**JAVASCRIPT EXTRAS**

**APIS**

User interfaces allow humans to interact with programs. Application Programming Interfaces(APIs), let programs, including scripts, talk to each other.

Browser – the DOM is an API

Scripts – Jquery is a javascript file with an API

Platforms – sites such as Facebook, Google and Twitter open their platforms so that you can access and update data they store (via websites and apps).

**HTML5 APIs**

Geolocation – how to tell where the user is located

LocalStorage/sessionStorage – store information in the browser

History – how to access items from the browsers history

*and many more!!*

When you write code that uses the HTML5 APIs, you may need to check if the browser supports that feature before your code tries to use it:

If(navigator.geolocation){

//code block

}else{

//code block

}

There is a library called ‘modernizr’ which deals with the cross browser issues (DL from modernizr.com)

**Geolocation API**

An increasing number of sites offer extra functionality to users who disclose their location. The users location can be requested using the geolocation API.

The geolocation API is available by default is any browser that supports it (just like the DOM is).

|  |  |
| --- | --- |
| Method | Returns |
| getCurrentPosition(success, fail) | Requests users position.  Success is a function to call if co-ordinates are retrieved  Fail is a function to call if co-ordinates are not retrieved |

E.g.

var elMap = document.getElementById(‘locaion’);

var msg = ‘sorry, we could not locate you’;

if(modernizr.geolocation){

navigator.geolocation.getCurrentPosition(success, fail);

elMap.textContent=’searching…’;

}else{

elMap.textContent=msg;

}

Function success(position){

Var location = position.coords.latitude+’,’+position.coords.longitude;

elMap.innerHTML=location;

}

Function fail(msg){

elMap.textContent=msg;

console.log(msg.code);

}

**Web storage API**

Web storage lets you store data in the browser.

2 different types: localStorage & sessionStorage

Both of these objects are implemented on the window object.

To save an item in the storage object, you use the setItem() method, which takes 2 parameters: name of the key and the value associated with it. E.g. localStorage.setItem(‘age’,12);

To retrieve a value from the storage object you use the getItem() method, passing it the key. E.g. var age = localStorage.getItem(‘age’);

E.g. if(window.localStorage){

Var txtUsername = document.getElementById(‘username’);

TxtUsername.value = localStorage.getItem(‘username’);

txtUsername.addEventListener(‘input’,function(){

localStorage.setItem(‘username’,txtUsername.value);

},false);

}

|  |  |
| --- | --- |
| Type | use when: |
| Session storage | Information that changes frequently  Information that is personal |
| localStorage | Information that only changes at set intervals  Information that the user might want to come back and use again(such as saving preferences/settings) |

**Jquery UI**

Jquery UI is a plug in (extra code added in to the jquery file). The jquery foundation maintain its own set of jquery plugins called jquery UI. They help create user interfaces such as widgets, effects and interactions. (download via jqueryui.com)

Jquery UI accordion

E.g. HTML

<div id=”prizes”>

<h3>1st prize</h3>

<div><p>first prize wins….</p></div>

<h3>2nd prize</h3>

<div><p>second prize wins….</p></div>

<h3>3rd prize</h3>

<div><p>third prize wins….</p></div>

</div>

JS

$(function(){

$(‘#prizes’).accordion();

}

Jquery UI tabs

E.g. HTML

<div id=”prizes”>

<ul>

<li><a href=”#tab-1”>1st prize</a></li>

<li><a href=”#tab-2”>2nd prize</a></li>

</ul>

<div id=”tab-1”><p>text here</p></div>

<div id=”tab-2”><p>text here</p></div>

</div>

JS

$(function(){

$(‘#prizes’).tabs()

}

Jquery UI form

Jquery UI introduces several form controls to make it easier for people to enter data in to forms. Below are 2 examples (slider and datepicker).

E.g HTML

<h2>find accommodation</h2>

<p id=’price’>

<label for=”amount”>price range:</label>

<input type=’text’ id=’amount’/>

</p>

<div id=”priceRange”></div>

<p>

<label for=’arrival’>Arrival:</label>

<input type=’text’ id=’arrival’ />

</p>

<input type=’submit’ value=’find a hotel’ />

JS

$(function(){

$(‘#arrival’).datepicker();

Var $amount = $(‘#amount’);

Var $range = $(‘#priceRange’);

$range.slider({

Range:true,

Min:0,

Max:400,

Values:[150,300],

Slide:function(e,ui){

$amount.val(‘$’+ui.values[0]+’ -$’+ui.values[1]);

}

});

$amount.val(‘$’+$range.slider(‘values’,0)+’ -$’+$range.slider(‘values,1);

**Angular JS**

Angular JS is a framework that makes it easier to create web apps. In particular it assists in creating apps that write, read, update and delete data in a database on a server.

