

HTML 5

Banu Prakash

banuprakashc@yahoo.co.in

Introduction to HTML5

- HTML5 is the latest iteration of HTML addressing modern needs and expectations of websites.
- It deals with things like semantic markup, providing information about content it describes.
- It is becoming new standard for all good web developers.
- It works on desktops, tablets, and mobile devices alike.

Introduction to HTML5

- W3C announced in July 2009 that HTML5 will be the official recommendation of web standard!
- Today, most of the major web browsers are aggressively wanting to support HTML5.
 - Even Microsoft Internet Explorer, which has the reputation of slow in adopting new standard is supporting HTML5 in its IE9.



Timeline of Web Technologies



HTML5 \sim = HTML + CSS + JS

Why HTML5?

- **It's the Future**

- The number one reason why you should start using HTML5 today is this: it's the future

- **Mobile**

- Mobile browsers have fully adopted HTML5 so creating mobile ready projects is as easy as designing and constructing for their smaller touch screen displays, hence the popularity of Responsive Design.
- There are some great meta tags that also allow you to optimize for mobile:
- Viewport: allows you to define viewport widths and zoom settings
- Full screen browsing: IOS specific values that allow Apple devices to display in full screen mode
- Home Screen Icons: like favicons on desktop, these icons are used to add favourites to the home screen of an IOS and Android mobile device

Why HTML5?

- **Legacy/Cross Browser Support**

- Fortunately, HTML5 is being built to make things easier and more cross browser friendly so in those older IE browsers that don't like the new tags we can just simply add a JavaScript shiv that will allow them to use the new elements:

```
<!--[if lt IE 9]>
```

```
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
```

```
<![endif]-->
```

Why HTML5?

- **Better Interactions**

- HTML5 also comes with great APIs that allow you to build a better user experience and a more dynamic web application :
 - Drag and Drop (DnD)
 - Offline storage database
 - Browser history management
 - Timed media playback
 - Canvas

Why HTML5?

- **Video and Audio Support**

- Forget about Flash Player and other third party media players, make your videos and audio truly accessible with the new HTML5 `<video>` and `<audio>` tags.
- HTML5's video and audio tags basically treat them as images;
- `<video src="url" width="640px" height="380px" autoplay/>`.

- **Accessibility**

- HTML5 makes creating accessible sites easier for two main reasons: semantics and ARIA. The new HTML headings like `<header>`, `<footer>`, `<nav>`, `<section>`, `<aside>`, etc. allow screen readers to easily access content.
- Before, your screen readers had no way to determine what a given `<div>` was even if you assigned it an ID or Class. With new semantic tags screen readers can better examine the HTML document and create a better experience for those who use them.
- ARIA (**Accessible Rich Internet Applications Suite**) is a W3C spec that is mainly used to assign specific “roles” to elements in an HTML document – essentially creating important landmarks on the page: header, footer, navigation or article, via role attributes.

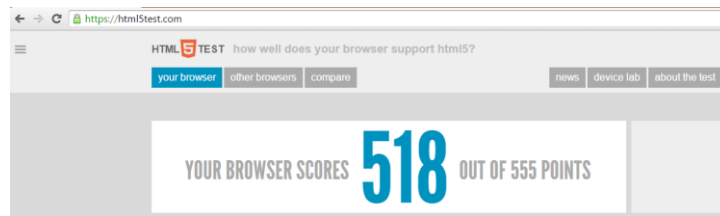
HTML5 in action

- **The DocType**

- Document Type Declaration in HTML5 is cool and simple. It is as simple as below:
 - `<!DOCTYPE html>`
- Unlike of different variant HTML4 or XHTML1 doctype that we are familiar with:-
- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`
- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`
- `<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">`
- `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
- `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`

HTML 5 Browser Support

- Using [<https://html5test.com/>] to test your Browser support
- Chrome



OVERVIEW

	Chrome	Firefox	Internet Explorer	Opera	Safari
Upcoming			TP 343		
Current	39 501	35 449	11 336	26 497	8.0 396

HTML 5 Browser Support

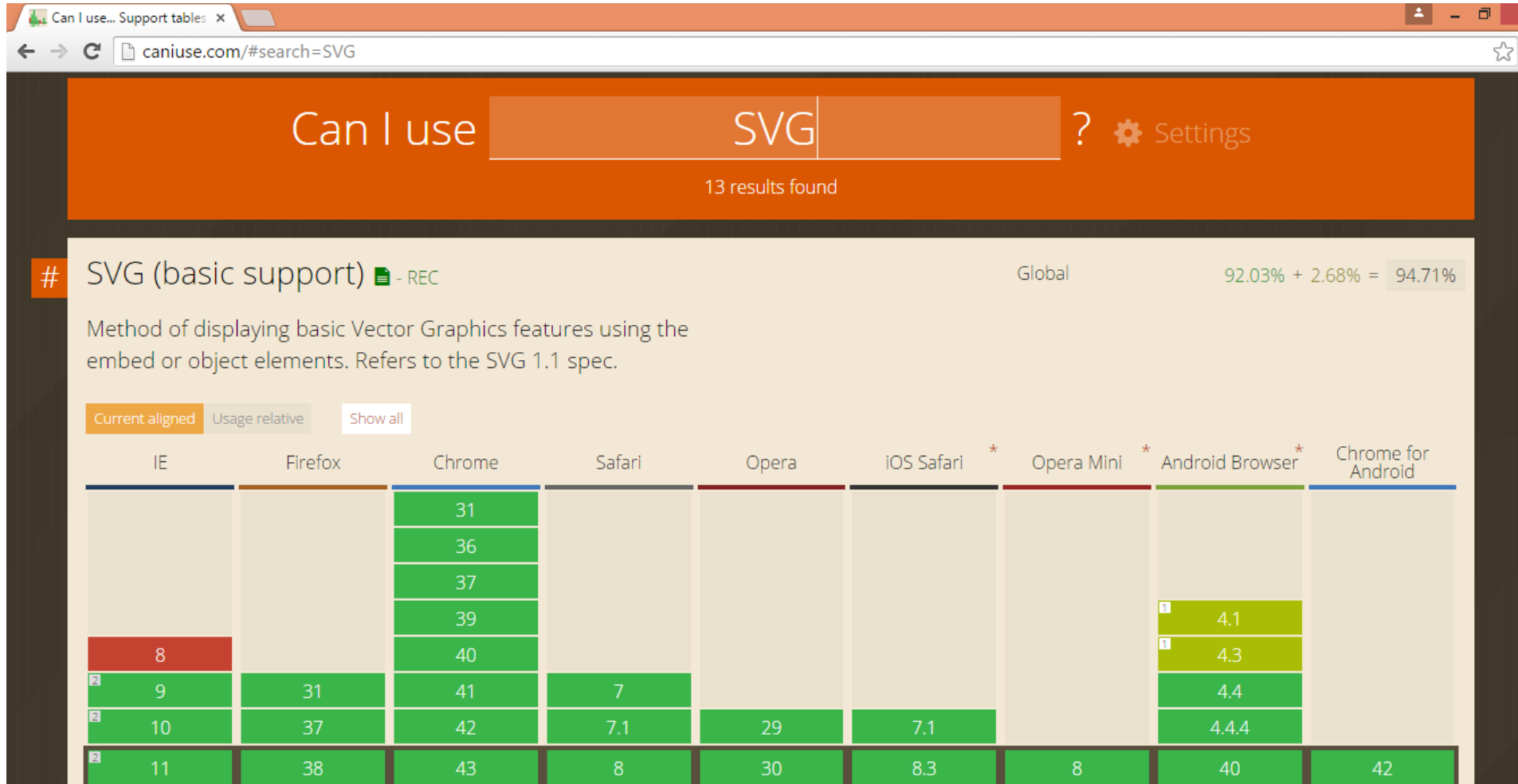
- Mobiles

Android	BlackBerry	Chrome	Firefox	iOS	Opera	Windows Phone
5.0 452	10.3 449	39 493	35 456	8.0 405	26 489	8.1 346

- Tablets

Android	BlackBerry	Chrome	Firefox	Internet Explorer	iOS	Opera
5.0 452	2.1 354	39 493	35 456	11 336	8.0 405	26 489

HTML 5 Browser Support



The screenshot shows the Can I use website interface for the search term "SVG". The header bar is orange and contains the text "Can I use SVG" and "13 results found". Below the header, the main content area has a light beige background. It features a section titled "# SVG (basic support) - REC" with a "Global" status and a usage percentage of "92.03% + 2.68% = 94.71%". A description states: "Method of displaying basic Vector Graphics features using the embed or object elements. Refers to the SVG 1.1 spec." Below the description are three tabs: "Current aligned" (selected), "Usage relative", and "Show all". The main table lists browser support for various versions of IE, Firefox, Chrome, Safari, Opera, iOS Safari, Opera Mini, Android Browser, and Chrome for Android. The table uses color coding: green for supported versions, red for unsupported versions, and yellow for versions with partial support. The table is structured as follows:

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		39						
		40						
8		41	7				4.1	
9	31	42	7.1	29	7.1		4.3	
10	37						4.4	
11	38	43	8	30	8.3	8	4.4.4	42

Backwards Compatibility for Older Browsers

- Shiv

- In old browsers HTML5 doesn't display correctly.
- This is where the HTML5Shiv comes in.
- It allows most of the old browsers to recognize the HTML5 tags and style them using CSS instead of HTML5.
- Therefore, consider include the shiv in every page so that anyone using an older browser can still see your website the way you intended.

```
<head>

  <!--[if lt IE 9]>
  <script
    src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
  </script>
  <![endif]-->

  <title>Title</title>
</head>
```

Shim

- In computer programming, a shim is a small library that transparently intercepts API calls and changes the arguments passed, handles the operation itself, or redirects the operation elsewhere.
- Shims typically come about when the behaviour of an API changes, thereby causing compatibility issues for older applications which still rely on the older functionality.
- In such cases, the older API can still be supported by a thin compatibility layer on top of the newer code. Web polyfills are a related concept.

Modernizr

- Modernizr is a small JavaScript library that detects the availability of native implementations for next-generation web technologies, i.e. features that stem from the HTML5 and CSS3 specifications.
- Many of these features are already implemented in at least one major browser (most of them in two or more), and what Modernizr does is, very simply, tell you whether the current browser has this feature natively implemented or not.

Modernizr

- Include Modernizr
 - `<script src="js/modernizr.js"></script> </head>`
- `<html class="no-js">`
 - Modernizr depends on JavaScript being enabled in the browser.
 - However, in the rare cases when JavaScript is disabled, it remains in the HTML markup, allowing you to create special style rules for such visitors if necessary.
 - the no-js class has been replaced by js , indicating that JavaScript is enabled.

```
<!DOCTYPE HTML><html class=" js flexbox canvas canvastext no-webgl  
no-touch no-geolocation no-postmessage no-websqlatabase no-indexeddb  
no-hashchange no-history draganddrop no-websockets rgba hsla multiplebgs  
backgroundsize borderimage borderradius boxshadow textshadow opacity  
cssanimations csscolumns cssgradients cssreflections csstransforms  
no-csstransforms3d csstransitions fontface generatedcontent video audio  
no-localstorage no-sessionstorage no-webworkers no-applicationcache svg  
no-inlinesvg smil svgclippaths"><head>  
<meta charset="utf-8">
```

banuprakashc@yahoo.co.in

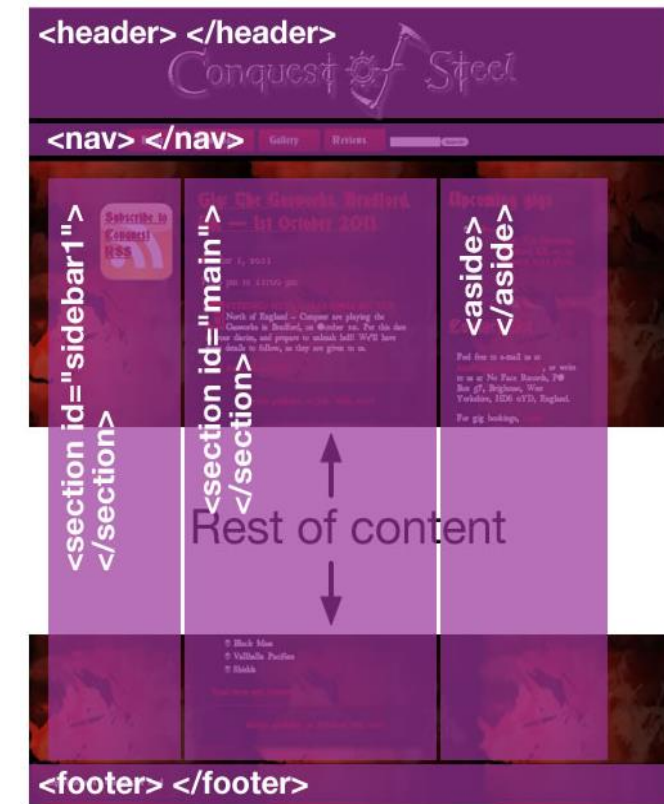
Modernizr

- Conditional code using Modernizr

```
if (Modernizr.localstorage) {  
    // script to run if local storage is  
    // supported  
} else {  
    // script to run if  
    // local storage is not supported  
}
```

