



# HyperText Markup Language



# Agenda

- Introduction
  - Html Review
  - HTML5 new elements
  - HTML5 API'S
- 



# Introduction



- HTML is a markup language for describing web documents.
- **Hypertext** is text which contains links to other texts.
- Markup language used to identify piece of document to do something, usually to describe it's logical structure.
- Markup indicators are often called 'tags'



# History



- ▶ Tim Berners-Lee, a physicist working at CERN labs | 1989 is the inventor of HTML, and is often credited as the inventor of www.
- ▶ HTML is based on SGML(Standard Generalized Markup Language), which existed since 1940's, but without the ability to link.
- ▶ Tim Berners-Lee proposed a way for enabling researchers from remote sites in the world to organize and pool together information.
- ▶ He suggested that you could actually link the text in the files themselves, through an agreed-upon markup language
- ▶ Today, the World Wide Web consortium (W3C), is an international consortium, where member organizations, a full-time staff and public work together to develop web standards. Tim Berners-Lee is the director of W3C.



# HTML Versions

- HTML – 1991
  - HTML 2.0 – 1995
  - HTML 3.2 – 1997
  - HTML 4.01 – 1999
  - XHTML – 2000
  - HTML5 - 2014
- 



# Editors

- Notepad++
  - Open Source
  - Line Numbering
  - Zoom in and out
  - Detection of modified files
  - Search and Replace
  - Syntax Highlighting



# Contd.,

- ▢ Brackets
  - ▢ Open Source
  - ▢ Quick Edit
    - ▢ CSS
    - ▢ Colors
    - ▢ Curves and more
  - ▢ JSLint
  - ▢ Live Preview

# Basic structure





# Document Type

- The **<!doctype>** declaration is not an HTML tag, it is an instruction to the web browser about what version of HTML the page is written in.
- HTML 5 - `<!DOCTYPE html>`
- The earlier versions of HTML are based on SGML, so they need a reference to the DTD. For example,
- HTML 4.01 Strict - `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`
- XHTML 1.1 - `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">`
- **Comments**
  - `<-- comment here - >`



# HTML Review



- Elements
  - `<tagname> Content </tagname>`
  - Empty elements
  - Nested elements
- Attributes
  - Specified in the start tag
  - Usually come in name/value pairs



# Elements

- Headings - `<h1>` to `<h6>`
- Paragraph - `<p>` .... `</p>`
- Line breaks - `<br>` `<hr>`
- Horizontal Rule - `<hr>`



# HTML Text Formatting

- `<b>` - bold
- `<strong>` - Important text
- `<i>` - italic text
- `<em>` - emphasized text
- `<mark>` - marked text
- `<small>` - small text
- `<del>` - deleted text
- `<ins>` - inserted text
- `<sub>` - subscript text
- `<sup>` - superscript text

# HTML Lists

- Lists are used to group related pieces of information together
  - Unordered list - `<ul>`
  - Ordered list - `<ol>`
  - Description list - `<dl>`
- `<li>`
- `<style = "list-style-type: disc/circle/square/none">` (for unordered list)
- `<style = "list-style-type: 1/a/A/i/I">` (for ordered list)
- `<dl>`
  - `<dt>`
  - `<dd>`



# Nested list

➤ <ul>

➤ <li>

➤ <li>

➤ <li>

➤ <ul>

➤ <li>

➤ <li>



# Horizontal list

- Horizontal lists are generally used to create menu's (navigation bars)
- 



# Applying styles on lists







# Tables

- ▶ Tables are used to arrange the data in the cells of rows and columns
- ▶ Tables are defined using `<table>` tag
- ▶ Rows can be added to tables using `<tr>`
- ▶ Data can be inserted into the rows using `<td>`
- ▶ Table headers can be created using `<th>`
- ▶ You can assign borders to the table using border attribute
- ▶ Most of the attributes supported in earlier versions are not supported in HTML5.
- ▶ Caption can be added to the table using `<caption>` tag, it must be inserted immediately after the `<table>` tag
- ▶ `<thead>`, `<tbody>` and `<tfoot>` to group the content and apply styles to them.



