CS381
Web Application Development

HTML5

These class notes are based on the material from our textbook,
Learning PHP, MySQL & JavaScript, 5th ed., by Robin Nixon

Dynamic User Interface

PHP, MySQL, JavaScript, CSS, and HTML5 are used to produce dynamic web content.

- PHP handles all the main work on the web server.
- MySQL manages all the data.
- CSS and JavaScript looks after web page presentation.
- JavaScript can also talk with a PHP code on the web server to update something.
- HTML5, such as the canvas, audio and video, and geolocation make web pages highly dynamic, interactive, and multimedia-packed.

Static Vs Dynamic

Static Web Pages	Dynamic Web Pages
▪ Display the same prebuilt content every time. ▪ May contain plain text, multimedia design,& videos. ▪ Standard HTML pages	▪ Produce different contents for different visitors. ▪ PHP, ASP, and JSP pages.

Client-side processing	Server-side processing
▪ Scripts executed by a browser. ▪ Done by the end users computer.	▪ Scripts executed by a web server. ▪ Uses databases or other data stores.

JavaScript

- Client-side scripting language that runs entirely inside the web browser.
- Brings dynamic functionalities to a website.
- Placed between <script> and </script> HTML tags.
- Gained new power when HTML elements got a more formal structured definition, called **the DOM (Document Object Model)**; an interface for HTML and XML.
- DOM represents the page as a document of objects so that programs can change the document structure, style, and content.

```
<html><head><title>Hello World</title></head><body>
<script>
        document.write("Hello World")
</script>
<noscript>Your browser doesn't support or has disabled
JavaScript</noscript>
</body></html>
```

JavaScript

- **When to use a semicolon (;)**
  - To separate more than one statement on a single line.
  - To write bookmarklets: small pieces of JavaScript which can be easily executed via "bookmarks" or "favorites" features in a web browser.

- **Automatic Semicolon Insertion**: When there is a place where a semicolon is needed, JavaScript adds it behind the scenes.

- **Including js Files** <script src="script.js"></script>

- **Head Scripts Use**
  - to execute a script when a page loads.
  - to write things such as meta tags into the <head> section. *Metadata is information about data.*
- **Debugging**
  - Chrome: Ctrl-Shift-J
  - Edge: F12

JavaScript is a very loosely typed language; the type is determined when a value is assigned and can change as the variable appears in different contexts.

Variables

Variables Names

- May include only the letters a-z, A-Z, 0-9, the $ symbol, and underscore (_).
- No other characters, such as spaces or punctuation, are allowed in a variable name.
- The first character of a variable name can be only a-z, A-Z, $, or _
- Reserved words (keywords) cannot be used as identifiers or property names.
- Names are case-sensitive. Count, count, and COUNT are all different variables.
- There is no set limit on variable name lengths.

String Variables

You may include a **single** quote within a **double-quoted** string or a double quote within a single-quoted string. But you must escape a quote of the same type by using the backslash character.

greeting = “Hello there“

warning = ‘Be careful‘

greeting = "\"Hello there\" is a greeting"

warning = '\'Be careful\' is a warning'

Functions

A function definition (called a function declaration, or function statement) consists of the

function name **(** *list of parameters*) **{** body **}**

```
function test(a, b )
{       return a + b;
}
```

function can also be created by a **function expression**. It also **can be anonymous**; does not have a name.

```
const sum = function(a, b)
{       return a + b;
}
```

Arrow function expressions

```
const sum = (a, b) => a + b;

document.write( sum(5, 6) );
```

Variables

Global Variables

- *Used* or declared outside of all functions. or
- *Used* within functions without declaration. (not good)

```
a = 123                                  // Global scope
var b = 456                              // Global scope
if (a == 123) var c = 789                // Global scope
```

Local Variables

- Parameters; except Arrays are passed to functions by reference.
- Declared within a function.

```
function test(t, w )                     // Local scope
{    a = 123                             // Global scope
     var b = 456                         // Local scope
     if (a == 123) var c = 789           // Local scope
}
```

Strict Mode **"use strict";** undeclared variables can not be used

- At the top: enables strict mode for an entire script.
- In function: enables strict mode for just this function.

```
function test()
{    "use strict";
     console.log(x);
     var x = 10;
     let y = 20;
     const name = 'Khaled';
}
```

Hoisting a mechanism where variables and function declarations are moved to the top of their scope before code execution.

var vs. let var is function scoped but let is block scoped.

unlike var, a let variable cannot be re-declared within its scope.

var is initialized as undefined, and let is not initialized.

const behaves just like let but it must be initialized with a value and cannot be changed.