HTML5 structural elements

- **<header>** :Used to contain the header content of a site
- **<footer>**: Contains the footer content of a site
- **<nav>**: Contains the navigation menu, or other navigation functionality for the page
- **<article>**: Contains a standalone piece of content that would make sense if syndicated as an RSS item, for example a news item.
- **<section>**: Used to either group different articles into different purposes or subjects
- **<aside>**: Defines a block of content that is related to the main content around it, but not central to the flow of it



```

<body>

  <header>
    <!-- header content goes in here -->
  </header>

  <nav>
    <!-- navigation menu goes in here -->
  </nav>

  <section id="sidebar1">
    <!-- sidebar content goes in here -->
  </section>

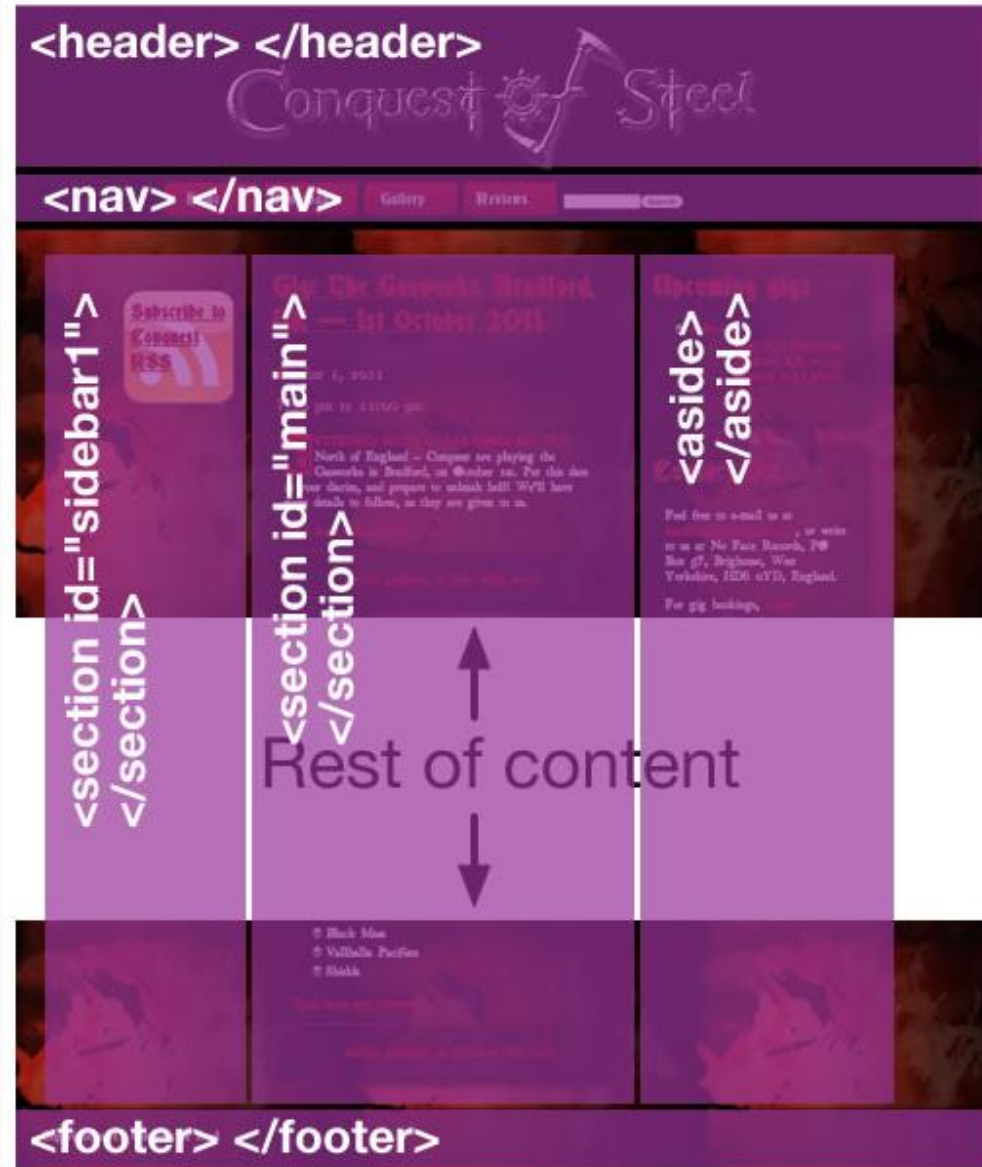
  <section id="main">
    <!-- main page content goes in here -->
  </section>

  <aside>
    <!-- aside content goes in here -->
  </aside>

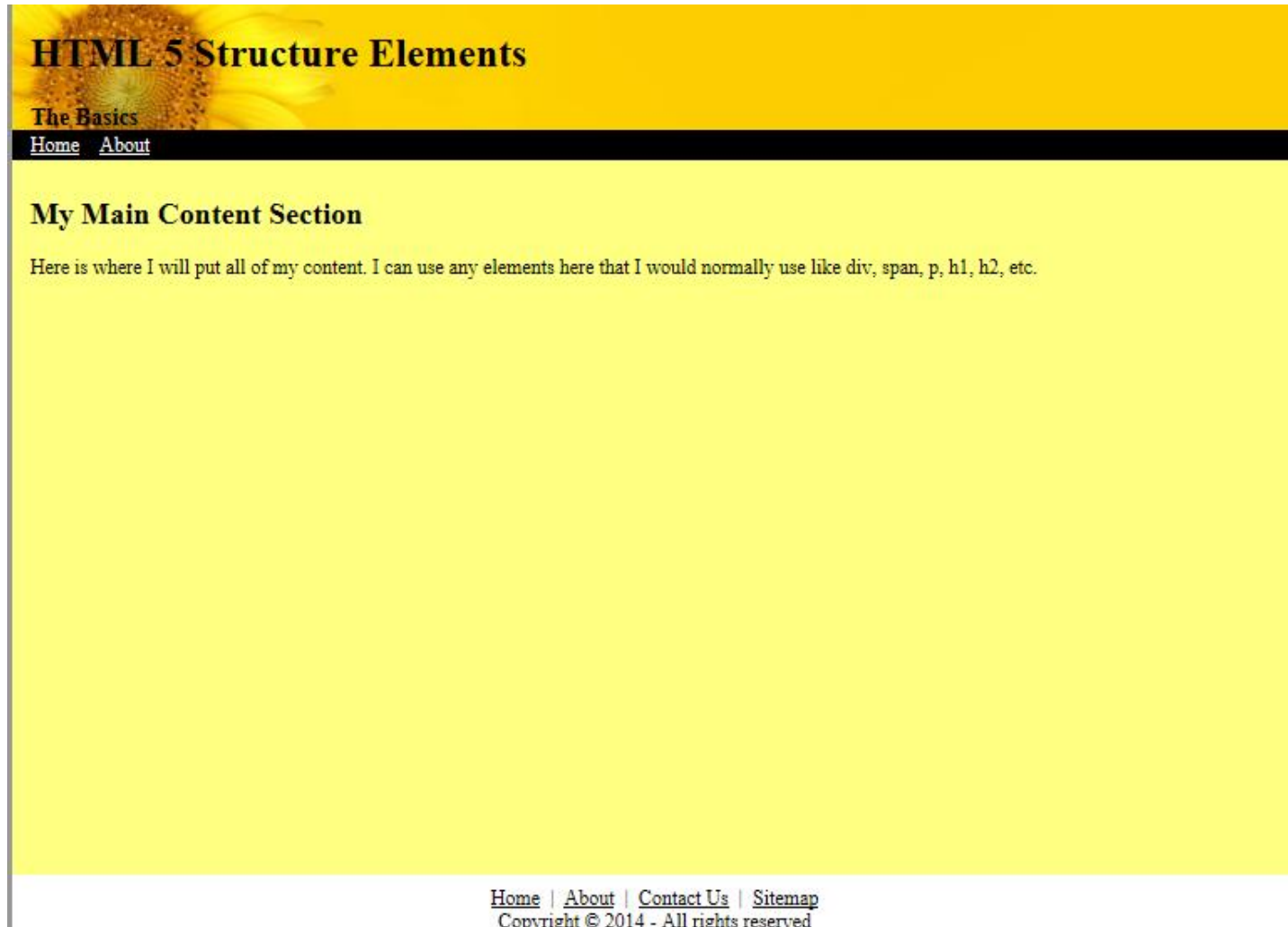
  <footer>
    <!-- footer content goes in here -->
  </footer>

</body>

```



Page built using HTML5 structural elements



HTML5 forms

- **Apart from HTML 4 form elements HTML5 has many new form elements**

[datetime-local](#)

A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601, with no time zone information.

[number](#)

This accepts only numerical value. The step attribute specifies the precision, defaulting to 1.

[range](#)

The range type is used for input fields that should contain a value from a range of numbers.

[email](#)

This accepts only email value. This type is used for input fields that should contain an e-mail address. If you try to submit a simple text, it forces to enter only email address in email@example.com format.

HTML5 form attributes

- **The placeholder attribute**

- HTML5 introduced a new attribute called placeholder.
 - This attribute on <input> and <textarea> elements provides a hint to the user of what can be entered in the field.
 - The placeholder text must not contain carriage returns or line-feeds.
- `<input type="text" name="search" placeholder="search the web"/>`

- **The required attribute**

- Now you do not need to have javascript for client side validations like empty text box would never be submitted because HTML5 introduced a new attribute “required” which would be used as follows and would insist to have a value:
 - `<input type="text" name="search" required/>`

HTML5 form attributes

- Min and Max values

```
<p>
Age
</p>
<p>
    <input type="number" min="18" max="100"/>
</p>
<p>
Date Of Birth
</p>
<p>
    <input type="datetime-local" name="db" min="1-1-1000" max="1-1-2000" />
</p>
```

Create Custom HTML5 Form Error Messages

- Using Title Attribute

```
<form>
  <label for=name>Your name</label>
  <input type="text"
    id="name"
    pattern="^([\u00c0-\u01ffa-zA-Z'\-]){2,50}$"
    title="Please enter only letters (hyphens and spaces may be included)"
    required>
  <input type="submit" value="Test">
</form>
```

Your name

Test



Please fill in this field.

Please enter only letters (hyphens and spaces may be included)

HTML 5 Form Polyfill

- HTML5Forms.js is a JavaScript polyfill that implements a subset of the HTML5
- Forms module in all browsers. The script will only add support for the different parts of the module when there doesn't exist a native implementation.
- HTML5Forms supports the following HTML5 input types:
 - range
 - date
 - datetime
 - datetime-local
 - week
 - color

HTML 5 Polyfill

- It also supports:
 - form validation (via "required" and "pattern" attributes)
 - the autofocus attribute (i.e. focusing on a particular form element onload)
 - the placeholder attribute (i.e. descriptive text of what should be in a form field)
 - the output tag (solves equations of form elements)
 - CSS styling of form validation states (simulates :invalid and :valid in unsupported browsers like IE9 and lower)
 - CSS styling of form elements that are not included in the CSS3 UI specification, but I think are useful for developers:
 - - .wf2_isBlank, .wf2_notBlank – these classes are applied to form field when a form element is blank/not blank respectively.
 - - .wf2_lostFocus -this class is applied to a form element when a form field loses focus.
 - - .wf2_submitAttempted – this class is applied to a <form> tag when a form submission is attempted.

Browsers don't style them in exactly the same way

	Windows	Mac	Linux
Firefox 4.0+ (native support)			
Safari 5.x- (polyfill)			Not Applicable
Chrome 17+ (native support)			
Opera (native support)			
IE8 (polyfill)			

Cross Browser Styling of HTML5 Forms — Even In Older Browsers

- You will need to download the HTML5Forms.js JavaScript library to add support for older browsers that don't support HTML5 Forms natively.
- data-webforms2-support is set to "validation" which tells html5Forms to load webforms2 polyfill to add support for older browsers, which is included with HTML5Forms.js

```
<script type="text/javascript"
    src="/path/to/shared/js/modernizr.com/Modernizr-2.5.3.forms.js">
</script>

<script type="text/javascript"
    data-webforms2-support="validation"
    src="/path/to/html5Forms/shared/js/html5Forms.js" >

</script>
```

Cross Browser Styling of HTML5 Forms — Even In Older Browsers

<head>

```
<link rel="stylesheet" type="text/css" href="css/formstyle.css">  
<script type="text/javascript" src="js/Modernizr-2.5.3.forms.js">  
</script>
```

```
<script data-webforms2-force-js-validation="true"  
      data-webforms2-support="validation"  
      src="js/html5Forms.js"  
      type="text/javascript">  
</script>
```

</head>

<body>

```
<form data-webforms2-force-js-validation="true">
```

Customizing the Error Messages

- Note that the default error messages may vary from browser to browser:
- Firefox has an attribute, x-moz-errormessage which can be used to set custom error messages easily.
- Using polyfill HTML5Forms.js for custom error – messages [data-errormessage]

```
<input type="text" name="fullName" value="" required="required" placeholder="Required information" data-errormessage="You forgot to fill out this field!"/>
```

Custom CSS styling

```
/*  
 * STEP 1: change the bubble so background is white and text is green  
 */  
  
/* Webkit */  
input::-webkit-validation-bubble-message {  
    color: green !important;  
    background: white;  
    border: 1px solid black;  
    padding: 18px;  
  
}  
  
/* Polyfill styles to do the same as above */  
.wf2_errorMsg {  
    color: green !important;  
    background: white;  
  
}  
  
/* Removes the icon that appears in Webkit browsers by default */  
input::-webkit-validation-bubble-icon {  
    display: none;  
  
}
```

Custom CSS styling

```
/*  
 * Step 2: The arrow  
 */  
  
input::-webkit-validation-bubble-arrow {  
    background: white;  
    border: solid black 1px;  
}  
  
/* Polyfill */  
.wf2_errorMsg:before {  
    /* By default, the image is 29x15, but you can change it and position it with CSS */  
    background: url('whiteValidationPointer.png') !important;  
}
```


Full Name :


Company Name :

You forgot to fill out this field!

Using CSS to Show the Validity of Form Fields As The User Types

```
<style type="text/css">
  input:required, textarea:required{
    background:url("css/asterisk_orange.png") no-repeat right top;
  }
  /* Polyfill */
  input[required] {
    background:url("css/asterisk_orange.png") no-repeat right top;
  }
</style>
```


Full Name : 


Company Name : 

STEP 2: Show the user when a form field has valid data in it with a green checkmark.

```
/* Browsers that implement HTML5 Forms Natively */
input[required]:valid, textarea[required]:valid {
  /* Make this important if you want IE10 to work right. */
  background:url("css/tick.png") no-repeat right top !important;
}


/* Polyfill */
input[required].wf2_valid {
  background:url("css/tick.png") no-repeat right top;
}
```


Full Name : 

Company Name : 

STEP 3: Show the user when a form field has invalid data in it with a red 'X'

```
/* Browsers that implement HTML5 Forms Natively */  
input:focus:invalid, textarea:focus:invalid {  
    background:url("css/cancel.png") no-repeat right top;  
}  
  
/* Polyfill */  
input.wf2_invalid:focus {  
    background:url("css/cancel.png") no-repeat right top;  
}
```

Full Name : 

Company Name : 

Video

```
<video width="600px" height="340px" id="movie">  
    <source src="f:\Now.You.See.Me.mp4" type="video/mp4">  
</video>
```

HTML5 Canvas

- The HTML `<canvas>` element is used to draw graphics, on the fly, via scripting (usually JavaScript).
- The `<canvas>` element is only a container for graphics. You must use a script to actually draw the graphics.
- Canvas has several methods for drawing paths, boxes, circles, text, and adding images.

