Internet Programming

Drawing and Canvas Element

Dan Brandon, Ph.D., PMP

Before HTML5

- Before HTML5, web programmers used tools such as applets and Flash to draw graphics objects and animate them
- Applets are small Java programs that embed inside web pages
- Sun Microsystems developed the Java, but Oracle Corporation acquired Sun in 2010
- In 2011, Adobe (Flash's owner) conceded that Flash was perhaps not the best user experience for mobile devices
- In 2016 Adobe, changed Flash's name to Animate to distance itself from the less-than-stellar reputation associated with Flash players
- Applets and Flash were not built into browser software
- So to use them, it meant end users were expected to install add-on software to their browsers
- That worked, but it was an annoyance; and to make matters worse, applets and Flash were known to be security risks

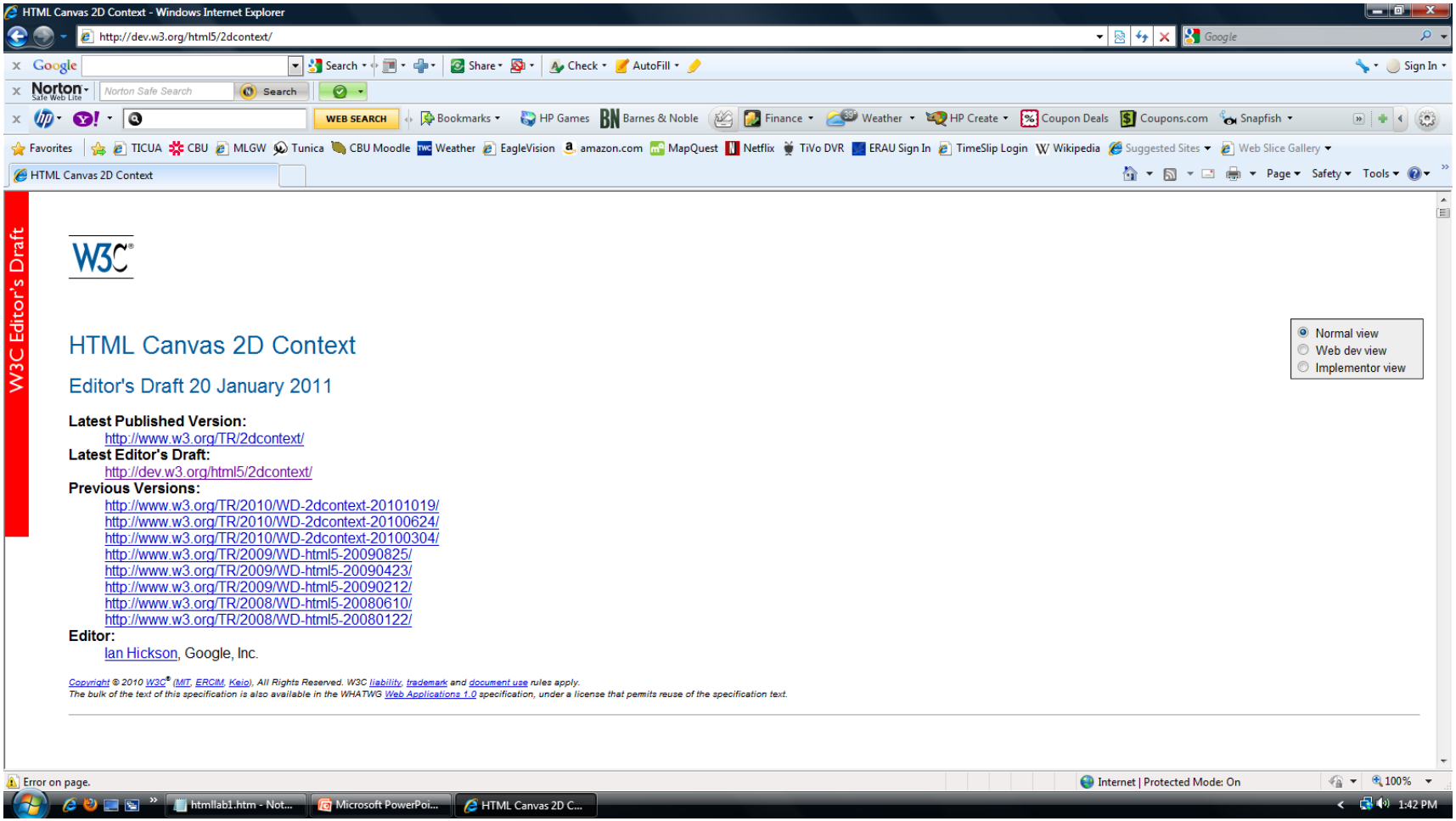
Flash vs HTML5 Drawing

- In designing HTML5, the WHATWG and W3C added Canvas to HTML
- Canvas allows you to draw graphics objects (lines, rectangles, circles, etc.) and animate them, but unlike applets and Flash, canvas is built into all modern browsers, so there's no add-on software
- Canvas was and is considered to be one of HTML5's most significant new features, and it has quickly gained widespread acceptance
- Canvas's acceptance is illustrated by Oracle phasing out its support of applets and Adobe phasing out its support of Flash
- Also, most browsers no longer support applets, and browsers now turn off Flash content by default
- HTML5 allows programmers to incorporate neat effects for Websites and apps on their desktops, tablets and smartphones without tailoring apps for specific hardware or online stores

HTML5 Canvas Element

- HTML syntax:
  - <canvas id="my_canvas" height="yyy" width="xxx">
    - Sorry - This page uses <code>canvas</code> and your browser doesn't support it.
  - </canvas>
  - Can use CSS positioning (or tables) to "place" your canvas on the HTML page, and set styles for the canvas
    - style="border:1px solid #d3d3d3;"
- Current browser support:
  - Chrome, Firefox, Opera, Safari, IE 9+
- Drawing accomplished via JavaScript
- Drawing objects: lines, arcs, shapes, images, and text
- Canvas API:
  - http://dev.w3.org/html5/2dcontext/

http://dev.w3.org/html5/2dcontext/



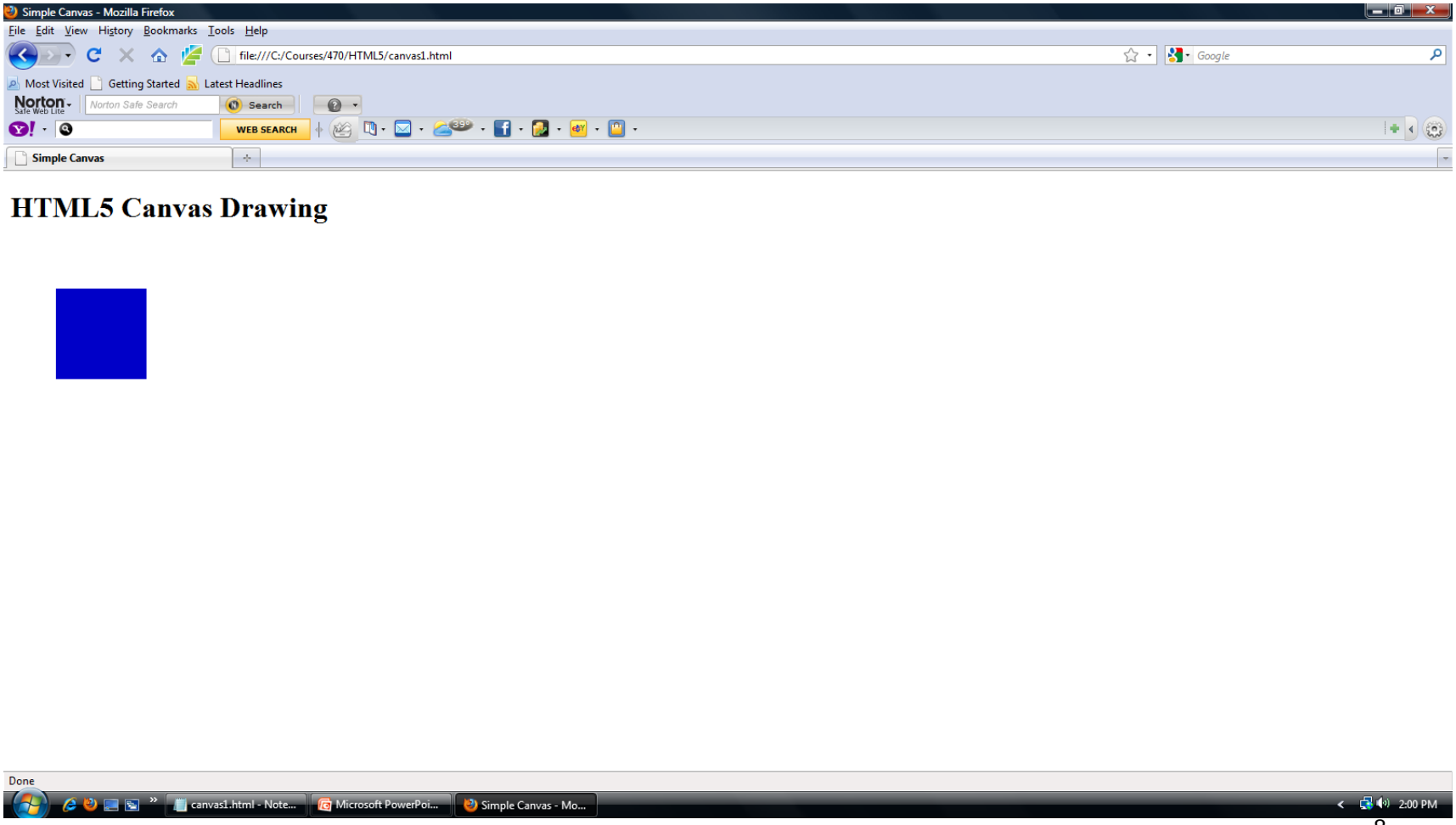
JavaScript Specification Syntax

- object.function
  - my_canvas.fillRect(x1, x2, x3, x4);
- object.attribute
  - my_canvas.fillstyle=xxxx;
- Documentation: attribute (argument) type specification:
  - Attribute type name // values (default)
    - Attribute DOMString lineCap; // “butt”, “round”, “square” (default “butt”)
  - Types:
    - any, DOMString, float, boolean

