Cardiff's School of Computer Science & Informatics

CM1102 "Web Applications"

JavaScript

Dr Natasha Edwards

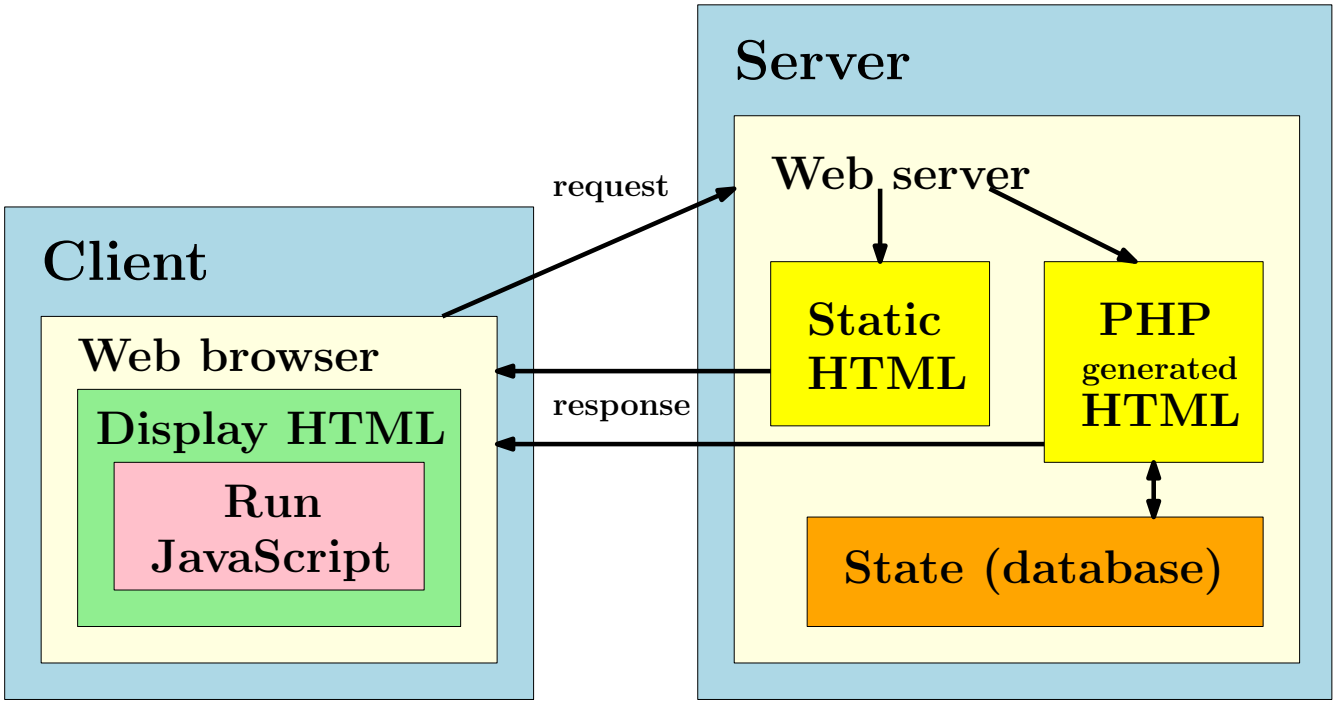
EdwardsN@cardiff.ac.uk

Learning Objectives

- Understand aspects and concepts related to web development, such as **server-side** and **client-side** processing and programming, **DOM**
- Learn how to start programming **JavaScript** by examining the main aspects of the syntax:
  - Variables, data types, operators, functions, conditional statements, looping
  - Event-based programming
  - Built-in objects and their methods
- Introduce HTML, JavaScript and DOM; and AXAJ.

Client-side vs server-side processing

- **Client-side**: processing is done locally on the client.
- **Server-side**: processing is done on server.



Server-side vs client-side processing

Server-side scripting/programming:

- Appropriate when non-trivial processing has to be carried out on the server.
- In general, when the **state** of the system is stored on the server, and the pages served to the user depend on the state:
  - Interactive access to centralised databases or other resources.
  - Social networks, CMSs, e-commerce, *etc.*

Client-side scripting/programming:

- Good for user interactivity, dynamic applications.
- Can be combined with access to server resources with AJAX

Client-side programing in web pages

- Various client-based methods of scripting/coding to increase interactivity:
  - Java Applets
  - Active X
  - Flash
  - VB Script
  - JavaScript
- We are going to consider JavaScript.

JavaScript

- Javascript was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.
- ECMA-262* is the official JavaScript standard. The standard is based on JavaScript (Netscape) and JScript (Microsoft).

  (* ECMA = European Computer Manufacturers Association)
- The development of the standard is still in progress.
- Aspects of JavaScript **syntax** are similar to that of Java and C, but the language is **not related** to Java!