What is HTML Canvas?

- A canvas is a rectangular area on an HTML page. By default, a canvas has no border and no content.

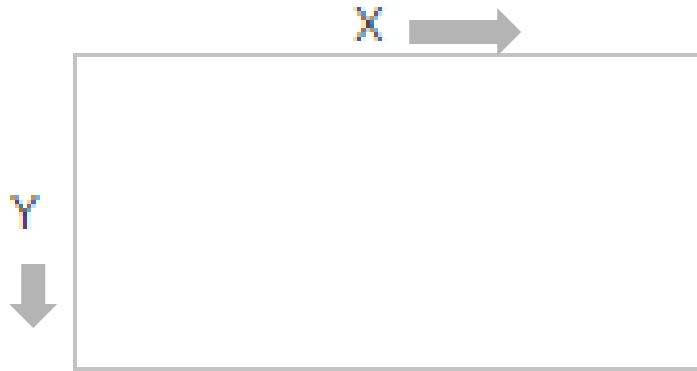
- The markup looks like this:

```
<canvas id="myCanvas" width="200" height="100">  
</canvas>
```

- Note: Always specify an id attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas.

HTML Canvas Coordinates

- Canvas Coordinates
 - The HTML canvas is a two-dimensional grid.
 - The upper-left corner of the canvas has the coordinates (0,0)



HTML Canvas Drawing

- Step 1: Find the Canvas Element
 - `document.getElementById("myCanvas");`
- Step 2: Create a Drawing Object : The `getContext()` is a built-in HTML object, with properties and methods for drawing:
 - `var ctx = canvas.getContext("2d");`
- Step 3: Draw on the Canvas.

```
<canvas id="myCanvas" width="200" height="100"
        style="border:1px solid #000000;">
</canvas>
<script>
    var canvas = document.getElementById("myCanvas");
    var ctx = canvas.getContext("2d");
    ctx.fillStyle = "#FF0000";
    ctx.fillRect(0,0,150,75);
</script>
```



HTML Canvas Drawing example

```
var canvas = document.getElementById('myCanvas');  
var ctx = canvas.getContext('2d');
```

```
ctx.fillStyle = "rgb(200,0,0)";  
ctx.fillRect (10, 10, 55, 50);
```



```
ctx.fillStyle = "rgba(0, 0, 200, 0.5)";  
ctx.fillRect (30, 30, 55, 50);
```

HTML Canvas Drawing

- Draw a Line

- moveTo(x,y) - defines the starting point of the line
- lineTo(x,y) - defines the ending point of the line

- Draw a Circle

- beginPath();
- arc(x, y, r, start, stop)
- Then use the stroke() method to actually draw the circle

```
var canvas = document.getElementById("myCanvas");  
var ctx = canvas.getContext("2d");  
ctx.beginPath();  
ctx.arc(95,50,40,0,2*Math.PI);  
ctx.stroke();
```

HTML Canvas Gradients

- Gradients can be used to fill rectangles, circles, lines, text, etc.
- There are two different types of gradients:
 - `createLinearGradient(x,y,x1,y1)` - creates a linear gradient
 - `createRadialGradient(x,y,r,x1,y1,r1)` - creates a radial/circular gradient
 - Once we have a gradient object, we must add two or more color stops.
 - The `addColorStop()` method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

```
var lingrad = canvasCtx.createLinearGradient(0,0,0,150);  
lingrad.addColorStop(0, 'red');  
lingrad.addColorStop(1, 'green');  
// assign gradients to fill and stroke styles  
canvasCtx.fillStyle = lingrad;  
// draw shapes  
canvasCtx.fillRect(10,10,130,130);
```



HTML Canvas Text

- Drawing Text on the Canvas
 - To draw text on a canvas, the most important property and methods are:
 - font - defines the font properties for the text
 - fillText(text, x, y) - draws "filled" text on the canvas
 - strokeText(text, x, y) - draws text on the canvas (no fill)

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.shadowOffsetX = 4;
ctx.shadowOffsetY = 4;
ctx.shadowBlur = 6;
ctx.shadowColor = 'rgba(0,0,255,0.5)';
ctx.font = "bold 40px Arial";
ctx.fillText("Banu Prakash", 300, 100);
```

Banu Prakash

Canvas - Images

- To draw an image on a canvas, use the following method:
 - `drawImage(image, x, y)`

```
var canvas = document.getElementById("myCanvas");  
var ctx = canvas.getContext("2d");  
var img = document.getElementById("logo");  
ctx.drawImage(img, 10, 10);
```

What is a Sprite Sheet?

- A sprite sheet consists of multiple frames in one image. The following sprite sheet has 10 frames. The width of the image is 460 pixels. Therefore the frame width is $460/10$ or 46 pixels.

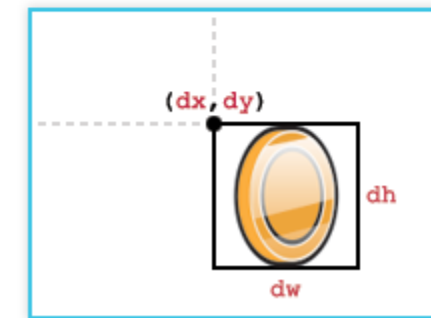
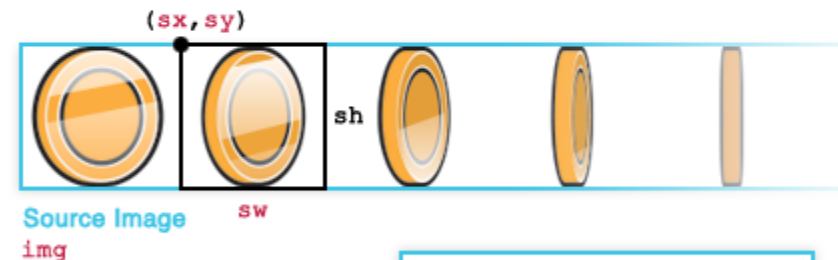


DrawImage Method

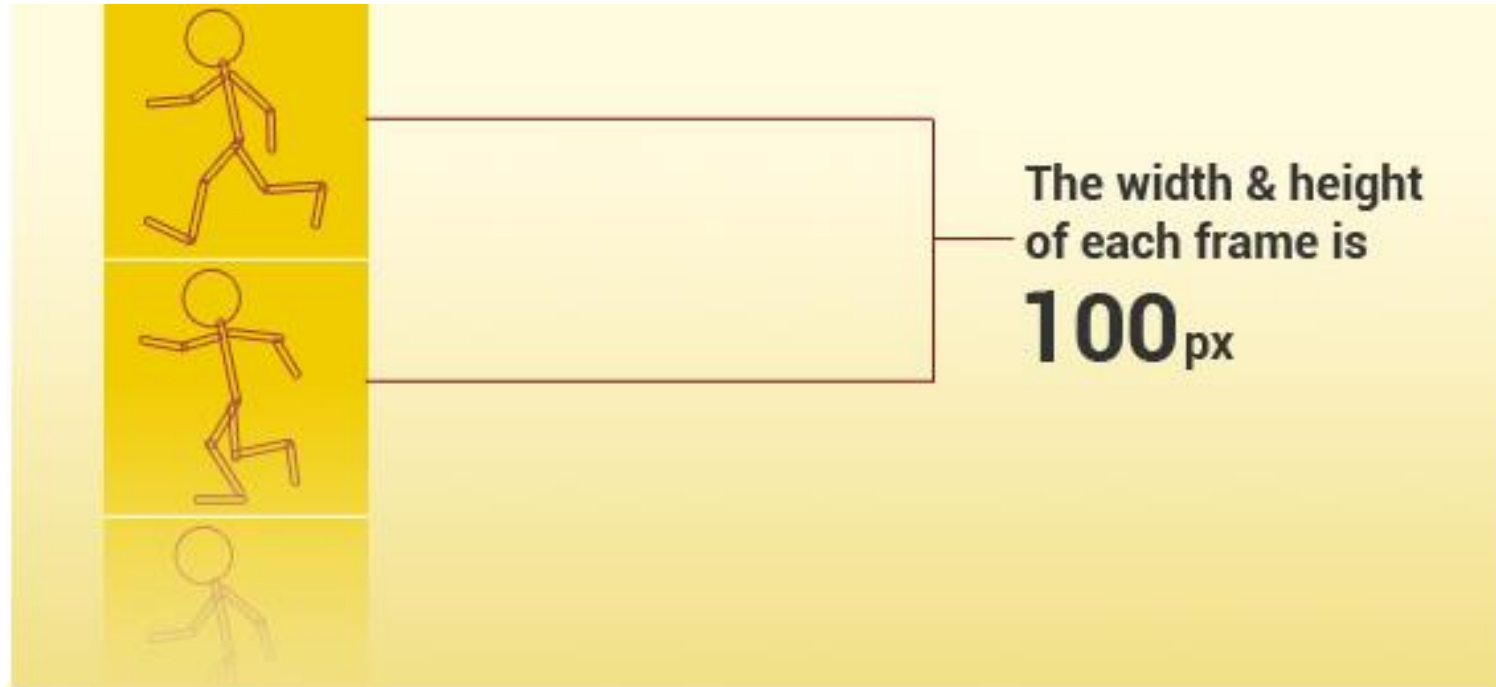
- The key to creating sprites from one image is that the context's drawImage method allows us to render a cropped section of the source image to the canvas.

```
context.drawImage(img, sx, sy, sw, sh, dx, dy, dw, dh)
```

<code>img</code>	Source image object	Sprite sheet
<code>sx</code>	Source x	Frame index times frame width
<code>sy</code>	Source y	0
<code>sw</code>	Source width	Frame width
<code>sh</code>	Source height	Frame height
<code>dx</code>	Destination x	0
<code>dy</code>	Destination y	0
<code>dw</code>	Destination width	Frame width
<code>dh</code>	Destination height	Frame height



Sprite Example



PNG sprite sequence

Sprite Example

```
<canvas id="myCanvas" width="100" height="100">
</canvas>
<script>
    var width = 100, height = 100, frames = 4,
        currentFrame = 0,
        canvas = document.getElementById("myCanvas");
    ctx = canvas.getContext("2d");
    image = new Image()
    image.src = 'images/sprite.png';
    var draw = function() {
        ctx.clearRect(0, 0, width, height);
        ctx.drawImage(image, 0, height * currentFrame, width, height, 0, 0,
            width, height);

        if (currentFrame == frames) {
            currentFrame = 0;
        } else {
            currentFrame++;
        }
    }
    setInterval(draw, 100);
</script>
```

Canvas Cheat sheet

HTML5 CANVAS ELEMENT

Html5 canvas element

```
<canvas id="myCanvas" width="500" height="300">
```

Html5 canvas element with fallback content

```
<canvas id="myCanvas" width="500" height="300">  
  your browser doesn't support canvas!  
</canvas>
```

2d context

```
var context = canvas.getContext('2d');
```

Webgl context (3d)

```
var context = canvas.getContext('webgl');
```

Canvas Cheat sheet

COLOR FORMATS

String

```
context.fillStyle = 'red';
```

Hex Long

```
context.fillStyle = '#ff0000';
```

Hex Short

```
context.fillStyle = '#f00';
```

RGB

```
context.fillStyle = 'rgb(255,0,0)';
```

RGBA

```
context.fillStyle = 'rgba(255,0,0,1)';
```

Canvas Cheat sheet

SHAPES

Draw rectangle

```
context.rect(x, y, width, height);  
context.fill();  
context.stroke();
```

Fill rectangle shorthand

```
context.fillRect(x, y, width, height);
```

Stroke rectangle shorthand

```
context.strokeRect(x, y, width, height);
```

Draw circle

```
context.arc(x, y, radius, 0, Math.PI * 2);  
context.fill();  
context.stroke();
```

Canvas Cheat sheet

PATHS

Begin Path

```
context.beginPath();
```

Line

```
context.lineTo(x, y);
```

Arc

```
context.arc(x, y, radius, startAngle, endAngle,  
counterClockwise);
```

Quadratic curve

```
context.quadraticCurveTo(cx, cy, x, y);
```

Bezier curve

```
context.bezierCurveTo(cx1, cy1, cx2, cy2, x, y);
```

Close Path

```
context.closePath();
```

Canvas Cheat sheet

TEXT

Fill Text

```
context.font = '40px Arial';  
context.fillStyle = 'red';  
context.fillText('Hello World!', x, y);
```

Stroke Text

```
context.font = '40pt Arial';  
context.strokeStyle = 'red';  
context.strokeText('Hello World!', x, y);
```

Bold Text

```
context.font = 'bold 40px Arial';
```

Italic Text

```
context.font = 'italic 40px Arial';
```

Text Align

```
context.textAlign = 'start|end|left|center|right';
```

Canvas Cheat sheet

STYLES

Fill

```
context.fillStyle = 'red';  
context.fill();
```

Stroke

```
context.strokeStyle = 'red';  
context.stroke();
```

Linear gradient

```
var grd = context.createLinearGradient(x1, y1, x2,  
y2);  
grd.addColorStop(0, 'red');  
grd.addColorStop(1, 'blue');  
context.fillStyle = grd;  
context.fill();
```

Radial gradient

```
var grd = context.createRadialGradient(x1, y1,  
radius1, x2, y2, radius2);  
grd.addColorStop(0, 'red');  
grd.addColorStop(1, 'blue');  
context.fillStyle = grd;  
context.fill();
```

TRANSFORMS

Translate

```
context.translate(x, y);
```

Scale

```
context.scale(x, y);
```

Rotate

```
context.rotate(radians);
```

Flip Horizontally

```
context.scale(-1, 1);
```

Flip Vertically

```
context.scale(1, -1);
```

Custom Transform

```
context.transform(a, b, c, d, e, f);
```

Set Transform

```
context.setTransform(a, b, c, d, e, f);
```

Shear

```
context.transform(1, sy, sx, 1, 0, 0);
```

Reset

```
context.setTransform(1, 0, 0, 1, 0, 0);
```

SVG (Scalable Vector Graphics)

- SVG stands for Scalable Vector Graphics
- SVG is used to define graphics for the Web
- SVG is a W3C recommendation
- The HTML `<svg>` Element
 - The HTML `<svg>` element (introduced in HTML5) is a container for SVG graphics.
 - SVG has several methods for drawing paths, boxes, circles, text, and graphic images

SVG

- Circle

```
<svg width="100" height="100">  
  <circle cx="50" cy="50" r="40"  
    stroke="green" stroke-width="4"  
    fill="yellow" />  
</svg>
```



- Rectangle

