**70-480 Exam Reference Notes**

**Chapter 1 – Implement and manipulate document structures and objects**

**Using HTML5 semantic markup**

|  |  |
| --- | --- |
| HTML5 Element | Description |
| *<article>* | Defines self-contained areas on a page |
| *<aside>* | Defines smaller content areas outside the flow of a webpage |
| *<figcaption>* | Defines the caption of a figure element |
| *<figure>* | Defines content that contains a figure, such as an image, chart, or picture |
| *<footer>* | Defines the bottom of a section or page |
| *<header>* | Defines the top of a section or page |
| *<hgroup>* | Defines a group of headings (H1–H6 elements) |
| *<mark>* | Defines text that should be highlighted |
| *<nav>* | Defines navigation to other pages in the site |
| *<progress>* | Defines the progress of the task |
| *<section>* | Defines the distinct content of a document |

**DOM Element Properties**

|  |  |
| --- | --- |
| Property | Description |
| childNodes | A collection of all child nodes of the parent element |
| firstChild | A reference to the very first child node in the list of nodes of the parent node. |
| lastChild | A reference to the very last child node in the list of the child nodes of the parent node |
| hasChildNodes | *True* if the parent element has any child nodes. *False* if it has none. Good to use before firstChild or lastChild. |

**JavaScript**

|  |  |
| --- | --- |
| Function | Action |
| Alert | Pop up dialog in browser, accesses the DOM |
| getElementById | Access a specific DOM element based on its CSS ID selector |
| getElementsByTagName | Access all elements of a specific type |
| getElementsByClassName | Get all elements of the same CSS class |
| removeChild | Takes an element and removes its first child |
| createElement | Makes a new element based on passed in parameter type |
| innerText | Sets the content for inside an element |
| appendChild | Adds a child element onto passed in element (after). Returns a reference to the new element appended to the child elements. |
| insertBefore | Parameters: new elements, node before which you want to append the new element. Inserts the new element before the node. |
| removeNode | Takes one Boolean parameter, if *True*, method does a deep removal (all children are also removed) |
| replaceNode, replaceChild | Same effect as removeNode & removeChild but they allow you to replace the target element with a new one.  Ex. Convert all <p> tags to <a> tags, use a for loop. |

Good to keep a unique ID reference for all of your DOM objects, especially if you remove an element and want to use it elsewhere later. This way you won’t lose references to your elements completely if the JavaScript variable goes out of scope.

**Implementing media controls**

<video> element

<video src=”thevideofile.mp4” autoplay> </video>

<video controls height="400" width="600" poster="picture.jpg">

<source src="samplevideo.ogv" type="video/ogg"/>

<source src="samplevideo.mp4" type="audio/mp4"/>

<object>

<p>Video is not supported by this browser.</p>

</object>

</video>

|  |  |
| --- | --- |
| Attribute | Description |
| Src | Specifies video to play (local or a URL). Not all browsers support the same video formats, MP4 is a good standard though. You can have multiple src’s in a video control tag to assist with this, and an object to notify the user if the browser doesn’t support the file. Browser goes through the source elements from top to bottom and plays the first one that it supports. Browsers that don’t support the video element will not show it at all. |
| Autoplay | Tells browser to start playing video as soon as it loads. If not included, video only plays when told to through player control or through JS. |
| Controls | Tells browser to include or hide built-in video controls |
| Height/width | Amount of space the video will occupy, without a definition the video will be played at its native size. |
| Loop | Tells browser if it should continuously play the video after it is finished playing the first time. |
| Poster | Specifies an image to show in the place for the video until the user starts to play the video. |

<audio> element

Essentially the same as the video element but for audio files. Both provide a standard way of displaying audio and video content for more straightforward markup.

**Implementing graphics with HTML5 <canvas> and SVG**

<canvas> provides a blank canvas area that can be drawn on dynamically.

No default visibility (like a div) on a blank HTML page.

Need to specify a size for the canvas.

Should have an onLoad JS event to cause graphics to render when the page is loaded.

Fixed (x, y) coordinate system, top left corner of the canvas is (0, 0). Width and height determine the x and y axis lengths.