# HTML Links

- Links in HTML can be used to access other resources
- In HTML, links are defined using `<a>` tag.
- `href` attribute is used to specify the resource to be linked
- Text between the tags will be visible part used for link
- Link can be an absolute value or local link
- Link colors (blue, purple, red)
- The `target` attribute specifies where to open the linked document
- It can (`_self`, `_blank`, `_top`, `_parent`, `framename`)
  
- `<a>` can be used to create bookmarks



# HTML Images




- `<img>` is used to define images
- `src` attribute specifies the address of the image
- `alt` attribute provides an alternate text for an image
- `width` and `height` are used to specify the size
  
- Images can be used as a link
- Image maps can be accessed using `img` tag.



# FORMS



# What are forms?

- *An HTML form is a section of document that contains interactive controls that enable a user to submit information to a web server.*
- 



# Controls

- Users interact with forms through named *controls*.
- Like buttons, menus, checkboxes, radio buttons, hidden controls, object controls, etc.,



# <form> element

- <form>
- ...
- ...
- form elements
- ....
- ...
- </form>



# Form attributes



Attribute	Description
Name	Specifies a name used to identify the form
Action	Specifies an address(url) where to submit the form
Method	specifies the http method used when submitting the form
Accept charset	Specifies the charset used in the submitted form
Auto complete	Specifies if the browser should auto complete the form
Enctype	Specifies the encoding of submitted data
Novalidate	Specifies that the browser should not validate the form
Target	Specifies the target of the address in the action attribute



# <form> elements

- There are different kinds of form elements we have, each one is used to provide some content to the form

Type	Description
Input	Defines an input control
Textarea	Defines a multiline input control
Label	Defines a label for an input element
Fieldset	Groups related elements in a form
Legend	Defines a caption for the <fieldset> element
Select	Defines a drop down list
Optgroup	Defines a group of related options in a drop-down list
Button	Defines a clickable button
Keygen	Defines a key-pair generator field
Output	Defines the result of a calculation

# <input> element

- Input element can be displayed in several ways, depending on the type attribute

Type	Description
Text	Defines one line text input field
Password	Defines a password field
Submit	Defines a button for submitting form data to a form-handler
Reset	Defines a reset button that will reset all form values to their default values
Radio	Defines a radio button
Checkbox	Defines a check box
Button	Defines a button
Color	Used for input fields that should contain a color
Number	Defines a numeric input field



## Contd.,

Type	Description
Number	Defines a numeric input field
Date	Is used for input fields that should contain a date
Range	Used for input fields that should contain a value within a range
Month	Allows a user to select a month and year
Week	Allows users to select a week and year
Time	Allows users to select time
Datetime-local	Specifies a date and time input field, with no time-zone
Email	Used for input fields that should contain an email address
Search	User for search fields
Tel	Used for input fields that should contain a telephone number
url	Used for input fields that should contain a url address

# <input> element attributes

Attribute	Description
Value	Specifies an initial value for an input field
Readonly	Specifies that the input field is read-only
Disabled	Specifies that the input field is disabled
Size	Specifies the size for the input field
Maxlength	Specifies the max allowed length for input field
Form	Specifies one or more forms an input field belongs to
Formaction	Specifies the url of the file that will process the input control when the form is submitted
Formenctype	Specifies how the form data should be encoded when submitted for forms with method post

## Contd.,

Attribute	Description
Formmethod	Defines the http method for sending form data to the action url
Formnovalidate	Overrides the novalidate attribute of the form element
Formtarget	specifies a name or a keyword that indicates where to display the response that is received after submitting the form
Height and width	specifies the height and width for an input type 'image' element
List	Refers to a datalist that contains predefined options for an <input> element
Min and max	specifies the min and max values for and input element
Multiple	specifies that the user is allowed to enter more than one value in the input element. It works with email and file



## Contd.,

Attribute	Description
Multiple	Specifies that the user is allowed to enter more than one value in the input element. It works with email and file
Pattern	Specifies a regular expression that the input elements value is checked against
Placeholder	Specifies a hint that describes the expected value of an input field
Required	Specifies that the input field must be filled
Step	Specifies the legal number interval for an input field



