KickStart @ HTML5

A Solution to Easy Learning And Rapid Web Development

fosswithumesh.wordpress.com

Overview

- HTML5 is the latest and most enhanced version of HTML.
- HTML is not a programming language, but rather a markup language.
- HTML5 is a standard for structuring and presenting content on the World Wide Web.
- HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).

The latest versions of Apple Safari, Google Chrome, Mozilla Firefox, and Opera all support many HTML5 features and Internet Explorer 9.0 will also have support for some HTML5 functionality.

New Features

- **New Semantic Elements**: These are like <header>, <footer>, and <section>.
- **Forms 2.0**: Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- **Persistent Local Storage**: To achieve without resorting to third-party plugins.
- **WebSocket** : A next-generation bidirectional communication technology for web applications.
- **Canvas**: This supports a two-dimensional drawing surface that we can program with JavaScript.
- **Audio & Video**: we can embed audio or video on our web pages without resorting to third-party plugins.
- **Geolocation**: Now visitors can choose to share their physical location with our web application.
- **Drag and drop**: Drag and drop the items from one location to another location on a the same webpage.

Compatibility

- HTML5 is designed, as much as possible, to be backward compatible with existing web browsers.
- New features build on existing features and allow we to provide fallback content for older browsers.
- The HTML 5 language has a "custom" HTML syntax that is compatible with HTML 4 and XHTML1 documents published on the Web

Syntax Rules

- HTML 5 does not have the same syntax rules as XHTML where we needed lower case tag names, quoting our attributes, an attribute had to have a value and to close all empty elements.

- But HTML5 is coming with lots of flexibility and would support the followings:

  - ✓ Quotes are optional for attributes.
  - ✓ Attribute values are optional.
  - ✓ Closing empty elements are optional.
  - ✓ Uppercase tag names.

Elements

- HTML5 elements are marked up using start tags and end tags. Tags are delimited using angle brackets with the tag name in between.
- The difference between start tags and end tags is that the latter includes a slash before the tag name.

  Example: <p>...</p>
- HTML5 tag names are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.
- Most of the elements contain some content like <p>...</p> contains a paragraph.
- Elements forbidden from containing any content at all are known as void elements. For example, br, hr, link and meta etc.

Attributes

- Elements may contain attributes that are used to set various properties of an element.
- Some attributes are defined globally that can be used on any element
- All attributes have a name and a value.
- A div element with an attribute named class using a value of "example":

```
<div class="example">...</div>
```

- Attributes may only be specified within start tags and must never be used in end tags.
- HTML5 attributes are case insensitive and may be written in all uppercase or mixed case.

Extras

- DOCTYPEs in older versions of HTML were longer but HTML 5 authors would use simple syntax to specify DOCTYPE as follows:

```
<!DOCTYPE html>
```

- HTML 5 authors can use simple syntax to specify Character Encoding as follows:

```
<meta charset="UTF-8">
```

- It's common practice to add a type attribute with a value of "text/javascript" to script elements but HTML 5 removes extra information required and we can use simply following syntax:

```
<script src="scriptfile.js"></script>
```

- HTML 5 removes extra information(type) required for <link> Tag and we can use simply following syntax:

```
<link rel="stylesheet" href="stylefile.css">
```

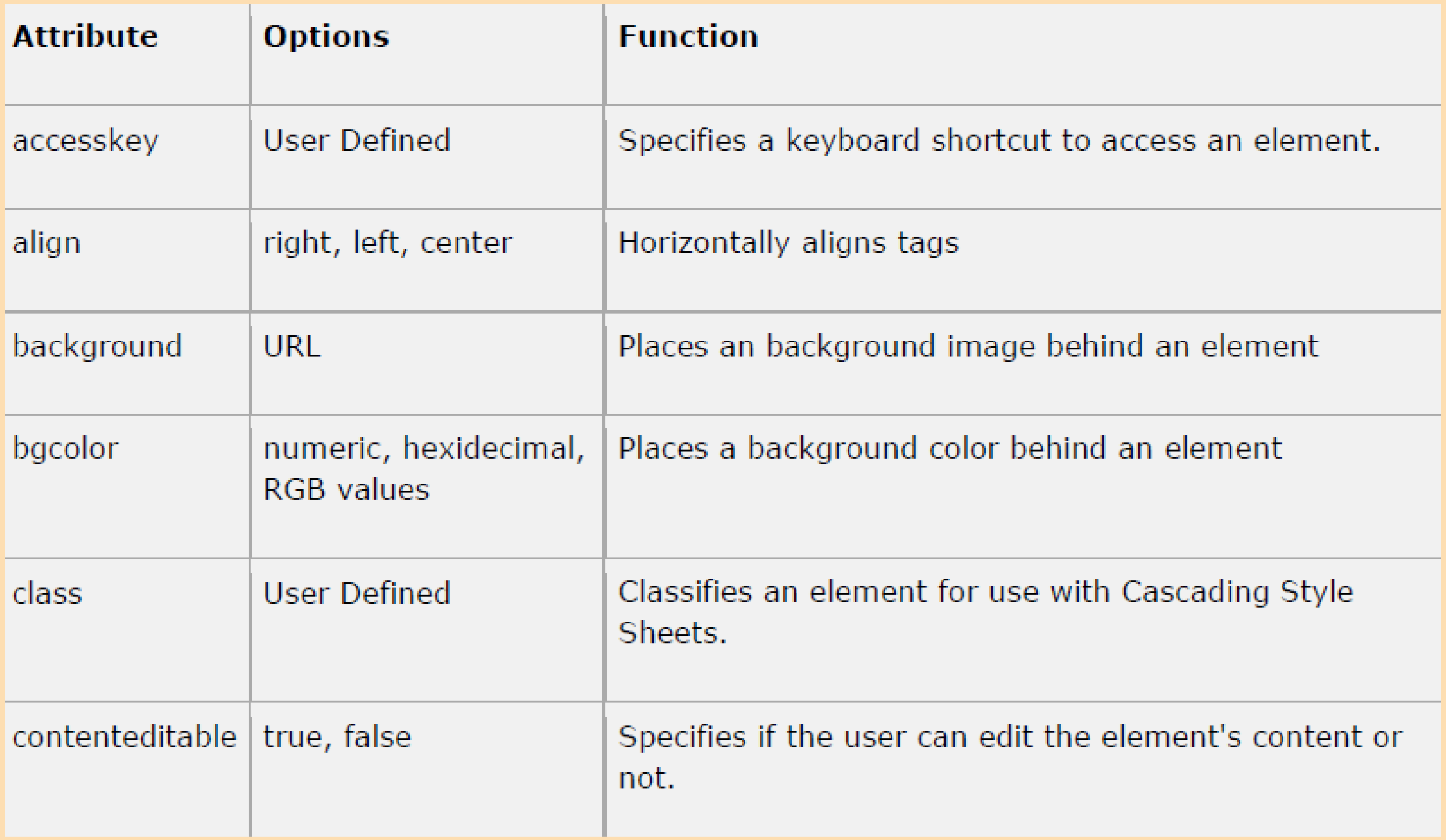
New Semantic Elements

- Section: This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- Article: This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- Aside: This tag represents a piece of content that is only slightly related to the rest of the page.
- Header: This tag represents the header of a section.
- Footer: This tag represents a footer for a section and can contain information about the author, copyright information, et cetera.
- Nav: This tag represents a section of the document intended for navigation.
- Dialog: This tag can be used to mark up a conversation.
- Figure: This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