Arrays and Types

Arrays

```
var toys = new Array('bat', 'ball', 'whistle');        var toys = ['bat', 'ball', 'whistle']

var face = [ ['R', 'G', 'Y'],  ['W', 'R', 'O'],  ['Y', 'W', 'G']  ]

t = ['R', 'G', 'Y'];      m= ['W', 'R', 'O'];      b = ['Y', 'W', 'G'];      face = [t, m, b]

document.write( face[0][1] )          ➔          G
```

undefined a variable has been declared but its value has not been assigned.

null null is a value.

typeof operator used to look up types

```
n = '838102050';          document.write( typeof n ) ==> string
```

String to number

```
n = "123";          n *= 1
```

Number to String

```
n = 123;          n += ""
```

HTML5

Introduced:

- canvas — to draw graphics.
- audio and video — to embed audio and video contents in a document.
- Semantic elements — header, footer, section, article, others.
- Geolocation — to locate a user's position.
- Web worker — a JS that runs in the background, without affecting the performance of the page.
- Local Storage — stores data locally, far in excess of the limited capabilities of cookies

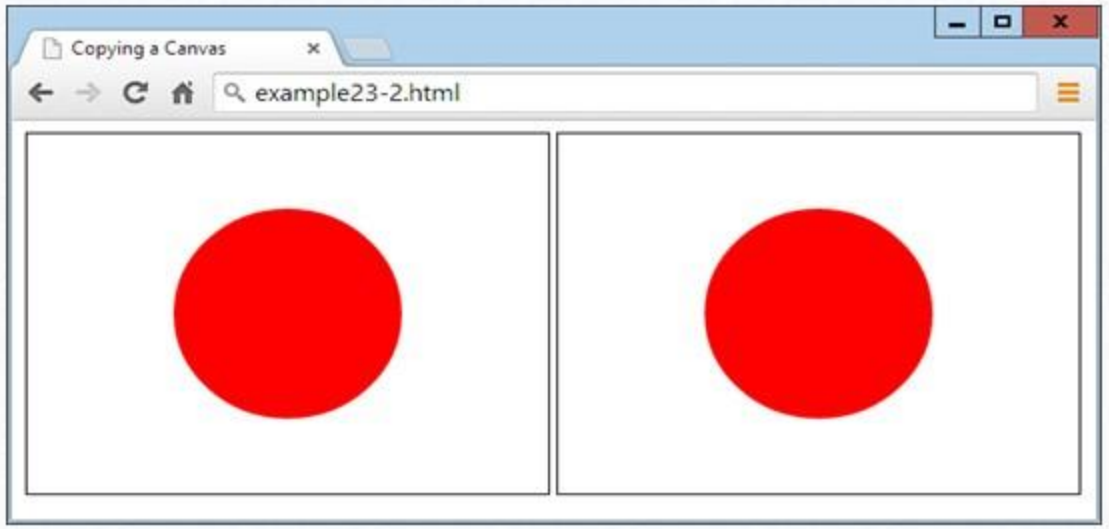
Canvas

- Originally introduced by Apple for the WebKit rendering engine.
- Enables us to draw graphics in a web page without having to rely on a plug-in such as Java or Flash.
- Created by using the <canvas> tag.

```
<!DOCTYPE html> <html>
<head><title>Canvas</title></head>

<body>
<canvas id='mycanvas' width='320' height='240'> This text is visible only in non-HTML5 </canvas> <!-- First Canvas -->

<script>
        var canvas = document.getElementById('mycanvas')      // returns the element object or null if no match.
        canvas.width = window.innerWidth - 30
        canvas.height = window.innerHeight - 20
        canvas.style.border = '1px solid black‘
</script>

</body>
</html>
```

toDataURL

```
<!DOCTYPE html> <html><head><title>Image toDataURL</title></head>

<body>
<canvas id='mycanvas' width='320' height='240'></canvas>
<img id='myimage'>
<script>
      var canvas = document.getElementById('mycanvas')
      canvas.style.border = '1px solid black'
      var ctx = canvas.getContext('2d')          // ('webgl')

      ctx.arc(160, 120, 70, 0, Math.PI * 2, false)
      ctx.fillStyle = 'red'
      ctx.fill()                                 // ctx.stroke()

      // copy image from convas
      var img = document.getElementById('myimage')
      img.style.border = '1px solid black'
      img.src = canvas.toDataURL()


</script>
</body>
</html>
```