# **NEW SEMANTIC ELEMENTS**



# Semantic Elements



Element	Description
Article	Defines an article in the document
Aside	Defines contents aside from the main content
Bdi	Part of text that might be formatted in a different direction from other text
Details	Defines additional details the user can view
Dialog	Defines a dialog box or window
Figure	Defines a self contained content, diagram, photo, etc.,
Figcaption	Defines a caption for the figure element
Footer	Defines a footer for the document or section
Header	Defines a header for the document or section
Main	Defines the main content of a document
Mark	Defines a marked or highlighted text





## Contd.,

Element	Description
Menuitem	Defines a command or menu item that the user can invoke from a popup menu
Meter	Defines a scalar measurement within a known range
Nav	Defines navigation links in the document
Progress	Defines progress of a text
Rp	Defines what to show in a browser that do not support ruby annotation
Rt	Defines explanation/ pronunciation of characters (for east Asian typography)
Ruby	Defines a ruby annotation
Section	Defines a section in the document
Summary	Defines a visible heading for a details element
Time	Defines date / time
Wbr	Defines a possible line break



# AUDIO AND VIDEO



# Audio

- ▶ `<audio>` - specifies a standard way to embed audio in a webpage.
- ▶ `controls` attribute adds audio controls
- ▶ `<source>` - defines multiple media elements
- ▶ Supported formats – mp3, wav and ogg
- ▶ HTML5 defines DOM methods, properties and events.



## <source>

- It specifies multiple media resources for either, <picture>, the <audio> or the <video> element.
- Commonly used to serve the same media content in multiple formats
- <audio id="myaudio" controls>
- <source src="ringtone.mp3">
- <source src="ringtone.wav">
- <source src="ringtone.ogg">
- </audio>



# HTML MediaElement API

- ▶ The HTMLMediaElement interface adds to HTMLMediaElement the properties and methods needed to support basic media-related capabilities that are common to audio and video.
- ▶ The HTMLVideoElement and HTMLAudioElement elements both inherit this interface.
- ▶ Properties
  - ▶ autoplay
  - ▶ audiotracks
  - ▶ buffered
  - ▶ controls
  - ▶ loop
  - ▶ muted
  - ▶ duration
  - ▶ error
  - ▶ src
  - ▶ volume



# Contd.,

## ➤ **Methods**

- Load()
- Play()
- Pause()
- canPlayType()
- addTextTrack()



# document.getElementById()

- Returns a reference to the element by its ID
- **Syntax**
  - `element = document.getElementById(id);`
- **Parameters**
  - `id` - is a case-sensitive string representing the unique ID of the element being sought.
- **Return Value**
  - `element` - is a reference to an Element object, or null if an element with the specified ID is not in the document.



# Controlling media playback

- Once you've embedded media into your HTML document using the new elements, you can programmatically control them from JavaScript code.
- For example, to start (or restart) playback, you can do this:
  - `var v = document.getElementsByTagName("video")[0];`
  - `v.play();`





# Example

- ▶ `<audio id="myaudio" controls>`
- ▶ `<source src="ringtone.mp3">`
- ▶ `</audio>`



# Video



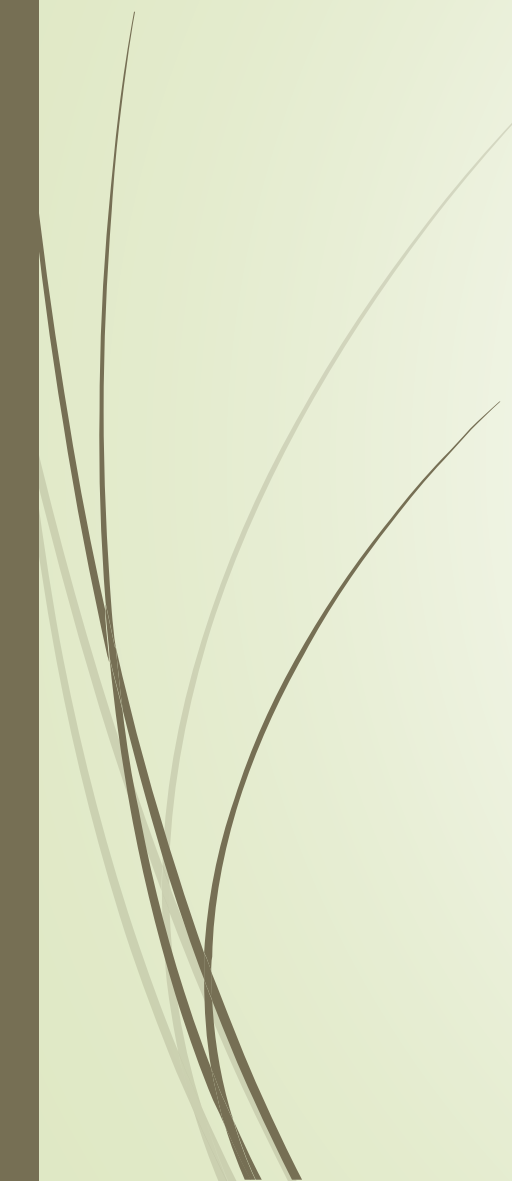
- ▶ `<video>` - specifies a standard way to embed video in a web page
- ▶ `controls` attribute adds controls
- ▶ `Width` and `height` attributes define the size
- ▶ `<source>` - defines multiple media elements
- ▶ Supported formats – mp4, webM and ogg
- ▶ `Autoplay` attribute is used to start the video automatically
- ▶ `<track>` - specifies text tracks for media elements
- ▶ HTML5 defines DOM methods, properties and events.