Context is currently only 2d but 3d will come in the future.

|  |  |
| --- | --- |
| Method | Description |
| beginPath | Rests/begins a new drawing path |
| moveTo | Moves the context to the point set in the beginPath method |
| lineTo | Sets the destination end point for the line |
| Stroke | Strokes the line, which makes the line visible |
| lineWidth | Line thickness property |
| strokeStyle | Color of the line, HEX or string |

Methods for drawing curves

|  |  |
| --- | --- |
| Method | Description |
| Arc | A standard arc based on a string and ending angle and a defined radius |
| QuadradicCurveTo | More complex arc that allows you to control the steepness of the curve, has one control point. |
| bezierCurveTo | Another complex arc that you can skew, has two control points. |

Each drawing method has styles applicable to it (lineWidth, strokeStyle, lineCap).

**Using path methods**

Always need a start and the end point when using the context object to draw.

Ending point for one stroke can be the starting point of another stroke.

Use the *beginPath* method for this (creates a new line), then draw all your lines, then the *closePath* method to close a line.

**Using the rect method**

Draws rectangles.

|  |  |
| --- | --- |
| Parameter | Description |
| X, Y | Starting point of the rectangle (top left corner) |
| Width | Rectangle’s width |
| Height | Height of the rectangle |

Center a rectangle by dividing your X and Y (width) parameters by 2 for the starting position of the rectangle.

**Using the fill method**

Fills shapes with colours, gradients, or patterns.

Sett the *fillStyle* property to a colour and call the *fill* method (before calling the *stroke* method).

*fillRect* is best for filling rectangles.

Logic is completely handled by the browser.

Creating a gradient involves using a new *CanvasGradient* object. First call the *createLinearGradient* methodto get a *CanvasGradient* object. On the *CG* object you define the color stops that you want to blend to create the gradient effect. Then you assign your *CG* object to the *fillStyle* property of the context.

To repeat a pattern, use an *onLoad* call to ensure that the image is repeated only after it is loaded by the browser. *createPattern(img, “repeat”);*

**Drawing images**

Use the *drawImage* method of the context object to draw an image on a canvas.

Methods takes an Image object and X & Y coordinates to define where the image should be drawn. Default IMG size if it’s resolution, but it is resizable (just mathematically alter the X & Y dimensions in the parameter calls).

**Drawing text**

strokeText(“string here”, x width, y height) – called on context object.

Always do after windows loads (window.onLoad).

Change font with .font = “…”

Use strokeStyle = “colour” to change the colour of the text.

textAlign =”center” for aligning text in the center. *Note:* Need to pass in X/2, Y/2 to center text in the center of the canvas.

**Scalable Vector Graphics (SVG)**

XML-based language for creating 2D graphics.

<svg> element.

Don’t lose quality when a user zooms in or out because they are vectors.

Can be accessed via the DOM.

Have attributes, styles, and event handlers.

SVG element provides a container to render graphics in.

Renders inline with the page’s layout.

In some cases SVG graphics are simpler than using the *canvas* element.

Performance can be a concern with SVG.

SVG’s can be rendered from existing graphics (image href).

SVG’s have their graphics rendered using a declarative syntax (*canvas* elements are drawn via JS through a reference to the context).

**Objective review**

1. D
2. A, C
3. B
4. C
5. B or C
6. C or D

**Applying styling to HTML elements programmatically (Objective 1.3)**

By default, all HTML elements flow statically from left to right in the same order that they are declared in the HTML page.

CSS provides more advanced options for element position.

*Absolute* or *relative* positioning can alter this flow.

*Absolute:* Element is placed at exact position specified

*Relative:* Element is positioned relative to its immediate left sibling’s coordinates.

*Top, Left, Right, & Bottom* are properties that take a distance parameter that specifies the relative distance of the object from a reference point based on the positioning attribute specified.

Default border or margins are ignored with *Absolute & Relative* positioning because the object is positioned where the positioning attributes direct the elements to be.

**Applying a transform**

Enable you to change an element’s appearance (larger, smaller, rotated, etc.).

