# 1. Overview of HTML and CSS

Hyper Text Markup Language – static language that describes structure and semantic meaning of web content. Every HTML page has the same basic structure:

* Doctype declaration
* Html section
  + Header
  + Body

## Gathering user input by using forms in HTML

<form> element

Action attribute defines how data is sent to the use:

* GET
* POST
* Accept-charset attribute – identifies character encoding
* Enctype attribute – MIME-type when encoding form data when post
* Target attribute – where action page displayed

### Form Controls

Text

Password

Hidden

Checkbox

Radio

Reset

Submit

Image – image for use as submit button

Button

File

<textarea>

<select> - drop-down list

<button>

### Form Layout elements

<fieldset> <legend>

<label>

# Overview of CSS

Selector {

property: value;

}

e.g.

section > h2

returns any h2 elements nested immediately under a section element

section + h2

returns any h2 elements immediately following and sharing the same parent element as a section element

section ~ h2

returns any h2 elements following and sharing the same parent element as a section element

### Cascading rules

**Importance** – you can ensure a certain property is always applied by appending the rule with !important

**Specifity** – styles with lease specific selector are applied first etc until most specific applied

**Source order** – applied in order in stylesheet

# 2. Creating and Styling HTML pages

## Styling an HTML5 page

### Understanding CSS Text Styles

* font-family
* font-size
* font-style
* font-weight
* font: (shorthand) – font-style, font-weight, font-size, font-family
* color
* opacity
* letter-spacing
* line-height
* text-align
* text-decoration
* text-transform

### The CSS Box Model

Margin

Border

Padding

Content

Order of top right bottom left: **TR**ou**BL**e

Page flow:

* visibility
* display
* position
* float
* overflow
* box-sizing

### Styling background in CSS

* background shortcut – only background-image is mandatory
* background-color
* background-position
* background-size
* background-repeat
* background-origin (content-box, padding-box- border-box)
* background-clip
* background-attachment
* background-image

# 3. Introduction to JavaScript

### Functions

Function arguments are option, you can still pass parameters into a function is arguments are not specified. They are available in an array: **arguments**.

## Introduction to the Document Object Model (DOM)

### Finding elements in the DOM

* **document.getElementById(IdString)** – returns single element
* **document.getElementsByName(NameString**) – returns an array of elements whose name attribute has the value specified

### Adding, Removing and Manipulating Objects in the DOM

Create new objects

* **document.createElement(tagname)**
* **document.createTextNode(string)**
* **document.createAttribute(name, value)**
* **document.createDocumentFragment**

Add it to the dom, find a parent element with document.getElementById then call one of the methods on that element:

* **appendChild(newNode)**
* **insertBefore(newNode, existingNode)**
* **replaceChild(newNode, existingNode)**
* **replaceData(offset, length, string)**

Removing:

* **removeChild(node)**
* **removeAttribute(attributeName)**
* **removeAttributeNode(node)**

### Handling Events (Important!)

* **addEventListener(eventName, listenderFunction, bubbles)**

var element = document.getElementById(“icon”);

Function ShowHelpText(){

Window.alert(‘some help text’);

};

element.addEventListener(“mouseover”, ShowHelpText, false);

## Introduction to JQuery

## Selecting elements and traversing the DOM using JQuery

Selection traversal with each:

$(document).ready(function(){

$(“h2”).each(function(){

this.style.color = “red”;

});

});

* **eq(index)** – returns the single element at the given index
* **each(function)** – iterates of a set of elements and applies the given function
* **filter(expression)** – returns a subset of elements e.g. $(“p”).filter($(“:first”));
* **find(selectorString)** – returns a subset of elements from those in the original set $(“form”).find(“input[type=text]”)
* **first() and last()**
* **next() and prev()**
* **size()**
* **slice(int, [int]**

Adding and removing elements:

* **addClass(className)**
* **append(htmlString)**
* **detach()**
* **html(htmlString)**
* **replaceWith(htmlString)**
* **val()**

### Handling Control Events by jQuery

* **bind(“event”, function)**
* **unbind()**
* **click(), dblclick()**
* **error()**
* **focus(), focusin(), focusout()**
* **keydown(), keyup(), keypress()**
* **hover(), mousedown(), mouseup(), mouseenter(), mouseleave(), mouseout(), mouseover(), mousemove()**
* **load(), unload()**
* **select()**