# CANVAS



# Canvas

- ▶ An HTML element used to draw graphics, using scripting language.
  - ▶ Canvas is a two-dimensional rectangular area with default width of 300px and height of 150px
  - ▶ The coordinates of top-left corner are (0,0) and bottom-right(width,height)
- 

# A Simple Canvas

- ▶ `<body>`
- ▶ `<canvas id="mycanvas" width="300px" height="200px"></canvas>`
- ▶ `</body>`
  
- ▶ `<script>`
- ▶ `window.onload=function(){`
- ▶ `var canvas = document.getElementById("myCanvas");`
- ▶ `var context = canvas.getContext("2d");`
- ▶ `// Drawing can be done here`
- ▶ `};`
- ▶ `</script>`



# Canvas Element

- The HTMLCanvasElement interface provides properties and methods for manipulating the layout and presentation of canvas elements.
- It also inherits the properties and methods of the HTMLElement interface.
- **Properties:**
  - HTMLCanvasElement.height
  - HTMLCanvasElement.width
- **Methods:**
  - HTMLCanvasElement.getContext()
  - HTMLCanvasElement.toDataURL()
  - HTMLCanvasElement.toBlob()




# HTMLCanvasElement.getContext()

- ▶ The HTMLCanvasElement.getContext() method returns a drawing context on the canvas, or null if the context identifier is not supported.
- ▶ **Syntax**
- ▶ `canvas.getContext(contextType, contextAttributes);`
- ▶ Return value
  - ▶ CanvasRenderingContext2D



# CanvasRenderingContext2D

- The CanvasRenderingContext2D interface is used for drawing graphics onto the canvas element.
  - It provides the 2D rendering context for the drawing surface of a <canvas> element.
- 





# Paths

- **Methods:**

- `context.moveTo(x,y);`
- `context.lineTo(x,y);`
- `context.stroke();`
- `context.fill();`
- `context.beginPath();`
- `context.closePath();`
- `context.arc(x,y,r,sAngle,eAngle,counterclockwise);`
- `context.arcTo(x1,y1,x2,y2,r);`
- `context.bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y);`
- `context.quadraticCurveTo(cpx,cpy,x,y);`
- `context.clip();`



# Line styles

- **Properties:**

- `context.lineCap="butt | round | square";`
- `context.lineJoin="bevel | round | miter";`
- `context.lineWidth=number;`
- `context.miterLimit=number;`



# Colors, Styles, and Shadows

## ■ **Properties:**

- `context.fillStyle=color | gradient | pattern;`
- `context.strokeStyle=color | gradient | pattern;`
- `context.shadowColor=color;`
- `context.shadowBlur=number;`
- `context.shadowOffsetX=number;`
- `context.shadowOffsetY=number;`

## ■ **Methods:**

- `context.createLinearGradient(x0,y0,x1,y1);`
- `context.createPattern(image,"repeat | repeat-x | repeat-y | no-repeat");`
- `context.createRadialGradient(x0,y0,r0,x1,y1,r1);`
- `gradient.addColorStop(stop,color);`