To use angular, first you include the angular.js script in your page and then it makes a set of tools available to you (just like jquery does). You can download angular and view the full API at angularjs.org.

E.g. HTML

<html **ng-app**>

<head>

<script src=”js/angular.js”></script>

</head>

<body>

<form>

<input **ng-model=”name”** type=”text” />

</form>

<div class=”postcard”>

<div>**{{name}}**</div>

</div>

\*\*no JS required in this example\*\*

**Look more in to angular at a later date.**

**Platform APIs**

Many large websites expose their API that allow you to access and update the data on their sites including facebook, twitter and google.

Each site offers different capability. E.g. facebook offer features such as allowing people to like sites or add comments and discussion at the bottom of a web page/ Google maps let you include various types of maps.

The syntax of an API will vary from platform to platform

Google maps API

Google maps allow you to show google maps in your web pages and customise the look of the maps.

API key = some APIs require that you register and request an API key in order to get data from their servers.

Basic map settings

Once you have included the google maps script in your [age, you can use their maps object. It lets you display google maps in your pages.

Creating a map:

The Map() object allows you to add a map. It requires 2 parameters:

1. The element in which you want the map drawn
2. A set of mapOptions that control how it is displayed given using object literal notation

Map options:

mapOptions – is created as an object literal before you call the Map(). 3 pieces of data are required:

1. Longitude and latitude
2. Zoom level
3. Type of map

E.g. function init() {

Var mapOptions = {

Center: new google.maps.LatLng(40.782710, -73.965310),

mapTypeId: google.maps.MapTypeId.ROADMAP,

zoom:13

};

Var venueMaps;

venueMaps = new

google.maps.Map(document.getElementById(‘map’),

mapOptions);

};

Function loadScript(){

Var script = document.createElement(‘script’);

Script.src=’http://maps.googleapis.com/maps/api/js?sensor=false&callback=init’;

Document.body.appendChild(‘script’);

}

Window.onload = loadScript;

\*\*NOTE – instead of the function load script, you can include <script sync defer src=”http://maps.googleapis.com/maps/api/js?sensor=false&callback=init”></script> in your HTML\*\*

\*\*note the init underlined runs the function. So this and the function name should be the same\*\*

Google map with a pin

Function initMap(){

Var uluru = {lat:25.363, lng:131.044};

Var map = new google.maps.Map(document.getElementById(‘map’), {

Zoom=4,

Center= uluru

});

Var marker = new google.maps.Marker({

Position:uluru,

Map: map

};

HTML = <script async defer src=”https://maps.googleapis.com/maps/api/js?key=API KEY&callback=initMap”><script/>

\*\*Ensure CSS styling is completed on map div (‘map’) – including height and width, otherwise you will not be able to see the map\*\*

*Look at pages 444-447 to see how to style google maps*

**Error handling & debugging**

The order in which statements are executed are can be complex. Some tasks cannot complete until another statement or function has been run.

The javascript interpreter processes one line of code at a time. When a statement needs data from another function, it stacks the new function on top of the current task.

When a statement has to call some other code in order to do its job, the new task goes to the top of the pile of things to do.

Understanding errors

If a javascript statement generates an error, then it throws an exception. At that point, the interpreter stops and looks for exception-handling code.

If you are anticipating that something in your script may cause an error, you can use a set of statements to handle the error

Error objects

Error objects can help you find where your mistakes are and browsers have tools to help you read them.

When there is an error, you can see all of the information in the JS console/error console of the browser:

* Name (type of error)
* Message (description of error)
* fileNumber (name of the JS file)
* lineNumber (line number of error)

There are 7 types of built-in error objects in javascript:

|  |  |
| --- | --- |
| Error | General error |
| syntaxError | Syntax hasn’t been followed |
| referenceError | Tried to reference a variable that is not declared |
| typeError | An unexpected data type that can’t be coerced |
| rangeError | Number not in acceptable range |
| URIError | encodeURI(), decodeURI() (and others) – used incorrectly |
| EvalError | Eval() function used incorrectly |

*E.g. syntaxError: unexpected token illegal errors.js:4*

The above shows the type of error, the name of the file (errors.js) and the line number(4).

\*\*look at p460-461\*\*

How to deal with errors

1. debug the script
2. handle errors gracefully (using try, catch, throw and finally statements)

Debugging

Where is the problem?

* Look at the error message, it tells you:
  + - Relevant script that caused problem
    - The line number (although error could be earlier, but this is the point in which the script stopped)
    - Type of error (although could be different)
* Check how far the script is running
* Use breakpoints where things are going wrong

What exactly is the problem?