# 4. Creating Forms to Collect and Validate User Input

### Declaring a Form in HTML5

<form name=”userLogin” action=”post” action=”action.aspx”>

<fieldset>

<legend>Enter your logjn</legend>

<div class=”someClass”>

<input id=”username” name=”username” type=”text” placeholder=”Username”/>

<label for=”username”>Username</label>

</div>

</fieldset>

<input type=”submit” value=”send” />

</form>

### HTML5 Input Attributes

* **number – supports max, min, step and value**
* **autofocus**
* **autocomplete**
* **required**
* **pattern**
* **placeholder**

## Validating user input

**Required** – works with text, search, url, tel, email, password, number, checkbox radio and file

**Number** attribute

**Pattern** attribute – text, search, url, email, password

## Validating user input by using JavaScript

Can use the onsubmit attribute of the form to run javascript validation.

Can also attach event listeners and use the setCustomValidity function to set an error message and stop data being submitted.

# 5. Communicating with a Remote Server

Some elements provide a src attribute and do an additional GET request image, iframe, script, video and audio.

## Using the XMLHttpRequestObject to Access Remote Data

**var request = new XMLHttpRequest();**

**var url =** [**http://contoso.com/resources/1**](http://contoso.com/resources/1)**;**

**request.open(“GET”, url);**

To transmit the request call the **send()** method – which does an async call.

Can use request.status to test for errors e.g.

**If (request.status != 200){ …**

### Consuming the Response

May be in JSON or xml can test the content-type returned:

**function getResponse(request){**

**var type = request.getResponseHeader(“Content-Type”);**

**switch(type){**

**case “text.xml” :**

**return request.responseXML;**

**default:**

**return request.responseText;**

**}**

**}**

### Handling an async response

Use the onreadystatechange handler:

**request.onreadystatechange = function(){**

**if (request.readyState == 4){**

**var response = JSON.parse(request.responseText);**

**}**

**}**

States:

0 – object not opened

1 – object has been opened

2 – object has sent a request

3 – object began receiving request

4 – object finished receiving response

### Transmitting data with a Request

Send with the send method: **send(data);**

Also set the content type e.g.

Request.setRequestHeader(“Content-Type”, “application/json”);

# 6. Styling HTML5 Using CSS3

## Fonts and Measurements

### @font-face

Used to specifiy a font to download and give it a name:

**@font-face{**

**font-family: ‘myFont’;**

**src: url(‘myFont.ttf’) format(‘truetype’);**

**}**

**body{**

**font-family: myFont, Arial;**

**}**

### Measurements

There are 6 units of absolute measurement:

* centimetres
* millimetres
* inches
* picas
* points
* pixels

7 units of relative measurements:

* em – current font size of the element
* ex – height of fonts lowercase x-height
* ch – width of the fonts 0 character
* rem – size of the font defined on the html element

Viewport relative (relative to size of browser window object)

* vw - 1% of the width of the viewport
* vh: 1% of the height of the viewport
* vmin – the smaller of vh and vw

### Implementing Text Effects

Paragraph indentation – text-indent: 3rem;

Line wrapping –hyphens: manual; word-wrap: break-word;

Text spacing: word-spacing: 2rem;

Shadow effects – text-shadow: 2px 2px 0 red;

(x-offset, y-offset, blur, color)

## Styling Block Elements

New box level support for:

### Outlines

Defines an outline box in addition to the four concentric boxes that make up the box model. Outline does not add to the total width or height, drawn above the margin box. Can overlap on a page.

* outline-width
* outline-style
* outline-color
* outline-offset

### Presentation

* border-radius
* overflow-x and overflow-y
* resize

### Multiple Column Layout

* column-count
* column-width
* column-gap
* column-rule

### Block Layout Models

* **block** – boxes formatted down the page respecting padding, border and margin
* **inline** – boxes formatted one after another based on the baseline of text content until the break onto another line. Ignores height and width
* **inline-block** – boxes formatted one after another but keep height and width
* **table** – identify blocks as tables, rows, colums and cells. Aligned by edges rather than content and sized to fit the computed table.
* **Flexbox** – allows flexing of elements to fix different viewports

## Pseudo-classes and Pseudo-elements

### Text Pseudo-Elements

* first-letter
* first-line
* before
* after
* selection

### Link and form Pseudo-Classes (Las Vegas fights Hells Angels)

* a:link
* a:visited
* a:focus
* a:hover
* a:active
* input: enabled
* input: disabled
* input: checked