Example

```
<!DOCTYPE html>
      <html>
            <head>
            <meta charset="utf-8">
            <title>...</title>
           </head>
      <body>
           <header>...</header>
            <nav>...</nav>
      <article>
            <section> ... </section>
       </article>
           <aside>...</aside>
           <footer>...</footer>
      </body>
      </html>
```

Standard Attributes



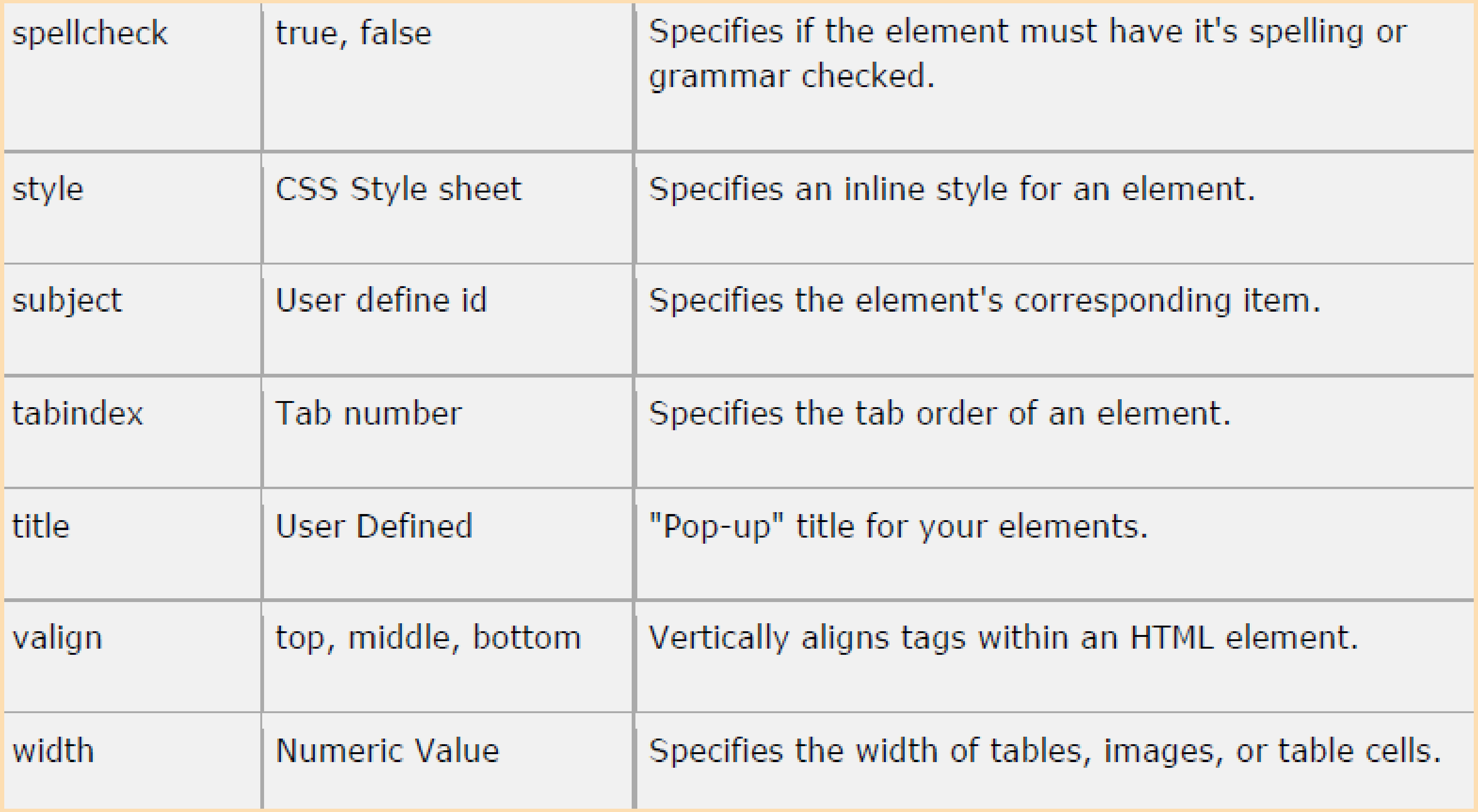
Attribute	Options	Function
accesskey	User Defined	Specifies a keyboard shortcut to access an element.
align	right, left, center	Horizontally aligns tags
background	URL	Places an background image behind an element
bgcolor	numeric, hexidecimal, RGB values	Places a background color behind an element
class	User Defined	Classifies an element for use with Cascading Style Sheets.
contenteditable	true, false	Specifies if the user can edit the element's content or not.

HTML

Standard Attributes

contextmenu	Menu id	Specifies the context menu for an element.
data-XXXX	User Defined	Custom attributes. Authors of a HTML document can define their own attributes. Must start with "data-".
draggable	true,false, auto	Specifies whether or not a user is allowed to drag an element.
height	Numeric Value	Specifies the height of tables, images, or table cells.
hidden	hidden	Specifies whether element should be visible or not.
id	User Defined	Names an element for use with Cascading Style Sheets.
item	List of elements	Used to group elements.
itemprop	List of items	Used to group items.

Standard Attributes



spellcheck	true, false	Specifies if the element must have it's spelling or grammar checked.
style	CSS Style sheet	Specifies an inline style for an element.
subject	User define id	Specifies the element's corresponding item.
tabindex	Tab number	Specifies the tab order of an element.
title	User Defined	"Pop-up" title for your elements.
valign	top, middle, bottom	Vertically aligns tags within an HTML element.
width	Numeric Value	Specifies the width of tables, images, or table cells.

HTML

Custom Attributes

- A new feature being introduced in HTML 5 is the addition of custom data attributes.
- A custom data attribute starts with data- and would be named based on your requirement. Following is the simple example:

```
<div class="example" data-subject="physics" data-level="complex"> ... </div>
```

- The above will be perfectly valid HTML5 with two custom attributes called data-subject and data-level.

