Advanced HTML5

Section 1: Introduction to HTML5

1.1 Introduction, Tags and Elements

HTML5 is the new standard and is a cooperation between W3C and the Web Hypertext Application Technology Working Group (WHATWG) for setting the modern HMTL5. HTML5 offers some new features that make webpage more dynamic.

HTML5 introduces are new sets of tags such as <header> and <section>, a <canvas> element for drawing 2D drawings, access to local storage (*alternative to cookies to store small amount of data in a user's browser*), new form controls like calendar, date and time, media functionality and geolocation.

Most modern browsers support most if not all of the HTML5 features.

A very useful website for learning more about the HTML5 tags and elements is the W3School website (https://www.w3schools.com/html/default.asp).

1.2 New Form Elements

The best new feature of HTML5 are the new form/input element tags which make it easy to create HTML forms for users to interact with a webpage/web app. Below are examples:

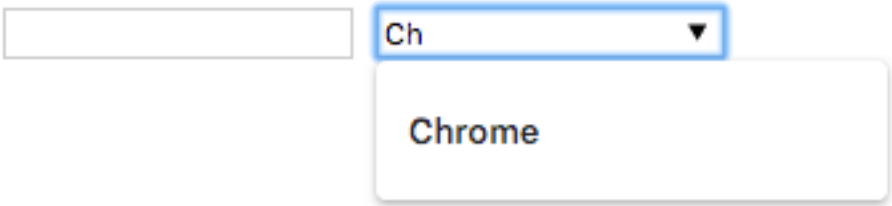
Example	Description
<input type="email" name="">	Allows the form to validate the email address format without the use of JavaScript
<input type="tel" name="">	Used to define a field for entering a telephone number with validation
<input type="color" name="">	Allows to select a colour from a colour picker
<input type="date" name="">	Allows the user to select a date from a popup calendar
<input type="range" name="" min="1" max="10">	Control over the number input (like a slider input)
<input type="time" name="">	Used to define a filed for entering a time
<input type="url" name="">	Used to define a field for entering a url/link

The **<datalist>** is a new form element which specifies a list of pre-defined options for an <input> element. It provides a autocomplete feature on the <input> elements. The

element will look like a text input element and the user will see a dropdown list of pre-defined options as they start to input the data (i.e. start typing). The <input> element's list attribute is used to bind it together with a <datalist> element, for example:

```
<input list="browser">

<datalist id="browser">
  <option value="Chrome">
  <option value="Edge">
  <option value="Firefox">
  <option value="Internet Explorer">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

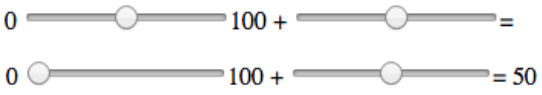


The **<keygen>** form element provides an advanced way to authenticate users on client side. The <keygen> tag specifies a key-pair generator field in a form and when the form is submitted two keys are generated, one private and one public. The private key is stored locally and the public key is sent to the server. The public key could be used to generate a client certificate to authenticate the user in the future. For example:

```
<form action="demo.php" method="GET">
  User: <input type="text" name="usr_name">
  Encryption: <keygen name="security">
  <input type="submit">
</form>
```

The **<output>** form element performs a calculation and shows the result in the <output> element. For example:

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
  <input type="range" id="a" value="50">100 +
  <input type="range" id="b" value="50">=
  <output name="x" for="a b"></output>
</form>
```



1.3 HTML5 Canvas

The canvas element is used to create lines, shapes, etc. on the webpage. The HTML is used to create the canvas while JavaScript is used to draw the 2d objects inside of the canvas.

The canvas element requires a width and height attributes. The id attribute allows us to target the element with our CSS and JavaScript.

```
<body onload="makecanvas()">
  <canvas id="canvas 1" width="400" height="300"></canvas>
</body>
```

The onload attribute on the <body> element is a lifecycle event which triggers on load of the html page and can be used to trigger off a JavaScript function. The JavaScript file must be imported using a <script> tag placed either in the <header> element or at the bottom of the <body> element tags depending on how early you want to load/import the JavaScript code.

Below is an example JavaScript code to write some text on the canvas:

```
function makecanvas() {
    // 1. Get Object
    var canvas1 = document.getElementById("canvas 1");
    var ctx1 = canvas1.getContext('2d');

    // 2. Set Properties
    ctx1.font = '32pt Arial';
    ctx1.fillStyle = 'DeepSkyBlue';
    ctx1.strokeStyle = 'black';

    // 3. Action
    ctx1.fillText("I Love HTML5", 45, 150);
    ctx1.strokeText("I Love HTML5", 45, 150);
}
```

Three Steps:
1. Get object (declare variables)

2. Set the object properties

3. Action i.e. use the object to draw on the canvas.

I Love HTML5

This is a very basic example demonstrating how HTML5 canvas can be used to create 2d objects on a webpage. More advanced JavaScript code can make more complicated objects with animation/particle effects.

1.4 HTML5 Feature Detection & Drag and Drop

Feature detection is finding specific properties or methods in a browser's DOM to detect the browser type and whether it supports a given operation. It is not a browser detection.

Disclaimer: Developers use to use "UA Sniffing" to detect the user's browser using the navigator.userAgent property. This would detect the actual browser but was unreliable. This is now obsolete and not recommended.