Declare a transform for an object in CSS, apply it via JS through window.onLoad

.rota

{

transform: rotate(90deg);

}

**Objective review – page 71**

1. A – Absolute: top-left corner
2. D – Scale
3. D – Visibility=’hidden’

**Implement HTMML5 APIs**

Web Storage API

* Storing webpage data locally
* Essentially replaces the concept of cookies
* *Local –* Persistent, data is stored on the user’s machine and is available even after the user closes their browser completely and then reopens it to your site. Use the *localStorage* object.
* *Session –* Available only for the duration that the browser is open. Use the *sessionStorage* object.
* Both objects provide the exact same APIs.
* Name-Value pairs. Can store strings.
* Data resides locally and stays local (good thing), unlike with cookies which need to go to the server
* Data stored in web storage is organized by *root domain*.
* Should do max 5 mb.

**AppCache API**

* Makes content and webpages available even when a web app is in offline mode.
* Stores files in the application cache in the browser.
* Limited storage.
* Available as a global object.
* <html manifest=”webApp.appcache”>………</html>
* If you want to change the entension from .appCache you must configure your server with a MIME type for the manifest
* Manifest file – each section may appear once, multiple times, or not at all
* *CACHE MANIFEST* – Explanation of file
  + *CACHE:* Lists all resources that must be cached offline
  + *NETWORK:* Lists all resources that must have an internet connection
  + *FALLBACK:* Provides instructions for the browser in the event that an item isn’t available in the cache and the browser is in offline mode.
* Look up AppCache events\*

**Using the Geolocation API**

* Var geoLocattor = window.navigator.geolocation;
* Code above for shorthand access to the API
* *getCurrentPosition* method
  + getCurrentPosition(positionCallback, [positionErrorCallBack], [positionOptions])
  + Gives the position of the user’s device
  + 1st parameter is required, 2nd and 3rd are optional
    1. Callback method that is called after position is determined
    2. Error callback
    3. Set special options for how the method behaves
       - enableHighAccuracy (more precise & resource intensive)
       - timeout (0 == infinite time to get location)
       - maximumAge (used a cached result if possible)
  + Browser security starts when location code is run
    - If not allowed by user, an exception is thrown
* *watchPosition* method
  + Continuously polls for the current position
  + Same parameters as *getCurrentPosition*
  + Holds a reference to the invoked *watchPosition* instance
  + Calls success callback method every time the Geo API detects a new location
  + Polling continues forever until stopped
    - Cancel polling through the *clearWatch* method

**Objective review (page 85)**

1. C
2. C
3. B
4. A
5. B

**Objective 1.5: Establish the scope of objects & variables – JavaScript**

Scope and Lifetime are vital

Declaring variables and instantiating objects consumes resources (mainly memory)

As memory usage increases, typically the usage of other system resources such as battery power are increased in their usage as well.

A user is less likely to use apps that eat up their resources unfairly.

*Var* keyword is used to declare a variable

Vars can be declared concurrently

*Var x = 0, y = 2, z = 3*

Until a *var* is initialized with the declaration of value for it, otherwise it is undefined.

“In Scope” is when a variable is available for use

Scope depends on where the var is declared.

* Global: Available throughout the web page. Any JS can access it.
* Local: Only available is a certain context (usually a method)
  + Don’t match names because locally scoped vars override higher-level variables of the same name (cant access the overridden values)
    - Use meaningful names to counteract this

Global Namespace

* Avoid it whenever possible
* Everything exposed by it is global
* Ex. Window
* Available to all code within an application session
  + With libraries, naming and security issues can arise
  + Name of classes in a namespace must be unique
  + Using the domain in reverse can be a good way to make a namespace
    - *com.microsoft*
  + Never implement your objects in the global namespace when making reusable JS libraries
  + Apply a namespace to custom objects to prevent conflicts in the global namespace

This (keyword)

* Used to reference the containing object directly
* Always refers to the object that contains the currently running code
* Context = *this* always refers to the object that contains the currently running code

Objective review

1. B
2. B
3. C

**Create and implement objects and methods**

Types:

1. Native – provided by JavaScript itself
   1. Can be static (don’t need to instantiate them)
2. Custom – for unique data constructs and behaviours

Objects can be based on other objects (subtypes)

Object inheritance – one object inherits all of the attributes and behaviours of another, but can also implement additional aspects that are unique to it

Objects encapsulate functionality and state information that is relevant for them

Functionality = methods

State information = properties

*New* keyword is used to instantiate an object. Tells runtime to allocate a new object of the type specified.