```
<svg width="200" height="200">  
  <rect  
    x="0" y="0"  
    width="100" height="100"  
    fill="blue" stroke="red"  
    stroke-width="5px"  
    rx="8" ry="8"  
    id="myRect" class="chart" />  
</svg>
```



SVG

- Rounded Rectangle

```
<svg width="400" height="180">  
  <rect x="50" y="20" rx="20" ry="20"  
    width="150" height="150"  
    style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />  
</svg>
```



SVG- transform

```
<svg>  
  <rect x="25" y="50" width="150" height="100"  
    fill="tomato"  
    stroke="lightgreen"  
    stroke-width="2"  
    transform="rotate(-45 100 100)" />  
</svg>
```



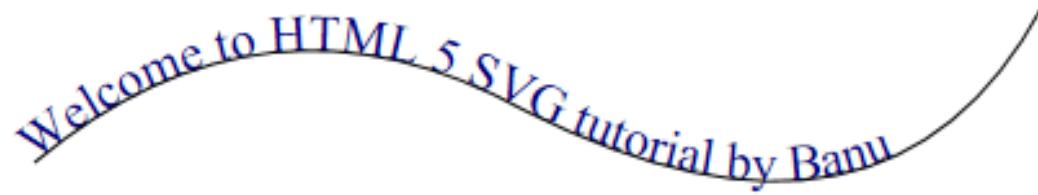
SVG Path - <path>

The <path> element is used to define a path.

The following commands are available for path data:

- M = moveto
- L = lineto
- H = horizontal lineto
- V = vertical lineto
- C = curveto
- S = smooth curveto
- Q = quadratic Bézier curve
- T = smooth quadratic Bézier curveto
- A = elliptical Arc
- Z = closepath

SVG Path - <path>



```
<svg>
<defs>
  <!-- Start at (10,90) end at (200,70) use control point (100,15) -->
  <!-- Continue from (200,70) end at (400,30) use control point (340,140) -->
  <path id = "s3" d = "M 10,90 Q 100,15 200,70 Q 340,140 400,30"/>
</defs>
<g fill = "navy">
  <text font-size = "20">
    <textPath xlink:href = "#s3">
      Welcome to HTML 5 SVG tutorial by Banu
    </textPath>
  </text>
  <use x = "0" y = "0" xlink:href = "#s3" stroke = "black" fill = "none"/>
</g>
</svg>
```

Comparison of Canvas and SVG

Canvas

- Resolution dependent
- No support for event handlers
- Poor text rendering capabilities
- You can save the resulting image as .png or .jpg
- Well suited for graphic-intensive games

SVG

- Resolution independent
- Support for event handlers
- Best suited for applications with large rendering areas (Google Maps)
- Slow rendering if complex (anything that uses the DOM a lot will be slow)
- Not suited for game applications

Fullscreen API

- Fullscreen API provides a programmatic way to request full screen display from the user, and exit full screen when desired.

```
function launchFullScreen(element) {  
  if(element.requestFullScreen) {  
    element.requestFullScreen();  
  } else if(element.mozRequestFullScreen) {  
    element.mozRequestFullScreen();  
  } else if(element.webkitRequestFullScreen) {  
    element.webkitRequestFullScreen();  
  }  
}
```

- 'requestFullScreen' on 'Element': API can only be initiated by a user gesture.

```
// Launch fullscreen for browsers that support it based!  
launchFullScreen(document.documentElement); // the whole page  
launchFullScreen(document.getElementById("videoElement")); // any individual element
```

Fullscreen API

- The exitFullscreen method (prefixed in older browsers) morphs the browser back into standard layout:

```
function exitFullscreen() {  
    if(document.exitFullscreen) {  
        document.exitFullscreen();  
    } else if(document.mozCancelFullScreen) {  
        document.mozCancelFullScreen();  
    } else if(document.webkitExitFullscreen) {  
        document.webkitExitFullscreen();  
    }  
}
```


Fullscreen CSS

- Browsers provide helpful fullscreen CSS control

```
:-webkit-full-screen {  
    /* properties */  
}  
  
:-moz-full-screen {  
    /* properties */  
}  
  
:-ms-fullscreen {  
    /* properties */  
}  
  
:full-screen { /*pre-spec */  
    /* properties */  
}  
  
/* deeper elements */  
:-webkit-full-screen video {  
    width: 100%;  
    height: 100%;  
}
```

File API

- HTML5 provides a standard way to interact with local files, via the File API specification.
- File API could be used to create a thumbnail preview of images as they're being sent to the server, or allow an app to save a file reference while the user is offline.
- You could use client-side logic to verify an upload's mimetype matches its file extension or restrict the size of an upload.
- API:
 - File - an individual file; provides readonly information such as name, file size, mimetype, and a reference to the file handle.
 - FileList - an array-like sequence of File objects. (<input type="file" multiple> or dragging a directory of files from the desktop).

File API

Choose Files 3 files

- CIJS008_001.pptx (application/vnd.openxmlformats-officedocument.presentationml.presentation) - 910943 bytes, last modified: 19/03/2015
- CIJS013_14_15_16_001.pptx (application/vnd.openxmlformats-officedocument.presentationml.presentation) - 1659535 bytes, last modified: 21/03/2015
- CWCP004.pptx (application/vnd.openxmlformats-officedocument.presentationml.presentation) - 281530 bytes, last modified: 20/03/2015

```
<body>
  <input type="file" id="files" name="files[]" multiple />
  <output id="list"></output>
  <script>

    function handleFileSelect(evt) {
      var files = evt.target.files; // FileList object
      // files is a FileList of File objects. List some properties.
      var output = [];
      for (var i = 0, f; f = files[i]; i++) {
        output.push('<li><strong>', escape(f.name),
                    '</strong> (', f.type || 'n/a', ') - ',
                    f.size, ' bytes, last modified: ',
                    f.lastModifiedDate ? f.lastModifiedDate.toLocaleDateString() : 'n/a',
                    '</li>');
      }
      document.getElementById('list').innerHTML = '<ul>' + output.join('') + '</ul>';
    }

    document.getElementById('files').addEventListener('change', handleFileSelect, false);
  </script>

</body>
```

Reading files

- FileReader includes four options for reading a file, asynchronously:
 - `FileReader.readAsBinaryString(Blob | File)` - The result property will contain the file/blob's data as a binary string.
 - `FileReader.readAsText(Blob | File, opt_encoding)` - The result property will contain the file/blob's data as a text string.
 - `FileReader.readAsDataURL(Blob | File)` - The result property will contain the file/blob's data encoded as a data URL.
 - `FileReader.readAsArrayBuffer(Blob | File)` - The result property will contain the file/blob's data as an ArrayBuffer object.

Reading files

```
<script>
function handleFileSelect(evt) {
    var files = evt.target.files; // FileList object
    // Loop through the FileList and render image files as thumbnails.
    for (var i = 0, f; f = files[i]; i++) {
        if (!f.type.match('image.*')) { // Only process image files.
            continue;
        }
        var reader = new FileReader();
        // Closure to capture the file information.
        reader.onload = (function(theFile) {
            return function(e) {
                // Render thumbnail.
                var span = document.createElement('span');
                span.innerHTML = ['<img class="thumb" src="", e.target.result,
                                '" title="", escape(theFile.name), "/>'].join('');
                document.getElementById('list').insertBefore(span, null);
            };
        })(f);
        reader.readAsDataURL(f); // Read in the image file as a data URL.
    }
}
document.getElementById('files').addEventListener('change', handleFileSelect, false);
</script>
```



Drag and Drop

```
<div id="drop_zone">Drop files here</div>
<output id="list"></output>
```

```
<script>
```

```
function handleFileSelect(evt) {
    evt.stopPropagation();
    evt.preventDefault();
    var files = evt.dataTransfer.files; // FileList object.
    // files is a FileList of File objects. List some properties.
    var output = [];
    for (var i = 0, f; f = files[i]; i++) {
        output.push('<li>', escape(f.name),
                    '(', f.type || 'n/a', ') - ',
                    f.size, '</li>');
    }
    document.getElementById('list').innerHTML = '<ul>' + output.join('') + '</ul>';
}
function handleDragOver(evt) {
    evt.stopPropagation();
    evt.preventDefault();
    evt.dataTransfer.dropEffect = 'copy'; // Explicitly show this is a copy.
}
// Setup the dnd listeners.
var dropZone = document.getElementById('drop_zone');
dropZone.addEventListener('dragover', handleDragOver, false);
dropZone.addEventListener('drop', handleFileSelect, false);
```

```
</script>
```

Drop files here

- Elton John - Sacrifice.mp3(audio/mp3) - 4222537
- Fugees - Killing Me Softly.mp3(audio/mp3) - 3838519
- George Michael - Last christmas.mp3(audio/mp3) - 6511474

Offline Storage

- **Why offline?**

- Airplane, road trip, deserted island
- Flaky connections (e.g. cafes, car, short daily trips)
- Better performance
- Consolidates the concept of permanent app you will have always available

- **HTML 5 offline storage**

- **Web Storage (`localStorage/sessionStorage`)**

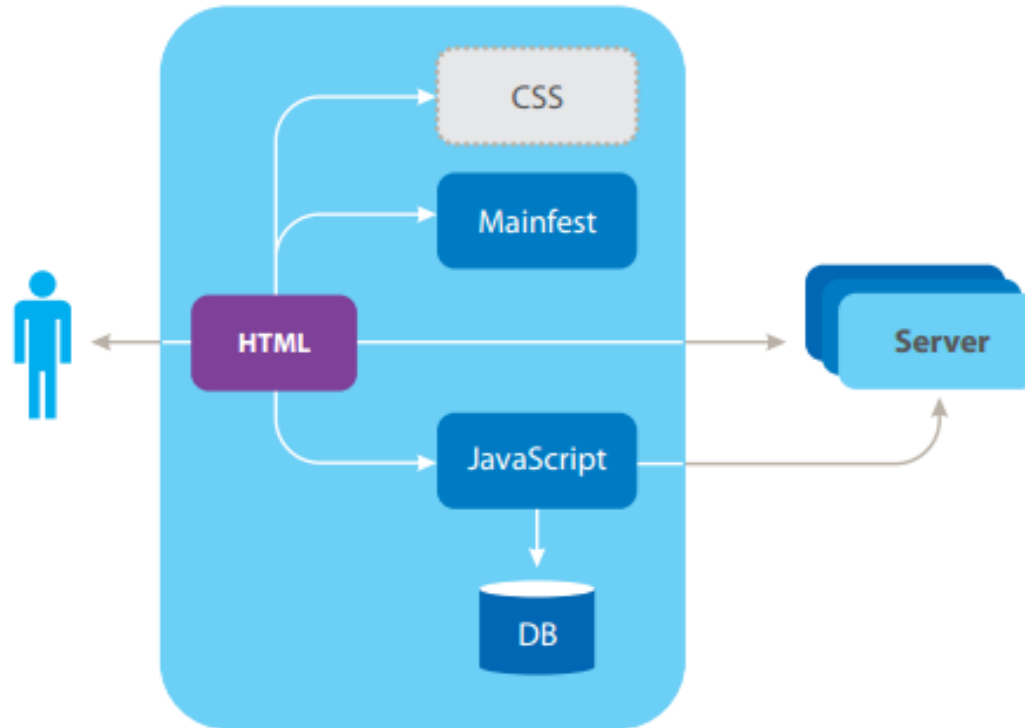
- Store unstructured data
 - Store miscellaneous data (user/game/app preferences)
 - Use cases of cookies but saving HTTP requests

- **IndexedDB/WebSQL**

- Store structured data
 - Handle simultaneous data operations (with transactions)

Offline Storage

- Architecture of an Offline Web Application



LocalStorage and SessionStorage

- Web Storage is an example of a NoSQL key-value store.
- Web Storage is basically a single persistent object called `localStorage`.
- There's also a second object called **sessionStorage** available, which works the same way, but clears when the window is closed.
 - `// Store`
`localStorage.setItem("lastname", "Smith");`
 - `// Retrieve`
`localStorage.getItem("lastname");`
 - `//Remove`
`localStorage.removeItem("lastname");`

WebSQL

- Open Database

```
var dbSize = 5 * 1024 * 1024; // 5MB
var db = openDatabase("Todo", "1", "Todo manager", dbSize);
```

Create Table

```
function createTable() {
    db.transaction(function(tx) {
        tx.executeSql("CREATE TABLE IF NOT EXISTS todo(ID INTEGER PRIMARY KEY ASC,
            todo TEXT, added_on DATETIME)", []);
    });
}
```

WebSQL

```
function addTodo() {
    var todoText = document.getElementById("todo");
    var txt = todoText.value;
    alert(txt);
    db.transaction(function(tx) {
        var addedOn = new Date();
        tx.executeSql("INSERT INTO todo(todo, added_on) VALUES (?,?)",
            [ txt, addedOn ], onSuccess, onError);
    });
    todoText.value = "";
}

function onError(tx, e) {
    alert("There has been an error: " + e.message);
}

function onSuccess(tx, r) {
    // re-render the data.
    // loadTodoItems is defined in Step 4a
    alert("Success :" + r);
    getAllTodoItems(loadTodoItems);
}
```

WebSQL

```
function getAllTodos(renderFunc) {
    db.transaction(function(tx) {
        tx.executeSql("SELECT * FROM todos", [], renderFunc,
            onError);
    });
}

function loadTodos(tx, rs) {
    var rowOutput = "";
    var todos = document.getElementById("todos");
    for (var i = 0; i < rs.rows.length; i++) {
        rowOutput += renderTodo(rs.rows.item(i));
    }

    todos.innerHTML = rowOutput;
}

function renderTodo(row) {
    return "<li>"
        + row.todo
        + " [<a href='javascript:void(0);' onclick='deleteTodo(" + row.ID + ")';>Delete</a>]</li>";
}
```

IndexedDB API

- Difference between Relational Database and indexedDB
 - indexedDB is an Object Store. It is not the same as a Relational Database.
 - In a traditional relational data store, we would have a table that stores data in the form of rows where as indexedDB creates an Object Store for a type of data that can store persist Javascript Objects.
 - Each Object Store can have a collection of Indexes that make it efficient to query and iterate across.
 - Unlike SQL, it is replaced with a query against an index which produces a cursor that you use to iterate across the result set.

Developing an Application using indexedDB

- Getting Reference to indexedDB