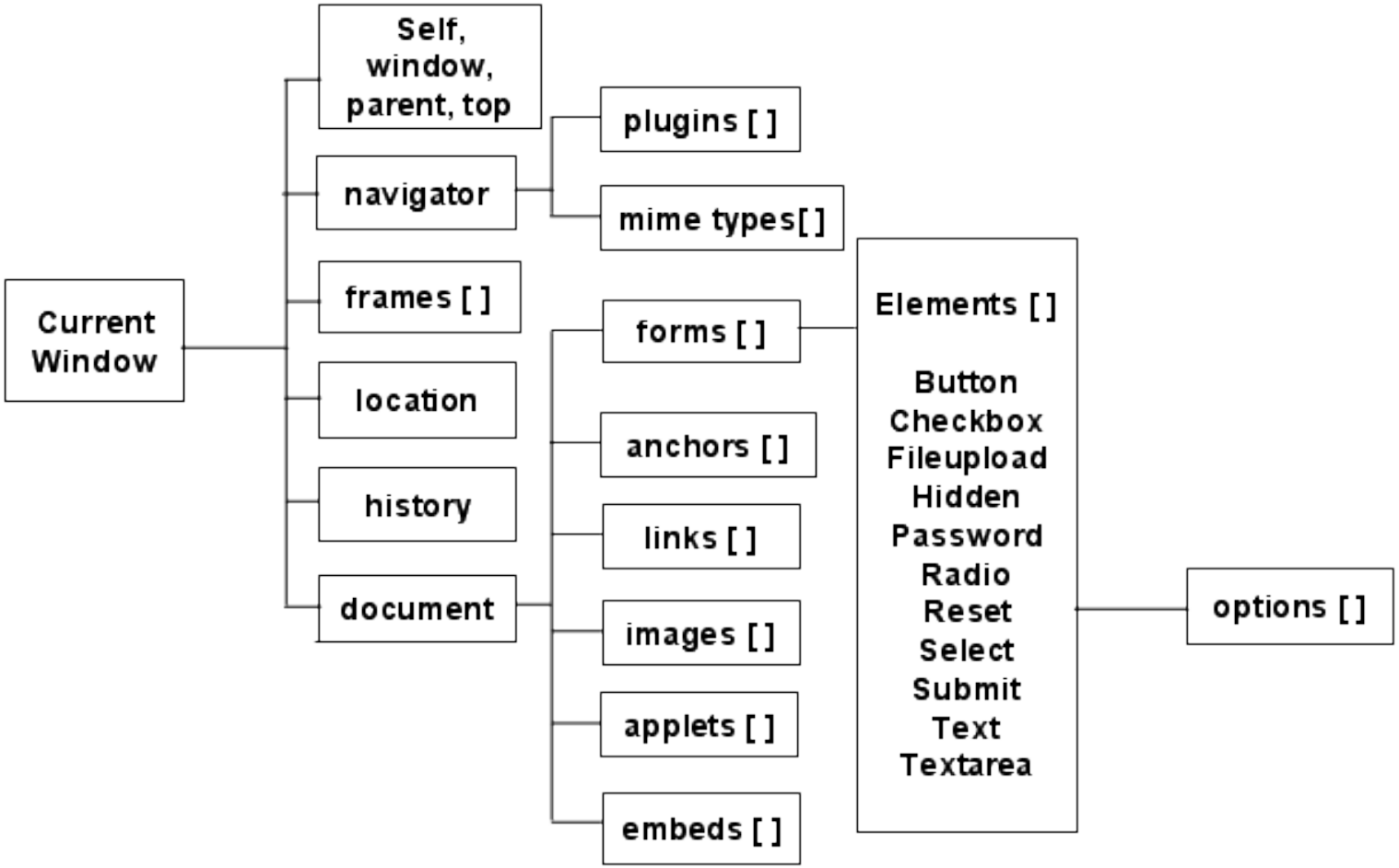
What is JavaScript useful for?

- Add content to a web page dynamically (as the web page is loading).
- Read and write HTML elements and hence change the content of an HTML element.
- Manipulate the content of a web page in response to user actions (respond to "events").
- React to user generated events:
  - For example, validate the contents of a form prior to submission to the server, *e.g.* make sure all fields are filled in.
  - Provide help with input.
- Turn on and off layers of a document.

Document Object Model (DOM)

- JavaScript can manipulate **objects** associated with a web page.
- The objects have methods. For example `writeln` in `document.writeln`.
- The objects are arranged into a hierarchy of objects called the Document Object Model (DOM).
- DOM is a convention for representing and interacting with objects in HTML, XHTML and XML documents.
- At the top of the hierarchy is the `window` object. The `document` object is a child of the `window` object.

Original view of DOM (first version)



Where to put JavaScript code?