## Enhancing Graphical Effects by Using CSS3

### Gradients:

* Linear gradient

Background: linear-gradient(direction, start-color, [mid-color-list], end-color);

* Radial gradient

Background: radial-gradient(position, shape, start-color, [mid-color-list], end-color);

### Transforming elements by using css3

* **Translate3d(x,y,z)** – moves the whole element along x, y and z axis

transform: translate3d(10px, 50px, 10px);

* **Scale3d(x,y,z)** – scales element by factors along x,y and z axis

transform: scale3d(2, 4, 0.5)

* **Rotate3d(x,y,z,a)** – rotates and element in 3d by angle a around the point x,y,z
* **Skew(a,b)** – skews an alement by angle a long x axis and b along y axis, degrees between 0 and 180

# 7. Creating Objects and Methods by Using Javascript

## Scoping and Hoisting

A variable has global scope if it is defined outside of a function.

A variable has function scope if it is declared inside of a function.

## Managing the Global Namespace

### Immediately invoked functions

This is a function that is defined and run immediately, any variables and functions created inside the function body are scoped to the immediately involved function. E.g.

**(function(){**

**//stuff here in immediate function scope**

**});**

### Namespacing

**var MyNamespace = {**

**myFunction1: function(params){**

**//Code**

**}**

**};**

### Strict mode

Without using var the variable will have global scope. To prevent this: “use strict”;

## Singleton Objects and Global Functions in Javascript

e.g. Math and JSON

### Using Prototypes

Prototypes give you a way to share functions between objects created e.g.

var Account = function(id, name){

this.id = id;

this.name = name;

this.balance = 0;

this.numTransactions = 0;

};

Account.protoype = {

Deposit: function(amount){

this.balance += amount;

this.numTransactions++;

}

}

### Using the Object.create method

The **Object** object has a **create()** method that enables you to create an object based on an existing prototype and optionally provide additional properties. This allows a form of inheritance.

**Object,create(prototypeObject, propertiesObject)**

Use **Object.getPrototypeOf(obj)** to get the prototype of an existing object

## Extending Objects

### Implementing encapsulation

Closures enable you to define encapsulated variables fr an object and expose the variables through a set of public accessor functions.

1. Declare variables without the this keyword
2. Declare the methods to get and set the variables and use the this keyword to ensure they are visible to external code

### Implementing and Chaining Prototypes

You can use several mechanisms to implement inheritance, e.g. **Object.create**. Another common approach is to make use of constructor function prototypes:

1. Define the base constructor and prototype
2. Define the derived constructor
3. Set the proptype property of the derived constructor to be an instance of the base object – this ensures the derived property has access to all the members of the base prototype
4. Reset the constructor property in the derived prototype so that it refers back to the derived constructor

On generic methods can use apply() to use a method on an object:

function SetColor(color){

this.color = color;

}

Set.Color.apply(p1, “[red]”);

# 8. Creating Interactive Pages by Using HTML5 APIs

## HTML5 File Interfaces

* **Blob** – represents immutable raw data, has a type attribute that indicates the media.
* **File** interface, has two read only attributes:
  + **name**
  + **lastModifiedDate**
* **FileList** – collection of file objects, two ways to obtain
  + Defined an <input type=’file’> element and handle the change event
  + Handle drop even on an elment
* **FileReader** – read a file or blob into a javascript variable

### FileReader Interface

* readAsText() – for text files
* readAsDataURL() – useful for contents of binary files
* readAsArrayBuffer()

Reads are async, number of events that typically would be handled:

* progress
* load
* about
* error
* loadend

### Implementing Drag and Drop

* set the **draggable** attribute for the element to true
* Handle the **dragstart** event – through the **dataTransfer** attribute on the event object, which represents a DataTrasnfer object – setData(mimeType, data)
* **setDragImage**(imgElement, s, y)
* **effectAllowed** (copy, move, link)
* **dropEffect**

## Incorporating Multimedia

**<video>**

* width and height
* poster
* controls
* autoplay
* loop
* muted
* src and type

Video interaction using javascript code:

* play()
* pause()
* paused
* currentTime
* duration
* volume
* playbackRate
* **loadedmetadata –** gets enough information to know duration
* **loadeddata –** when all the video data has been loaded
* **timeupdate –** fires during playback to indicate the current time