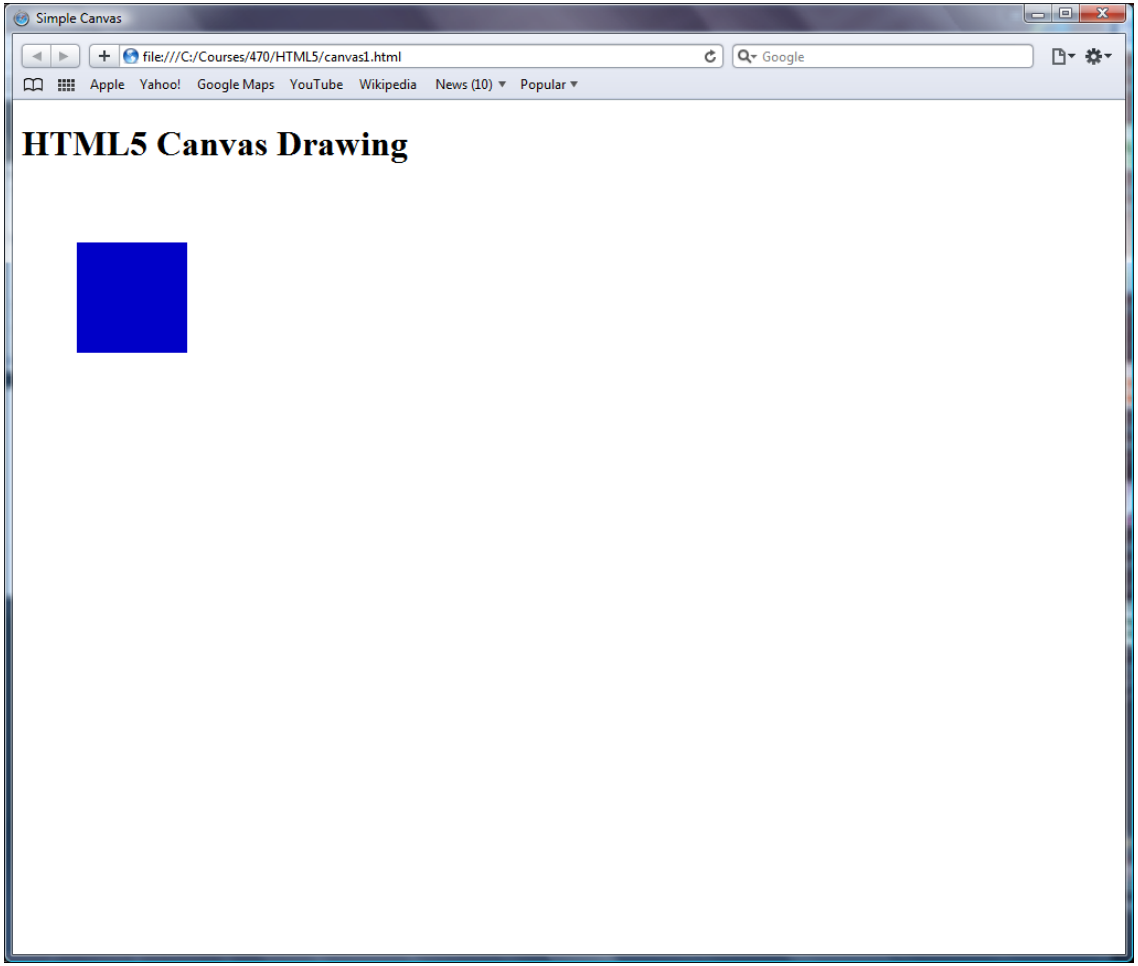
Simple Example

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>Simple Canvas</title>
                <script type="text/javascript">
                        function init() {
                                var x = document.getElementById("my_canvas");
                                var y = x.getContext("2d");
                                y.fillStyle = "rgba(0, 0, 200, 1)";
                                y.fillRect(50, 50, 100, 100);
                        }
                </script>
        </head>
        <body onload="init()">
                <h1>HTML5 Canvas Drawing</h1>
                <canvas id="my_canvas" width="500" height="500"></canvas>
        </body>
</html>
```

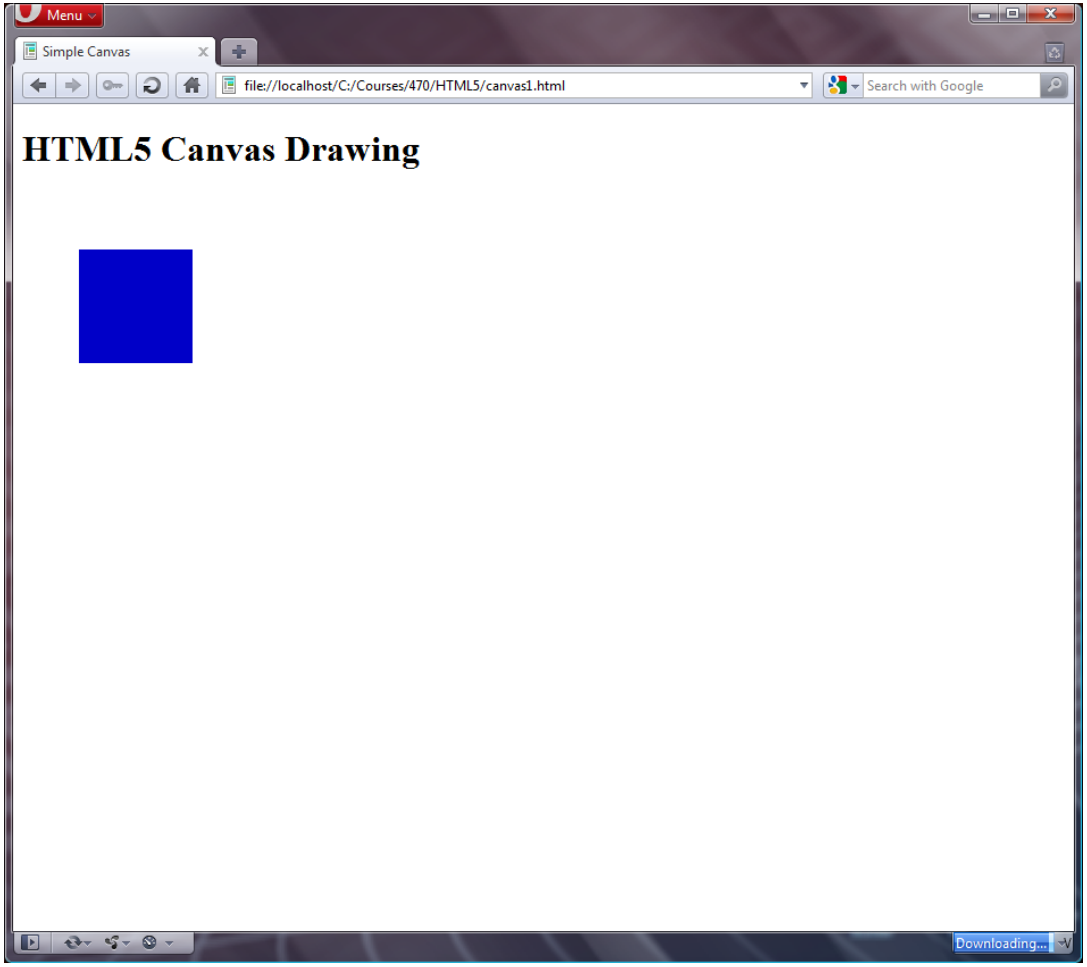
Firefox


Simple Canvas - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Courses/470/HTML5/canvas1.html
Most Visited Getting Started Latest Headlines
Norton Safe Web Lite
Norton Safe Search
Search
WEB SEARCH
Simple Canvas
HTML5 Canvas Drawing
Done
canvas1.html - Note...
Microsoft PowerPoi...
Simple Canvas - Mo...
2:00 PM

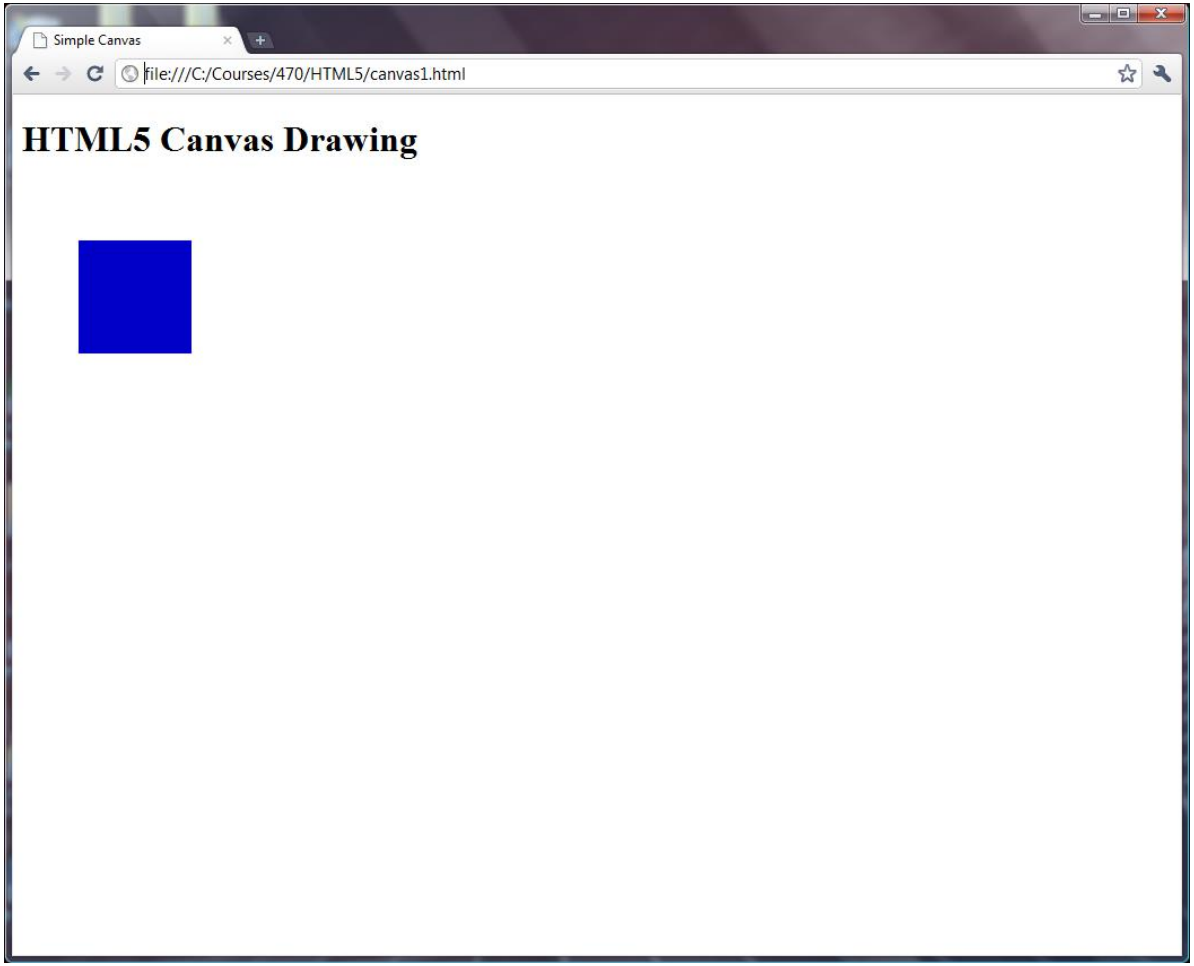
Safari (Ctrl-O)



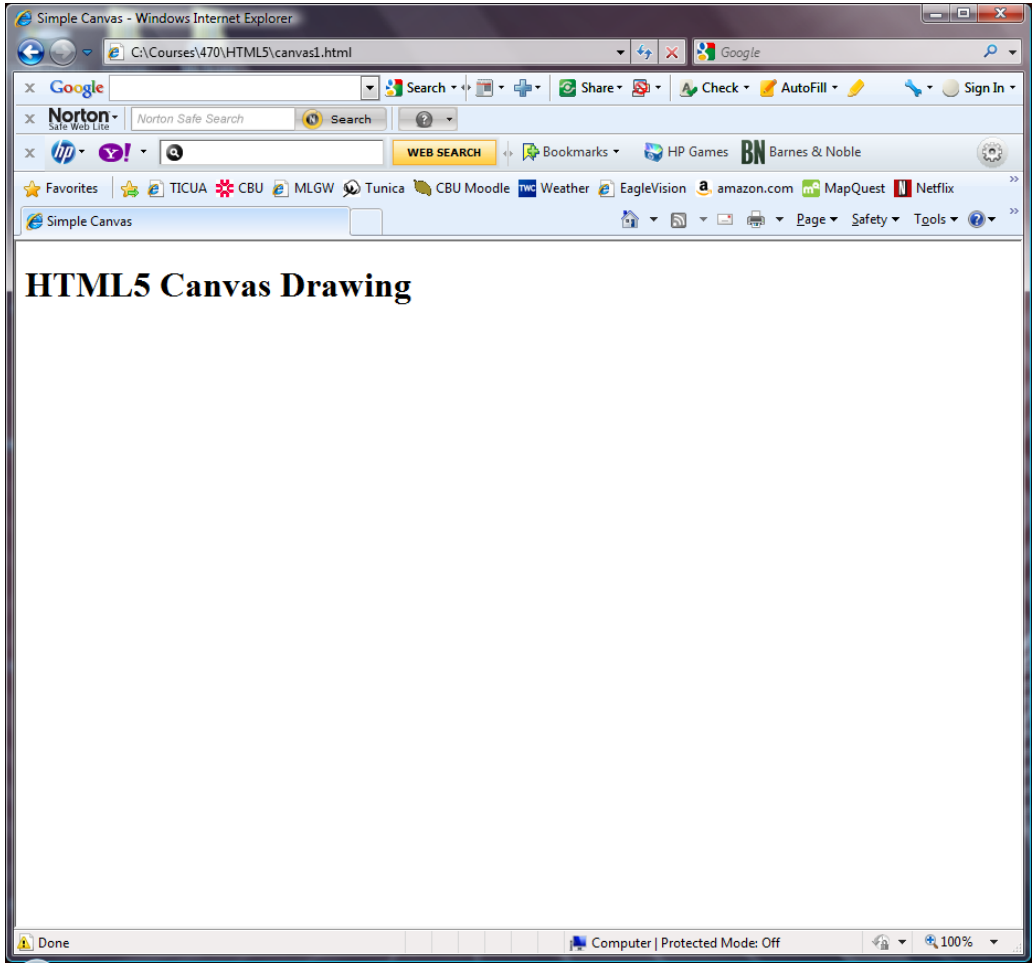
Opera (Ctrl-O)


Menu
Simple Canvas
file://localhost/C:/Courses/470/HTML5/canvas1.html
Search with Google
HTML5 Canvas Drawing
Downloading...

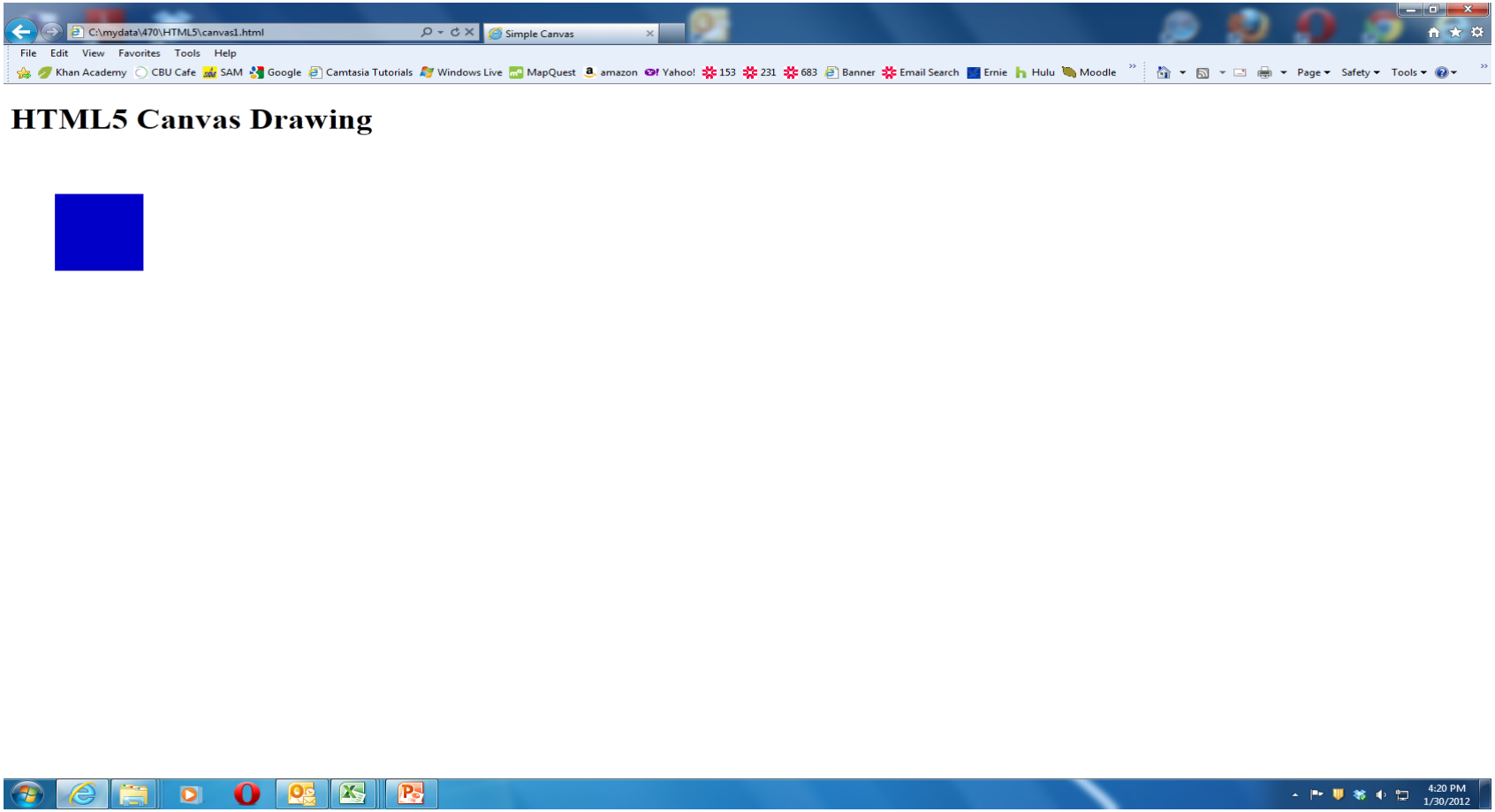
Chrome (Ctrl-O)



IE 8


Simple Canvas - Windows Internet Explorer
C:\Courses\470\HTML5\canvas1.html
Google
Search
Share
Check
AutoFill
Sign In
Norton
Norton Safe Search
Search
WEB SEARCH
Bookmarks
HP Games
Barnes & Noble
Favorites
TICUA
CBU
MLGW
Tunica
CBU Moodle
Weather
EagleVision
amazon.com
MapQuest
Netflix
Simple Canvas
Page
Safety
Tools
HTML5 Canvas Drawing
Done
Computer | Protected Mode: Off
100%

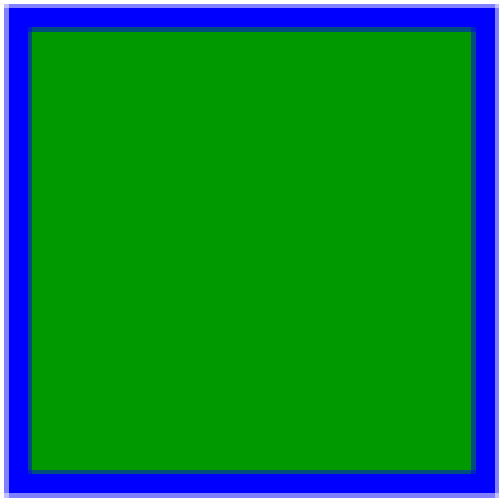
IE 9+


C:\mydata\470\HTML5\canvas1.html
Simple Canvas
File Edit View Favorites Tools Help
Khan Academy CBU Cafe SAM Google Camtasia Tutorials Windows Live MapQuest amazon Yahoo! 153 231 683 Banner Email Search Ernie Hulu Moodle
Page Safety Tools
HTML5 Canvas Drawing
4:20 PM
1/30/2012

Canvas Styles

- Drawing style
  - attribute any fillStyle; // (default black)
  - attribute any strokeStyle; // (default black)
- Line style
  - attribute DOMString lineCap; // “butt”, “round”, “square” (default “butt”)
  - attribute DOMString lineJoin; // “butt”, “round”, “square” (default “butt”)
  - attribute float lineWidth; // (default 1)
  - attribute float miterLimit; // (default 10)

Blue Stroke & Green Fill



fillStyle

- You could directly specify the color as follows:
  - fillStyle="green";
- If you want more control over the colours, you can either specify the hexadecimal value of the colour, the value can range from 000000 to FFFFFF
  - fillStyle = "AFFF2F";
- Alternately, you can specify the value of red, green and blue in decimal, in this case, each of the values will range from 0 to 255:
  - fillStyle="rgb(255,100,100)";
- If you are specifying the text color using rgb value, you can also specify the transparency of the color by passing a fourth parameter; the value of this parameter will range from 0 to 1 with 0 being fully transparent and 1 being fully opaque
  - fillStyle = "rgba(255,100,100,0.5)";

lineCap and lineJoin

Value	Description
butt	Default. A flat edge is added to each end of the line
round	A rounded end cap is added to each end of the line
square	A square end cap is added to each end of the line

Draw a line with rounded end caps:

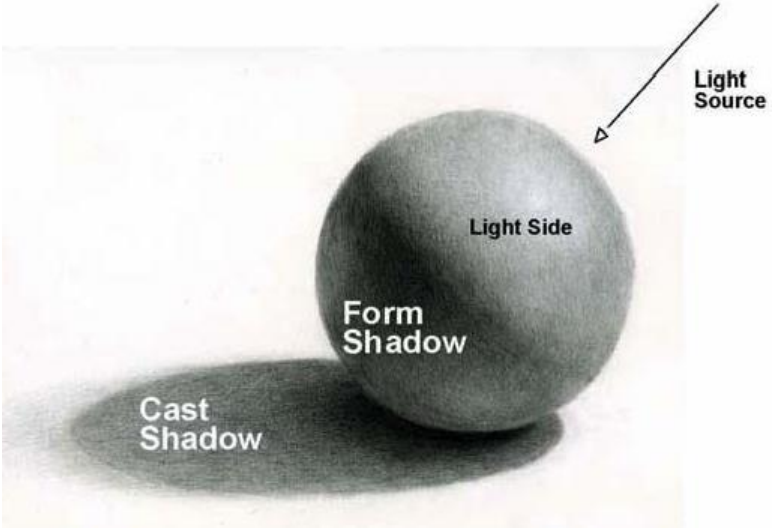
Create a rounded corner when the two lines meet:

Line Styles

- The *lineWidth* attribute gives the width of lines, in coordinate space units. On getting, it must return the current value. On setting, zero, negative, infinite, and NaN values must be ignored, leaving the value unchanged; other values must change the current value to the new value.
- When the context is created, the lineWidth attribute must initially have the value 1.0.
- The *lineCap* attribute defines the type of endings that UAs will place on the end of lines. The three valid values are butt, round, and square. The butt value means that the end of each line has a flat edge perpendicular to the direction of the line (and that no additional line cap is added). The round value means that a semi-circle with the diameter equal to the width of the line must then be added on to the end of the line. The square value means that a rectangle with the length of the line width and the width of half the line width, placed flat against the edge perpendicular to the direction of the line, must be added at the end of each line.
- On getting, it must return the current value. On setting, if the new value is one of the literal strings butt, round, and square, then the current value must be changed to the new value; other values must ignored, leaving the value unchanged.
- When the context is created, the lineCap attribute must initially have the value butt.
- The *lineJoin* attribute defines the type of corners that UAs will place where two lines meet. The three valid values are bevel, round, and miter.
- On getting, it must return the current value. On setting, if the new value is one of the literal strings bevel, round, and miter, then the current value must be changed to the new value; other values must be ignored, leaving the value unchanged.
- When the context is created, the lineJoin attribute must initially have the value miter.
- A join exists at any point in a subpath shared by two consecutive lines. When a subpath is closed, then a join also exists at its first point (equivalent to its last point) connecting the first and last lines in the subpath.
- In addition to the point where the join occurs, two additional points are relevant to each join, one for each line: the two corners found half the line width away from the join point, one perpendicular to each line, each on the side furthest from the other line.