```
//prefixes of implementation that we want to test  
window.indexedDB = window.indexedDB || window.mozIndexedDB  
                || window.webkitIndexedDB || window.msIndexedDB;  
  
//prefixes of window.IDB objects  
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction  
                        || window.msIDBTransaction;  
  
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange  
                    || window.msIDBKeyRange
```

Creating a database

- Creating a database is done by calling the `open()` method of the `indexedDB` object.
- `var request = window.indexedDB.open("newDatabase", 1);`

```
var db;
request.onerror = function(event) {
  console.log("error: ");
};

request.onsuccess = function(event) {
  db = request.result;
  console.log("success: "+ db);
};
```

Creating a database

- This event handles the event whereby a new version of the database needs to be created
- Either one has not been created before, or a new version number has been submitted via the `window.indexedDB.open` line

```
request.onupgradeneeded = function(event) {  
    var db = event.target.result;  
    var objectStore = db.createObjectStore("customers", {keyPath: "id"});  
    for (var i in customerData) {  
        objectStore.add(customerData[i]);  
    }  
}
```


Adding Data to Object store

- Once the data source is created, we can store data into it as follows

```
function add() {  
    var request = db.transaction(["customers"], "readwrite")  
        .objectStore("customers")  
        .add({ id: "103", name: "Rahul", age: 21, email: "rahul@gmail.com" });  
  
    request.onsuccess = function(event) {  
        alert("Data has been added to your database.");  
    };  
    request.onerror = function(event) {  
        alert("Unable to add data is already exist in your database! ");  
    }  
}
```

Read data

```
function read() {  
    var transaction = db.transaction(["customers"]);  
    var objectStore = transaction.objectStore("customers");  
    var request = objectStore.get("103");  
    request.onerror = function(event) {  
        alert("Unable to retrieve data from database!");  
    };  
    request.onsuccess = function(event) {  
        // Do something with the request.result!  
        if(request.result) {  
            alert("Name: " + request.result.name + ", Age: "  
                + request.result.age + ", Email: " + request.result.email);  
        } else {  
            alert("Data couldn't be found in your database!");  
        }  
    };  
}
```

Read all

```
function readAll() {  
    var objectStore = db.transaction("customers").objectStore("customers");  
  
    objectStore.openCursor().onsuccess = function(event) {  
        var cursor = event.target.result;  
        if (cursor) {  
            alert("Name for id " + cursor.key + " is "  
                + cursor.value.name + ", Age: "  
                + cursor.value.age + ", Email: " + cursor.value.email);  
            cursor.continue();  
        }  
        else {  
            alert("No more entries!");  
        }  
    };  
}
```

Delete

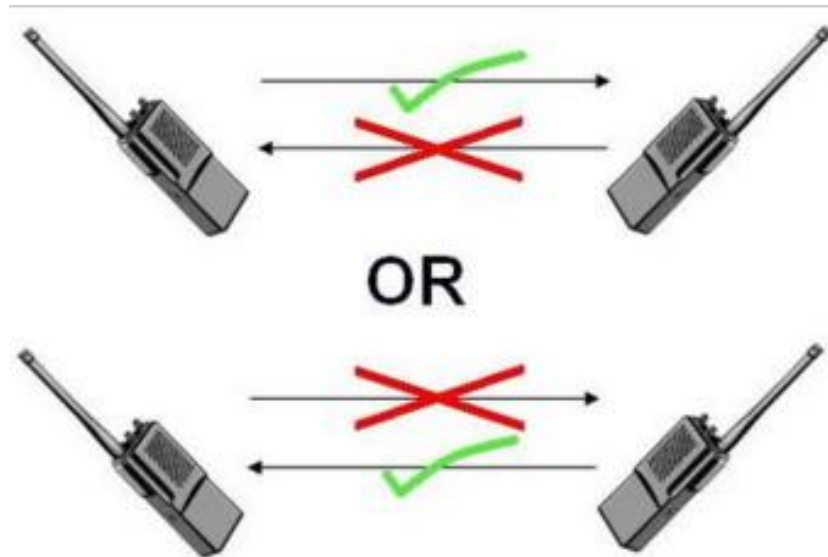
```
function remove() {  
    var request = db.transaction(["customers"], "readwrite")  
        .objectStore("customers")  
        .delete("103");  
    request.onsuccess = function(event) {  
        alert("entry has been removed from your database.");  
    };  
}
```

Duplex (telecommunications)

- A duplex communication system is a point-to-point system composed of two connected parties or devices that can communicate with one another in both directions.
- There are two types of duplex communication systems:
 - full-Duplex and half-Duplex.

Duplex (telecommunications)

- A half-duplex (HDX) system provides communication in both directions, but only one direction at a time (not simultaneously).



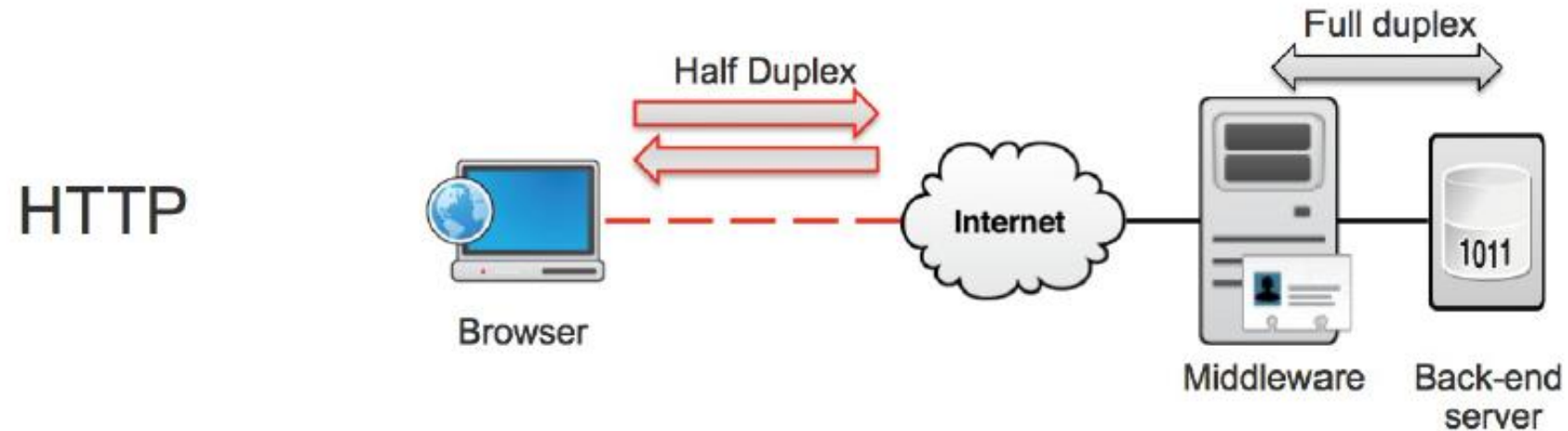
Duplex (telecommunications)

- A *full-duplex* (FDX) system, or sometimes called *double-duplex*, allows communication in both directions, and, unlike half-duplex, allows this to happen simultaneously.



HTTP Web Architecture

- Middleware is the glue between HTTP and TCP



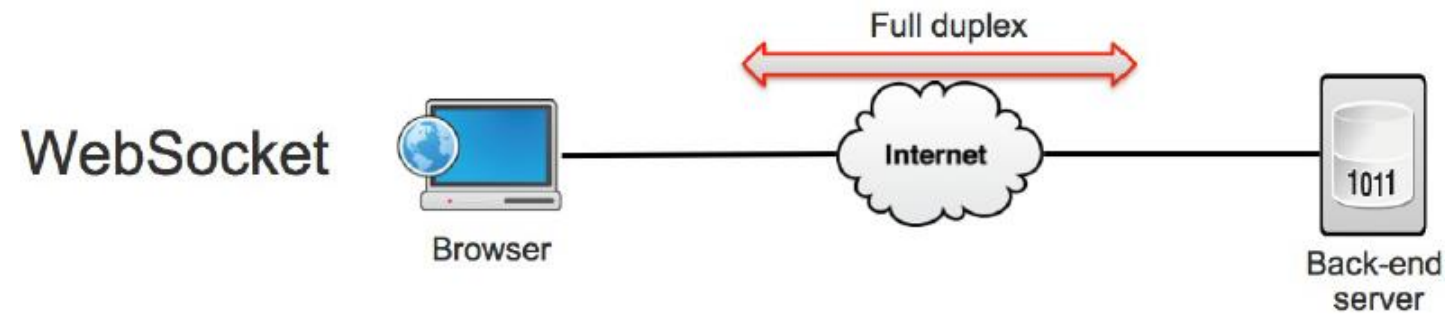
The Legacy Web Stack

- Designed to serve static documents
 - HTTP
 - Half duplex communication
- High latency
 - Bandwidth intensive
 - HTTP header traffic approx. 800 to 2000 bytes overhead per request/response
- Complex architecture
 - Ajax applications use various “hacks” to simulate real-time communication
 - Polling / long polling



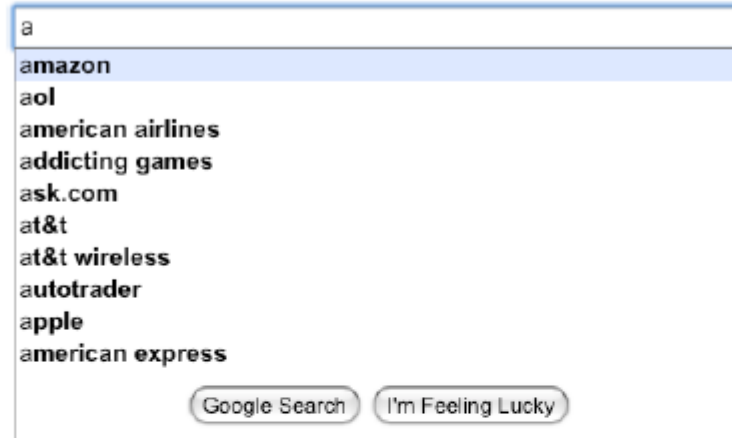
HTML5 WebSocket

- Provides a full-duplex, single socket over the Web



HTTP versus WebSockets

- Example: Entering a character in a search field with auto suggestion



	HTTP traffic*	WebSocket Traffic*
Google	788 bytes, plus 1 byte	2 bytes, plus 1 byte
Yahoo	1737 bytes, plus 1 byte	2 bytes, plus 1 byte

* Header information for each character entered into search bar

WebSockets reduces bandwidth overhead up to 1000x

HTTP versus WebSockets

“Reducing kilobytes of data to 2 bytes...and reducing latency from 150ms to 50ms is far more than marginal. In fact, these two factors alone are enough to make WebSocket seriously interesting to Google.”

—Ian Hickson (Google, HTML5 spec lead)

The New Web Stack

- Designed for full-duplex high performance transactional Web
 - HTTP & HTML5 WebSocket
 - Full duplex communication
- Lower latency
- Reduced bandwidth



WebSocket API

```
//Create new WebSocket  
var mySocket = new WebSocket("ws://  
www.WebSocket.org");
```

```
// Associate listeners  
mySocket.onopen = function(evt) {  
    alert("Connection open...");  
};
```

```
mySocket.onmessage = function(evt) {  
    alert("Received message: " + evt.data);  
};
```

```
mySocket.onclose = function(evt) {  
    alert("Connection closed...");  
};
```

```
// Sending data  
mySocket.send("WebSocket Rocks!");
```

```
// Close WebSocket  
mySocket.close();
```

WebSocket API

// Sending String

```
connection.send('your message');
```

```
    // Sending canvas ImageData as ArrayBuffer
```

```
    var img = canvas_context.getImageData(0, 0, 400, 320);
```

```
    var binary = new Uint8Array(img.data.length);
```

```
    for (var i = 0; i < img.data.length; i++) {
```

```
        binary[i] = img.data[i];
```

```
    }
```

```
    connection.send(binary.buffer);
```

// Sending file as Blob

```
var file = document.querySelector('input[type="file"]').files[0];
```

```
connection.send(file);
```

WebSocket API

- Attributes
 - url
 - the url passed to the constructor, it's a readonly attribute.
 - readyState
 - CONNECTING, (numeric value 0) The connection has not yet been established.
 - OPEN, (numeric value 1) The WebSocket connection is established
 - CLOSING, (numeric value 2) The connection is going through the closing handshake.
 - CLOSED, (numeric value 3) The connection has been closed or could not be opened.

WebSocket API

- Methods
 - `send(data)`
 - This method will send its data parameter to the server.
 - `close()`, calling this method will end the connection.
- Events
 - `open`
 - is fired (or called) when the connection is established.
 - `message`
 - is fired (or called) when a message is received..
 - `Error`
 - is fired (or called) when an error occurred.
 - `Close`
 - is fired (or called) when the connection is close.

Cross Origin Communication

- Being a modern protocol, cross origin communication is baked right into WebSocket.
- While you should still make sure only to communicate with clients and servers that you trust, WebSocket enables communication between parties on any domain.
- The server decides whether to make its service available to all clients or only those that reside on a set of well defined domains.


Proxy Servers

- Every new technology comes with a new set of problems.
- In the case of WebSocket it is the compatibility with proxy servers which mediate HTTP connections in most company networks.
- The WebSocket protocol uses the HTTP upgrade system (which is normally used for HTTP/SSL) to "upgrade" an HTTP connection to a WebSocket connection. Some proxy servers do not like this and will drop the connection.

The Server Side

- Keeping a large number of connections open at the same time requires an architecture that receives high concurrency at a low performance cost. Such architectures are usually designed around either threading or so called non-blocking IO
- Node.js
 - Socket.IO
 - WebSocket-Node
 - ws
- Java
 - Jetty
 - Apache Tomcat 8
- NET
 - SuperWebSocket

Use Cases

- Multiplayer online games
- Chat applications
- **Financial Apps** 
- Live sports ticker
- social networking



SSE: Server Sent Events

- Conventional web applications generate events which are dispatched to the web server. For example a simple click on a link requests a new page from the server.
- The type of events which are flowing from web browser to the web server may be called client-sent events.
- Along with HTML5, WHATWG [Web Hypertext Application Technology Working Group (WHATWG)] Web Applications 1.0 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
- Using SSE you can push DOM events continuously from your web server to the visitor's browser.
- The event streaming approach opens a persistent connection to the server, sending data to the client when new information is available, eliminating the need for continuous polling.
- Server-sent events standardizes how we stream data from the server to the client.

SSE vs WebSockets

- SSE: Server-Sent Events
 - HTTP based API dedicated to Push.
 - Allows the server to send data to client (one way communication)
 - SSE being based on HTTP, it is more compliant with various elements of existing IT infrastructure (load balancers, firewalls, ...)
- WebSockets
 - TCP based protocol providing full duplex communication link between client and server.

Receive Server-Sent Event Notifications

- Create a new EventSource object, and specify the URL of the page sending the updates
- Each time an update is received, the onmessage event occurs

