HTML5 & Javascript

Odds and Ends

Where are we now? What have we missed?

* In the past 9 lectures, we've covered a lot of Javascript and HTML and a little jQuery and CSS along the way
  * There can't be much left, surely (don't call me Shirley!)
  * In fact, we've scratched the surface
    * Never looked at regular expressions
    * Never touched structured error handling
    * No examples of data validation
    * No sorting (ordering)
    * Didn't do much analysis of events, cookies, screen, history…
* 12 weeks is not really enough time (never mind 10)
  * You'll need to keep going on your own and in other modules
    * e.g. Introduction to Programming for Mobile Devices module

Some PRACTICAL odds and ends

* Before we review the module (next week), there are a few items that ought to be discussed
  * All things that you might do in your project
    * i.e. this is what most of you said in the stage 1 submission
* In no particular order
  * Searching a data set
  * Error handling
  * Drag & Drop (particularly images)
  * Validating input data
  * Sorting a data set into some order

Searching a data-set

* We'll consider ***appointments*** to be a typical data-set
  * Not too far off the truth
* First cut…

```
for (var index=0; index<appointments.length; index+=1) {
        if(appointments[i].<something> = searched-value) {
                do-something!
        }
}
```

* <something> is a problem
  * What are we really searching?
* searched-value is unspecified – is that OK?
  * What are we looking for – a string of text?, a date?...
* do-something WHAT SHOULD WE DO!!!!!

What are objects for?

* We already expect an appointment to know its date and time, generate table rows, etc, etc.
* How about expecting it to decide whether it matches a search term…

```
Appointment.prototype.contains = function(search-term) {
    if( this.toString().indexOf(search-term) > -1){
        return true;
    } else {
        return false;
    }
}
```

By using toString(), we can match anything in the appointment

* Now we can do:

```
var matches = [];
for (var index=0; index<appointments.length; index+=1) {
    if(appointments[i].contains( search-term )) {
        matches.push(appointments[i]);
    }
}
```

Error handling

* Javascript is quite resilient to errors, so it can be difficult to crash! a page
  * However, this is a bad thing, because the user can continue working on with garbage data
* It is useful to detect errors
  * Good for de-bugging
  * Good for keeping the user informed
  * Good for avoiding garbage-in, garbage-out problems
    * See later

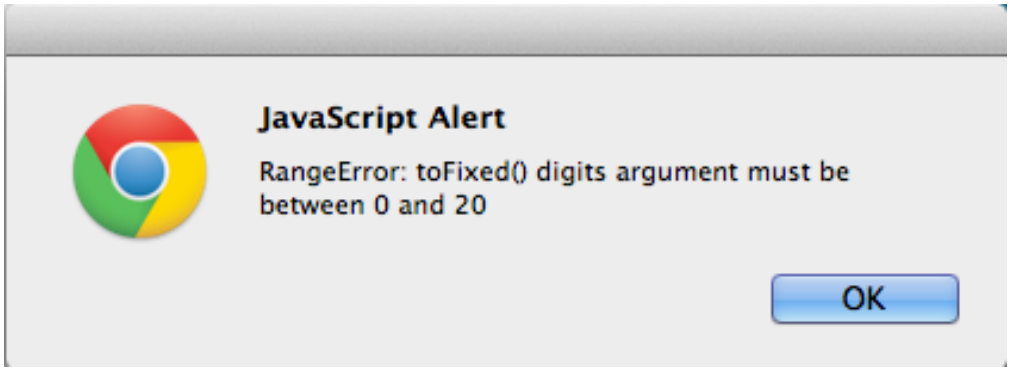
Try this

* Basic principle
  * Identify error-prone code (usually after user-input, collecting data from outside, trying to access devices etc.)
  * Use try..catch to detect problems, e.g.

```
try {
   // Do something dodgy…
} catch (error) {
  // Handle errors here.
}
```

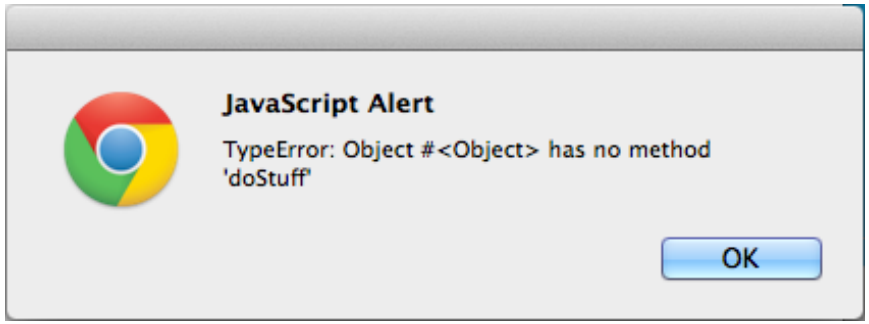
Error Handling Example

```
function doBadStuff() {
    var input;
    try {
        var pi = 3.14159268;
        pi.toFixed(100);
    } catch(err) {
        alert(err);
    }
}
```