- A filled triangle connecting these two opposite corners with a straight line, with the third point of the triangle being the join point, must be rendered at all joins. The lineJoin attribute controls whether anything else is rendered. The three aforementioned values have the following meanings:
- The bevel value means that this is all that is rendered at joins.
- The round value means that a filled arc connecting the two aforementioned corners of the join, abutting (and not overlapping) the aforementioned triangle, with the diameter equal to the line width and the origin at the point of the join, must be rendered at joins.
- The miter value means that a second filled triangle must (if it can given the miter length) be rendered at the join, with one line being the line between the two aforementioned corners, abutting the first triangle, and the other two being continuations of the outside edges of the two joining lines, as long as required to intersect without going over the miter length.
- The miter length is the distance from the point where the join occurs to the intersection of the line edges on the outside of the join. The miter limit ratio is the maximum allowed ratio of the miter length to half the line width. If the miter length would cause the miter limit ratio to be exceeded, this second triangle must not be rendered.
- The miter limit ratio can be explicitly set using the *miterLimit* attribute. On getting, it must return the current value. On setting, zero, negative, infinite, and NaN values must be ignored, leaving the value unchanged; other values must change the current value to the new value.
- When the context is created, the miterLimit attribute must initially have the value 10.0.

Shadows

- attribute DOMString shadowColor; // (default 0)
- attribute float shadowBlur; // (default 0)
- attribute float shadowOffsetX // (default 0)
- attribute float shadowOffsetY; // (default 0)

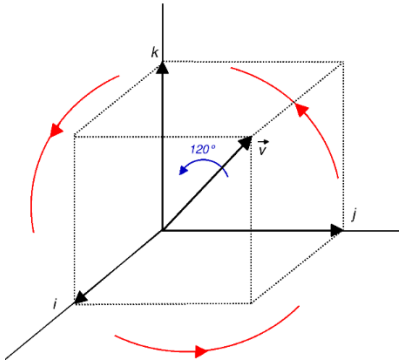


Shadows (con't)

- The *shadowColor* attribute sets the color of the shadow. When the context is created, the shadowColor attribute initially must be fully-transparent black. On getting, the serialization of the color must be returned.
- On setting, the new value must be parsed as a CSS <color> value and the color assigned. If the value cannot be parsed as a CSS <color> value then it must be ignored, and the attribute must retain its previous value. [CSSCOLOR]
- The *shadowOffsetX* and *shadowOffsetY* attributes specify the distance that the shadow will be offset in the positive horizontal and positive vertical distance respectively. Their values are in coordinate space units. They are not affected by the current transformation matrix.
- When the context is created, the shadow offset attributes must initially have the value 0.
- On getting, they must return their current value. On setting, the attribute being set must be set to the new value, except if the value is infinite or NaN, in which case the new value must be ignored.
- The *shadowBlur* attribute specifies the level of the blurring effect. (The units do not map to coordinate space units, and are not affected by the current transformation matrix.) When the context is created, the shadowBlur attribute must initially have the value 0.
- On getting, the attribute must return its current value. On setting the attribute must be set to the new value, except if the value is negative, infinite or NaN, in which case the new value must be ignored.
- *Shadows are only drawn if* the opacity component of the alpha component of the color of shadowColor is non-zero and either the shadowBlur is non-zero, or the shadowOffsetX is non-zero, or the shadowOffsetY is non-zero.
- When shadows are drawn, they must be rendered as follows:
  - Let *A* be an infinite transparent black bitmap on which the source image for which a shadow is being created has been rendered.
  - Let *B* be an infinite transparent black bitmap, with a coordinate space and an origin identical to *A*.
  - Copy the alpha channel of *A* to *B*, offset by shadowOffsetX in the positive *x* direction, and shadowOffsetY in the positive *y* direction.
  - If shadowBlur is greater than 0:
    - Let σ be half the value of shadowBlur.
    - Perform a 2D Gaussian Blur on *B*, using σ as the standard deviation.
  - User agents may limit values of σ to an implementation-specific maximum value to avoid exceeding hardware limitations during the Gaussian blur operation.
  - Set the red, green, and blue components of every pixel in *B* to the red, green, and blue components (respectively) of the color of shadowColor.
  - Multiply the alpha component of every pixel in *B* by the alpha component of the color of shadowColor.
  - The shadow is in the bitmap *B*, and is rendered as part of the drawing model described below.
  - If the current composition operation is copy, shadows effectively won't render (since the shape will overwrite the shadow).

Transformations

- rotate (float angle);
  - angle in radians
    - To calculate from degrees to radians: *degrees**Math.PI/180.
- scale (float x, float y);
- translate (float x, float y);



Rectangles

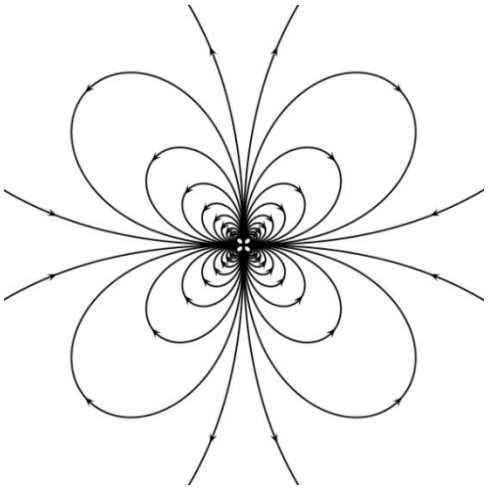
- strokeRect (float x, float y, float w, float h)
  - Rectangle without fill - x, y is upper left corner
- fillRect (float x, float y, float w, float h)
- clearRect (float x, float y, float w, float h)

Lines

- beginPath()
  - The context always has a current path, and there is only one current path
  - A *path* has a list of zero or more subpaths
  - Each subpath consists of a list of one or more points, connected by straight or curved lines, and a flag indicating whether the subpath is closed or not
    - A closed subpath is one where the last point of the subpath is connected to the first point of the subpath by a straight line (polygon)