```
<script>
  var source = new EventSource('events');
  source.onmessage = function(e) {
    document.body.innerHTML += e.data + '<br>';
  };
</script>
```


Server-Side Code Example using NodeJS

```
function sendSSE(req, res) {
  res.writeHead(200, {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive'
  });
  var id = (new Date()).toLocaleTimeString();
  // Sends a SSE every 5 seconds on a single connection.
  setInterval(function() {
    constructSSE(res, id, (new Date()).toLocaleTimeString());
  }, 5000);
  constructSSE(res, id, (new Date()).toLocaleTimeString());
}

function constructSSE(res, id, data) {
  res.write('id: ' + id + '\n');
  res.write("data: " + data + '\n\n');
}
```

Id	Type	Data	Time
11:56:51 AM	message	11:56:51 AM	11:56:51.763
11:56:51 AM	message	11:56:56 AM	11:56:56.706
11:56:51 AM	message	11:57:01 AM	11:57:01.707
11:56:51 AM	message	11:57:06 AM	11:57:06.708
11:56:51 AM	message	11:57:11 AM	11:57:11.708
11:56:51 AM	message	11:57:16 AM	11:57:16.710

Web workers

- This specification defines an API for running scripts in the background independently of any user interface scripts.
- This allows for long-running scripts that are not interrupted by scripts that respond to clicks or other user interactions, and allows long tasks to be executed without yielding to keep the page responsive.
- Workers (as these background scripts are called herein) are relatively heavy-weight, and are not intended to be used in large numbers.
- Generally, workers are expected to be long-lived, have a high start-up performance cost, and a high per-instance memory cost

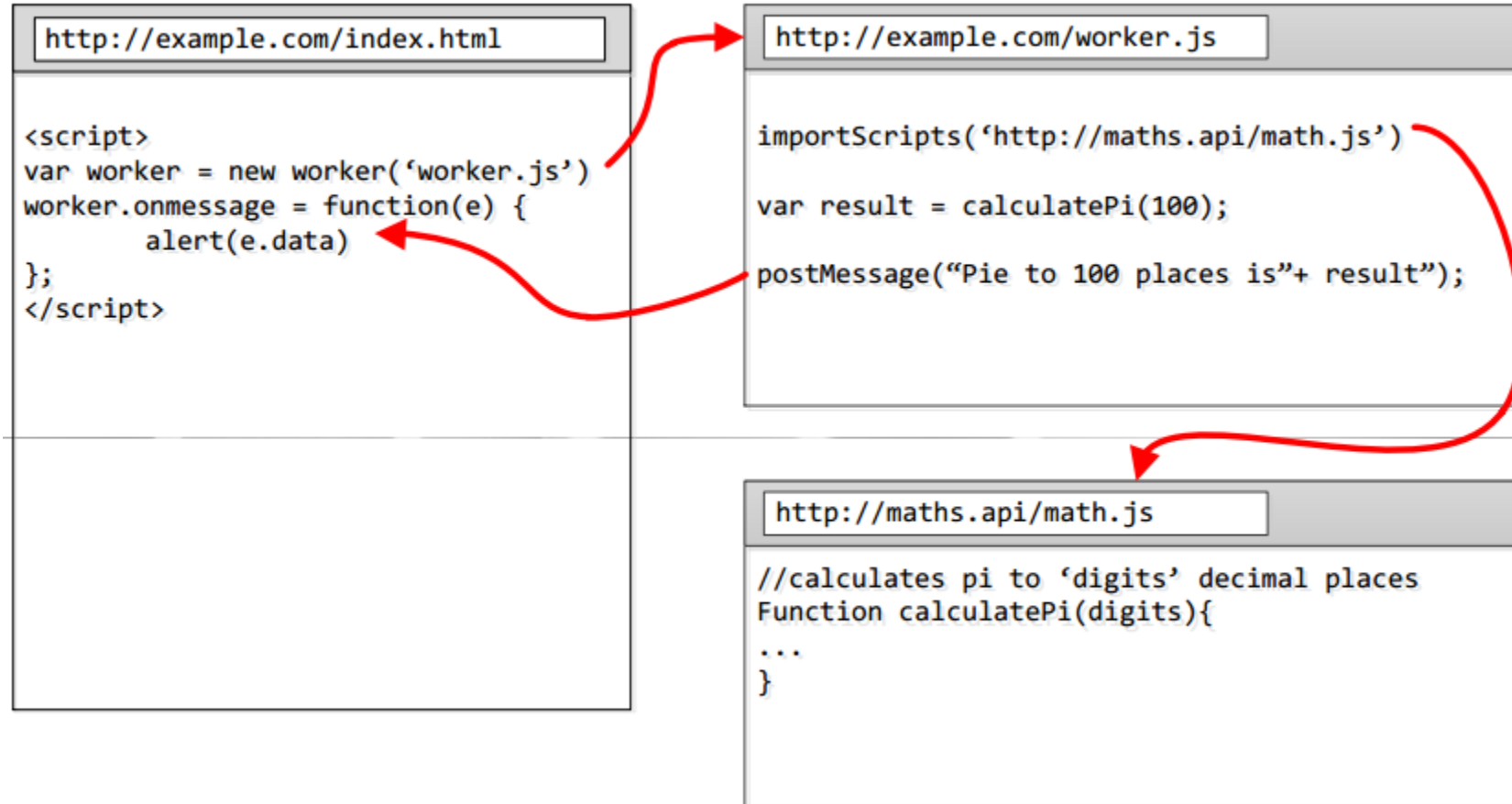
Worker Global Scope

- Workers have their own JavaScript context, separate from the renderer
 - Global scope (this) is NOT window
 - No DOM Access
 - No window
 - No document
 - No cookies
 - No Storage

Worker Global Scope

- Common functions (across all implementations)
- `postMessage`
- Event support
 - `addEventListener`
 - `dispatchEvent`
 - `removeEventListener`
- `importScripts`
- `Location`
- `Navigator`
- `XMLHttpRequest`

Web workers example



Web workers example

- Basic usage:

```
var worker= new Worker('worker.js');
```

- `postMessage()`
- sends data between worker and renderer
 - Though other channels exist: `openDatabase`, `XMLHttpRequest`, `MessageChannel`

Geolocation API

- The Geolocation API defines a high-level interface to location information, such as latitude and longitude, which is linked to the device hosting the implementation.
- The API exposes three methods that belong to the `window.navigator.geolocation` object.
- The methods provided are:
 - `getCurrentPosition`
 - `watchPosition`
 - `clearWatch`
- To obtain the device's location, `getCurrentPosition` and `watchPosition` make an asynchronous request.
- The difference between these methods is that `getCurrentPosition` performs a one-time request, while `watchPosition` monitors the device's location for changes and notifies the application when a location change takes place.
- The position object that's returned from the success callbacks of `getCurrentPosition` and `watchPosition` contains a `timestamp` and `coords` property.
- The `coords` property is an object containing the location's latitude, longitude, altitude, accuracy, altitudeAccuracy, heading, and speed.

Geolocation API

```
var geolocation = null;

if (window.navigator && window.navigator.geolocation) {
    geolocation = window.navigator.geolocation;
}

if (geolocation) {
    geolocation.getCurrentPosition(function(position) {
        console.log(position);
    });

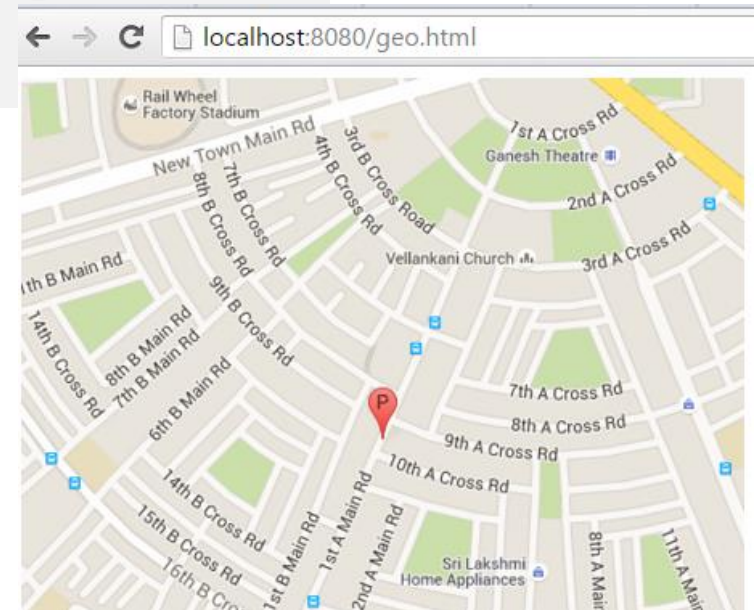
    var identifier = geolocation.watchPosition(function(position) {
        console.log(position);
    });

    console.log(identifier);
}
```

- Check using <http://www.latlong.net/c/?lat=20.09&long=78.58>

Using Google Maps

```
var image_url = "http://maps.google.  
com/maps/api/staticmap?sensor=false&center=" + position.coords.  
latitude + "," +  
position.coords.longitude + "  
    &zoom=16&size=500x500&markers=color:red|label:P|" +  
position.coords.latitude + ',' + position.coords.longitude;  
  
document.getElementById("map").src = image_url;
```



HTML5 History API

- The HTML5 history API is just a handful of methods on the `window.history` object, plus one event on the `window` object.
- In a supported browser, navigating the Next and Previous links in the photo gallery will update the content in place and update the URL in the browser location bar, without triggering a full page refresh.
- In unsupported browsers — or, indeed, supported browsers where the user has disabled scripting — the links simply function as regular links, taking you to a new page with a full page refresh.

The history.pushState()

- function takes three parameters:
 - state can be any JSON data structure. It is passed back to the popstate event handler, which you'll learn about in just a moment.
 - title can be any string. This parameter is currently unused by major browsers.
 - url can be, well, any URL. This is the URL you want to appear in the browser's location bar.
- Calling history.pushState will change the URL in the browser's location bar.

The history.pushState()

- Normally when the user navigates to a new page (with a full page refresh), the browser pushes the new URL onto its history stack and downloads and draws the new page.
- When the user presses the back button, the browser pops one page off its history stack and redraws the previous page.
 - But what happens now that you've short-circuited this navigation to avoid a full page refresh?
 - Well, you've faked "moving forward" to a new URL, so now you also need to fake "moving backward" to the previous URL.
 - And the key to faking "moving backwards" is the popstate event.

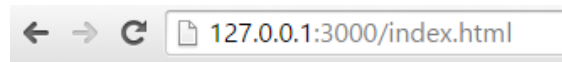
The `replaceState()` and `popState()` method

- **`history.replaceState()`**
 - operates exactly like `history.pushState()` except that `replaceState()` modifies the current history entry instead of creating a new one.
 - `replaceState()` is particularly useful when you want to update the state object or URL of the current history entry in response to some user action.
- **The `popstate` event**
 - A `popstate` event is dispatched to the window every time the active history entry changes. If the history entry being activated was created by a call to `pushState` or affected by a call to `replaceState`, the `popstate` event's `state` property contains a copy of the history entry's state object.
- **The current state**
 - You can read the state of the current history entry without waiting for a `popstate` event :
`var currentState = history.state;`

History API Example - 1

- To begin with you will need to create an HTML file that contains some navigation links, a `<h1>` for the page title and a `<div>` for the content.

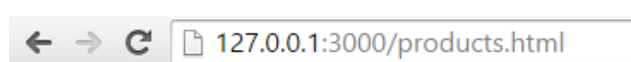
```
<nav>
  <ul>
    <li><a href="index.html" class="load-content">Home</a></li>
    <li><a href="products.html" class="load-content">Products</a></li>
    <li><a href="contact.html" class="load-content">Contact</a></li>
  </ul>
</nav>
<h1 id="title"></h1>
<div id="content"></div>
```



- [Home](#)
- [Products](#)
- [Contact](#)

Home Page

This is the home page.



- [Home](#)
- [Products](#)
- [Contact](#)

Products

Buy some of our great products!

History API Example - 2

```
window.onload = function() {  
  
    // Content for the pages.  
    // Note: You would probably want to load the page content using  
    // AJAX in a real application.  
    var pages = {  
        index: {  
            title: "Home Page",  
            content: "This is the home page."  
        },  
        products: {  
            title: "Products",  
            content: "Buy some of our great products!"  
        },  
        contact: {  
            title: "Contact",  
            content: "Say hello! We love to chat."  
        }  
    };  
  
    // Get references to the page elements.  
    var navLinks = document.querySelectorAll('.load-content');  
    var titleElement = document.getElementById('title');  
    var contentElement = document.getElementById('content');
```

History API Example - 3

```
// Update the page content.
var updateContent = function(stateObj) {
    // Check to make sure that this state object is not null.
    if (stateObj) {
        document.title = stateObj.title;
        titleElement.innerHTML = stateObj.title;
        contentElement.innerHTML = stateObj.content;
    }
};

// Attach click listeners for each of the nav links.
for (var i = 0; i < navLinks.length; i++) {
    navLinks[i].addEventListener('click', function(e) {
        e.preventDefault();
        // Fetch the page data using the URL in the link.
        var pageURL = this.attributes['href'].value;
        var pageData = pages[pageURL.split('.')[0]];
        // Update the title and content.
        updateContent(pageData);
        // Create a new history item.
        history.pushState(pageData, pageData.title, pageURL);
    });
}
```


History API Example - 4

```
// Update the page content when the popstate event is called.  
window.addEventListener('popstate', function(event) {  
    updateContent(event.state)  
});  
  
// Load initial content.  
updateContent(pages.index);  
  
// Update this history event so that the state object contains  
// the data for the homepage.  
history.replaceState(pages.index, pages.index.title, '');
```

App Cache

- List resources that you want to take offline