- Code and function definitions are placed inside the `<script>` element.
  - Can also use `<script type="text/javascript">`, but modern browsers assume the script is JavaScript by default
  - `language="javascript"` attribute is deprecated (not part of current standards).
- Functions (inside the `<script>` element) are best defined within the `head` element.
- Functions can be called by JavaScript commands inside `body` element and in combination with event properties in HTML tags.

Where to put JavaScript code?

- For modularity and repeated use of functions, save to a file and reference as follows:

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

- Some consider using external scripts as best practice

Browsers that do not support JavaScript

- Browsers that do not support or disable JavaScript will display JavaScript as page content.
- To prevent them from doing this the HTML comment tag can be used to "hide" the JavaScript. Just add an HTML comment tag `<!--` before the first JavaScript statement, and a `-->` (end of comment) after the last JavaScript statement.
- Can also insert:

```
<noscript>
  Your browser does not support JavaScript!
</noscript>
```

Hello World!

```
<html>
<head>
<title>Hello World!</title>
<body>
  <script>
      document.write("Hello World!");
  </script>
 </body>
 </head>
</html>
```

Variables

- Variables are used to store data
- Variables are declared using the `var` keyword or a variable name; values are assigned using the equals operator `=`, e.g.

```
var name, age;
var today = "Tuesday";
var myName = "Natasha";
```

- Variable is *undefined* of no value is given.

Variables: Rules

- Use of **letters**, digits, and underscores are allowed.
- Must begin with a letter or underscore *(similar to some other programming languages)*.
- Names are case-sensitive.
- As Python, the typing is dynamic.
- Do not call two variables the same name with different cases to distinguish them (for example, `username` and `UserName`) because this is a common source of confusion later.
- Use descriptive names to make your code easier to understand.

Variables: Lifetime

- When declared in a function, the variable can only be accessed within that function.
- When you exit the function, the variable is destroyed.
  - = **local** variables
- If variable declared outside a function, all the functions on the page can access it.
  - = **global** variables
- The lifetime of these variables starts when they are declared, and ends when the page is closed.

Numbers, Strings, Arrays

```
var a = 42;
a = 3.14159;
var aStr = "This is a 'bStr' in aStr";
var bStr = "This is a \"cStr\" in a bStr";
var myColors = new Array("red","green","blue");
var things = ["red", "green", 2, 66];
var stuff = array(100);
```

Arrays are heterogeneous collections: its elements can have different types! Indexing is zero-based.

Operators

- **Arithmetic**
  - addition (`x+2`); subtraction (`x+2`); multiplication (`x*2`); division (`x/2`); modulus (`x%2`); increment (`x++`); decrement (`x--`);
- **Assignment**
  - The basic assignment operator is `=`; used to assign a value to the variable
  - Shorthand assignment operators: `+= -= *= /= %=`, e.g. `x+=y` is equivalent to `x=x+=y`
- **Comparison**
  - To compare two operands and then return either true or false: `==, !=, >, <, >=, <=`
- **Logical (or Boolean)**
  - AND (`&&`), OR (`??`), NOT (`!`)

String Operators

- You can use the + operator to concatenate strings, e.g.

```
var firstName = "Jane";
var lastName = "Doe";
name = firstName + lastName;
```

The output will be: *..... ??*

- You can also use the comparison operators we have just seen to compare strings.

Functions

Again, a familiar concept:

```
function add(a, b)
{
  var sum = a + b;
  return sum;
}
var theSum = add(2, 4);

function printAll() {
  for (var i = 0; i < arguments.length; i++) {
    document.writeln(arguments[i] + "<br />");
  }
}
printAll(42, "Phasor");
```

Conditional statements

- `if`
- `if ... else`
- `switch`

Conditional statements

```
var pints = 5;
if (pints > 2) {
  document.writeln("You cannot drive a car!<br />");
  if (pints > 6) {
    document.writeln("Call a cab!<br />");
  } else {
    document.writeln("Ride your bicycle!<br />");
  }
} else if (pints > 0)
  {
    document.writeln("Drive very cautiously!<br />");
  } else {
    document.writeln("It is ok to drive!<br />");
  }
```

Example: average age

```
<html><head></head><body>
<script>
<!--
  var age, averageAge, counter, total, ageValue;
  total = 0; ageValue = 1; counter = 0;

  document.writeln("<h2>Average age of the class</h2>");
  while (ageValue > 0) {
    age = window.prompt("Enter age (0 to end): ","0");
    ageValue = parseInt(age);
    if (ageValue > 0) {
      total = total + ageValue;
      counter++;
    }
  }
  averageAge = total/counter;
  document.writeln("<h2>Average = " + averageAge +"</h2>");
//-->
</script></body></html>
```

Loops

- **The familiar `while` loop:**

```
maxval = 5; count = 0;
while(count < maxval) {
  document.writeln("value is " + count);
  document.writeln("<br>");
  count = count + 1;
}
```

