
selector {
    property: value;
}

selector {
    property: value;
    property: value;
}
...
```

CSS Selectors

- You can select elements by tag name, ID, class, and other attributes.

```
* {...}          // Select everything
p {...}          // Select all paragraphs.
#toolbar {...}   // Select element with id="toolbar".
.section {...}   // Select all elements with
                 //   class="section ...".
[type="text"] {...} // Elements with type="text".
```

- We can combine these (no spaces!):

```
p.section {...}           // <p class="section ...">
input[type="text"] {...} // <input type="text">
.section.highlighted {...} // Multiple classes
.section[x="5"][y="10"]    // Class & attributes
```

- *#id* is already unique, so no point combining with others.

CSS Selector Combinators

- Selectors can also look at the tree structure:

```
p .section {...}                                    // Example A
```

```
.section[x="5"] *[type="text"] {...} // Example B
```

  - Two selectors separated by a space – matches the second anywhere within the first.
  - Example A: searches for <p> first, *then* searches for `class="section ..."` within those particular paragraphs only.
  - Example B: finds elements with `type="text"` *within* elements with `class="section ..."` and `x="5"`.

- Can also perform a simple "or" operation:

```
p, .section {...}
```

  - Two selectors separated by a comma – matches either one.

CSS Selectors: Pseudo-Classes and Pseudo-Elements

- "Pseudo-class" selectors identify elements by their state:

```
a:visited {...} // Select links (<a> elements) that
                // have been visited.
```

- "Pseudo-elements" can be created and added into the document by CSS.
  - `::before` or `::after` existing elements.
  - Purely for stylistic purposes.

```
// Selects/creates a new pseudo-elements before
// each .section element.
.section::before {
    content: "Welcome to a new section...";
    ...
}
```

CSS Declarations

There are a huge number of properties. Some include:

- `border:` *thickness style colour*`;`
  `border-`*position*`:` *thickness style colour*`;`

```
border: 2px solid red;
border-left: 1px dashed gray;
```

- `font-family:` *font [backup-font ...]*`;`

```
font-family: monospace;
```

```
font-family: Ariel Helvetica sans-serif;
```

- `position:` *static | absolute | fixed | relative | sticky | ...*`;`
  `width:` *distance*`;` `height:` *distance*`;`
  `left:` *distance*`;` `top:` *distance*`;`
  - Where and how big should the element(s) be?

CSS and the DOM

- We often want to dynamically change how things look at runtime.
- You *can* set CSS properties dynamically:

```
let element = ...;
element.style.color = "blue";
```

- *However*, probably easier and neater to have fixed CSS declarations, and instead change the class/attributes:

```
let element = ...;
element.className = "new_class";
element.setAttribute("x", "abc");
```

- Such changes take effect at the end of the current event.
  - Practically all your JavaScript code responds to events.
  - When each one is done, the page will repaint itself if needed.

JavaScript: The World's Best Worst Language

- Online documentation:
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript
  - https://www.w3schools.com/js/default.asp
- JavaScript $\neq$ Java. (Not even close!)
  - The name "JavaScript" came from a commercial branding exercise.
  - Java and JavaScript are not otherwise related.
  - JavaScript follows a standard called "ECMAScript".
- Interpreted. All web browsers contain JavaScript interpreters.
- Dynamically-typed, like Python.
  - There are datatypes, but you don't declare them.
- Superficially C-like syntax: `if(...) {...}`, etc.
- Object oriented, *sort-of*.

JavaScript as a Platform

- Not everyone wants to write in JavaScript itself.
- You can treat it as a *platform* instead.
  - You can compile, or "transpile", most other languages *into JavaScript*.
  - (In particular, we'll look at TypeScript later on.)
- Thus, we can (sometimes) view JavaScript as a kind of virtual machine language.
  - More comparable to the JVM rather than Java itself.
  - Though not as efficient.
- Since 2017, there is also WebAssembly.
  - A "proper" virtual machine language, designed to support C/C++ (and everything else too).
  - Fills this role more efficiently.
- *But*, regardless of WebAssembly and transpiling, JavaScript itself is still important.

Alerts, Logging and Debugging

- Being interpreted, all errors happen at runtime!
- Extremely important to know where to look.
  - If running in a web browser (the usual case!) press F12, and view the "console".
  - This is where most error and warning messages are shown.
- You can log messages to the console as follows:

```
console.log("Hello world");
```

- You can also pop-up an alert message:

```
alert("Out of Cheese Error.");
```

  - This can be used for debugging, but also (perhaps?) for genuine error messages.
  - Can be annoying, because it prevents the user doing anything else until they press "OK".

NodeJS

- NodeJS is a standalone implementation of JavaScript – i.e. not in a web browser.
- You can use it to play around with the language:

```
[user@pc]$ node
```

```
> console.log("Hello world");
```

- Although web-browser-specific things won't work.
  - There's no window, document, alert, etc.
- It can be used to write server-side code, or other miscellaneous scripts.

JavaScript – the Hairy Bits

- JavaScript has "optional" semicolons, which means:
  - Don't split a line if both parts would be valid statements.
  - e.g. this *looks* like it returns an array, but it doesn't!

```
function getArray()
{
    return                      // Returns nothing.
        [1, 2, 3, 4, 5];        // Separate "expression
}                               //   statement".
```

- Use === and !== (rather than == and !=).
  - Only === tests for exact equality.
  - == first converts both sides to the same type.
  - `0 == "0"`, `0 == []`, `0 == ""`, `0 == [" 000 "]`
  - `"0" != []`, `"0" != ""`. Obviously...
- Mixed-mode arithmetic is even more bizarre.
  - Though fortunately you should never need to do it.

JavaScript Objects

- JavaScript objects are maps of key-value pairs.
  - (Probably implemented as a hash-table underneath.)