  - Subpaths with fewer than two points are ignored when painting the path

Lines (con't)

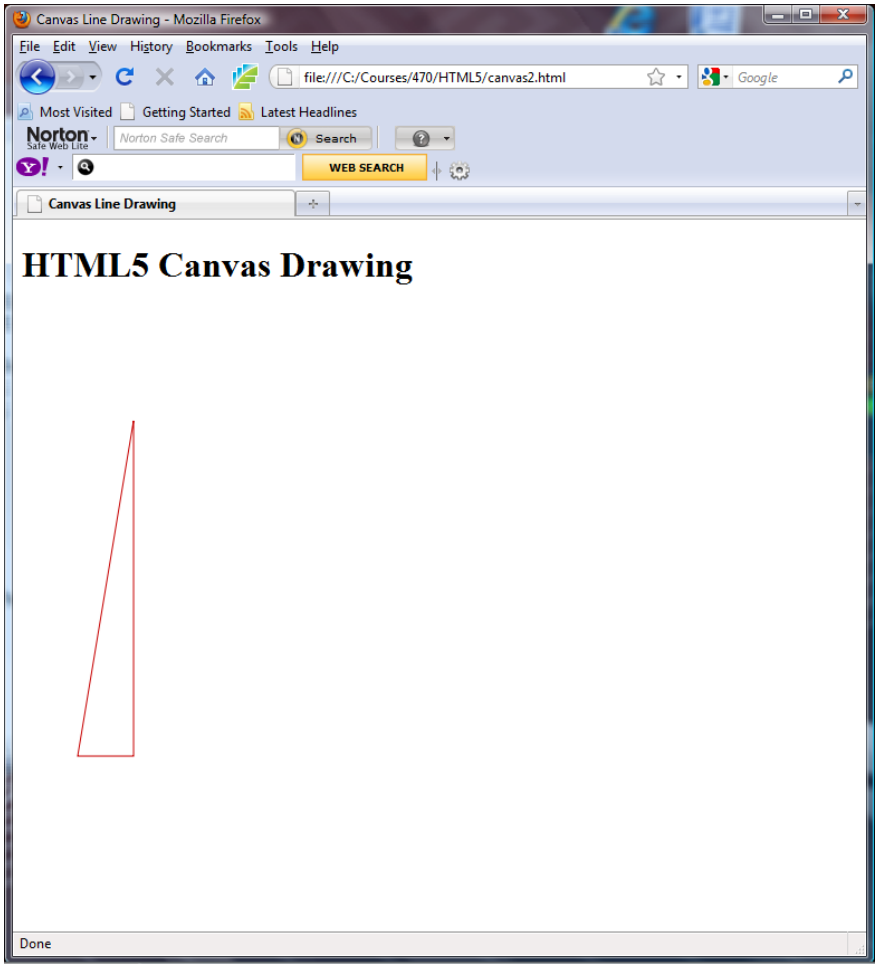
- moveTo (float x, float y);
- lineTo (float x, float y);
- closePath()
- stroke()
- fill()
- clip()



Line Example

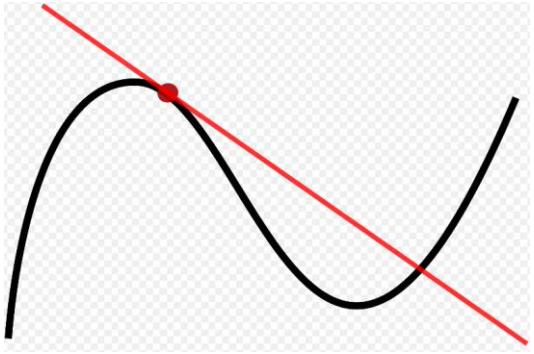
```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>Canvas Line Drawing</title>
                <script type="text/javascript">
                        function init() {
                                var x = document.getElementById("my_canvas");
                                var y = x.getContext("2d");
                                y.beginPath();
                                y.strokeStyle = "rgba(200, 0, 0, 5)";
                                y.moveTo (100, 400);
                                y.lineTo (100, 100);
                                y.lineTo (50, 400);
                                y.closePath();
                                y.stroke();
                        }
                </script>
        </head>
        <body onload="init()">
                <h1>HTML5 Canvas Drawing</h1>
                <canvas id="my_canvas" width="500" height="500"></canvas>
        </body>
</html>
```

Line Example (con't)



Curves & Shapes

- arc (float x, float y, float radius, float startAngle, float endAngle, boolean anticlockwise)
  - angles in radians (0 is at the 3 o'clock position of the arc's circle)
- arcTo (float x1, float y1, float x2, float y2, float radius)
- bezierCurveTo (float cp1x, float cp1y, float cp2x, float cp2y, float x, float y)
- quadraticCurveTo (float cpx, float cpy, float x, float y)
- boolean isPointInPath(x, y)



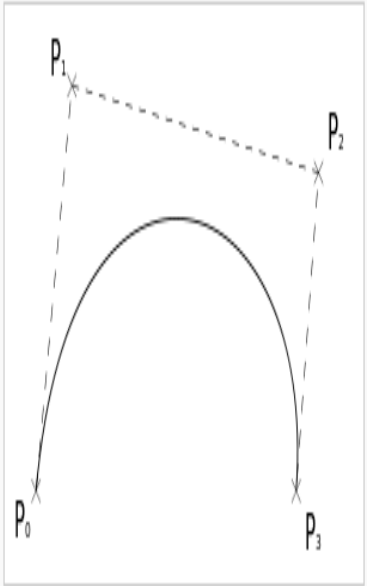
Bézier curve

From Wikipedia, the free encyclopedia

A **Bézier curve** is a parametric curve frequently used in computer graphics and related fields. Generalizations of Bézier curves to higher dimensions are called Bézier surfaces, of which the Bézier triangle is a special case.

In vector graphics, Bézier curves are used to model smooth curves that can be scaled indefinitely. "Paths", as they are commonly referred to in image manipulation programs,[note 1] are combinations of linked Bézier curves. Paths are not bound by the limits of rasterized images and are intuitive to modify.

Bézier curves are also used in the time domain, particularly in animation, user interface[note 2] design and smoothing cursor trajectory in eye gaze controlled interfaces.[1] For example, a Bézier curve can be used to specify the velocity over time of an object such as an icon moving from A to B, rather than simply moving at a fixed number of pixels per step. When animators or interface designers talk about the "physics" or "feel" of an operation, they may be referring to the particular Bézier curve used to control the velocity over time of the move in question.

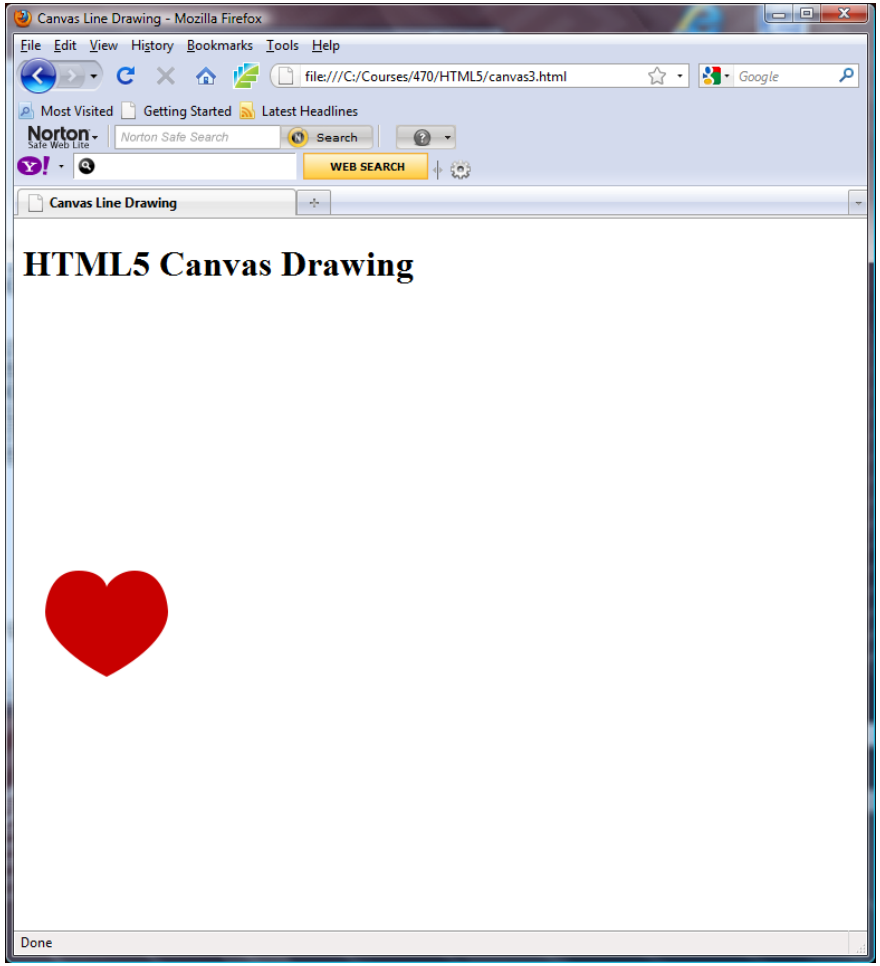


Cubic Bézier curve with four control points

Curves & Shapes Example

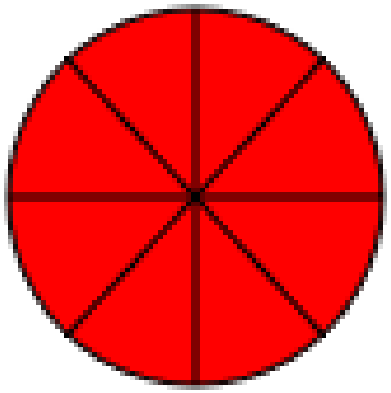
```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>Canvas Line Drawing</title>
                <script type="text/javascript">
                        function init() {
                                var x = document.getElementById("my_canvas");
                                var y = x.getContext("2d");
                                y.fillStyle = "rgba(200, 0, 0, 5)";
                                y.beginPath();
                                y.moveTo (75, 250);
                                y.bezierCurveTo(75, 247, 70, 235, 50, 235);
                                y.bezierCurveTo(20, 235, 20, 272.5, 20, 272);
                                y.bezierCurveTo(20, 290, 40, 312, 75, 330);
                                y.bezierCurveTo(110, 312, 130, 290, 130, 272);
                                y.bezierCurveTo(130, 272.5, 130, 235, 100, 235);
                                y.bezierCurveTo(85, 235, 75, 247, 75, 250);
                                y.closePath();
                                y.fill();
                        }
                </script>
        </head>
        <body onload="init()">
                <h1>HTML5 Canvas Drawing</h1>
                <canvas id="my_canvas" width="500" height="500"></canvas>
        </body>
</html>
```

Curves & Shapes Example (con't)



Exercise

- Draw a basketball



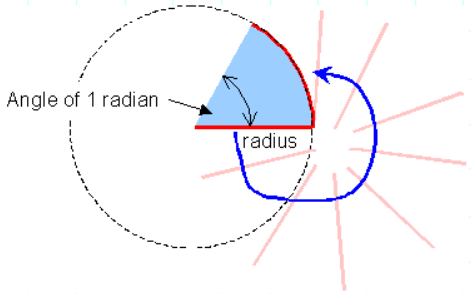
- Hint follows →

Exercise (con't)

- HINT
- arc (float x, float y, float radius, float startAngle, float endAngle, boolean anticlockwise)
  - angles in radians (0 is at the 3 o'clock position of the arc's circle)

One radian is the angle of an arc created by wrapping the radius of a circle around its circumference.

In this diagram, the radius has been wrapped around the circumference to create an angle of 1 radian. The pink lines show the radius being moved from the inside of the circle to the outside:



The radius 'r' fits around the circumference of a circle exactly 2π times. That is why the circumference of a circle is given by:

$$\text{circumference} = 2\pi r$$

So there are 2π radians in a complete circle, and π radians in a half circle.

Thinking…

?

Don’t look ahead !

Exercise (con't)

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<H1>Basketball</H1>
<canvas id="myCanvas" width="200" height="100"></canvas>
<SCRIPT>
        var c = document.getElementById("myCanvas");
        var ctx = c.getContext("2d");
        ctx.beginPath();
        ctx.arc(95,50,40,0,2*Math.PI);
        ctx.fillStyle="red";
        ctx.fill();
        ctx.moveTo(67,20);
        ctx.lineTo(123,80);
        ctx.moveTo(123,20);
        ctx.lineTo(67,80);
        ctx.moveTo(135,50);
        ctx.lineTo(55,50);
        ctx.moveTo(95,10);
        ctx.lineTo(95,90);
        ctx.stroke();
</SCRIPT>
</BODY>
</HTML>
```