# Rectangles



- `context.rect(x,y,width,height);`
- `context.fillRect(x,y,width,height);`
- `context.strokeRect(x,y,width,height);`
- `context.clearRect(x,y,width,height);`



# Text styles

- *Properties:*

- *font, textalign, textbaseline*

- *Methods:*

- `context.fillText(text,x,y,maxWidth);`

- `context.strokeText(text,x,y,maxWidth);`

- `context.measureText(text).width;`



# Image Drawing

- ▶ `context.drawImage(img,x,y);`
- ▶ `context.drawImage(img,x,y,width,height);`



# Transformations

- `context.scale(scalewidth,scaleheight);`
- `context.rotate(angle);`
- `context.translate(x,y);`
- `context.transform(a,b,c,d,e,f);`
- `context.setTransform(a,b,c,d,e,f);`



**SVG**





# Scalable Vector Graphics (SVG)

- **Scalable Vector Graphics (SVG)** is an XML-based markup language for describing two-dimensional vector graphics. SVG is essentially to graphics what HTML is to text.
- A vector image can be scaled up or down to any extent without losing the image quality.
- SVG images and their behaviors are defined in XML files




# Advantages of SVG

- SVG images can be searched, indexed, scripted, and compressed.
- SVG images can be created and modified using JavaScript in real time.
- SVG images can be printed with high quality at any resolution.
- SVG content can be animated using the built-in [animation elements](#).
- SVG images can contain [hyperlinks](#) to other documents.



# Creating SVG Images

- SVG images can be created with any text editor, but it is often more convenient to create SVG images with a drawing program, like [Inkscape](#).
- 



# SVG Elements

- `<svg width, height> </svg>`
- `<g> </g>`
- `<use> </use>`
- `<defs> </defs>`

# SVG Text

- `<text x,y,font-family, font-color, cursive> </text>`
- `<svg width="620" height="100">`
- `<text x="30" y="90" fill="#ED6E46" font-size="100" font-family="'Leckerli One', cursive">Watermelon</text>`
- `</svg>`



# SVG Shapes



- SVG has some predefined shape elements that can be used by developers:
  - Rectangle <rect height, width, x, y, rx, ry, fill>
  - Circle <circle cx, cy, r, fill>
  - Ellipse <ellipse cx, cy, rx, ry>
  - Line <line x1, y1, x2, y2, stroke, stroke-width>
  - Polyline <polyline points, stroke, stroke-width>
  - Polygon <polygon points, fill>
  - Path <path d="M, L, H, V, C, Q, >



# SVG Stroke Properties

- Stroke= color
- stroke-width: length
- stroke-linecap: butt, round, square, inherit
- stroke-dasharray: value
- Stroke- dashoffset: value
- Stroke – opacity: value

# Example:

- `<svg>`
- `<circle cx="75" cy="75" r="75" fill="#ED6E46" />`
- `</svg>`
  
- `<svg>`
- `<rect width="200" height="100" fill="#BBC42A" />`
- `</svg>`





# SVG Filter Elements

- `<feBlend>` - filter for combining images
- `<feColorMatrix>` - filter for color transforms
- `<feComponentTransfer>` - color component wise re-mapping
- `<feComposite>` - combination of two input images
- `<feConvolveMatrix>`
- `<feDiffuseLighting>`
- `<feDisplacementMap>`
- `<feFlood>`
- `<feGaussianBlur>`



# Contd.,

- `<feImage>`
- `<feMerge>`
- `<feMorphology>`
- `<feOffset>` - filter for drop shadows
- `<feSpecularLighting>`
- `<feTile>`
- `<feTurbulence>`
- `<feDistantLight>` - filter for lighting
- `<fePointLight>` - filter for lighting
- `<feSpotLight>` - filter for lighting



# SVG Blur, Drop Shadows, Gradients

- SVG Blur Effects
  - <defs> and <filter>
- SVG Drop Shadows
  - <defs> and <filter>
- SVG Gradients
  - There are two main types of gradients in SVG:
  - Linear -<linearGradient>
  - Radial - <radialGradient>