The best way to detect HTML5 and CSS3 features are by using third party JavaScript libraries such as Modenizr (https://modernizr.com/). Modernizr does not try to detect the browser but it detects certain features whether supported by the client browser. Once implemented we can use simple programming such as the below examples:

```
if (document.querySelector) {
    element = document.querySelector(selectors);
}
```

To implement Modernizr in our webpage is to download the file and include it the <head> section and add the class="no-js" to the <html> element tag.

```
<!DOCTYPE html>
<html lang="en" class="no-js">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HTML5</title>
    <script src="modernizr-2.0.6.min.js"></script>
  </head>
```

There are other third party JavaScript feature detection libraries but Modernizr is one of the best ones out there.

Modernizr creates properties for all of the features that it test for and we can refer to these properties to check whether the browser supports the object and its properties.

```
if (Modernizr.canvas) {
    console.log("Yes, canvas is support")
} else {
    console.log("No, canvas is not support")
}
```

If statement can be used to check if for example the HTML5 canvas property is supported by the browser and if so run some block of JavaScript code else run another block of JavaScript code.

The drag and drop syntax requires the element to have a draggable attribute to be set to true. Once the element is set to draggable we would need to specify what will happen when the element is dragged. The ondragstart attribute calls the function. Within the function we have the setData( ) function which takes in two arguments of the data type and the value of the drag data.

```
<img src="" draggable="true" ondragstart="drag(event)">
```

```
function drag(e) {
    e.dataTranfer.setData("Text", e.target.id);
}
```

By default, data/elements cannot be dropped in other elements. To allow a drop, we need to prevent the default handling of the element and this is done with the event.preventDefault( ) function. So the syntax for the function would now look like:

```
function drag(e) {
    e.preventDefault();
    var data = e.dataTranfer.getData("Text");
    e.target.appendChild(document.getElementById(data));
}
```

The .appendChild method will do the drop of the dragged element.

Below is a sample code of implementing both the feature detection and the drag and drop HTML5 features.
Disclaimer: The CSS and JavaScript code are added in the <head> element tags for demonstration and is not considered good practice.

```
<!DOCTYPE html>
<html lang="en" class="no-js">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/modernizr/2.8.3/modernizr.min.js"></script>
    <title>HTML5</title>
    <style>
      #div1 {
        width: 250px;
        height: 250px;
        padding: 10px;
        border: 1px #aaaaaa solid;
        float: left;
      }
    </style>

    <script type="text/javascript">
      if (window.FileReader && Modernizr.draganddrop) {
        function allowDrop(e) {
          e.preventDefault();
        }

        function drag(e) {
          e.dataTransfer.setData("Text", e.target.id);
        }

        function drop(e) {
          e.preventDefault();
          var data = e.dataTransfer.getData("Text");
          e.target.appendChild(document.getElementById(data));
        }
      } else {
        document.write("This browser does NOT support drag and drop");
      }
    </script>
  </head>
  <body>
    <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    <img id="drag1" src="html5.jpg" alt="HTML5 Logo" draggable="true" ondragstart="drag(event)">
  </body>
</html>
```

1.5 Multimedia

HTML5 introduces a much simpler way of adding videos and audio clips in web pages. Previously plugins such as Flash/Shockwave was required to display videos in a web page. It is now as simple as including an image.

The types of media (MIME Types) supported for videos are Mp4, WebM and Ogg while for audio Mp3, Ogg and Wav are supported.

HTML5 Element Tags	Description
<video></video>	Define a video or audio content
<source></source>	Define multiple sources for the video/audio element(s)
<track></track>	Defines text tracks in media player

Important Note: Javascript is needed for extra functionality and controls.

JavaScript Methods	Description
addTextTrack( );	Adds a new track
canPlayType( );	Checks if the browser can play a file type
load( );	Reloads the audio/video element
play( );	Starts playing the audio/video
pause( );	Pauses the playing of the audio/video

1.6 SVG in HTML5

Scalable Vector Graphic (SVG) is used to define vector-based graphics on a web page. Graphics are defined in Extensive Markup Language (XML) and vector graphics do not lose any quality whether resized smaller or larger. SVG are recommended by the W3C.

SVG is mostly useful for vector type diagrams like Pie Charts, Two-dimensional Graphs in a X,Y coordinate system, etc.

Most modern web browsers can display SVG just like they can when displaying PNG, GIF and JPG files.

HTML5 allows embedding of SVG directly using the <svg></svg> element tag.

```
<svg height="130" xmlns="http://www.w3.org/2000/svg">
  <circle id="circle1" cx="70" cy="70" r="50" fill="blue"/>
</svg>
```

We can also animate SVG using the <animate /> element tag nested within the <sgv></svg> tags as seen in the below example:

```
<svg width="auto" xmlns="http://www.w3.org/2000/svg">
  <rect id="rectangle" x="20" y="10" width="100" height="100" fill="red" />
  <animate
    attributeName="x"
    xlink:href="#rectangle"
    begin="0s"
    dur="5s"
    from="30"
    to="500"
    fill="freeze"
  />
</svg>
<svg width="auto" xmlns="http://www.w3.org/2000/svg">
  <circle id="circle" cx="70" cy="70" r="50" fill="blue" />
  <animate
    attributeName="cx"
    xlink:href="#circle"
    begin="0s"
    dur="5s"
    from="30"
    to="500"
    repeatCount="indefinite"
  />
</svg>
```

The attributeName is the attribute of the SVG we wish to animate. In this example we are moving the rectangle on the X axis from start position 30 to end position 500 on the X axis.

The freeze value will freeze the element at the end position rather than jumping back to the start position when the anumation duration ends.

We can use the repeatCount property on <animate /> to repeat the animation for a number of time or indefinitely.

We can chain multiple <animate /> tags on a single SVG to perform multiple animation such as moving the element from left and right and changing the colour as it moves.