- **And the `do...while` loop:**

```
maxval = 5; count = 0;
do {
  document.writeln("value is " + count);
  document.writeln("<br>");
  count = count + 1;
} while (count < maxval);
```

Loops

- `for` loop is also available:

```
<html><head></head><body>
    <script>
      <!--
      var maxval = 5;
      var myArray = new Array(maxval);
      for (j=0; j<maxval; j++) { myArray[j] = 2*j;
      for (j=0; j<maxval; j++) {
        document.writeln("value " + j + " is " +
            myArray[j]);
        document.writeln("<br>");
      }
//-->
</script></body></html>
```

Events

- JavaScript is an **event-driven** language.
- You can write a block of Javascript code that does something in response to the action of the user, e.g.:
  - Open a new browser window with annoying ads when a user clicks on a link.
  - Validate user input from a form when user clicks the submit button.
  - Cause graphic effects such as rollovers when mouse moves over an element.
- Multiple types of event, associated with particular types of element.

Events

Functions are often called in response to events associated with HTML elements.

- `onLoad` is associated with the `body` element.
- `onSubmit` is associated with the `form` element.
- `onMouseOver`, `onMouseDown`, `onClick`, `onBlur`, `onFocus`... in response to mouse events.

Some useful events

- `onClick`: mouse button is clicked on an element (used with the `button` and `link` elements).
- `onMouseOver`/`onMouseOut`: mouse moves into/out of an element (with `link`, `image` and `layer`).
- `onMouseDown`/`onMouseUp`: mouse button is pressed/released.
- `onLoad`/`onUnload`: when browser loads/finishes with a document (used with the `body` element).
- `onFocus`/`onBlur`: when an element is selected/deselected (i.e. user clicks on it/selects another element) with the mouse (`form` elements: `input`, `select`, `textarea` and `links <a>`).
- `onSubmit`: when the submit button pressed in a form.

Example: change background colour

```
<head>
 <title>Example</title>
  <script>
 <!--
  function change(col) {
    if(col=="red") {
      document.body.style.backgroundColor = "#ff0000";
    }
    if(col=="blue") {
      document.body.style.backgroundColor = "#0000ff";
    }
  }
 //-->
 </script>
</head>
<body>
 <form>
 <input type="button" value="Red" onClick=change("red")>
 <input type="button" value="Blue" onClick=change("blue")>
 </form>
</body>
```

Example: rollover

```
<html>
 <head>
  <script>
   function mouseOver(myImage) {
        myImage.src = 'fp.jpg';
     }
   function mouseOut(myImage){
        myImage.src = "kirk.jpg";
     }
  </script>
</head>
<body>
 <img src="kirk.jpg" onmouseover="mouseOver(this)"
                                 onmouseout="mouseOut(this)" />
</body>
</html>
```

Built-in objects

- String
- Array
- Math
- Date
- Window

String manipulation

Methods of `String` objects:

- `charAt(index)` returns the character at position `index` in the string.
- `concat("string"[, "string"[, "..."]])` concatenates: `astring.concat(bstring)` adds `bstring` to the end of `astring`. Can also do with `astring + bstring`.
- `length` returns the number of characters in the string.
- `split(separator)` breaks the string apart whenever it encounters the separator character (and returns pieces in an array), like `explode` in PHP.
- `toUpperCase`/`toLowerCase` convert the string to upper/lower case.

String manipulation

Methods of `String` objects:

- `indexOf("search"[, offset])` searches for the string in the first parameter and returns the start of the target string, returns -1 if unsuccessful. Offset is start of search and = 0 by default but can be set.
- `substr(index[, length])` returns a substring which starts at the character indicated by the `index` parameter. Can specify length of string returned.
- `substring(index1[, index2])` returns the set of characters which start at `index1` and continues up to `index2 - 1`.

Some Array methods and properties

- `length` returns the number of elements. So, `stuff.length` equals 100. The value could be reset with `stuff.length = 200`.
- `pop` (use with `var avalue = stuff.pop`) removes the last element of the array.
- `push(element)` adds an element to the end of the array.
- `shift/unshift(element)` remove and add elements to beginning of array.
- `reverse` reverses the order of the array elements.
- `sort` sorts the array.

Mathematical functions

Methods of `Math` object:

- `abs(value)` returns the absolute value (modulus).
- `sin(value)`/`cos(value)`/`tan(value)` trigonometric functions.
- `parseInt(value)`/`parseFloat(value)` converts a string to an integer/floating point number.
- `ceil(value)`/`floor(value)` familiar floor and ceiling functions.
- `max(value1, value2)` returns the larger/smaller of its two arguments.
- `random()` returns a pseudorandom number between 0 and 1 (uniformly distributed).