Text

- attribute DOMString font; // (default 10px sans-serif)
- attribute DOMString textAlign; // “start”, “end”, “left”, “right”, “center” (default “start”)
- attribute DOMString textBaseline; // “top”, “hanging”, “middle”, “alphabetic”, “ideographic”, “bottom” (default “alphabetic”)
- TextMetrics measureText(DOMString)
- fillText(DOMString text, float x, float y, optional float maxWidth)
- strokeText(DOMString text, float x, float y, optional float maxWidth)

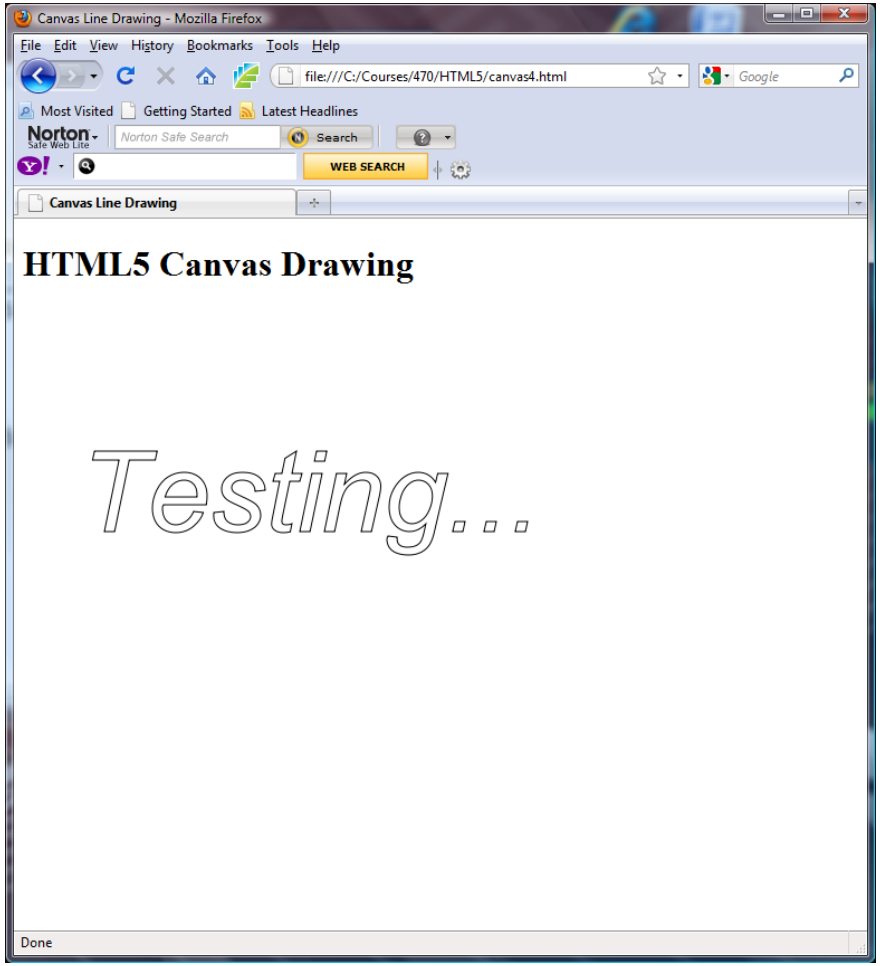
Text (con’t)


top of bounding box
top of em square
hanging baseline
middle
alphabetic baseline
ideographic baseline
bottom of em square
bottom of bounding box
Á ÿ ∫ Ω 私達 आ