* When you have set break points, you can see if variables around them have the values you would expect them to. If not, look earlier in the script
* Break down/break out parts of the code to test smaller pieces of the functionality
  + - Write values of variables in the console
    - Call functions from the console to check if they are returning what you would expect them to
    - Check if objects exist and have the methods/properties that you think they do
* Check the number of parameters for a function, or the number of items in an array

Browser developer tools and JS console

Press f12 or options>tools>developer tools/JS console

\*\*Revisit the rest of the chapter once starting to complete more complex code where these methods would be necessary\*\*

*---------------------------P468-p485 not yet covered---------------------------*

**Form enhancement and validation**

The form element

|  |  |
| --- | --- |
| **Property** | **Description** |
| Action | The URL the form is submitted to |
| Method | If it is to be sent via GET or POST |
| Name | Rarely used, more common to select a form by the value of its ID attribute |
| Elements | A collection of the elements in the form that users can interact with. They can be accessed via index numbers or the values of their name attributes |
| **Method** |  |
| Submit() | This has the same effect as clicking the submit button on a form |
| Reset() | Resets the form to the initial values it had when the page loaded |
| **Event** |  |
| Submit | Fires when the form is submitted |
| Reset | Fires when the form is reset |

DOM methods such as; getElementById() can be used for forms. However, the DOM also has something called ‘forms collection’ – which holds a reference to each of the <form> elements that appear on a page. These are accessed via index numbers or ‘name’ attributes.

E.g. document.forms.login (access a form with the *name* attribute ‘login’)

Document.forms[1].elements[0] (access 2nd form and 1st form control)

Document.forms[1].elements.password (access 2nd form and elements whose *name* attribute is ‘password’

Form controls

|  |  |
| --- | --- |
| **Property** | **Description** |
| Value | In a text input, it is the text the user entered; otherwise, it is the value of the *value* attribute |
| Type | When a form control has been created using the <input> element, this defines the type of the form element (e.g. text, password, checkbox…) |
| Name | Gets or sets the value of the *name* attribute |
| defaultValue | The initial value of a text box or text area when the page is rendered |
| Form | The form that the control belongs to |
| Disabled | Disables the <form> element |
| Checked | Indicates which checkbox or radio buttons have been checked. This property is a Boolean |
| defaultChecked | Whether the checkbox or radio button was checked or not when the page loaded |
| Selected | Indicates that an item from a select box has been selected. This property is a Boolean |
| **Method** |  |
| Focus() | Gives an element focus |
| Blur() | Removes focus from an element |
| Select() | Selects and highlights text content of an element (e.g. text inputs, text areas, and passwords) |
| Click() | Triggers *click* event upon buttons, checkboxes, and file upload |
| **Event** |  |
| Blur | When the user leaves field |
| Focus | When the user enters a field |
| Click | When the user clicks on an element |
| Change | When the value of an element changes |
| Input | When the value of an <input> or <textarea> element changes |
| Keydown, keyup, keypress | When the user interacts with a keyboard |

The submit Event & Getting form vaues

(function() {

Var form = document.getElementById(“login”) //form id

Form.addEventListener(‘submit’,function(e) {

e.preventDefault();

var elements = this.elements;

var username = elements.username.value

var msg = “Welcome “+username;

document.getElementById(“message”).textContent=msg;

},false);

}());

How to show password

(function(){

Var pwd = document.getElementById(“pwd”); //password input ID

Var show = document.getElementById(“showPwd”); //checkbox ID

Show.addEventListener(“change”,function(e) {

Var target = e.target;

If(target.checked) {

Pwd.type = “text”;

}else{

Pwd.type=”password”;

}

},false);

}());

Disabling submit button

* Disables the button when:
  + The script first loads. The change event then checks when the password changes and enables submit if the password is given a value.
  + The form has been submitted (to prevent the form been sent multiple times).

(function() {

Var form = document.getElementById(“login”);

Var pwd = document.getElementById(“pwd”);

Var submit = document.getElementById(“submit”);

Var submitted = false; //known as a flag variable

Submit.disabled = true; //disables submit on page load

//on input

Pwd.addEventListener(“input”, function(e) {

Var target = e.target;

Submit.disabled = submitted || !target.value;

//above disables if previously submitted or no value in pwd input

},false);

//on submit

Form.addEventListener(“submit”,function(e) {

If(submitted||submit.disabled) {

e.preventDefault();

return;

}//if form submitted or submit btn disabled then prevent submit btn firing and continue code

Submit.disabled = true;

Submitted = true;

//demo purposes

e.preventDefault();

alert(“password is “ + pwd.value);

},false);

}());

Checkboxes

(function() {

Var form = document.getElementById(“login”);

Var elements = form.elements;

Var options = element.genre; //name of checkboxes

Var all = document.getElementById(“all”); //id of *‘all’* checkbox

Function updateAll() {

For(var i = 0; i<options.length; i++) {

Options[i].checked = all.checked;