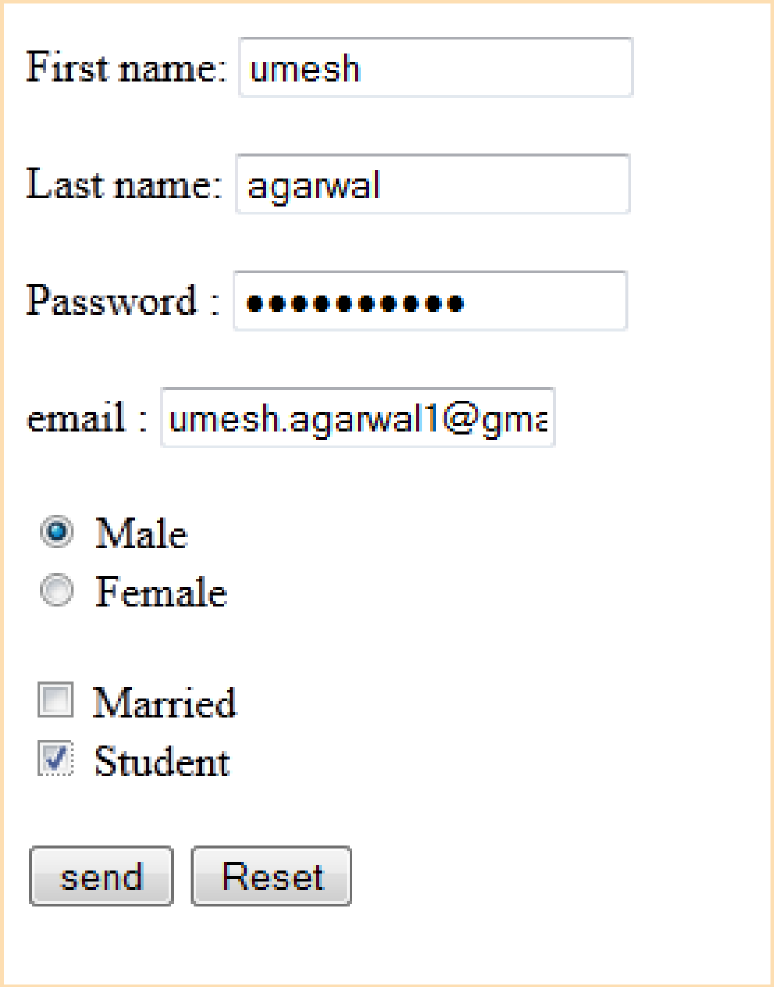
Old Input Elements

Type	Description
text	A free-form text field.
password	A free-form text field for sensitive information.
checkbox	A set of zero or more values from a predefined list.
radio	An enumerated value.
submit	A free form of button initiates form submission.
file	An arbitrary file with a MIME type and optionally a file name.
image	A coordinate, relative to a particular image's size.
hidden	An arbitrary string that is not normally displayed to the user.
select	An enumerated value, much like the radio type.
textarea	A free-form text field, nominally with no line break restrictions.
button	A free form of button which can initiates any event related to button.

Example

```
<!DOCTYPE HTML>
<html>
<body>
        </form>
        <form action="http://example.com/cgiscript.pl" method="post">
         <p>
                 <label for="firstname">First name: </label> <input type="text" id="firstname“>  <br ><br >
                  <label for="lastname">Last name: </label> <input type="text" id="lastname“>  <br ><br >
                 <label for="password">Password : </label> <input type="password" id="password“>  <br > <br >
                  <label for="email">email : </label> <input type="text" id="email“>  <br><br>
                 <input type="radio" name="sex" value="male"> Male  <br>
                 <input type="radio" name="sex" value="female"> Female   <br> <br>
                 <input type="checkbox" name="Married" value="1"> Married   <br>
                 <input type="checkbox" name="Student" value="2"> Student   <br> <br>
                 <input type="submit" value="send">
                 <input type="reset">
        </p>
        </form>
</body>
</html>
```

Output



First name: umesh

Last name: agarwal

Password : ●●●●●●●●●●●

email : umesh.agarwal1@gma

● Male
○ Female

☐ Married
☑ Student

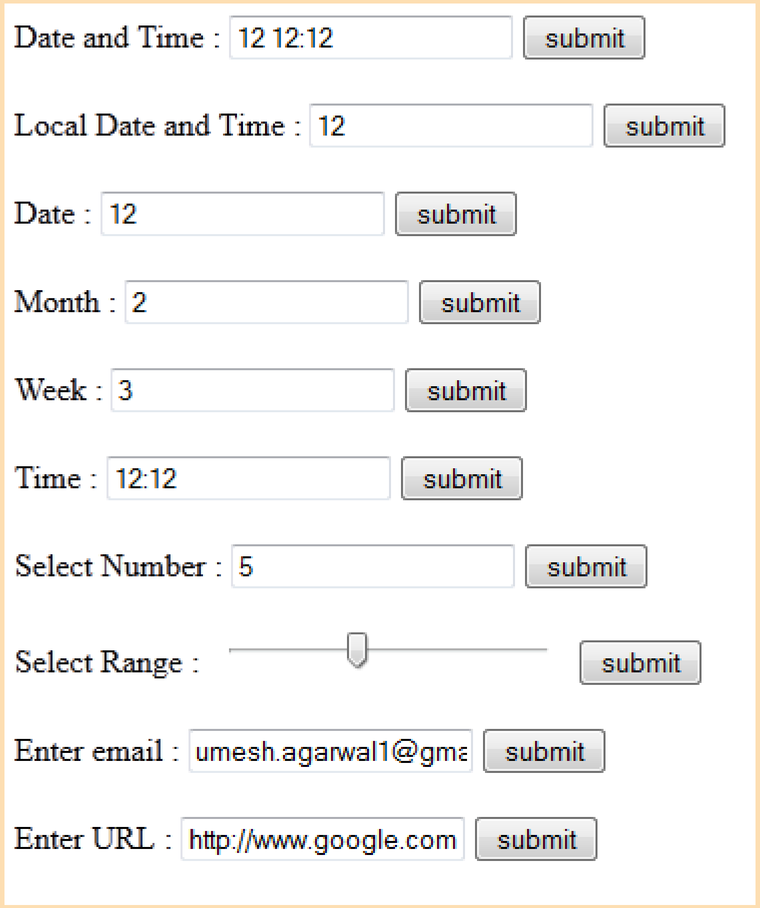
New Input Elements

Type	Description
datetime	A date and time (year, month, day, hour, minute, second, fractions of a second) encoded according to ISO 8601 with the time zone set to UTC.
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.
date	A date (year, month, day) encoded according to ISO 8601.
month	A date consisting of a year and a month encoded according to ISO 8601.
week	A date consisting of a year and a week number encoded according to ISO 8601.
time	A time (hour, minute, seconds, fractional seconds) encoded according to ISO 8601.
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.
url	This accepts only URL value. This type is used for input fields that should contain a URL address. If you try to submit a simple text, it forces to enter only URL address either in http://www.example.com format or in http://example.com format.