Text Example

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>Canvas Line Drawing</title>
                <script type="text/javascript">
                        function init() {
                                var x = document.getElementById("my_canvas");
                                var y = x.getContext("2d");
                                y.font = "italic 100px sans-serif";
                                y.strokeText("Testing...", 50, 200);
                        }
                </script>
        </head>
        <body onload="init()">
                <h1>HTML5 Canvas Drawing</h1>
                <canvas id="my_canvas" width="500" height="500"></canvas>
        </body>
</html>
```

Text Example (con't)



Images

- drawImage(HTMLImageElement image, float dx, float dy, optional float dw, float dh)
- drawImage(HTMLImageElement image, float sx, float sy, float sw, float sh, float dx, float dy, float dw, float dh)
- drawImage(HTMLCanvasElement image, float dx, float dy, optional float dw, float dh)
- drawImage(HTMLCanvasElement image, float sx, float sy, float sw, float sh, float dx, float dy, float dw, float dh)
- drawImage(HTMLVideoElement image, float dx, float dy, optional float dw, float dh)
- drawImage(HTMLVideoElement image, float sx, float sy, float sw, float sh, float dx, float dy, float dw, float dh)

Images (con't)

context . `drawImage`(***image***, ***dx***, ***dy***)

context . `drawImage`(***image***, ***dx***, ***dy***, ***dw***, ***dh***)

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

Draws the given image onto the canvas. The arguments are interpreted as follows:

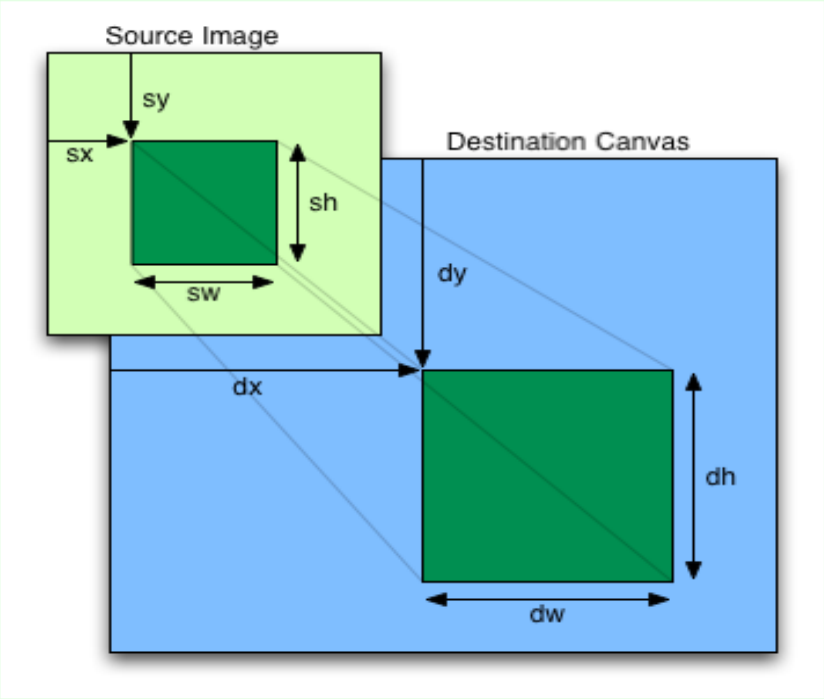
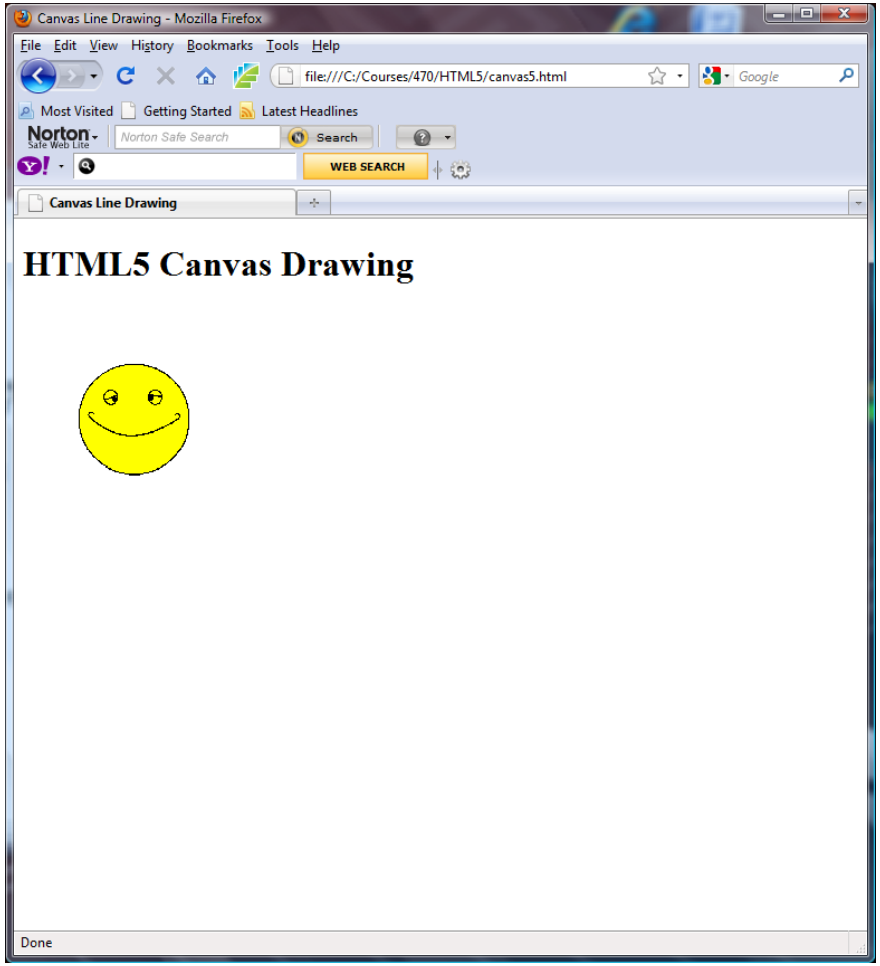


Image Example

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>Canvas Line Drawing</title>
                <script type="text/javascript">
                        function init() {
                                var x = document.getElementById("my_canvas");
                                var y = x.getContext("2d");
                                var i = new Image();
                                i.src = "smile.gif";
                                y.drawImage (i, 50, 50);
                        }
                </script>
        </head>
        <body onload="init()">
                <h1>HTML5 Canvas Drawing</h1>
                <canvas id="my_canvas" width="500" height="500"></canvas>
        </body>
</html>
```

Image Example (con't)



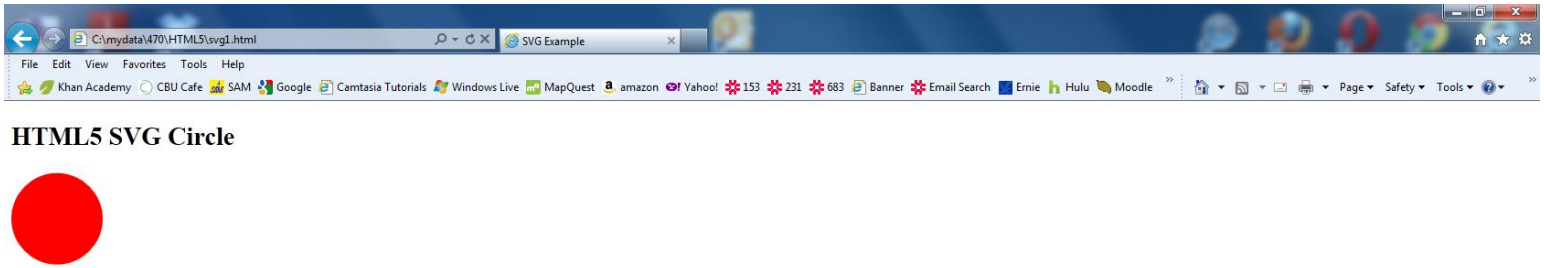
Embedding SVG in HTML5

- If you want the crisp edges of vector-drawn art (as opposed to a bit mapped image), HTML 5 allows you to embed SVG
- SVG (scalable vector graphics) is embedded directly using **<svg>...</svg>** tag which has following simple syntax:
  - <svg xmlns="http://www.w3.org/2000/svg">
  - ...
  - </svg>

SVG Example

- <!DOCTYPE html>
- <html>
- <head>
  - <title>SVG Example</title>
  - <meta charset="utf-8" />
- </head>
- <body>
- <h2>HTML5 SVG Circle</h2>
- <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 in IE9+


C:\mydata\470\HTML5\svg1.html
SVG Example
File Edit View Favorites Tools Help
Khan Academy CBU Cafe SAM Google Camtasia Tutorials Windows Live MapQuest amazon Yahoo! 153 231 683 Banner Email Search Ernie Hulu Moodle
Page Safety Tools
HTML5 SVG Circle