# Example:

- `<svg width="405" height="105">`
- `<defs>`
- `<linearGradient id="Gradient1" x1="0" y1="0" x2="100%" y2="0">`
- `<stop offset="0%" stop-color="#BBC42A" />`
- `<stop offset="100%" stop-color="#ED6E46" />`
- `</linearGradient>`
- `</defs>`
- `<rect x="2" y="2" width="400" height="100" fill="url(#Gradient1)" stroke="#333333" stroke-width="4px" />`
- `</svg>`

# Comparison

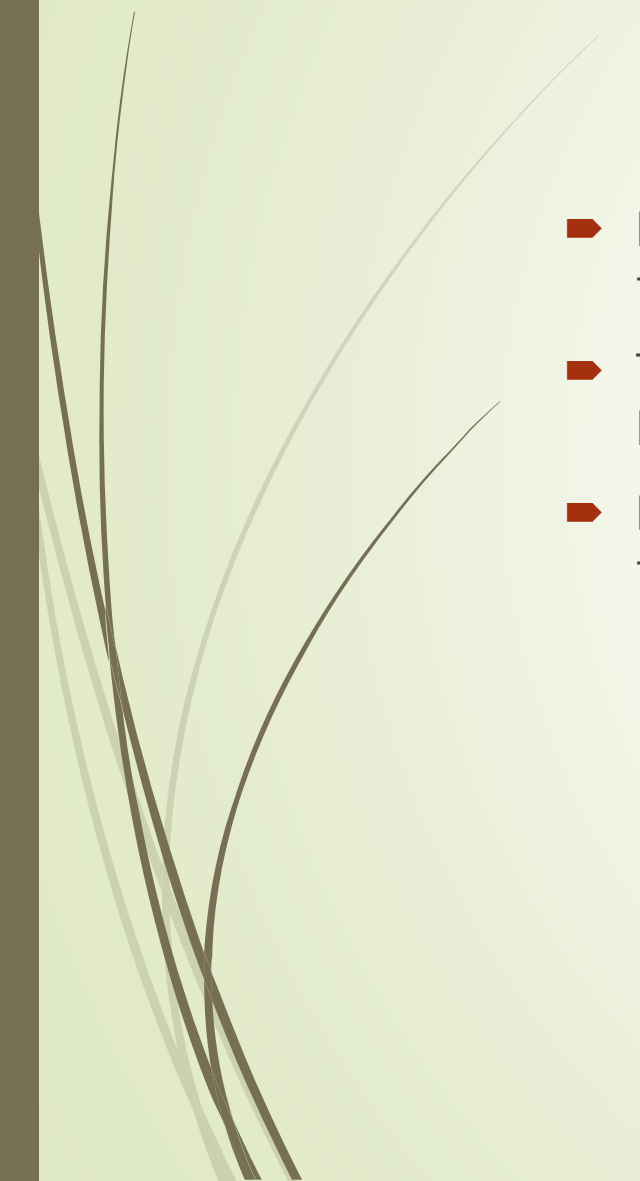
SVG	Canvas
Vector based (composed of shapes)	Raster based (composed of pixel)
Multiple graphical elements, which become the part of the DOM	Single HTML element similar to <code>&lt;img&gt;</code> in behavior
Modified through script and CSS	Modified through script only
Give better performance with smaller number of objects or larger surface, or both	Give better performance with smaller surface or larger number of objects, or both
Better scalability — can be printed with high quality at any resolution	Poor scalability — not suitable for printing on higher resolution





# **DRAG AND DROP**



# Drag and Drop

- HTML Drag and Drop interfaces enable applications to use drag and drop features in browsers.
  - The HTML drag and drop interfaces are DragEvent, DataTransfer, DataTransferItem and DataTransferItemList.
  - HTML drag and drop uses the DOM event model and drag events inherit from mouse events.
- 



Event	On Event Handler	Description
<u><a href="#">dragstart</a></u>	<u><a href="#">ondragstart</a></u>	Fired when the user starts dragging an element or text selection.
<u><a href="#">drag</a></u>	<u><a href="#">ondrag</a></u>	Fired when an element or text selection is being dragged.
<u><a href="#">dragenter</a></u>	<u><a href="#">ondragenter</a></u>	Fired when a dragged element or text selection enters a valid drop target.
<u><a href="#">dragover</a></u>	<u><a href="#">ondragover</a></u>	Fired when an element or text selection is being dragged over a valid drop target (every few hundred milliseconds).
<u><a href="#">dragleave</a></u>	<u><a href="#">ondragleave</a></u>	Fired when a dragged element or text selection leaves a valid drop target.
<u><a href="#">dragend</a></u>	<u><a href="#">ondragend</a></u>	Fired when a drag operation is being ended (for example, by releasing a mouse button or hitting the escape key).
<u><a href="#">drop</a></u>	<u><a href="#">ondrop</a></u>	Fired when an element or text selection is dropped on a valid drop target.