Date

- Create a new `Date` object: `new Date()`
- Common Methods of the `Date` object:
- `date()`: returns a `Date` object
- `getDate()`: returns the date of a `Date` object (from 1 to 31)
- `getDay()`: returns the day of a `Date` object (from 0 to 6; 0=Sunday, 1=Monday, and so on)
- `getMonth()`: returns the month of a `Date` object (from 0 to 11; 0=January, 1=February, and so on)
- `getFullYear()`: returns the year of a `Date` object (four digits)
- `getYear()`: returns the year of a `Date` object using only two digits (from 0 to 99)

Window

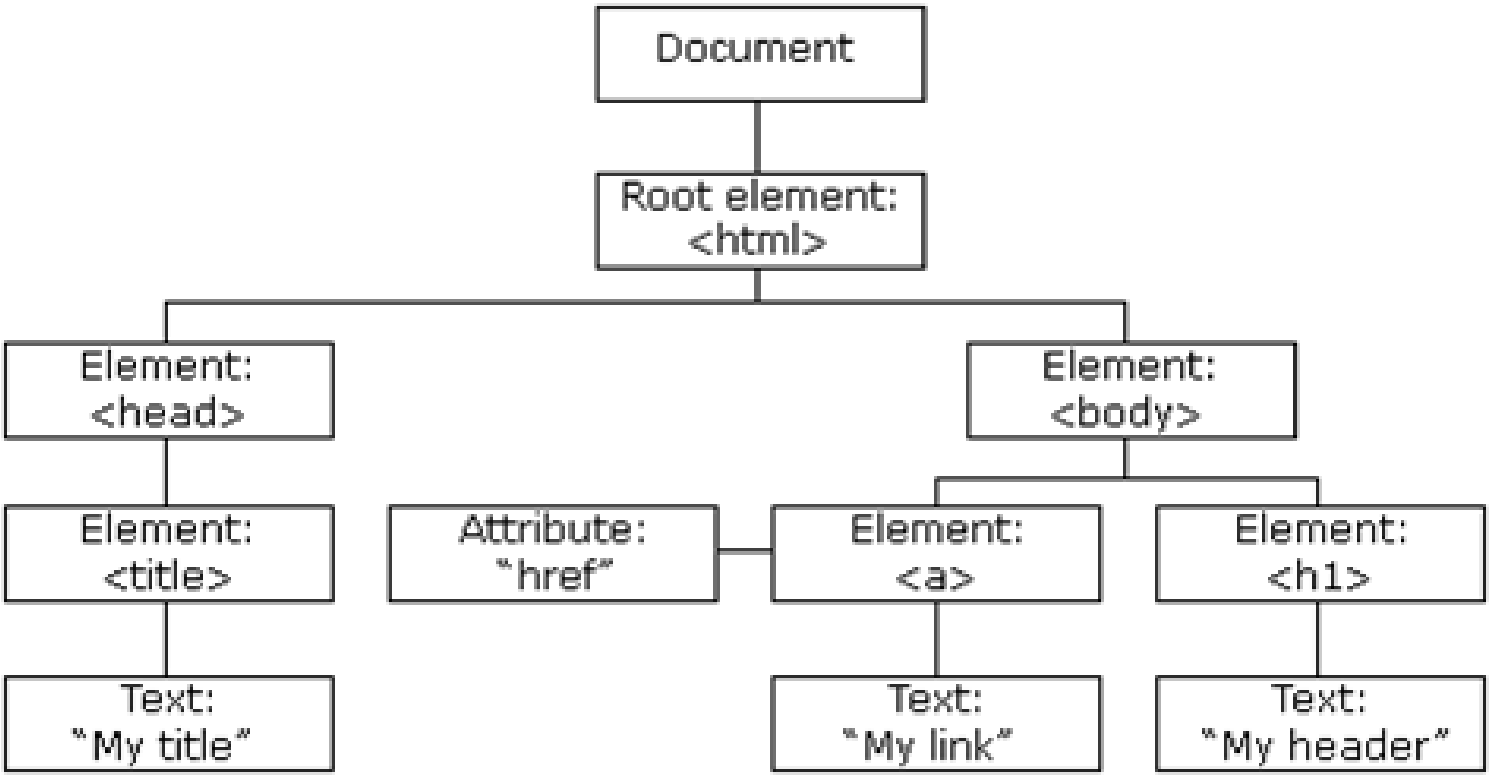
Some methods of `Window` object

- `alert()`: displays an alert box containing a message and an OK button
- `back()`: same effect as the browser?s Back button
- `blur()`: removes focus from the current window
- `close()`: closes the current window or another window if a reference to another window is supplied
- `prompt()`: Creates a dialog box for the user to enter an input

HTML, JavaScript & DOM

- The HTML containers (e.g. `<head>...</head>`) and their contents can be viewed as a **tree** with the HTML container as the **root** of the tree and its contents forming branches extending downward from the root.
- The second **layer** of the HTML DOM tree has two branches — one for the `<head>` container and the other for the `<body>` container.
- The lower layers of the HTML DOM tree branch further for each of the containers and text contained within them.
- The lowest layer of the tree ("**leaves**") corresponds to the text content of the document and any empty containers.
- Each item in the tree is referred to as a **node**.