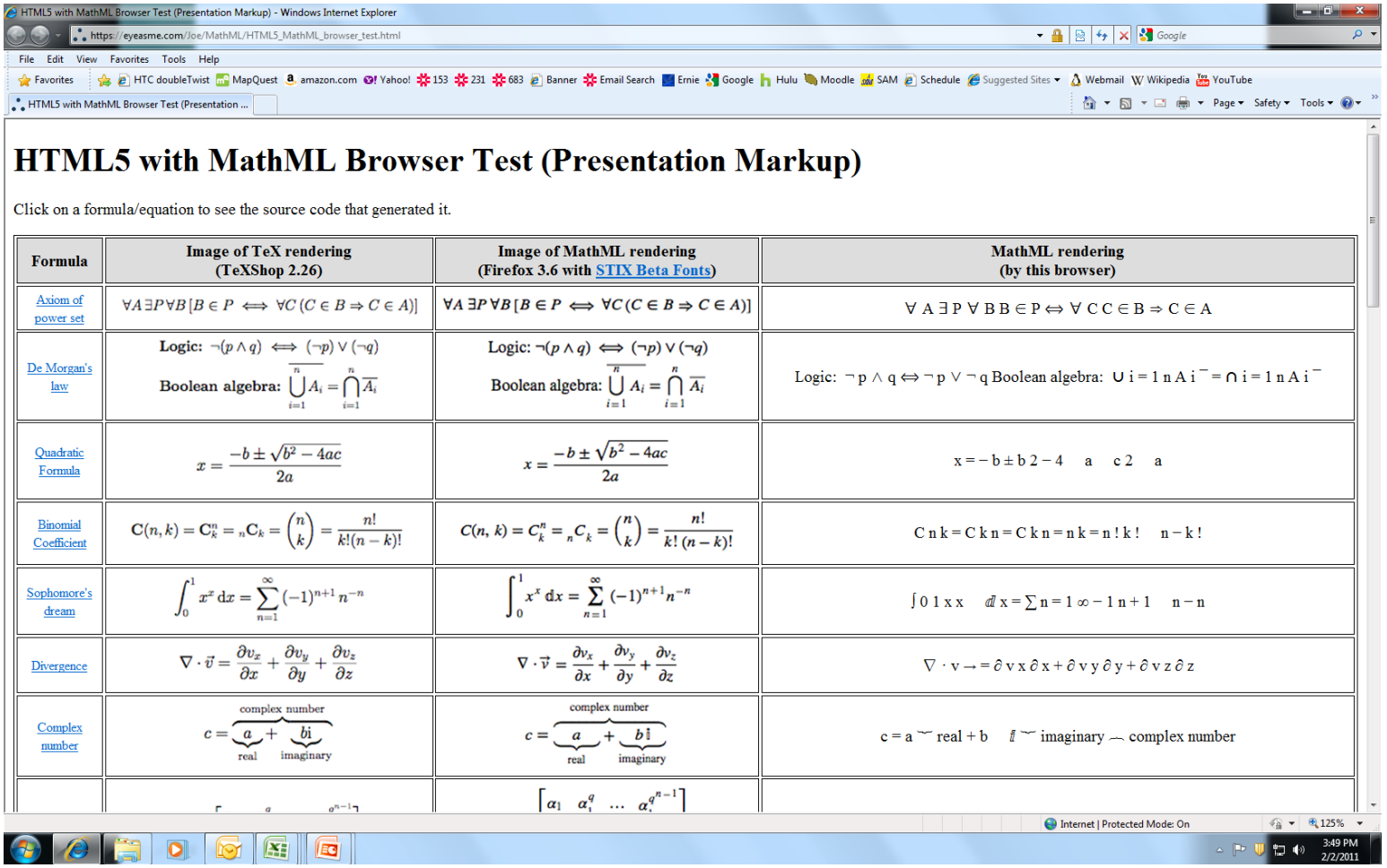

4:27 PM
1/30/2012

Embedding MathML in HTML5

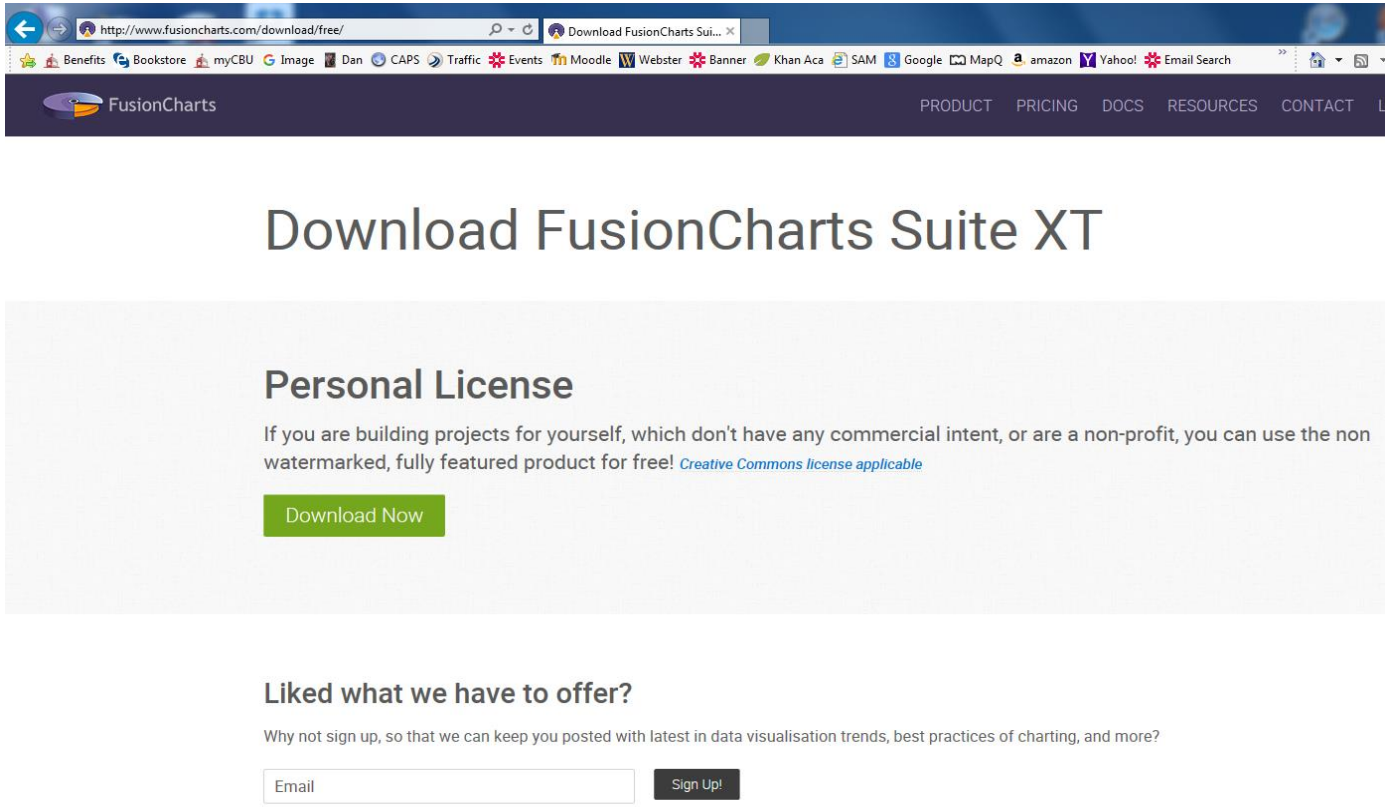
- The syntax of HTML5 allows for MathML elements to be used inside a document using $...$ tags
- To express $a^2 + b^2 = c^2$
  - <!doctype html><html>
  - <head> <meta charset="UTF-8"> <title>Pythagorean theorem</title> </head>
  - <body>
  - <math xmlns="http://www.w3.org/1998/Math/MathML"> <mrow> <msup><mi>a</mi><mn>2</mn></msup> <mo>+</mo> <msup><mi>b</mi><mn>2</mn></msup> <mo>=</mo> <msup><mi>c</mi><mn>2</mn></msup> </mrow> </math>
  - </body></html>

MathML Browser Test

[https://eyeasme.com/Joe/MathML/HTML5_MathML_browser_test.html]



FusionCharts



Example HTML/JS Code

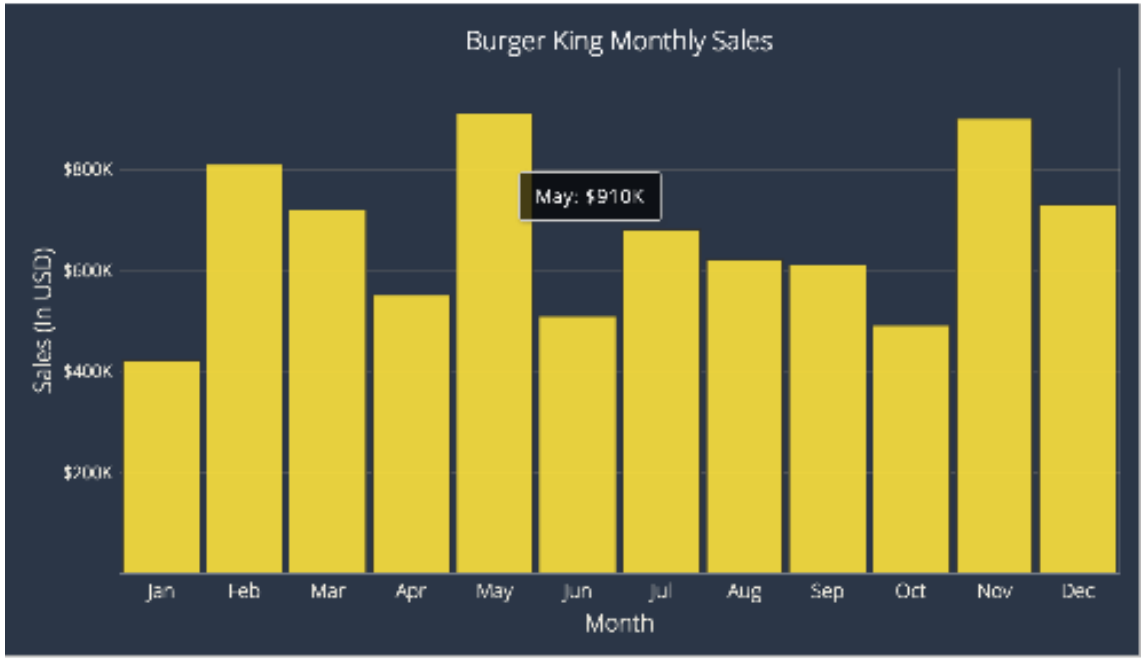
```
<html>
  <head>
    <!-- FusionCharts core library -->
      <script type="text/javascript" src="/path/to/fusioncharts.js"></script>
      <script type="text/javascript">
            FusionCharts.ready(function() {
                  var salesChart = new FusionCharts({
                        type: "column2d",                renderAt: "chart-container",
                        width: "100%",                   height: "500",
                        dataFormat: "json",
                        dataSource: {
                              "chart": {   "caption": "Burger King Monthly Sales",
                              "captionFontColor": "#fff",
                              // more chart configuration options  },
```

Example (con’t)

```
"data": [{
                                                    "label": "Jan",
                                                     "value": "420000"
                                                                },
                                                   // more data
                                                   ]
                                                    }
                                                  });
                                          salesChart.render();
                              });
                </script>
    </head>
<body>
    <div id="chart-container">Your chart will render here!</div>
</body>
</html>
```