# Drag Event



- The DragEvent interface is a DOM event that represents a drag and drop interaction.
- Properties:
  - `DragEvent.dataTransfer`
- Methods:
  - `DragEvent()`



# DataTransfer

- The DataTransfer object is used to hold the data that is being dragged during a drag and drop operation.
- **Properties**
  - DataTransfer.dropEffect
  - DataTransfer.effectAllowed
  - DataTransfer.files
  - DataTransfer.items
  - DataTransfer.types



# Contd.,

## ➤ **Methods**

- `DataTransfer.clearData()`
- `DataTransfer.getData()`
- `DataTransfer.setData()`
- `DataTransfer.setDragImage()`
- `DataTransfer.addElement()`



# DataTransferItem


- The DataTransferItem object represents one drag data item.
- **Properties:**
  - DataTransferItem.kind (String / file)
  - DataTransferItem.type (MIME)
- **Methods:**
  - DataTransferItem.getAsFile()
  - DataTransferItem.getAsString()



# **GEO LOCATION**




# Geolocation API


- The Geolocation API is used to determine the location information associated with the hosting device.
  - User is asked for the permission to use the location information.
- 



# Navigator.geolocation

```
partial interface Navigator {  
  readonly attribute Geolocation geolocation;  
};
```



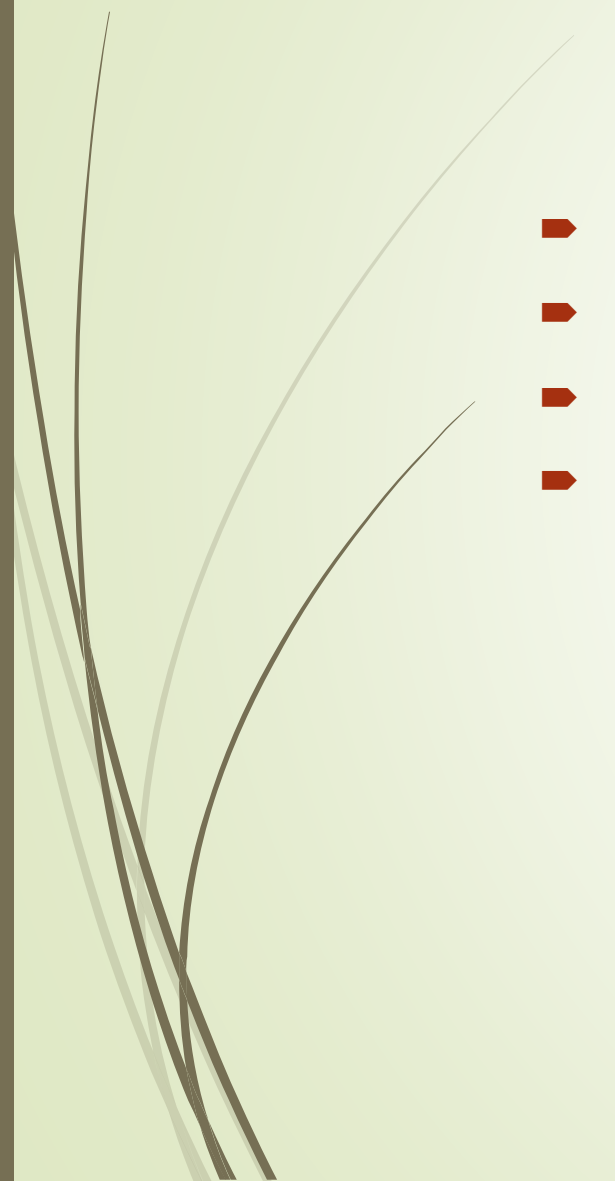


```
interface Geolocation {  
    void getCurrentPosition(PositionCallback successCallback,  
        optional PositionErrorCallback errorCallback,  
        optional PositionOptions options);  
  
    long watchPosition(PositionCallback successCallback,  
        optional PositionErrorCallback errorCallback,  
        optional PositionOptions options);  
  
    void clearWatch(long watchId);  
};  
  
callback PositionCallback = void (Position position);  
  
callback PositionErrorCallback = void (PositionError positionError);
```