**<audio> manu similarities to <video>**

HTML5 Geolocation API

* navigator.geolocation.getCurrentPosition()
* navigator.geolocation.watchPosition() and navigator.geolocation.clearWatch();

Both getCurrentPosition and watchPosition take the following params:

* callback function
* optional error callback function
* optional PositionOptions object:
  + enableHighAccuracy
  + timeout
  + maximumAge

callback function recives an object with cords property:

* latitude
* longitude
* accuracy – meters
* altitude
* alititudeAccuracy
* heading
* speed

### Detecting the Context for Page

Maybe useful to poll less often or pause e.g. game

Document.hidden

document,visibilityState

visibilitychange event

Can determine network connectivity:

document.onLine

online event

offline event

# 9. Adding Offline Support to Web Applications

## Maintaining Session State Information by Using Cookie s

Persisting Session Data by Using Session Storage

Session storage is a browser-persistence that can store text data on the device running the browser. When the browser is closed the session store is cleared automatically. You can access it via the window object and test for it:

If (window.sessionStorage){

//do stuff

}

* sessionStorage **.setItem(“myKey”, “text value”;**
* sessionStorage **.getItem(“myKey”)**
* **can use array notation and pseudo properties for each key**
* sessionStorage **.removeItem**

### Persisting data across sessions by using Local Storage

Similar syntax to above but uses the localStorage API:

* localStorage.**setItem**
* localStorage.**getItem**
* localStorage.**removeItem**
* localStorage.**clear()**

### Handling storage events:

The storage API which session and local storage confirm include a single event called **storage** which you can uses to notifies a web page of changes e.g. call a server when data is modified:

Function myStorageCallback(e){

Alert(“Key: “ + e.key + “ Changed to: “ + e.newValue);

}

window.addEventListener(“storage”, myStorageCallback, true);

## Storing Structured Data by Using in Indexed Database API

IndexedDb provides an efficient mechanism for storing structured data held locally on the device running the browser. You access IndexedDB by using the **indexedDB** property of the window object.

You connect to a database by creating a request objet that references the open() function. If the database does not exist it is created.

**var db;**

**var openRequest = indexedDB.open(“contosoDB”);**

**openRequest.onsuccess = function(event){**

**db = event.target.result;**

**};**

**openRequest.onerror = function(event){**

**alert(“Error “ + event.target.errorCode);**

**};**

You define an object store by using the createObjectStore() function; specifying a key and an object to add to the store.

var myObject = {

id: “1”

};

var myObjectStore = db.createObjectStore(“myObjects”, { keyPath: “id” });

You can use the add() method on an object store to add an object, put() to modify an existing object and delete() to remove it.

## Adding offline support by using the Application Cache

Application cache is a client-side storage mechanism that enables the developer to explicitly declare which static files should be cached by the browser. You can use this mechanism to create website that run just as well offline as they do online.

The file should have the .manifest file extension – you may need to configure the web server to server this type of file by adding the MIME time: text/cache-manifest.

The file starts with the line:

**CACHE MANIFEST**

And may have the following sections:

* **CACHE** – resources listed here will be downloaded into the users browser then the cached version used
* **NETWORK** – resources will always be downloaded if the network is available
* **FALLBACK** – resources here are not cached but you provide an alternative URL for them should the server become unavailable

Application cache is available through the window object. Events include:

* **checking** – when the browser examines the application cache for updates
* **downloading** – when downloading resources to the application
* **updateready** – when new version of cached objects for a webpage have been downloaded
* **obsolete** – fires if the manifest file no longer available and the application cache nolonger valid for the current web page
* **cached** – fires when the application cache is ready and available for use
* **error**
* noupdate
* progress

Also has a numeric status property:

1. Uncached – page not associated with a cache, no resources downloaded
2. Idle – all cached resources downloaded
3. Checking – cache is being checked for updates to download
4. Downloading
5. Updateready
6. Obsolete

### Triggering Resource Updates

To force an update to get the new version of an existing resource you must make a significant change to the manifest file Updating the last modified date is not enough, common to use a comment with a version number, then update this:

#version=1.2.3

You can use the update() function to check for updates, existing resources will be used until the page is reloaded or you invoke the swapCache() function e.g.

If (applicationCache.status == 4){

applicationCache.swapCache();

}

## Testing for Network Connectivity

You can detect the network connectivity in an application by using the **online** property of the **navigator** object.