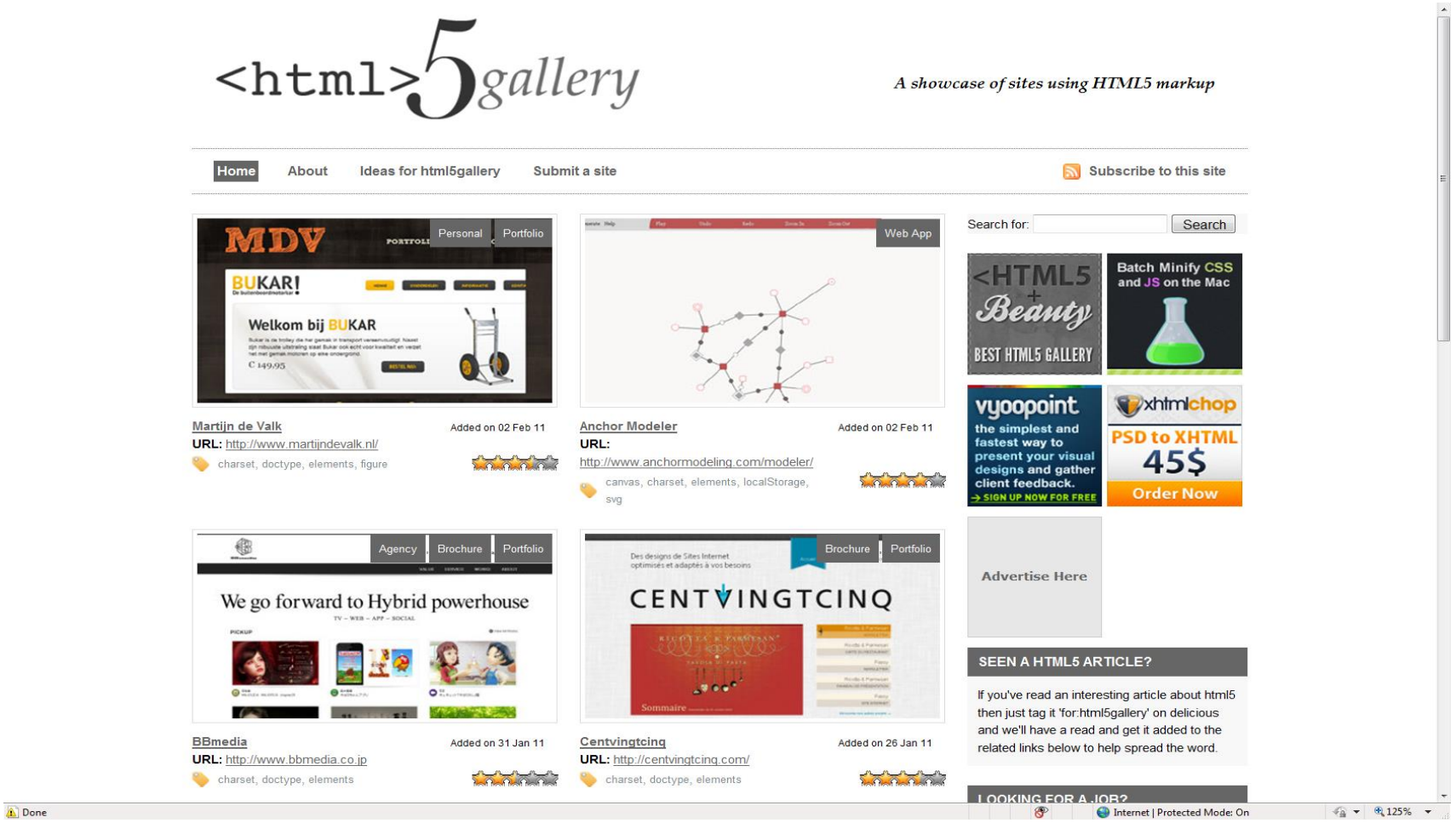
Json format embed data, can also be in file

Example (con’t)



HTML5 Gallery

[http://html5gallery.com/]



Manipulating Content

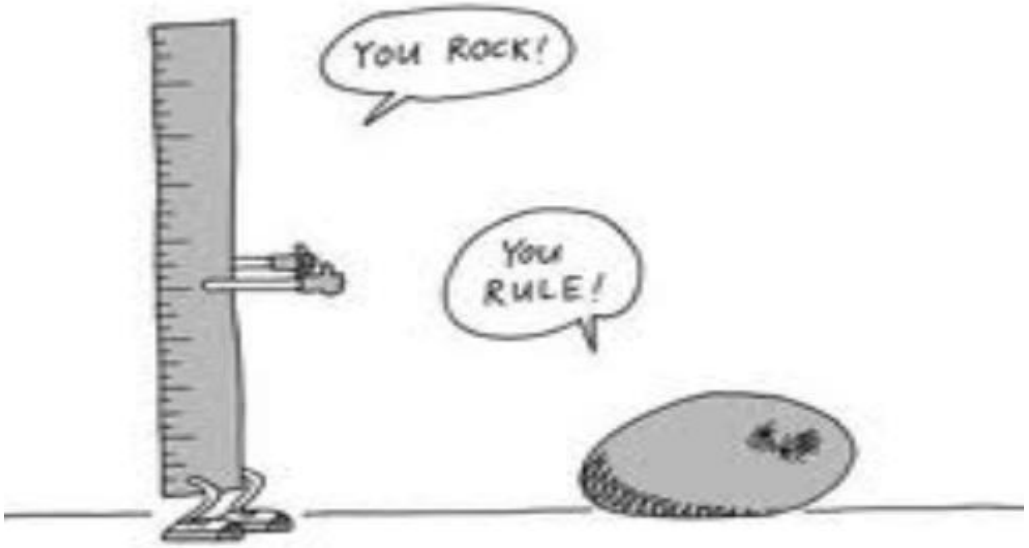
- Manipulating images and other content (cut, past, copy, drag, drop, etc.) is covered in a later lesson

References

- HTML5 Canvas: Native Interactivity and Animation for the Web by Steve Fulton and Jeff Fulton
- Core HTML5 Canvas: Graphics, Animation, and Game Development (Core Series) by David Geary
- HTML5 Canvas: From Noob to Ninja by Kirupa Chinnathambi

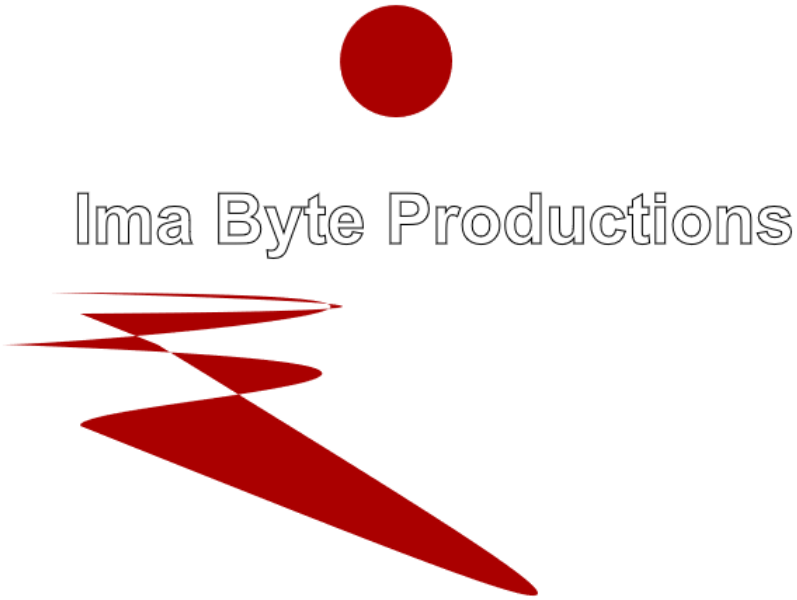
Homework

- Draw something nontrivial → Examples
- Submit files



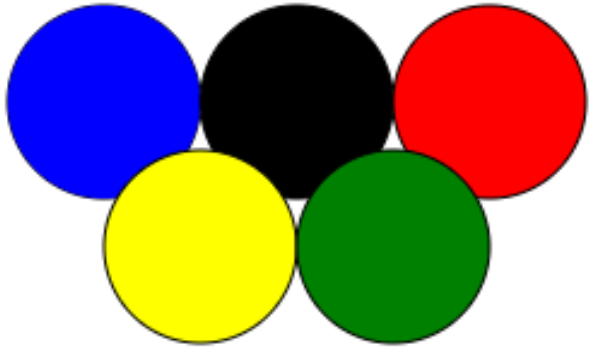
Example

```
<HTML>
<HEAD>
 <SCRIPT type="text/javascript">
 function init() {
  var x = document.getElementById("my_canvas");
  var y = x.getContext("2d");
  y.font = "bold 50px sans-serif";
  y.strokeText("Ima Byte Productions", 70, 200);
  y.fillStyle = "rgba(170, 0, 0, 5)";
  y.beginPath();
  y.moveTo (75, 250);
  y.bezierCurveTo(500, 247, 70, 235, 50, 235);
  y.bezierCurveTo(500, 235, 70, 272.5, 20, 272);
  y.bezierCurveTo(500, 290, 70, 312, 75, 330);
  y.bezierCurveTo(500, 500, 500, 500, 130, 272);
  y.closePath();
  y.fill();
  var c = document.getElementById("my_canvas");
  var ctx = x.getContext("2d");
  ctx.beginPath();
  ctx.arc(300,70,40,0,2*Math.PI);
  ctx.fillstyle="red";
  ctx.fill();
 }
 </SCRIPT>
</HEAD>
<BODY onload="init()">
 <canvas id="my_canvas" width="1000" height="800"></canvas>
</BODY>
</HTML>
```



Example

```
<html>
<head>
<meta charset="utf-8">
<title>Canvas Line Drawing</title>
<script LANGUAGE="JavaScript">
function init() {
var c = document.getElementById("my_canvas");
var ctx = c.getContext("2d");
ctx.fillStyle="blue"; ctx.beginPath();
ctx.arc(45, 300, 40, 0, 2 * Math.PI);
ctx.stroke(); ctx.fill();
var c = document.getElementById("my_canvas");
var ctx = c.getContext("2d");
ctx.fillStyle="black"; ctx.beginPath();
ctx.arc(125, 300, 40, 0, 2 * Math.PI);
ctx.stroke();  ctx.fill();
var c = document.getElementById("my_canvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(205, 300, 40, 0, 2 * Math.PI);
ctx.fillStyle= "red";
ctx.fill(); ctx.stroke();
var c = document.getElementById("my_canvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(85, 360, 40, 0, 2 * Math.PI);
ctx.fillStyle= "yellow";
ctx.fill(); ctx.stroke();
var c = document.getElementById("my_canvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.arc(165, 360, 40, 0, 2 * Math.PI);
ctx.fillStyle= "green";
ctx.fill(); ctx.stroke();
var x = document.getElementById("my_canvas");
var y = x.getContext("2d");
y.font = "20px Arial";
y.strokeText("Olympics Logo", 270, 330);
}
</script>
</head>
<body onload="init()">
<canvas id="my_canvas" width="800" height="500"></canvas>
</body>
</html>
```



Example

```
<head size=36px>Flag of Cuba</head><br><br>
<canvas id="cuba" width="600" height="300" style="border:1px solid grey">
        Your browser does not support HTML5.
</canvas>

<script>
        drawCubanFlag("cuba",0,0,300);

        //Draw Cuban flag
        function drawCubanFlag(canvasId, x, y, height)
        {
                var context = document.getElementById(canvasId).getContext("2d");

                var width = height * 2.0;
                var triangleWidth = (height * Math.sqrt(3))/2;

                context.fillStyle = "white";
                context.fillRect(0,0, width, height);

                for (var idxStripe = 0; idxStripe < 5; idxStripe += 2)
                {
                        context.fillStyle = "blue";
                        context.fillRect(0, idxStripe * height/5, width, height/5);
                }

                context.beginPath();
                context.moveTo(0,0);
                context.lineTo(triangleWidth, height/2);
                context.lineTo(0,height);
                context.fillStyle = "red";
                context.fill();

                drawStar(context, 100, height/2, 5, 50, 15);
                context.fillStyle = "white";
                context.fill();
        }

        function drawStar(context, xCenter, yCenter, nPoints, outerRadius, innerRadius) {
    context.beginPath();
    for (var ixVertex = 0; ixVertex <= 2 * nPoints; ++ixVertex) {
      var angle = ixVertex * Math.PI / nPoints - Math.PI / 2;
      var radius = ixVertex % 2 == 0 ? outerRadius : innerRadius;
      context.lineTo(xCenter + radius * Math.cos(angle), yCenter + radius * Math.sin(angle));
    }
  }
</script>
```