Example

```
<!DOCTYPE HTML>
<html>
<body>          <form>
                Date and Time : <input type="datetime" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Local Date and Time : <input type="datetime-local" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Date : <input type="date" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Month : <input type="month" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Week : <input type="week" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Time : <input type="time" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Select Number : <input type="number" min="0" max="10" step "1" value="5" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Select Range : <input type="range" min="0" max="10" step "1" value="5" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Enter email : <input type="email" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                Enter URL : <input type="url" name="newinput" />
                            <input type="submit" value="submit" /> <br><br>
                </form>
</body>
</html>
```

Output



Date and Time : 12 12:12 submit

Local Date and Time : 12 submit

Date : 12 submit

Month : 2 submit

Week : 3 submit

Time : 12:12 submit

Select Number : 5 submit

Select Range : submit

Enter email : umesh.agarwal1@gma submit

Enter URL : http://www.google.com submit

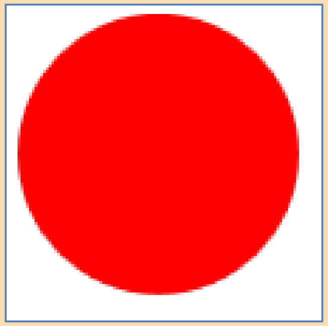
SVG

- SVG stands for **Scalable Vector Graphics** and it is a language for describing 2D-graphics and graphical applications in XML and the XML is then rendered by an SVG viewer.
- SVG is mostly useful for vector type diagrams like Pie charts, Two-dimensional graphs in an X,Y coordinate system etc.
- Most of the web browsers can display SVG just like they can display PNG, GIF, and JPG.
- HTML5 allows embeding SVG directly using <svg>...</svg> tag which has following simple syntax:

```
<svg xmlns="http://www.w3.org/2000/svg"> ..... </svg>
```

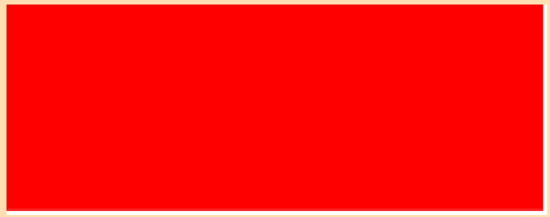
SVG Circle

```
<!DOCTYPE html>
    <head>
    <title>SVG</title>
    <meta charset="utf-8" />
    </head>
<body>
    <svg id="svgelem" height="200"
    xmlns="http://www.w3.org/2000/svg">
    <circle id="redcircle" cx="50" cy="50"
    r="50" fill="red" />
    </svg>
</body>
</html>
```



SVG Rectangle

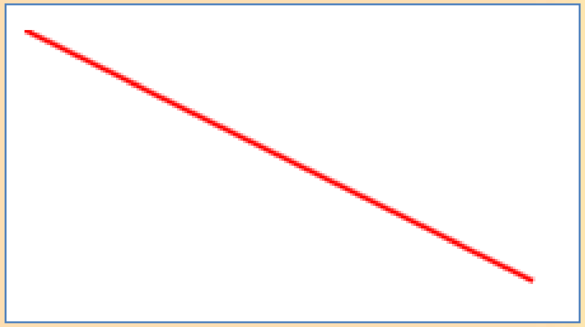
```
<!DOCTYPE html>
    <head>
    <title>SVG</title>
    <meta charset="utf-8" />
    </head>
<body>
    <svg id="svgelem" height="200"
    xmlns="http://www.w3.org/2000/svg">
    <rect id="redrect" width="300"
    height="100" fill="red" />
    </svg>
</body>
</html>
```



fosswithumesh.wordpress.com

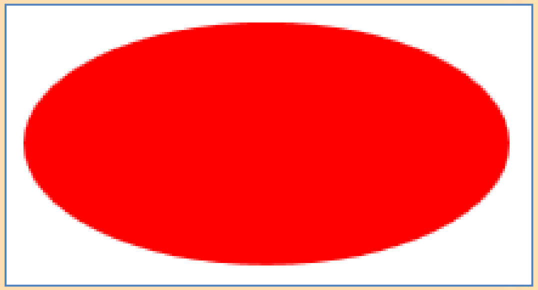
SVG Line

```
<!DOCTYPE html>
    <head>
    <title>SVG</title>
    <meta charset="utf-8" />
    </head>
<body>
    <svg id="svgelem" height="200"
    xmlns="http://www.w3.org/2000/svg">
    <line x1="0" y1="0" x2="200" y2="100“
    style="stroke:red;stroke-width:2"/>
    </svg>
</body>
</html>
```



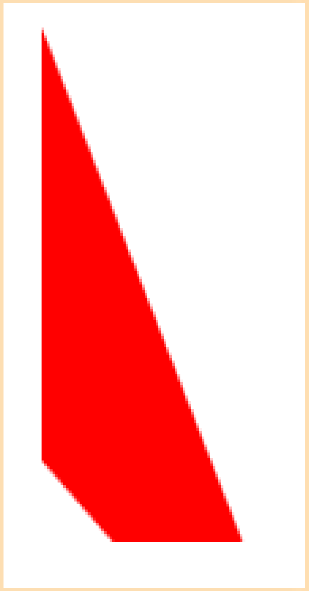
SVG Ellipse

```
<!DOCTYPE html>
    <head>
    <title>SVG</title>
    <meta charset="utf-8" />
    </head>
<body>
    <svg id="svgelem" height="200"
    xmlns="http://www.w3.org/2000/svg">
    <ellipse cx="100" cy="50" rx="100" ry="50"
    fill="red" />
    </svg>
</body>
</html>
```



SVG Polygon

```
<!DOCTYPE html>
    <head>
    <title>SVG</title>
    <meta charset="utf-8" />
    </head>
<body>
    <svg id="svgelem" height="200"
    xmlns="http://www.w3.org/2000/svg">
    <polygon points="10,10,123,300,10, 170,50"
    fill="red" />
    </svg>
</body>
</html>
```



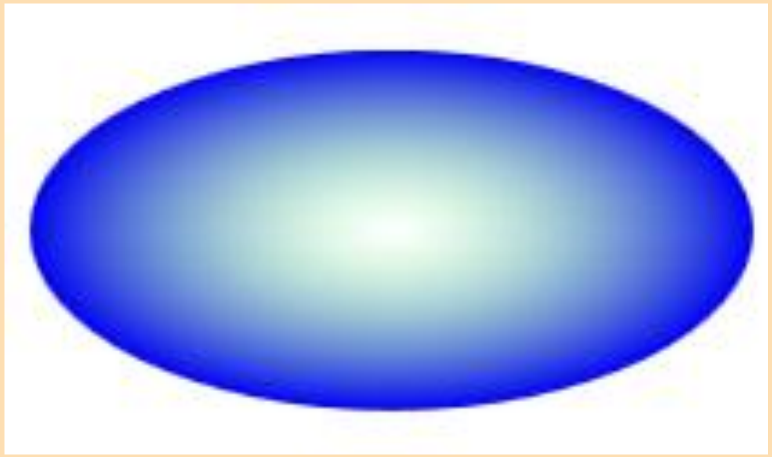
HTML 5

fosswithumesh.wordpress.com

SVG Gradients

```
<!DOCTYPE html>
    <head>
    <title>SVG</title>
    <meta charset="utf-8" />
    </head>
<body>
    <svg id="svgelem" height="200" xmlns="http://www.w3.org/2000/svg">
          <defs>
    <radialGradient id="gradient" cx="50%" cy="50%" r="50%" fx="50%" fy="50%">
          <stop offset="0%" style="stop-color:rgb(200,200,200); stop-opacity:0"/>
          <stop offset="100%" style="stop-color:rgb(0,0,255); stop-opacity:1"/>
    </radialGradient>
          </defs>
    <ellipse cx="100" cy="50" rx="100" ry="50" style="fill:url(#gradient)" />
    </svg>
</body>
</html>
```