HTML DOM



Source: `https://www.w3schools.com/js/js_htmldom.asp`

DOM objects

- The are several DOM methods to access and update the content of the DOM.
- The DOM nodes have relations of **parent**, **child** and **sibling**.
- If a `<div>` element contains a paragraph `<p>` the paragraph is a **child** of the `div` and the `div` is the **parent** of the paragraph.
- If the `div` contains two or more paragraphs (or any other elements) those paragraphs (or other elements) are **siblings** because they have the same **parent**.
- The text inside an element is a text node and it is a child of the parent element.

JS & Dymanic HTML

JavaScript can:

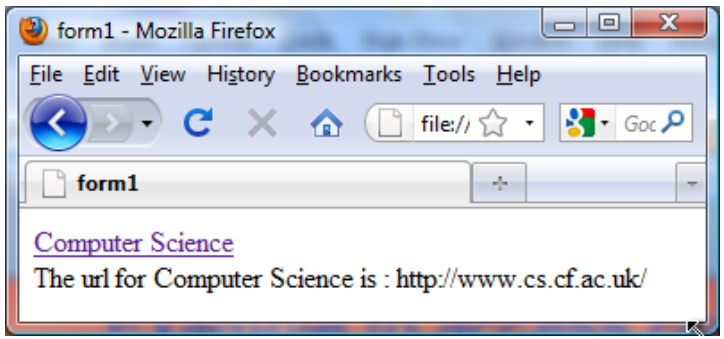
- change all the **HTML elements** in the page
- change all the **HTML attributes** in the page
- change all the **CSS styles** in the page
- remove existing **HTML elements** and **attributes**
- add new **HTML elements** and **attributes**
- react to all existing **HTML events** in the page
- create new **HTML events** in the page

Accessing document data

Objects in the DOM correspond to elements and properties/attributes in the HTML.

```
<a id="mylink" href="http://www.cs.cf.ac.uk">Computer Science
</a><br />
<script>
  url = document.getElementById('mylink').href;
  content = document.getElementById('mylink').innerHTML;
  document.write("The url for " +content+ " is : " + url);
 </script>
<body> </html>
```

`getElementById` references the `id` property. `innerHTML` accesses the content.



Accessing document data

In another example:

```
<form name="myform" action="..." >
<input type="text" id="address" />
. . .
</form>
```

The property associated with the input element with id="address" (or name="address") can be alternatively accessed as:

```
document.forms[0].elements[0].value;
document.forms["myform"].elements["address"].value;
```

`elements[0]` here assumes the address input element is the first in the form. Using explicit names is more reliable than numbered elements.

Manipulating DOM objects

- The values of style attributes can be accessed as children of the style sub-object.

```
anElement = document.getElementById("theId");
anElement.style.left = 20;
```

- This refers to the value of the **left** position attribute for the element named **anElement**.
- The document content of an element can be accessed using **innerHTML**.

```
anElement.innerHTML = "Hello there";
```

- This changes the text inside the contained whose **id** is **"theId"**.
- Can also be changed by other DOM methods.

Manipulating DOM objects

```
<html>
<head><title>Movement</title>
<style>
  #para1 {position: absolute; font-size = 16pt;
  font-family: arial, sans-serif;}
</style>
<script>
     function move(x, y) {
      var thePara = document.getElementById("para1");
      thePara.style.top = y;
      thePara.style.left = x;
      thePara.style.fontSize = "24pt";
      thePara.style.color = "red";
      thePara.innerHTML = "I told you not to click!";
      }
</script>
</head>
<body>
<p id="para1" onclick="move(200,100)">
     Do not click here!
</p> </body>
</html>
```

Manipulating DOM objects

- Elements can be created and then attached to the DOM:

```
newNode = document.createElement("p");
```

- Text content to put in the new element may be a literal string or come from another variable, or generated dynamically:

```
sometext = "Give us full power!";
othertext = anode.value;
```

- To create a node to contain the text:

```
textNode =
  document.createTextNode(sometext);
```

Manipulating DOM objects

- To make the **textNode** a child of the **newNode** (paragraph):

```
newNode.appendChild(textNode);
```

- To insert the paragraph with its text in an existing ** ** node in the page, first access the node:

```
theDivNode =
      document.getElementByID("myDiv");
```