Another

```
function doBadStuff() {
    var input;
    try {
        var obj = {};
        obj.doStuff();
    } catch(err) {
        alert(err);
    }
}
```



Handling real errors

* The problem is that in many situations, Javascript will just carry on and forgive anyway – the previous examples are fairly unlikely
* E.g Get the user to enter their date of birth in a text box, then make this into a date…

```
var dateinput = document.getElementById("dob").value;
var dob = new Date(dateinput);  // User can enter any old rubbish.
```

* To get around this, you need to detect the error and deal with it in code

Coding around an error

```
function errorAction(id) {
   var element = document.getElementById(id);
   element.style.backgroundColor = “red”;
   element.focus();  //  puts the cursor into the error element.
}

function processInput() {
   var dateinput = document.getElementById(“dob”).value;
   var dob = new Date(dateinput);
   if( isNaN( dob.valueOf() ) {
     // This isn’t a valid date – get the user to re-enter.
     errorAction(“dob”);
   } else {
     // do normal stuff…
   }
}
```

HTML5 Drag & Drop

* Although awkward to set up, drag & drop is by far the easiest way to make an app interact with the system that hosts it
  * Drag a file into a browser page, the page can access the file
    * This is pretty safe – drag & drop is a deliberate act, so difficult to spoof

```
<!DOCTYPE html>
<head>
   <script type="text/javascript" src="drag&drop.js">
   </script>
</head>
   <img id="target" width="100px" height="100px"/>
</body>
</html>
```

Uses a file called drag&drop.js, which has a single function – setupDragDrop( )
Call this in window.onload( ) and pass in the ID of the target <img> element and a function to call.
Get a demo from Moodle (Samples & Demos)

```
// A typical callback function…
function justDroppedAnImage (imgData) {
   currentObject.imageSource = imgData;
}
```

Validating Input Data

* HTML5 provides well for inputting a range (<input type="range"> or an integer (<input type="number">)
* It doesn't do so well for numbers with a decimal point, a time of day, postcodes etc.
  * For these you need to validate in code.
  * The general principle is garbage-in-garbage-out
    * i.e. Let the user enter invalid data – suffer for it.
    * A couple of optional attributes for <input> help:
      * <input required… /> This needs to be entered
      * <input pattern="[A-Z]{1,2}[0-9R][0-9A-Z]? [0-9][ABD-HJLNP-UW-Z]{2}"/> (a UK Postcode)
      * <input placeholder="Enter your first name"… />
    * For validation, these need to be used inside a <form> tag.
      * See Moodle – input validation, for an example.
      * See http://quickstart.developerfusion.co.uk/quickstart/howto/doc/regexcommon.aspx and http://englishblog.flepstudio.org/tutorials/flash-cs3/sintax/useful-list-of-regular-expressions-with-actionscript-3-regexp-class/ for other useful patterns.

Sorting an array

* Javascript arrays already come with a built in sort() method:

  var arr = ["delta", "kilo", "alpha", "gamma", "beta", "epsilon"] ;

  arr.sort() -> ["alpha", "beta", "delta", "epsilon", "gamma", "kilo"]

* This works for known types (strings, dates, numbers) but how would you sort an array of, say, appointments?
* For this you will need to provide a sorting function:

```
function sortFunction(a, b) {
    // Compare a & b.  Return -1 if a<b, 0 if same, 1 if a>b
}
```

* Now call `array.sort(sortFunction);`
* So, how to compare two appointments for sorting by date & time…

```
function orderAppointments(appt1, appt2) {
    if(appt1.datetime < appt2.datetime) return -1;
    if(appt1.datetime === appt2.datetime) return 0;
    return 1;
}
```

* Now we can do…

```
appointments.sort(orderAppointments );
```

* Note – don't try to sort the on-screen data in a table. Sort the underlying array and then re-display it!

A Neat Trick

* See http://www.javascriptkit.com/javatutors/arraysort.shtml for this
* You can use the array.sort() method to randomize the items in an array (sometimes quite useful, especially in games). Just have this as the sort operation…

```
myarray.sort(function() {return 0.5 - Math.random()});
```

All done!

* Next week, there will be
  * a review of the module and
  * a sample test paper
    * (remember the real paper will be on the following week)
* Any Questions for Now?