```
CACHE MANIFEST
```

```
/static/stickies.html
```

```
/media/deleteButton.png
```

```
/media/deleteButtonPressed.png
```

```
/css/stickies.css
```

```
/js/stickies.js
```

```
<body manifest="./cache.manifest">
```

```
</body>
```

Search Engine Optimization (SEO)

- Search engine optimization (SEO) is a ways and means to help improve the amount of traffic that reaches your site from having your site appear in search engine results.
- SEOs rely on traditional HTML optimization as a standard tool in their fight to improve search rankings.
- Various forms of SEO have been around for many years. Ten years ago, if you wanted Yahoo and Alta Vista to index your site, you would have to perform rudimentary tasks such as using <META NAME> tags.
 <META NAME="keywords" CONTENT="web development, webmaster, web design, tips and tricks, HTMLGoodies, HTML">
 - These tags provided the basic information that let search engines know what your site was about, who wrote or created it, and when it was published.

SEO - KEYWORDS

- A keyword is a term that is used to match with the query a person enters into a search engine to find specific information.
- Most people enter search phrases that consist of two to five words. Such phrases may be called search phrases, keyword phrases, query phrases, or just keywords.
- Good keyword phrases are specific and descriptive.

SEO – KEYWORDS OPTIMIZATION

- **Keyword Frequency**

- This is calculated as how often does a keyword appear in a website title or description. You do not want to go overboard with frequency, however, since on some engines if you repeat a word too many times, you will be penalized for "**spamming**" or keyword stuffing.
- In general though, repeat your keyword in the document as many times as you can get away with, and up to **3-7 times** in your list of **metatags**.

- **Keyword Weight**

- It refers to the number of keywords appearing on your web page compared to the total number of words appearing on that same page.
- Some search engines consider this while determining the rank of your website for a particular keyword search.
- One technique that often works well is to create some smaller pages, generally just a paragraph long that emphasizes a particular keyword.
- By keeping the overall number of words to a minimum, you can increase the "weight" of the keyword you are emphasizing.

SEO – KEYWORDS OPTIMIZATION

- **Keyword Proximity**

- It refers to the placement of keywords on a web page in relation to each other or, in some cases, in relation to other words with a similar meaning as the queried keyword.
- For search engines, that grade a keyword match by keyword proximity, the connected phrase **home loans** will outrank a citation that mentions **home mortgage loans** assuming that you are searching only for the phrase "**home loans**".

- **Keyword Prominence**

- It is a measure of how early or high up on a page, the keywords are found.
- Having keywords in the first heading and in the first paragraph first **20 words or so on a page** are best.

SEO – KEYWORDS OPTIMIZATION

- **Keyword Placement**

- Where your keywords are placed on a page is very important.
- For example, in most engines, placing the keywords in the Title of the page, or in the Heading tags will give it more relevancy.
- On some engines, placing keywords in the link text, the part that is underlined on the screen in a browser, can add more relevancy to those words.
- Here is a list of places where you should try to use your main keywords.
 - Keywords in the <title> tags.
 - Keywords in the <meta name="description">.
 - Keywords in the <meta name="keyword">.
 - Keywords in <h1> or other headline tags.
 - Keywords in the keywords link tags.
 - Keywords in the body copy.
 - Keywords in alt tags.
 - Keywords in <!-- insert comments here> comments tags.
 - Keywords in the URL or website address.

SEO – KEYWORDS OPTIMIZATION

- **Finding Keywords**

- The potential words, people would use to find your product or service.
- The problems that your prospective customers may try to solve with your product or service.
- Keyword tags on competitor's websites.
- Visible page copy on competitor's websites.
- Related search suggestions on top search engines.
- Using an online tool such as **Google Keyword Tool**
- By analyzing your website carefully and finding out proper keywords. This task can be done by expert **SEO copywriters**.

SEO – KEYWORDS OPTIMIZATION

- **Word Stemming**

- Google uses a feature called word stemming that allows all forms of the word - singular, plural, verb form as well as similar words to be returned for a given search query.
- So if someone types in "house plans", not only the pages that are optimized for that phrase but the pages that contain all variations of that phrase are returned.
 - For example, "house plan", "house planning", "house planner".

Keywords in Special Places

- **Keywords in URLs and File Names**

- The domain name and the whole URL of a site tell a lot about it.
- The presumption is that if your site is about dogs, you will have “dog”, “dogs”, or “puppy” as part of your domain name.
- For instance, if your site is mainly about adopting dogs, it is much better to name your dog site “dog-adopt.net” than “animal-care.org”
 - In the first case you have two major keywords in the URL, while in the second one you have no more than one potential minor keyword.
- Strike a balance between the keywords in the URL and site usability, which says that more than 3 words in the URL is a way too much.

Keywords in Special Places

- **Keywords in Page Titles**

- The page title is another special place because the contents of the <title> tag usually gets displayed in most search engines, (including Google).
- Unlike URLs, with page titles you can get wordy. If we go on with the dog example, the <title> tag of the home page for the <http://dog-adopt.net> can include something like this: <title>Adopt a Dog – Save a Life and Bring Joy to Your Home</title>, <title>Everything You Need to Know About Adopting a Dog</title> or even longer.

- **Keywords in Headings**

- Normally headings separate paragraphs into related subtopics and from a literary point of view, it may be pointless to have a heading after every other paragraph but from SEO point of view it is extremely good to have as many headings on a page as possible, especially if they have the keywords in them.

SEO usage

- **Meta Description Tag**
- `<meta name="keywords"`
 `content="KEYWORD1 KEYWORD2 KEYPHRASE1 etc. about 30 to 40 unique words">`
- Use the following tips for preparing good meta keywords tags.
 - Use synonyms.
 - Use unique keywords.
 - No need to repeat any given phrase.
- **Robots Meta Tag**
 - The important metatag that you may need sometime is the Robots Metatag which looks like this:
 - `<meta name="robots" content="noindex,nofollow">`
 - Using the above metatag, you can tell a spider or a robot that you do not want some of your pages indexed, or that you do not want your links followed.

SEO usage

- **Anchor**

- Use descriptive anchor text for all your text links.

`Anchor Text`

- The anchor title should have appropriate keywords.
 - Anchor title helps the site visitors using a balloon, and displaying written text.
 - The Anchor Text is another important part, which should be selected very carefully because this text is used not only for search engines but also for navigation purpose.

SEO Benefits From HTML5

- HTML5 has several new tags that will help classify important content
 - <article>.
 - <section>.
 - <article>.
 - <header>.
 - <hgroup>
 - <footer>
 - <nav>
 - <aside>
 - <video>

```
<body>
  <header>
    Header content goes here
  </header>
  <nav>
    Navigation menu goes here
  </nav>
  <footer>
    Footer content goes here
  </footer>
</body>
```

Microdata

- Microdata and JSON-LD are two different ways to mark up your data using the schema.org vocabulary.
- It's best to choose either microdata or JSON-LD and avoid using both types on a single page or email.
- Google prefers microdata for web content.
- Microdata defines five HTML attributes that can be applied to any HTML5 tag.
 - **Itemscope** - Indicates the element is a microdata element and its child elements are part of its microdata format.
 - **Itemtype** - Defines the vocabulary to be used by the microdata format.
 - **Itemid** - The unique identifier of the item, if defined by the microdata vocabulary.
 - **Itemprop** - An individual data element.
 - **Itemref** - Allows a microdata element to reference another element on the page to define it by either HTML id or by itemid.

Microdata

- **schema.org** is a collaboration by Google, Microsoft, and Yahoo! to improve the web by creating a common vocabulary for describing the data on the web.
- If you add schema.org markup to your HTML pages, many companies and products—including Google search—will understand the data on your site. Likewise, if you add schema.org markup to your HTML-formatted email, other email products in addition to GMail might understand the data.

Microdata example

```
<article itemscope itemtype="http://schema.org/Event">
  <h1 itemprop="summary">HTML 5 Training</h1>
  