# Position interface



- interface Position {
- readonly attribute Coordinates coords;
- readonly attribute DOMTimeStamp timestamp;
- };



# Coordinates interface

```
interface Coordinates {  
    readonly attribute double latitude;  
    readonly attribute double longitude;  
    readonly attribute double? altitude;  
    readonly attribute double accuracy;  
    readonly attribute double? altitudeAccuracy;  
    readonly attribute double? heading;  
    readonly attribute double? speed;  
};
```



# PositionError interface

```
interface PositionError {  
    const unsigned short PERMISSION_DENIED = 1;  
    const unsigned short POSITION_UNAVAILABLE = 2;  
    const unsigned short TIMEOUT = 3;  
    readonly attribute unsigned short code;  
    readonly attribute DOMString message;  
};
```



# PositionOptions interface

```
dictionary PositionOptions {  
    boolean enableHighAccuracy = false;  
    unsigned long timeout = 0xFFFFFFFF;  
    unsigned long maximumAge = 0;  
};
```



# "one-shot" position request.

```
function showMap(position) {  
    // Show a map centered at (position.coords.latitude,  
    position.coords.longitude).  
}  
  
// One-shot position request.  
navigator.geolocation.getCurrentPosition(showMap);
```



# requesting repeated position updates.

```
function scrollMap(position) {  
    // Scrolls the map so that it is centered at (position.coords.latitude,  
    position.coords.longitude).  
}  
  
// Request repeated updates.  
var watchId = navigator.geolocation.watchPosition(scrollMap);  
  
function buttonClickHandler() {  
    // Cancel the updates when the user clicks a button.  
    navigator.geolocation.clearWatch(watchId);  
}
```



## requesting repeated position updates and handling errors

```
function scrollMap(position) {  
    // Scrolls the map so that it is centered at (position.coords.latitude,  
    position.coords.longitude).  
}  
  
function handleError(error) {  
    // Update a div element with error.message.  
}  
  
// Request repeated updates.  
var watchId = navigator.geolocation.watchPosition(scrollMap, handleError);  
  
function buttonClickHandler() {  
    // Cancel the updates when the user clicks a button.  
    navigator.geolocation.clearWatch(watchId);  
}
```



# Googlemaps API







# **WEB STORAGE**



# HTML5 Web Storage

- HTML5 allows web applications to store data in the browser.
- **Cookies** are also used to store data, but there are few drawbacks
  - storage limit
  - slow's down the web application performance.



# What is web storage?

- ▶ Client side database that allows users to store data in the form of key/value pairs.
  - ▶ It can store up to 10MB of data per domain.
  - ▶ Simple API to retrieve/write data into the local storage.



# Types of Web Storage

- **Local storage:**
  - Stores data with no expiration date
- **Session storage:**
  - Stores data for one session



# To save data into local storage

```
function setSettings() {  
    if ('localStorage' in window && window['localStorage'] !== null) {  
        //use localStorage object to store data  
    } else {  
        alert('Cannot store user preferences as your browser do not support  
local storage');  
    }  
}
```



## Contd.,

- The `setItem('key','value')` allows us to write the data into the local storage.
- `QUOTA_EXCEEDED_ERR` exception will be thrown if the storage limit exceeds 5MB.
- The `getItem('Key')` helps in retrieving the data stored in the database.
- The `length` function retrieves the total number of values in the storage area.
- The local storage area can be cleared by using the `clear()` function or `removeItem('key')` function.



# **WEB WORKERS**



# Web Workers

- **Web Workers provide a simple means for web content to run scripts in background threads.**
- 





# Web Workers API



- A worker is an object created using a constructor that runs a named JavaScript file.
  - Worker thread
    - Shared workers
    - Dedicated workers
  - Global context
- `postMessage()`



# Terminating a worker

- `//terminating from main thread`
- `myWorker.terminate();`
  
- `//terminating from worker`
- `close();`