Overloaded constructor – having multiple constructors with different parameters for an object

**Creating custom objects**

You should apply proper object-oriented practices when developing JS apps

Usually involves creating custom objects to encapsulate functionality within logical entities

Properties of an object represent its state

Methods of an object provide its behaviour

*Prototype* – provides a definition of the object so that you can construct the object using the *new* keyword, can take parameters to initialize object state (use *this* for everything in the prototype)

Creation of an instance of an object is based on the object’s prototype

**Implementing inheritance**

JS doesn’t use classes – everything is an object

Inheritance is like an extension of functionality

var popupBook = Object.create(Book.protoType,{ hasSound: {value:true}, showPopUp:{ value: function showPop() {

//do logic to show a popup

}

}

});

If *null* is passed then the object uses only those functions or properties specified in the second parameter

If an object prototype is specified, the object is created with all the properties and functions declared on that object prototype

**Objective review pg. 101**

1. D
2. B
3. D

**jQuery**

**Chapter 2 – Implement program flow**

*Program Flow* – Providing users with certain website functions only under certain conditions

* *Conditional*
  + Base on evaluating state to make a decision as to which code should run
  + Enables the application to examine the state of an object or variable to decide which code path to process
  + *If Else, Switch, Ternary Operators*
* *Iterative*
  + The ability to process lists & collections of information systematically and consistently
* *Behavioural*
  + An event or callback in which specific logic should be applied based on user engagement with the web application or the completion of another task
* Flow is using *Conditional + Iterative + Behavioural*
* *Exception Handling* 
  + Exception Handling constructs provide the ability to run specific logic in the case of an error in the program

Without *Program Flow* in JS, programs would run from top to bottom in the order in which the code was written

|  |  |  |
| --- | --- | --- |
| Operator | Type | Description |
| > | Conditional | Evaluates whether the value on the left is greater than the value on the right |
| < | Conditional | Evaluates whether the value on the right is greater than the value on the left |
| >= , <= | Conditional | Evaluates the same as > or < but with the additional logic that the values can also be equal |
| != | Conditional | Evaluates whether the values aren’t equal |
| == | Conditional | Evaluates whether the values are equal independent of the underlying data type |
| === | Conditional | Evaluates whether the values are equal both in value and underlying data type |
| && | Logical | The AND logical operator, in which the expressions on both sides must evaluate to true |
| || | Logical | The OR logical operator, in which at least one expression on either side must evaluate to true |

*Switch* statements are usually good to use if you have many *If* cases (less messy)

Cannot use the Logical && operator with *Switches*, but you can use || by omitting the break; between a case and the case after it (so you don’t have to always use the explicit || operator to have an OR).

*Ternary operator* – Shorthand mechanism for an *If* statement

<expression> ? <true part>: <false part>

*Arrays –* JS objects, created with the *new* keyword

var anArray = new Array(); //default sized

var anArray = new Array(5); //5 spots, 0-4, undefined values

var anArray = new Array('soccer', 'basketball', …, 'badminton');

If you make an array with a set size (5) but assign a higher index value (5<) then the array will automatically resize to accommodate the requested length

*Length property –* provides info on how long the array is (how many elements the array has allocated at the time of the property’s evaluation)

*Concat method –* combines 2 or more arrays into a single array

*indexOf –* provides a way to find the index of a known element, ascending order – parameters (term, index to start search from). Returns -1 if the index is not found.

*lastIndexOf –* find the index of a known element in a bottom to top (descending) order

*join –* Puts all the elements in an array into a single string separated by a specific delimiter

*reverse –* Reverses the sequence of all elements in the array

*sort –* sequences array items in ascending order

*slice –* takes out one or more items in an array and moves them to a new array

*splice –* provides a way to replace items in an array with new items – parameters (start index, # of items, item values CSV)

*Queue –* First-In-First-Out type of collection. New items are added to the end of the list.

* *Shift method –* removes and returns the first element of the array
* *Unshift method –* inserts new elements at the beginning of the array

*Stack –* Last-In-First-Out type of collection. New items are added to the beginning of the list.

* *Pop method –* removes the last item from the array
* *Push method –* appends the specified items to the end of the array