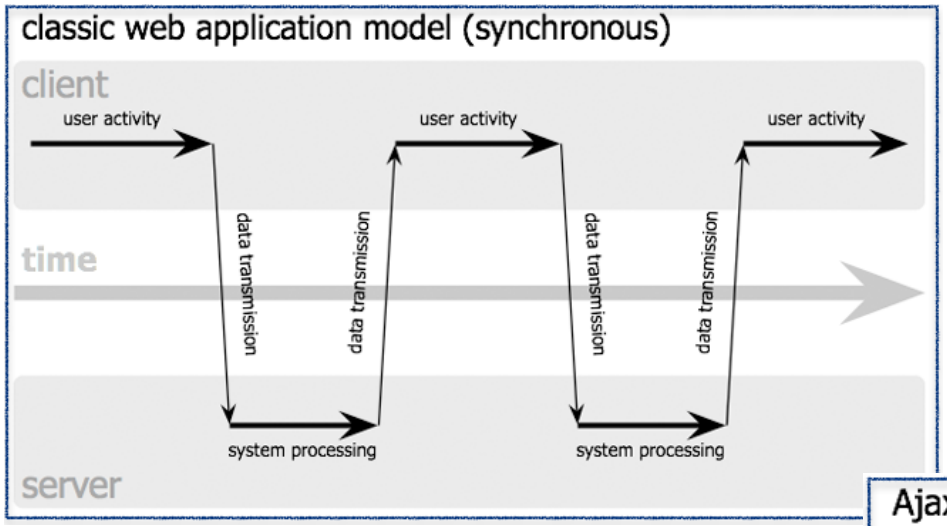
- Then append the **newNode** to it:

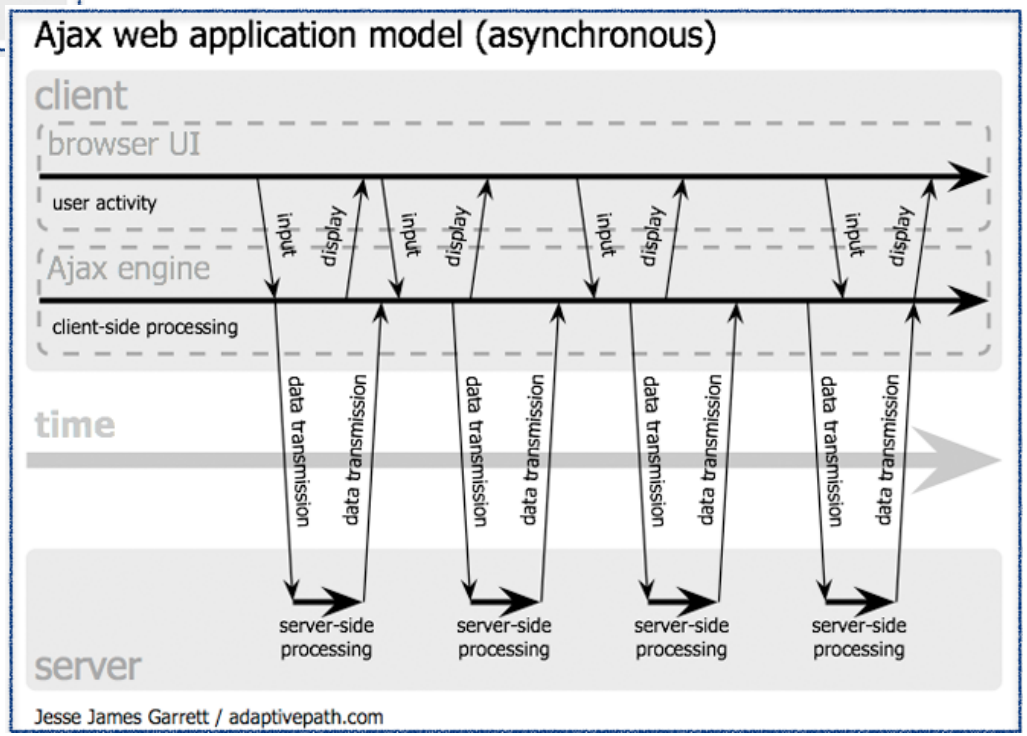
```
theDivNode.appendChild(newNode);
```