Output



Canvas

- HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript.
- It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.
- Here is a simple <canvas> element which has only two specific attributes width and height plus all the core HTML5 attributes like id, name and class etc.

```
<canvas id="mycanvas" width="100" height="100"></canvas>
```

- The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it.
- The canvas element has a DOM method called getContext, used to obtain the rendering context and its drawing functions. This function takes one parameter, the type of context 2d.
- The latest versions of Firefox, Safari, Chrome and Opera all support for HTML5 Canvas

Canvas Examples

- Drawing Rectangles
- Drawing Paths
- Drawing Lines
- Drawing Bezier
- Drawing Quadratic
- Using Images
- Create Gradients
- Styles and Colors
- Text and Fonts
- Pattern and Shadow
- Canvas States
- Canvas Translation
- Canvas Rotation
- Canvas Scaling
- Canvas Transform
- Canvas Composition
- Canvas Animation

Drawing Rectangles

- There are three methods that draw rectangles on the canvas:

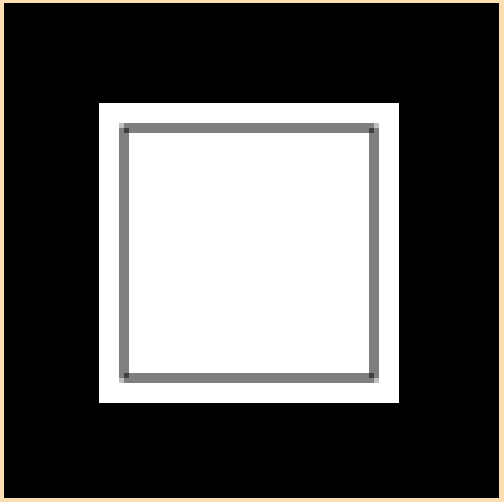
SN	Method and Description
1	**fillRect(x,y,width,height)** This method draws a filled rectangle
2	**strokeRect(x,y,width,height)** This method draws a rectangular outline
3	**clearRect(x,y,width,height)** This method clears the specified area and makes it fully transparent

- Here x and y specify the position on the canvas (relative to the origin) of the top-left corner of the rectangle and *width* and *height* are width and height of the rectangle.

Example

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">
function drawShape(){
        // Get the canvas element using the DOM
         var canvas = document.getElementById('mycanvas');
// Make sure we don't execute when canvas isn't supported
        if (canvas.getContext){
// use getContext to use the canvas for drawing
         var ctx = canvas.getContext('2d');
                // Draw shapes
                ctx.fillRect(25,25,100,100);
                ctx.clearRect(45,45,60,60);
                ctx.strokeRect(50,50,50,50);
                } else {
                alert('You need Safari or Firefox 1.5+ to see this demo.');
        }
 }
</script>
 </head>
 <body onload="drawShape();">
        <canvas id="mycanvas"></canvas>
 </body>
 </html>
```

Output



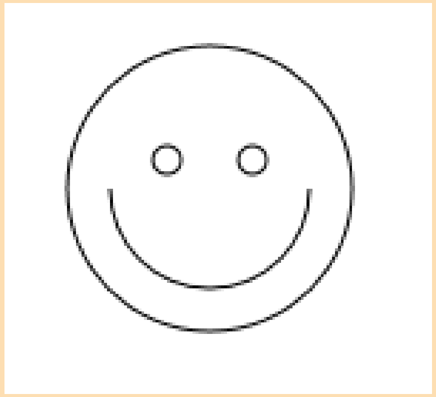
Drawing Paths

SN	Method and Description
1	**beginPath()** This method resets the current path.
2	**moveTo(x, y)** This method creates a new subpath with the given point.
3	**closePath()** This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath.
4	**fill()** This method fills the subpaths with the current fill style.
5	**stroke()** This method strokes the subpaths with the current stroke style.
6	**arc(x, y, radius, startAngle, endAngle, anticlockwise)** Adds points to the subpath such that the arc described by the circumference of the circle described by the arguments, starting at the given start angle and ending at the given end angle, going in the given direction, is added to the path, connected to the previous point by a straight line.

HTML

Example

```
 // Draw shapes
  ctx.beginPath();
 ctx.arc(75,75,50,0,Math.PI*2,true);  // Outer circle
ctx.moveTo(110,75);
  ctx.arc(75,75,35,0,Math.PI,false);   // Mouth
 ctx.moveTo(65,65);
 ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
  ctx.moveTo(95,65);
ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
  ctx.stroke();
```



fosswithumesh.wordpress.com

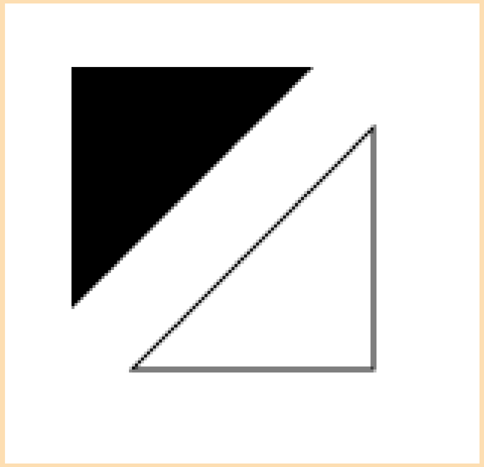
Drawing Lines

Line Methods:

- beginPath()
- moveTo(x, y)
- closePath()
- fill()
- stroke()
- **arc(x, y, radius, startAngle, endAngle, anticlockwise)**
  Adds points to the sub path such that the arc described by the circumference of the circle described by the arguments, starting at the given start angle and ending at the given end angle, going in the given direction, is added to the path, connected to the previous point by a straight line.

Example

```
// Filled triangle
   ctx.beginPath();
   ctx.moveTo(25,25);
   ctx.lineTo(105,25);
   ctx.lineTo(25,105);
   ctx.fill
// Stroked triangle
   ctx.beginPath();
   ctx.moveTo(125,125);
   ctx.lineTo(125,45);
   ctx.lineTo(45,125);
   ctx.closePath();
   ctx.stroke();
```



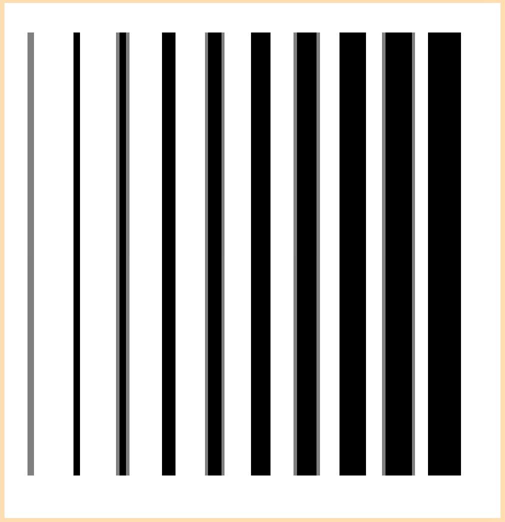
Drawing Lines

Line Properties

SN	Property and Description
1	**lineWidth [= value]** This property returns the current line width and can be set, to change the line width.
2	**lineCap [= value]** This property returns the current line cap style and can be set, to change the line cap style. The possible line cap styles are butt, round, and square
3	**lineJoin [= value]** This property returns the current line join style and can be set, to change the line join style. The possible line join styles are bevel, round, and miter.
4	**miterLimit [= value]** This property returns the current miter limit ratio and can be set, to change the miter limit ratio.

Example