```
let myObj = {       // Very similar syntax to JSON
    name: "Earth",
    result: 42
};
```

- Classes exist. Use them as normal.
  - They have constructors, methods, fields, inheritance, etc.
  - *But* you can make objects *without* classes!
  - Classes weren't in the language originally.
  - We're going to gloss over them in this unit.
- You can access object fields in two ways:
  - `myObj["name"]` or `myObj.name`
  - So, you can use an object as an OO-type object, *or* a map, or both.

JavaScript Functions

- JavaScript functions *are values*. You can call them, but also...
- Define anonymously, and assign to variables:

```
let g = function(x, y) { return x + y; }
let z = g(5, 10);
```

- Assign to object fields:

```
let myObj = {
    name: "Earth",
    show: function()
    {
        console.log(this.name);
    }
};

myObj.show(); // Prints "Earth".
```

JavaScript Functions

- Pass them to other functions.

```
function a(i) {...}

function b(theFunction, times)
{
    for(let i = 0; i < times; i++)
    {
        theFunction(i); // Calls a().
    }
}

b(a, 10); // Pass function a to function b.
```

JavaScript Functions

- Return them from other functions.

```
function makeFunction(x)
{
    function a(y)
    {
        console.log(x * y); // Captures x from the
    }                       // outer function.

    return a; // Returns function a.
}

theFunction = makeFunction(5);
theFunction(10); // Prints 50.
```

JavaScript in HTML

- Typically kept in a `.js` file, included with a `<script>` tag:

```
<html>
    <head>
        <script src="mycode.js"></script>
        ...
    </head>
    <body>...</body>
</html>
```

- Somewhat frustratingly, the browser executes this code *before* the page has finished loading!
  - You won't reliably be able to find parts of the HTML.
  - So...

JavaScript – window.onLoad

- Fortunately, your code can subscribe to onLoad events generated by the window (or document) objects:

```
function initialiseThings()
{
    ...
}

window.onload = initialiseThings;
```

- window.onload is expected to be a function, and will be called when everything has been loaded.
  - Now your code knows that the UI is complete.
  - It can get references to whatever elements it needs, and set up other event handlers.

jQuery

- Technically a third-party library, but so widely used as to be practically standard.
- But you do still have to load it; e.g.:

```
...
<script src="https://code.jquery.com/jquery-3.3.1.min.js">
                                                    </script>
<script src="mycode.js"></script>
...
```

- Often loaded directly from `jquery.com` (as above), or (e.g.) `ajax.googleapis.com`.[1]
- But it's freely-distributable, so you can serve it from your own server too.

[1] See https://developers.google.com/speed/libraries/#jquery, https://code.jquery.com/.

jQuery

- Makes it easier and quicker to manipulate the HTML.
- Adds in a new function called "$".
  - This is the entry point into the jQuery API.
  - (Fun fact: `$` is accepted as a valid variable/method/class-name character in many C-like languages.)
- Often, you call it like this:

```
$("div.section p").doSomething();
```

  - The parameter here is a CSS-like selector.
  - Returns an object that represents *all* the matching elements.
  - The returned object has methods for manipulating all those elements simultaneously.
- You can also stuff an existing element into it:

```
let element = ...;
$(element).doSomething();
```

jQuery Events

- jQuery provides alternate ways to set event handlers.
- On the page loading:

```
$(window).on("load", initialiseThings);
```

- On button clicks (instead of onclick="..."):

```
$("#myButton").on("click", function() {...});
```

```
$("input[type=text]").on("click", updateData);
```

  - Implied loop. Sets event handler *for every matching element*.

- Quite likely we'll have nested calls:

```
$(window).on("load", function()
{
    $("input[type=text]").on("click", function() {...});
    ...
});
```

jQuery Iteration

- You can iterate over a jQuery object like an array. However, you can also do this:

```
$("input[type=text]").each(function(index, obj)
{
    console.log(index + ": " + obj.value);
});
```

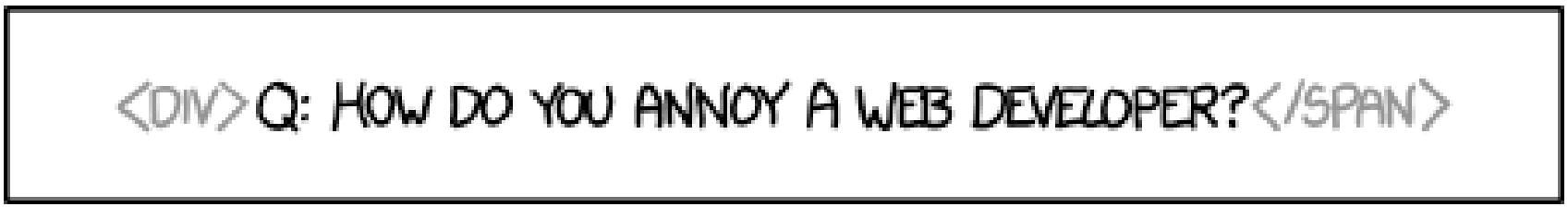
jQuery Modification

- You can trigger style changes by, say, adding/removing classes:

```
$("input[type=text]").addClass("highlighted");
...
$("input[type=text]").removeClass("highlighted");
```

- You can even replace whole sections of HTML dynamically:

```
$("#toolbar").replaceWith(
    '<div id="toolbar">Exciting new things</div>');
```



(https://xkcd.com/1144/)