AJAX

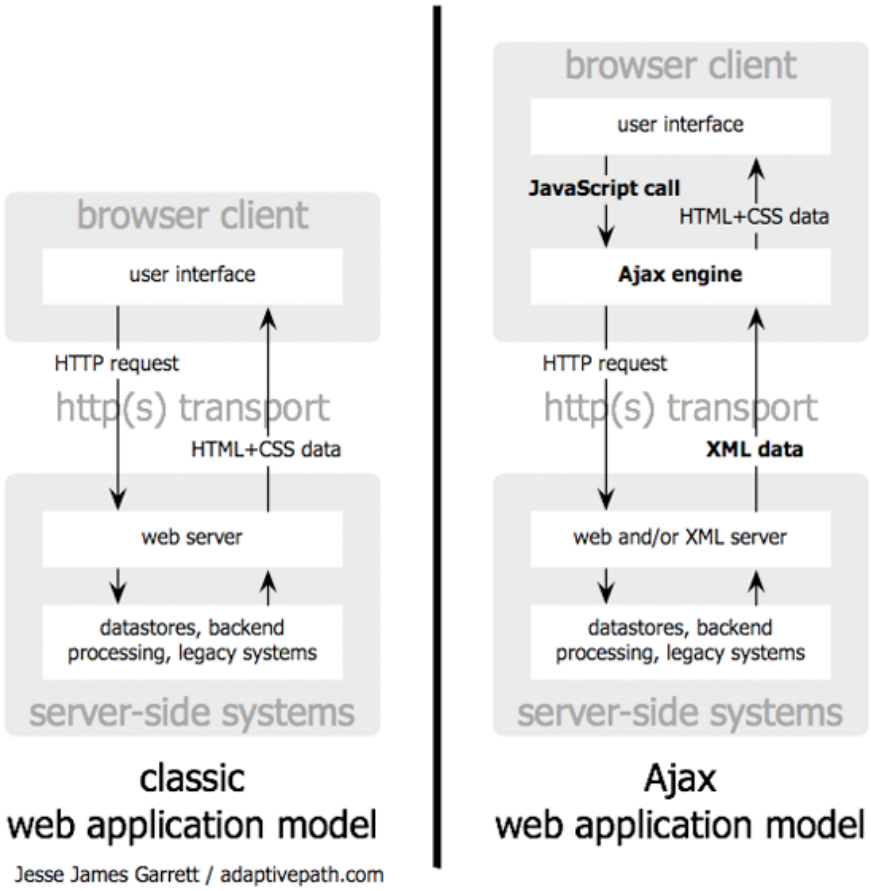
- = Asynchronous JavaScript And XML
- AJAX apps resemble deskop apps
- not a single technology, but a combination of several web technologies:
  - HTML
  - CSS
  - Javascript and DOM
  - XMLHttpRequest object (to support asynchronous communication with server)
  - XML (to return data from server); or JSON; or some object in the form of HTML code

Synchronous vs asynchronous model





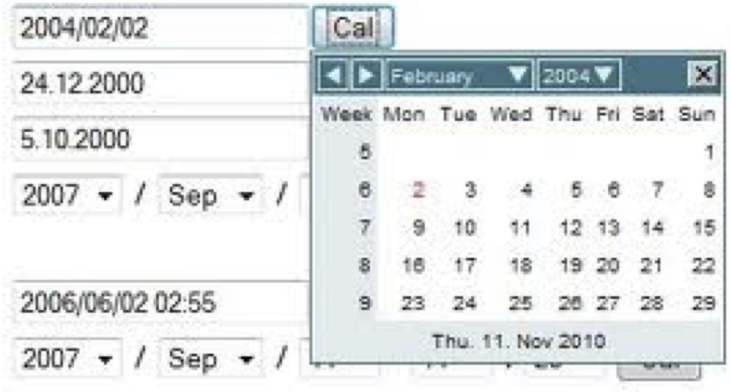
Classic vs AJAX model



AJAX examples

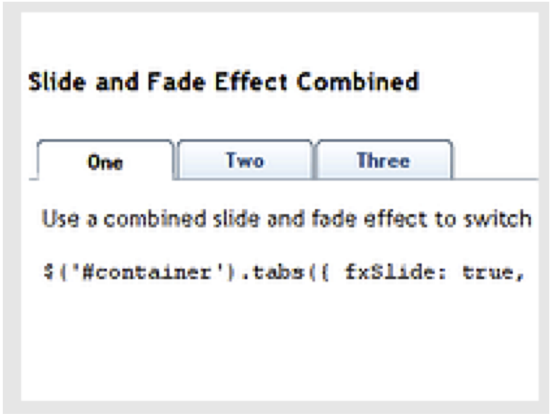
Forms

Calendar



Image/ element Carousel

Tabs/ Sliders



Reading and resources:

- R. Larsen (2013). *Beginning HTML and CSS*, Wiley
  - Ch.10
- JavaScript Tutorial on w3schools.com:
  https://www.w3schools.com/js/default.asp
- JavaScript HTML DOM on w3schools.com:
  https://www.w3schools.com/js/js_htmldom.asp

Extending Your Knowldge

JavaScript-based technologies and frameworks, e.g. JQuery, JSON, Bootstrap, Semantic UI, etc.

- For example, see:
  - R. Larsen (2013). *Beginning HTML and CSS*, Wiley
    - Chs.11, 12
  - Tutorials on https://www.w3schools.com
  - also, 'Top 22 Best Free HTML5 Frameworks for Responsive Web Development 2017'