```
 for (i=0;i<10;i++)
   {
    ctx.lineWidth = 1+i;
    ctx.beginPath();
ctx.moveTo(5+i*14,5);
    ctx.lineTo(5+i*14,140);
    ctx.stroke();
   }
```



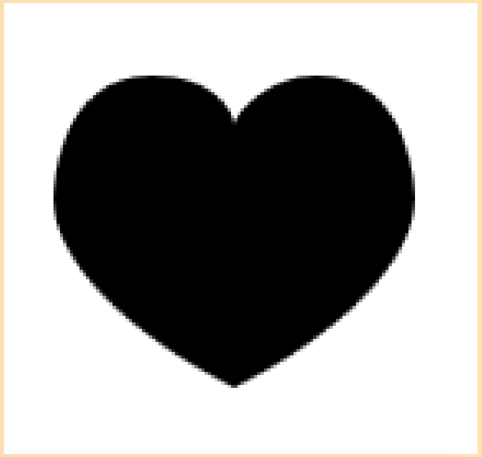
Drawing Bezier Curves

- beginPath()
- moveTo(x, y)
- closePath()
- fill()
- stroke()
- **bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)**

  This method adds the given point to the current path, connected to the previous one by a cubic Bezier curve with the given control points.
- The x and y parameters in bezierCurveTo() method are the coordinates of the end point. cp1x and cp1y are the coordinates of the first control point, and cp2x and cp2y are the coordinates of the second control point.

Example

```
ctx.beginPath();
ctx.moveTo(75,40);
ctx.bezierCurveTo(75,37,70,25,50,25);
ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
ctx.bezierCurveTo(20,80,40,102,75,120);
ctx.bezierCurveTo(110,102,130,80,130,62.5);
ctx.bezierCurveTo(130,62.5,130,25,100,25);
ctx.bezierCurveTo(85,25,75,37,75,40);
ctx.fill();
```

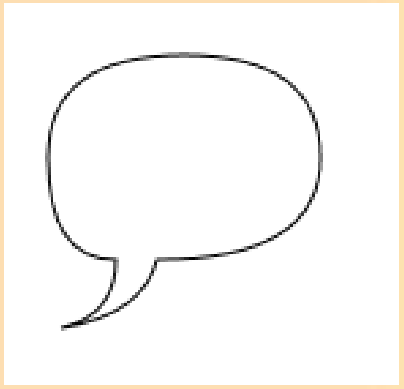


Drawing Quadratic Curves

- beginPath()
- moveTo(x, y)
- closePath()
- fill()
- stroke()
- **quadraticCurveTo(cpx, cpy, x, y)**
  This method adds the given point to the current path, connected to the previous one by a quadratic Bezier curve with the given control point.
- The x and y parameters in quadraticCurveTo() method are the coordinates of the end point. cpx and cpy are the coordinates of the control point.

Example

```
ctx.beginPath();
ctx.moveTo(75,25);
ctx.quadraticCurveTo(25,25,25,62.5);
ctx.quadraticCurveTo(25,100,50,100);
ctx.quadraticCurveTo(50,120,30,125);
ctx.quadraticCurveTo(60,120,65,100);
ctx.quadraticCurveTo(125,100,125,62.5);
ctx.quadraticCurveTo(125,25,75,25);
ctx.stroke();
```

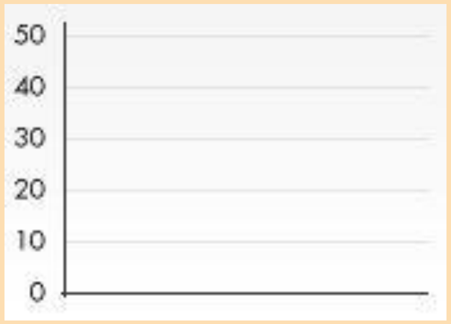


Using Images

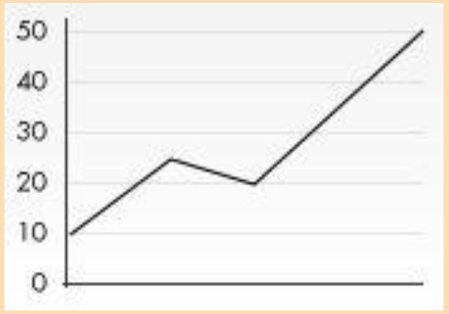
- beginPath()
- moveTo(x, y)
- closePath()
- fill()
- stroke()
- **drawImage(image, dx, dy)**
  This method draws the given image onto the canvas. Here *image* is a reference to an image or canvas object. x and y form the coordinate on the target canvas where our image should be placed.

Example

```
var img = new Image();
img.src = '/images/backdrop.jpg';
img.onload = function(){
  ctx.drawImage(img,0,0);
  ctx.beginPath();
  ctx.moveTo(30,96);
  ctx.lineTo(70,66);
  ctx.lineTo(103,76);
  ctx.lineTo(170,15);
  ctx.stroke();
```



images/backdrop.jpg.



Gradients

- **addColorStop(offset, color)**

  This method adds a color stop with the given color to the gradient at the given offset. Here 0.0 is the offset at one end of the gradient, 1.0 is the offset at the other end.
- **createLinearGradient(x0, y0, x1, y1)**

  This method returns a CanvasGradient object that represents a linear gradient that paints along the line given by the coordinates represented by the arguments. The four arguments represent the starting point (x1,y1) and end point (x2,y2) of the gradient.
- **createRadialGradient(x0, y0, r0, x1, y1, r1)**

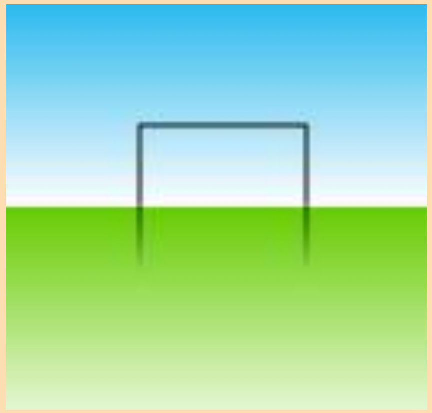
  This method returns a CanvasGradient object that represents a radial gradient that paints along the cone given by the circles represented by the arguments. The first three arguments define a circle with coordinates (x1,y1) and radius r1 and the second a circle with coordinates (x2,y2) and radius r2.

Linear Gradient Example