  <p itemprop="description">This training gives you a chance to
    learn about various new features of HTML 5 and CSS3.</p>
  <p>2014 November 6,
    <time itemprop="startDate" datetime="2014-11-06T08:30+01:00">8:30</time>
    &ndash;
    <time itemprop="endDate" datetime="2014-11-06T20:30+01:00">20:30</time></p>
  <p itemprop="location" itemscope
    itemtype="http://schema.org/Organization">
    <span itemprop="name">Mindtree</span><br>
    <span itemprop="address" itemscope
      itemtype="http://schema.org/PostalAddress">
      <span itemprop="streetAddress">Global Tech Park, Mysore Road.</span><br>
      <span itemprop="postalCode">560087</span>
      <span itemprop="addressLocality">Bangalore</span><br>
    </span>
  </p>
  <span itemprop="geo" itemscope itemtype="http://schema.org/Geo">
    <meta itemprop="latitude" content="70.047893" />
    <meta itemprop="longitude" content="44.4491" />
  </span>
</article>
```

Microdata example

localhost:3000/banu-microdata.html

B a n u ' s T r a i n

Attend Ba


HTML 5 Training

This training gives you a chance to learn about various new features of HTML 5 and CSS3.

2014 November 6, 8:30 - 20:30

Mindtree
Global Tech Park, Mysore Road
560087 Bangalore

Microdata (JSON) X

ITEM
type: <http://schema.org/Event>
properties:
summary: HTML 5 Training
photo: 

description: This training gives you a chance to learn about various new features of HTML 5 and CSS3.
startDate: 8:30
endDate: 20:30
location: ITEM 1
geo: ITEM 2

ITEM 1
type: <http://schema.org/Organization>
properties:
name: Mindtree
address: ITEM 1

ITEM 2
type: <http://schema.org/PostalAddress>
properties:
streetAddress: Global Tech Park, Mysore Road.
postalCode: 560087
addressLocality: Bangalore

What Is Web Accessibility?

- *Web accessibility* basically means that people with disabilities can use the Web.



[Web accessibility participant using computer with headstick]

Why Web Accessibility?

- The Web is providing unprecedented access to information and interaction for people with disabilities.
- It provides opportunities to participate in society in ways otherwise not available.
 - With accessible websites, people with disabilities can do ordinary things can do more things themselves, without having to rely on others.
 - People who are blind can read the newspaper (through screen readers that read aloud text from the computer), and so can people with cognitive disabilities who have trouble processing written information.
 - People who are deaf can get up-to-the-minute news that was previously available only to those who could hear radio or TV, and so can people who are blind and deaf (through dynamic Braille displays).
 - People with quadriplegia who cannot move their arms or legs can shop online to get groceries, gadgets, and gifts delivered.
 - People who cannot speak can participate in online discussions, such as through blog comments.

Web Content Accessibility Guidelines

- Web Content Accessibility Guidelines (WCAG) is developed through the W3C process in cooperation with individuals and organizations around the world, with a goal of proving a single shared standard for web content accessibility that meets the needs of individuals, organizations, and governments internationally.
- The WCAG documents explain how to make web content more accessible to people with disabilities. Web "content" generally refers to the information in a web page or web application, including:
 - natural information such as text, images, and sounds
 - code or markup that defines structure, presentation, etc.

Who WCAG is for?

- WCAG is primarily intended for:
 - Web content developers (page authors, site designers, etc.)
 - Web authoring tool developers
 - Web accessibility evaluation tool developers
 - Others who want or need a standard for web accessibility
- WCAG 2.0 is approved as an ISO standard: ISO/IEC 40500:2012.

Web Accessibility Evaluation Tools

- Web accessibility evaluation tools are software programs or online services that help you determine if web content meets accessibility guidelines.
- <http://www.w3.org/WAI/ER/tools/>
 - Provides a list of evaluation tools that you can filter to find ones that match your particular needs.
 - To determine what kind of tool you need and how they are able to assist you

Web Accessibility Page title

- Page titles [<title> within the <head>] are:
 - shown in the window title bar in some browsers
 - shown in browsers' tabs when there are multiple web pages open
 - shown in search engine results
 - used for browser bookmarks / favourites
 - read by screen readers
- What to do:
 - Look at the page's title (or with a screen reader, listen to it).
 - Look at titles of other pages within the website.
- What to check for:
 - Check that there is a title that adequately and briefly describes the content of the page.
 - Check that the title is different from other pages on the website, and adequately distinguishes the page from other web pages.

Web Accessibility Page title

- Example:
- Poor titles:
 - Welcome to home page of Acme Web Solutions, Inc.
 - Acme Web Solutions, Inc. | About Us
 - Acme Web Solutions, Inc. | Contact Us
 - Acme Web Solutions, Inc. | History
- Better page titles:
 - Acme Web Solutions home page
 - About Acme Web Solutions
 - Contact Acme Web Solutions
 - History of Acme Web Solutions

Web Accessibility: Alt Text

- Alternative text equivalents, called alt text, are a clear example of web accessibility. Web pages often include images, but some people cannot see images.
- People who cannot see images can get the information contained in the images when web developers include alternative text equivalents for images.
- An alternative text equivalent provides the same functional information in text as the image provides visually.
- The markup for image alt text looks like:
- **``**
- Alt text is:
 - Read by screen readers and voicing browsers
 - Displayed in text browsers
 - Displayed in graphical browsers when images are not downloaded

Web Accessibility: Contrast ratio

- Some people cannot read text if there is not sufficient contrast between the text and background, for example, light gray text on a light background.
- High contrast (for example, dark text on light background or bright text on dark background) is required by some people with visual impairments, including many older people who lose contrast sensitivity from ageing.

Some people cannot read text if there is not sufficient contrast between the text and background. For others, bright colors (high luminance) are not readable; they need low luminance.

Some people cannot read text if there is not sufficient contrast between the text and background. For others, bright colors (high luminance) are not readable; they need low luminance.

Figure: Dark text on light background, and yellow text on black background.

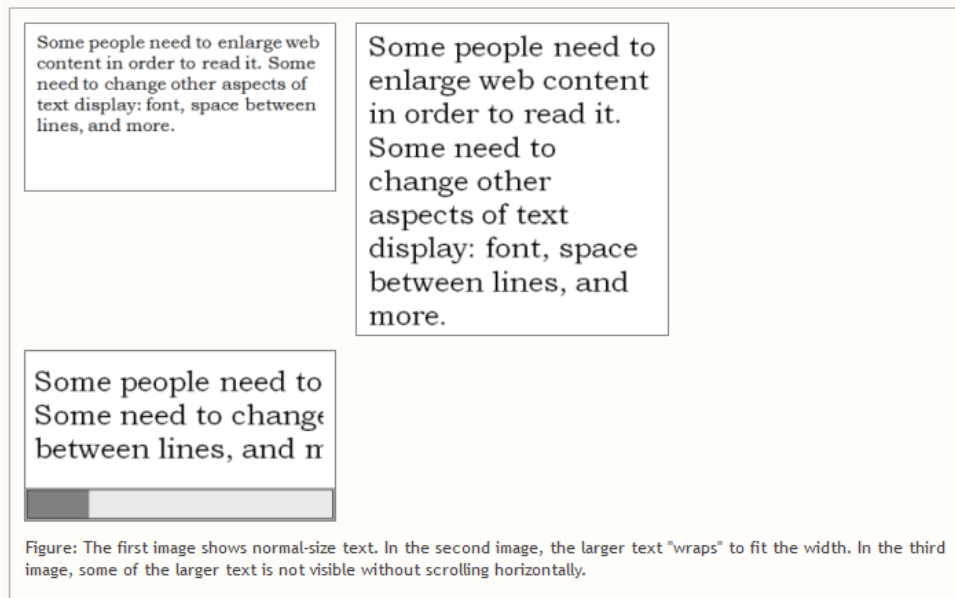
- Web browsers should allow people to change the color of text and background, and web pages need to work when people change colors.

Web Accessibility: Resize text

- Some people need to enlarge web content in order to read it.
- Some need to change other aspects of text display: font, space between lines, and more.
 - Most browsers allow users to change text size through:
 - text size settings (usually through Options or Preferences)
 - text-only zoom
 - page zoom (which also zooms images, buttons, etc.)
- When pages are not designed properly, they can be unusable when the text size is changed, especially when it is changed through text-only zoom or text settings.
- Sometimes columns and sections overlap, the space between lines disappears, lines of text become too long, or text disappears.

Web Accessibility: Resize text

- When text size is increased, sometimes part of the sentences are not visible and users have to scroll horizontally to read a sentence, as shown in the third example below.
- Most people cannot effectively read text that requires horizontal scrolling, and some disabilities make this impossible



Web Accessibility: Resize text

- What to do:
 - Increase the text size.
- What to check for:
 - All text gets larger.
 - Text doesn't disappear or get cut off.
 - Text, images, and other content do not overlap.
 - All buttons, form fields, and other controls are visible and usable.
 - Horizontal scrolling is not required to read sentences or "blocks of text".
 - It is best practice that when text size is increased, all the text in a sentence is visible.
 - It is acceptable to have to scroll horizontally to get to different sections of a page.

Keyboard access and visual focus

- Many people cannot use a mouse and rely on the keyboard to interact with the Web.
- People who are blind and some sighted people with mobility impairments rely on the keyboard or on assistive technologies and strategies that rely on keyboard commands, such as voice input.
- Websites need to enable people to access all content and functionality — links, forms, media controls, etc. — through a keyboard.
- Keyboard focus should be visible and should follow a logical order through the page elements.
- Visible keyboard focus could be a border or highlight, as shown below, that moves as you tab through the web page.

Forms, labels, and errors

- Form fields and other form controls usually have visible labels, such as "E-mail Address:" as the label for a text field.
- Check that the labels are positioned correctly.
- Check that any fields that are required/mandatory are clearly indicated.
 - Check that the indicator does not rely on colour alone, for example, if required fields were only indicated by red coloured labels, they would not be accessible to people who do not see the different colours.
 - Check that the indicator (such as asterisks (*)) is included in the marked up field label for text boxes and drop-down lists, or legend for radio buttons and checkboxes
- Visit
 - <http://www.w3.org/WAI/demos/bad/before/survey>
 - <http://www.w3.org/WAI/demos/bad/after/survey>

Designing Accessibility with HTML5

- **Accessible Rich Internet Applications**

- ARIA uses a set of roles, states and properties to expose a Web page to the accessibility APIs.
- These roles, states and properties are assigned on a page's elements, which are exposed to the ATs.
- Most current AT tools—including JAWS, NVDA and VoiceOver—support ARIA

ARIA Roles

- Roles indicate the type of element in a meaningful way.
- Suppose a screen reader comes across an HTML element on a page that includes `role=navigation`. The screen reader will know this HTML element is for navigation, and the user will be able to access navigation directly instead of tabbing through all the links.
- ARIA role attributes are applied to HTML elements like this:
- **`<div role="XXX"> </div>`**
 - Here “XXX” is a value that depends on the type of the HTML element and its role on the page.
 - It can take a number of values—such as a form, navigation, search or article—based on the content it represents.
 - There are three types of roles:
 - Landmark roles act as navigational landmarks.
 - Structural roles define the document’s structure and help organize content.
 - Widget roles consist of standalone UI widgets as well as composite widgets that are containers of two or more standalone widgets.

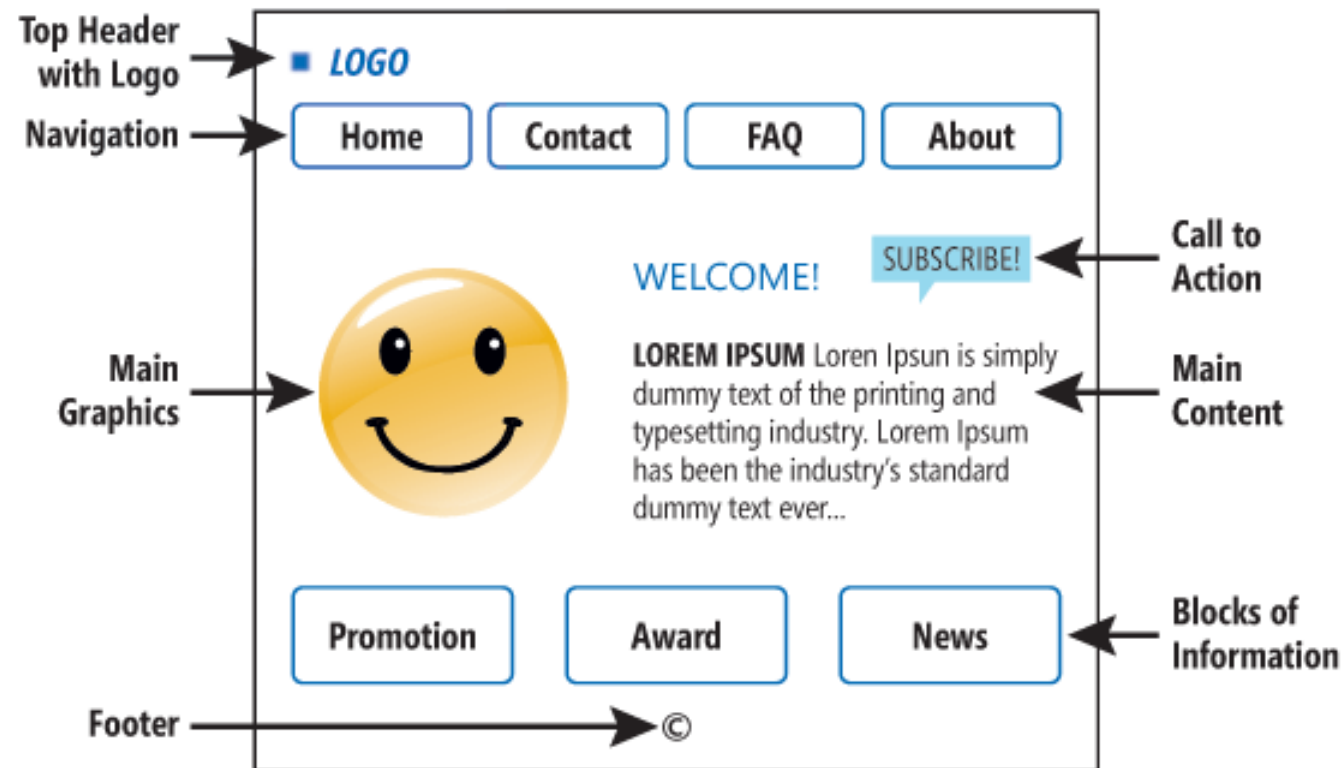
ARIA States

- An ARIA state is a dynamic property of an HTML element that represents data associated with the object but doesn't affect the essential nature of the element.
- The following shows the attribute aria-hidden:

```
<div aria-hidden="true">  
  <p>Paragraph text here </p>  
</div>
```
- This code will hide the paragraph from a screen reader.
- Visit: [http://www.w3.org/TR/wai-aria/states and properties](http://www.w3.org/TR/wai-aria/states_and_properties) for more details

ARIA Accessible Web Site

- Standard Layout for a Homepage



ARIA FORM-Example

```
<article>
  <h2>Contact <span>Form</span></h2>
  <form id="contacts-form" action="" method="post">
    <fieldset>
      <div class="field">
        <label for="name" >Name </label>
        <input id="name" placeholder="Banu Prakash" autofocus required
          aria-required="true" type="text" value="" />
      </div>
      <div class="field">
        <label for="email">E-mail</label>
        <input id="email" placeholder="john@msn.com" type="email" required
          aria-required="true" value=""/>
      </div>
      <div class="field">
        <label for="message">Message</label>
        <textarea id="message"
          placeholder="Write your message Here!" required
          aria-required="true" ></textarea>
      </div>
      <div><a href="#" onclick="submit()">Send Your Message!</a></div>
    </fieldset>
  </form>
</article>
```

STANDARDS

- W3C standards define an Open Web Platform for application development that has the unprecedented potential to enable developers to build rich interactive experiences that are available on any device.
- W3C develops these technical specifications and guidelines through a process designed to maximize consensus about the content of a technical report, to ensure high technical and editorial quality, and to earn endorsement by W3C and the broader community.

W3C Markup Validator

- What is Markup Validation?
 - Most pages on the World Wide Web are written in computer languages (such as HTML) that allow Web authors to structure text, add multimedia content, and specify what appearance, or style, the result should have.
 - As for every language, these have their own grammar, vocabulary and syntax, and every document written with these computer languages are supposed to follow these rules.
 - The (X)HTML languages, for all versions up to XHTML 1.1, are using machine-readable grammars called DTDs, a mechanism inherited from SGML.
 - However, Just as texts in a natural language can include spelling or grammar errors, documents using Markup languages may (for various reasons) not be following these rules.
 - The process of verifying whether a document actually follows the rules for the language(s) it uses is called validation, and the tool used for that is a validator.
 - A document that passes this process with success is called valid.


Is validation some kind of quality control?

- Validity is one of the quality criteria for a Web page
 - The fact that the W3C Markup Validator says that one page passes validation does not mean that W3C assesses that it is a good page.
 - It only means that a tool (not necessarily without flaws) has found the page to comply with a specific set of rules. No more, no less.
- This is also why the "valid ..." icons should never be considered as a "W3C seal of quality".

Why should I validate my HTML pages?









- Browsers accept Web pages and try to display them even if they're not legal HTML.
 - Usually this means that the browser will try to make educated guesses about what you probably meant.
 - The problem is that different browsers (or even different versions of the same browser) will make different guesses about the same illegal construct.
 - Worse, if your HTML is really pathological, the browser could get hopelessly confused and produce a mangled mess, or even crash.
- That's why you want to make sure your pages are legal HTML.
- The best way to do that is by running your documents through one or more HTML validators.


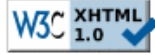

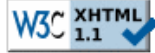




"valid" icons







- My document  use your "valid" icon?
- To show readers that one has taken some care to create an interoperable Web page, a "W3C valid" badge may be displayed (here, the "valid XHTML 1.0" badge) on any page that validates.
- We encourage you to use the XHTML code below (or its HTML equivalent), but you may use a different code to integrate the icon within your web page as long as the icon is used as a link to revalidate the Web page it is in.
- Sample code is as follows:

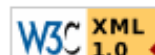
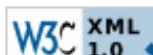
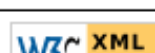
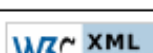
```
<p>  
  <a href="http://validator.w3.org/check?uri=referer">  
    </a>  
</p>
```

List of all W3C Validation Icons

Document type	"Gold"	Blue
HTML 2.0		
HTML 3.2		
HTML 4.0		
HTML 4.01		


Document type	"Gold"	Blue
XHTML 1.0		
XHTML 1.1		
XHTML Basic 1.0		
XHTML-Print 1.0		

Document type	"Gold"	Blue
CSS		
CSS 1		
CSS 2		

Version	"Gold"	Blue
XML 1.0		
XML 1.1		

W3C Markup Validation Service

- This validator checks the markup validity of Web documents in HTML, XHTML, SMIL, MathML, etc.



The screenshot shows the W3C Markup Validation Service interface. At the top, a browser address bar displays 'validator.w3.org'. Below this is a blue header with the W3C logo and the text 'Markup Validation Service' and 'Check the markup (HTML, XHTML, ...) of Web documents'. Three tabs are visible: 'Validate by URI' (selected), 'Validate by File Upload', and 'Validate by Direct Input'. Under the 'Validate by URI' tab, there is a section titled 'Validate by URI' with the text 'Validate a document online:'. Below this is a text input field labeled 'Address:'. A link 'More Options' is visible below the input field. At the bottom right, there is a 'Check' button. The footer of the page displays the email address 'banuprakashc@yahoo.co.in'.

W3C Markup Validation Service

- Validate by URI

Validate a document online:

Address:

► [More Options](#)

Validation Output: 5 Errors

✖ Line 239, Column 66: Duplicate attribute class.

```
... <a class="global" href="javascript:void(0)" class = "reg-pointer">GLOBAL</a>
```

⚠ Line 275, Column 41: Element name gcse:searchbox-only cannot be represented as XML 1.0.

```
<gcse:searchbox-only></gcse:searchbox-only>
```

✖ Line 275, Column 41: Element gcse:searchbox-only not allowed as child of element div in this context. (Suppressing further errors from this subtree.)

```
<gcse:searchbox-only></gcse:searchbox-only>
```

Content model for element [div](#):
[Flow content](#).

✖ Line 310, Column 93: & did not start a character reference. (& probably should have been escaped as &.)

```
...ale-microsite/index.php?utm_source=ref&utm_medium=cta&utm_campaign=digital_tra...
```

✖ Line 310, Column 108: & did not start a character reference. (& probably should have been escaped as &.)

```
...index.php?utm_source=ref&utm_medium=cta&utm_campaign=digital_transformation" da...
```

W3C Markup Validation Service

- Validate by direct input

Validate by direct input

Enter the Markup to validate:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">
  </head>
  <body>
    <div class="container">
      <header class="jumbotron">
        <h1>Welcome to SOBIS </h1>
        <h3> Spring and RESTful training </h3>
      </header>
    </div>
    <nav class="row">
```

► More Options

Check

- Validate by File Upload

Validate by File Upload

Upload a document for validation:

File: homePage.html

► More Options

Check