}

};

All.addEventListener(“change”, updateAll, false);

Function clearAllOptions(e) {

Var target = e.target;

If(!target.checked) {

All.checked = false;

};

};

For(var i = 0; i<options.length; i++) {

Options[i].addEventListener(“change”, clearAllOptions, false);

};

}());

***Shorcut if else statement***

Example:

Var name = “ali”;

(name == “ali” ? alert (“hi ali”) : alert(“I don’t know ”+name));

Radio button

Show textbox if ‘other’ selected

(function() {

Var form, options, other, otherText, hide;

Form = document.getElementById(“login”);

Options = form.elements.heard; //radio btn name

Other = document.getElementById(“other”); //*other* radio btn

OtherText = document.getElementById(“other-text”); //input box

otherText.className = “hide”; //css class with *display: none;*

for(var i = 0; i<options.length; i++) {

options[i].addEventListener(“click”, change, false);

}

Function change() {

Hide = other.checked? ‘’ : ‘hide’;

otherText.className = hide;

if(hide){

otherText.value = ‘’;

};

};

}());

Select boxes

The <select> element is a bit more complex. It’s DOM node has a number of extra properties and methods:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Options | A collection of all the <option> |
| selectedIndex | Index numbers of the option that is currently option |
| Length | Number of options |
| selectedOptions | A collection of all the selected <option> elements |
| **Method** |  |
| Add(option, before) | Adds an item to the list: 1st parameter is the new option, 2nd parameter is the element it should go before (goes at end if no value given). |
| Remove(index) | Removes item from list |

(function() {

//cach the 2 select boxes

Var type = document.getElementById(“equipmentType”);

Var model = document.getElementById(“model”);

//create objects for the model select box

Var cameras = {

Canon: “750d canon”,

Nikon: “Nikon D3400”

}

Var cars = {

BMW: “BMW 3 series”,

Merc: “Mercedes C Class”,

Audi: “Audi A5”

}

//when user changes the ‘type’ select box

Type.addEventListener(“change”, function() {

If(this.value === “change”) { //change is first option

Model.innerHTML = ‘<option>Choose type first</option>’;

Return;

}

Var models = getModels(this.value);

//loop through options

Var options = ‘<option> Please choose a model</option>;

For(var *key* in *models*) {

Options += ‘<option value = “’ + key + ‘”>’ +models[key] + ‘</option>’;

};

Model.innerHTML = options;

Function getModels(equipmentType) {

If(equipmentType ===”cameras”) {

Return cameras;

}else if(equipmentType===”cars”) {

Return cars;

};

};

}, false);

}());

\*\* the *in* operator returns true if the specified property is in the specified object, otherwise false\*\*

HTML5 form attributes & elements

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| Autofocus | Focus element when page loaded |
| Placeholder | Light grey text as a temporary value to input element |
| Required | Checks that the field has a value – could be text entered or an option selected |
| Min | Minimum permitted number |
| Max | Maximum permitted number |
| Step | Intervals by which numbers should increase or decrease |
| Value | Default value |
| Pattern | Lets you specify a regular expression to validate a value |
| Novalidate | Used on <form> element to disable HTML5 built in form validation |
| Autocomplete | On by default: shows list of past entries (disable for credit card no./sensitive data) |

<input type=”search”>

<input type=”telephone”> //or URL, email

<input type =”number”>

<input type=”range” min=”0” max=”12” steps=”2” value=”6”>

<input type =”color”> //color picker

<input type=”date”> //month, week, time, datetime

**Summary**

APIS

1. What does API stand for and what is it?
2. Use the geolocation API to show image of whereabouts on screen
3. Use localStorage API in an example
4. What are the differences between localStorage and sessionStorage
5. Using the jquery UI API plugin, use the:
   * 1. Accordion widget
     2. Tabs widget
6. Give a jquery ui form example using datepicker and slider methods
7. Use the google maps API to show a location on screen with a marker

Error Handling & Debugging

1. What are the 7 types of built in error objects?
2. How do you access developer tools

*Chapter not yet finished*

Form Enhancement & Validation

1. List 4 properties associated with forms
2. List 2 methods associated with forms
3. List 2 events associated with forms
4. How do you access a form and its elements via the DOM
5. Form controls – list:
   1. 9 properties
   2. 4 methods
   3. 6 events
6. Give practical examples on:
   1. How to hide and show password (change type)
   2. Disable the submit btn if previously submitted, or form not filled
   3. Check all checkboxes when ‘select all’ checkbox ticked
   4. Show text input when ‘other’ radio btn selected
   5. Create 2 select boxes – first showing the category cars & cameras. The second showing the models related to the category picked.
7. List 10 HTML5 form attributes and 6 HTML5 input type elements

Extra

1. How do you type a shorthand if/else statement?