```
// Create Linear Gradients
   var lingrad = ctx.createLinearGradient(0,0,0,150);
   lingrad.addColorStop(0, '#00ABEB');
   lingrad.addColorStop(0.5, '#fff');
   lingrad.addColorStop(0.5, '#66CC00');
   lingrad.addColorStop(1, '#fff');

   var lingrad2 = ctx.createLinearGradient(0,50,0,95);
   lingrad2.addColorStop(0.5, '#000');
   lingrad2.addColorStop(1, 'rgba(0,0,0,0)');

   // assign gradients to fill and stroke styles
   ctx.fillStyle = lingrad;
   ctx.strokeStyle = lingrad2;
   // draw shapes
   ctx.fillRect(10,10,130,130);
   ctx.strokeRect(50,50,50,50);
```

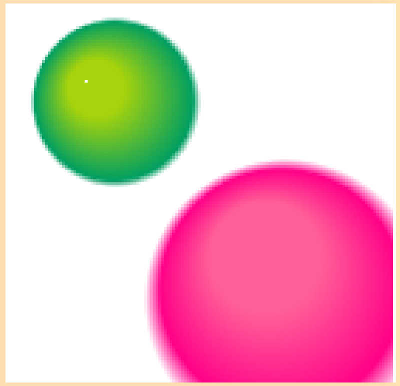


fosswithumesh.wordpress.com

Radial Gradient Example

```
// Create gradients
   var radgrad =
      ctx.createRadialGradient(45,45,10,52,50,30);
   radgrad.addColorStop(0, '#A7D30C');
   radgrad.addColorStop(0.9, '#019F62');
   radgrad.addColorStop(1, 'rgba(1,159,98,0)');

   var radgrad2 =
      ctx.createRadialGradient(105,105,20,112,120,50);
   radgrad2.addColorStop(0, '#FF5F98');
   radgrad2.addColorStop(0.75, '#FF0188');
   radgrad2.addColorStop(1, 'rgba(255,1,136,0)');
// draw shapes
   ctx.fillStyle = radgrad2;
   ctx.fillRect(0,0,150,150);
   ctx.fillStyle = radgrad;
   ctx.fillRect(0,0,150,150);
```



fosswithumesh.wordpress.com

Styles and Colors

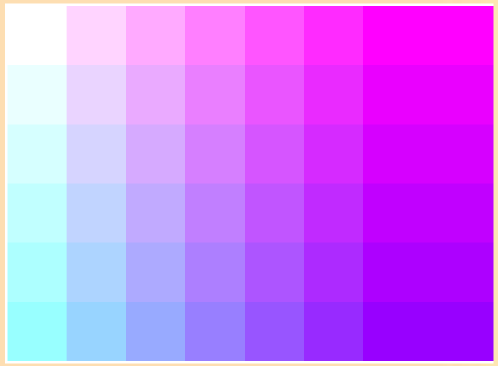
- **fillStyle**

  This attribute represents the color or style to use inside the shapes.
- **strokeStyle**

  This attribute represents the color or style to use for the lines around shapes.
- By default, the stroke and fill color are set to black which is CSS color value #000000.

A fillStyle Example

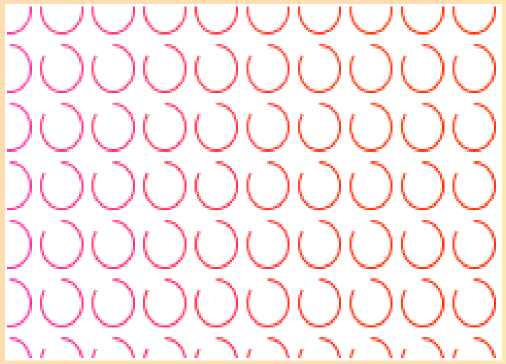
```
for (var i=0;i<7;i++)
{
      for (var j=0;j<7;j++)
{
       ctx.fillStyle='rgb(' + Math.floor(255-20.5*i)+ ','+
                       Math.floor(255 - 42.5*j) + ',255)';
        ctx.fillRect( j*25, i* 25, 55, 55 );
}
}
```



fosswithumesh.wordpress.com

A strokeStyle Example

```
for (var i=0;i<10;i++){
     for (var j=0;j<10;j++){
      ctx.strokeStyle='rgb(255,'+ Math.floor(50-2.5*i)+','+
                   Math.floor(155 - 22.5 * j ) + ')';
      ctx.beginPath();
      ctx.arc(1.5+j*25, 1.5 + i*25,10,10,Math.PI*5.5, true);
      ctx.stroke();
     }
   }
```



fosswithumesh.wordpress.com

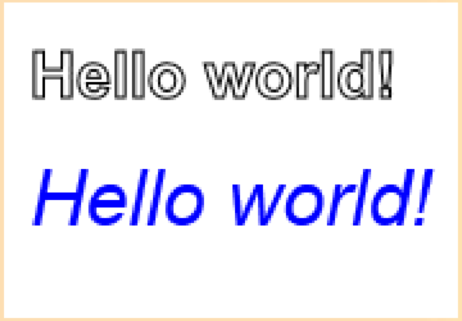
Text and Fonts

SN	Property and Description
1	**font [= value]** This property returns the current font settings and can be set, to change the font.
2	**textAlign [= value]** This property returns the current text alignment settings and can be set, to change the alignment. The possible values are start, end, left, right, and center.
3	**textBaseline [= value]** This property returns the current baseline alignment settings and can be set, to change the baseline alignment. The possible values are top, hanging, middle , alphabetic, ideographic and bottom
4	**fillText(text, x, y [, maxWidth])** This property fills the given text at the given position indicated by the given coordinates x and y.
5	**strokeText(text, x, y [, maxWidth])** This property strokes the given text at the given position indicated by the given coordinates x and y.

Example

```
ctx.fillStyle    = '#00F';
ctx.font         = 'Italic 30px Sans-Serif';
ctx.textBaseline = 'Top';
ctx.fillText  ('Hello world!', 40, 100);

ctx.font         = 'Bold 25px Sans-Serif';
ctx.strokeText('Hello world!', 40, 50);
```



fosswithumesh.wordpress.com

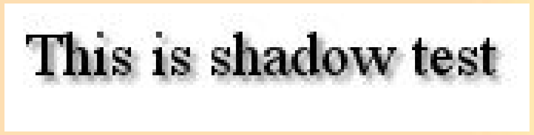
Create Shadows

SN	Property and Description
1	**shadowColor [= value]** This property returns the current shadow color and can be set, to change the shadow color.
2	**shadowOffsetX [= value]** This property returns the current shadow offset X and can be set, to change the shadow offset X.
3	**shadowOffsetY [= value]** This property returns the current shadow offset Y and can be set, change the shadow offset Y.
4	**shadowBlur [= value]** This property returns the current level of blur applied to shadows and can be set, to change the blur level.

Example

```
ctx.shadowOffsetX = 2;
ctx.shadowOffsetY = 2;
ctx.shadowBlur = 2;
ctx.shadowColor = "rgba(0, 0, 0, 0.5)";

ctx.font = "20px Times New Roman";
ctx.fillStyle = "Black";
ctx.fillText("This is shadow test", 5, 30);
```


This is shadow test

Canvas - Composition

- HTML5 canvas provides compositing attribute **globalCompositeOperation** which affect all the drawing operations.
- We can draw new shapes behind existing shapes and mask off certain areas, clear sections from the canvas using **globalCompositeOperation** attribute.

Attribute	Description
source-over	This is the default setting and draws new shapes on top of the existing canvas content.
source-in	The new shape is drawn only where both the new shape and the destination canvas overlap. Everything else is made transparent.
source-out	The new shape is drawn where it doesn't overlap the existing canvas content.
source-atop	The new shape is only drawn where it overlaps the existing canvas content.

Canvas - Composition

Attribute	Description
lighter	Where both shapes overlap the color is determined by adding color values.
xor	Shapes are made transparent where both overlap and drawn normal everywhere else.
destination-over	New shapes are drawn behind the existing canvas content.
destination-in	The existing canvas content is kept where both the new shape and existing canvas content overlap. Everything else is made transparent.
destination-out	The existing content is kept where it doesn't overlap the new shape.
destination-atop	The existing canvas is only kept where it overlaps the new shape. The new shape is drawn behind the canvas content.
darker	Where both shapes overlap the color is determined by subtracting color values.

Example

```
<!DOCTYPE HTML>
<html>
<head>
<script type="text/javascript">

var compositeTypes = [
  'source-over','source-in','source-out','source-atop',
  'destination-over','destination-in','destination-out',
  'destination-atop','lighter','darker','copy','xor'
];

function drawShape(){
  for (i=0;i<compositeTypes.length;i++){
    var label = document.createTextNode(compositeTypes[i]);
    document.getElementById('lab'+i).appendChild(label);
    var ctx = document.getElementById('tut'+i).getContext('2d');

    // draw rectangle
    ctx.fillStyle = "#FF3366";
    ctx.fillRect(15,15,70,70);

    // set composite property
    ctx.globalCompositeOperation = compositeTypes[i];

    // draw circle
    ctx.fillStyle = "#0066FF";
    ctx.beginPath();
    ctx.arc(75,75,35,0,Math.PI*2,true);
    ctx.fill();
  }
}
```

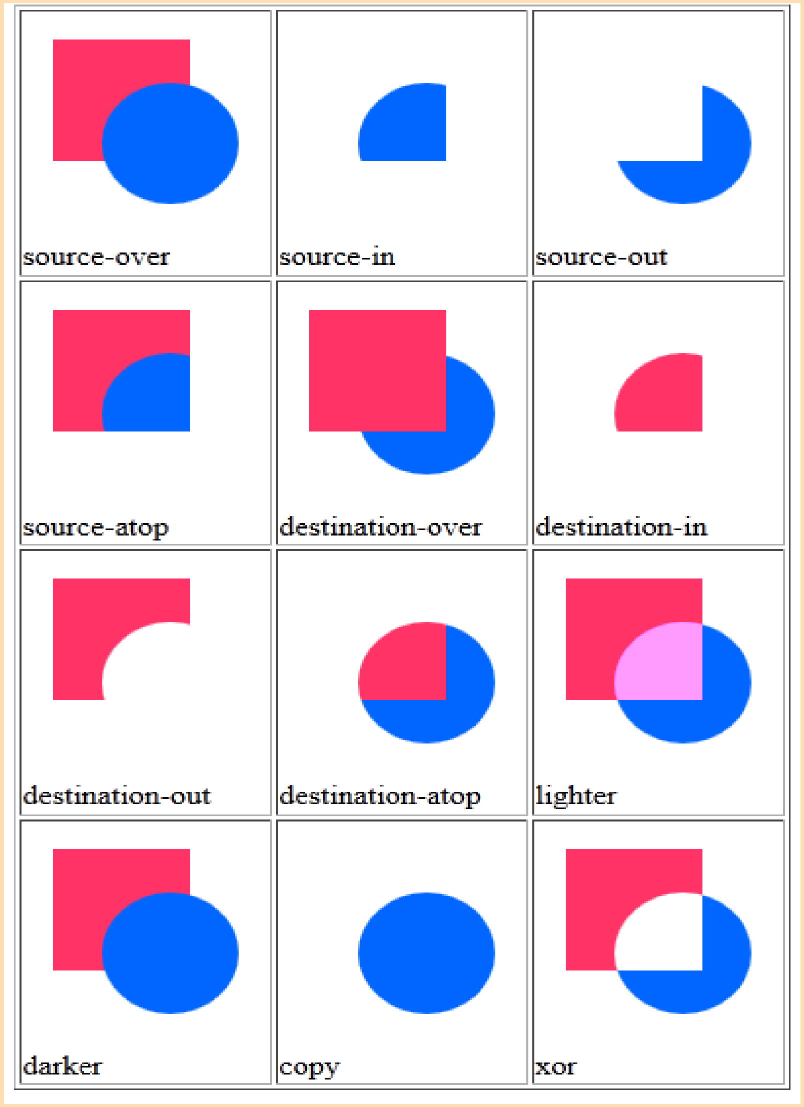
Example

```
</script>
</head>
<body onload="drawShape();">
<table border="1" align="center">
<tr>
 <td><canvas id="tut0" width="125" height="125"></canvas><br/>
    <label id="lab0"></label>
 </td>
 <td><canvas id="tut1" width="125" height="125"></canvas><br/>
    <label id="lab1"></label>
 </td>
 <td><canvas id="tut2" width="125" height="125"></canvas><br/>
    <label id="lab2"></label>
 </td>
</tr>
<tr>
 <td><canvas id="tut3" width="125" height="125"></canvas><br/>
    <label id="lab3"></label>
 </td>
 <td><canvas id="tut4" width="125" height="125"></canvas><br/>
    <label id="lab4"></label>
 </td>
 <td><canvas id="tut5" width="125" height="125"></canvas><br/>
    <label id="lab5"></label>
 </td>
</tr>
<tr>
```

Example

```
 <td><canvas id="tut6" width="125" height="125"></canvas><br/>
    <label id="lab6"></label>
 </td>
 <td><canvas id="tut7" width="125" height="125"></canvas><br/>
    <label id="lab7"></label>
 </td>
 <td><canvas id="tut8" width="125" height="125"></canvas><br/>
    <label id="lab8"></label>
</tr>
<tr>
 </td>
 <td><canvas id="tut9" width="125" height="125"></canvas><br/>
    <label id="lab9"></label>
 </td>
 <td><canvas id="tut10" width="125" height="125"></canvas><br/>
    <label id="lab10"></label>
 </td>
 <td><canvas id="tut11" width="125" height="125"></canvas><br/>
    <label id="lab11"></label>
 </td>
</tr>
</table>
</body>
</html>
```

Output


source-over
source-in
source-out
source-atop
destination-over
destination-in
destination-out
destination-atop
lighter
darker
copy
xor

Thank You For Your Patience!

Thank You!

HTML

fosswithumesh.wordpress.com

Queries?

https://twitter.com/umesh_agarwal1

https://fosswithumesh.wordpress.com

http://www.facebook.com/umeshagarwal007

https://github.com/umeshagarwal

umesh.agarwal1@gmail.com

Irc Nick - umesh