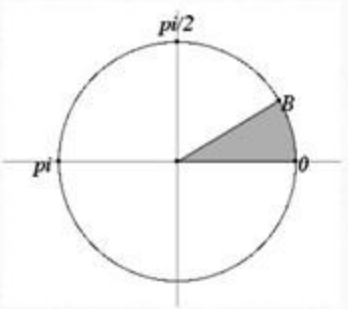


Example Displaying the Japanese flag:

radius of 70 pixels, beginning at an angle of 0 degrees (which is the **righthand** edge of the circle as you look at it).

Angle is provided in radian.

false: indicates a clockwise direction



Canvas Drawing

Specifying an Image Type

img.src = canvas.toDataURL('image/jpeg', 0.4)

- The default is PNG (image/png)
- This produces a JPEG with compression value of 0.4, 0 (lowest quality) To 1 (highest quality).

Drawing Rectangles

top-left followed by the width and height in pixels.

- **ctx.fillStyle = 'blue'** — **ctx.fillRect(20, 20, 600, 200)**
- **ctx.strokeStyle = 'green'** — **ctx.strokeRect(60, 60, 520, 120)**
- **ctx.clearRect(40, 40, 560, 160)** — draws a rectangle in which all the color values (red, green, blue, and alpha transparency) have been set to 0

Canvas Animation 1

```
<!DOCTYPE html><html>
<head>    <title>Canvas Animation 1</title>   <script src="Canvas Animation 1.js"></script></head>
<body>    <canvas id='mycanvas' width='800' height='600'></canvas></body></html>



// Canvas Animation 1.js
window.onload = function()                    // after loading the whole page (window.onload event), start the script
{
      var canvas = document.getElementById('mycanvas')
      canvas.style.border = '1px solid black'
      var ctx = canvas.getContext('2d')
      ctx.fillStyle = "Red"

            canvas.onmousemove = function(event)        // onmousedown,        mouseup
            {
                  ctx.clearRect(0, 0, canvas.width, canvas.height)

                  ctx.beginPath()
                  ctx.arc(event.x, event.y, 20, 0, Math.PI*2, false)
                  ctx.fill()
            }
}
```

Canvas Animation 2

```
<!DOCTYPE html> <html>
<head>
	<title>Canvas Animation 2</title>
	<meta http-equiv="refresh" content="1">
</head>
<body>
<canvas id='mycanvas' width='800' height='600'></canvas>
<script>
	var canvas = document.getElementById('mycanvas');
	canvas.style.border = '1px solid black';
	var ctx = canvas.getContext('2d');
	ctx.fillStyle = 'Green';

	var x = Math.random() * canvas.width;
	var y = Math.random() * canvas.height;
	var r = 10;

	ctx.beginPath();
	ctx.arc(x, y, r, 0, Math.PI*2, false);
	ctx.fill();
</script>
</body>
</html>
```

http-equiv:
an attribute provides an HTTP header for the value of the content attribute.

Redirect:
<meta http-equiv="refresh" content="5; url=https://example.com/">

Canvas Animation 3

```
// Canvas Animation 3.js
var canvas = document.getElementById('mycanvas')
canvas.style.border  = '1px solid black'
var ctx = canvas.getContext('2d')
ctx.fillStyle  = "Red"


var x = Math.random()  * canvas.width
var y = Math.random()  * canvas.height
var r = Math.random()  * 50
var dx = dy = Math.random()  * 10
var anim
draw()

function  draw()
{           ctx.clearRect(0,  0, canvas.width,  canvas.height)
            ctx.beginPath()
            ctx.arc(x,  y,  r,  0,  Math.PI*2,  false)
            ctx.fill()
            if( x+r > canvas.width  || x-r < 0)          dx = -dx
            if( y+r > canvas.height || y-r < 0)          dy = -dy
            x += dx
            y += dy
            anim = window.requestAnimationFrame(draw)    // request the animation function to be called before performing the next repaint.
}                                                        // integer, the request id, that identifies the entry in the callback list.

canvas.onmousedown  = function(event)          {   window.cancelAnimationFrame(anim);     }
canvas.ondblclick  = function(event)           {   window.requestAnimationFrame(draw);    }
```

```
<!DOCTYPE html><html><head>  <title>Canvas Animation 3</title>
<script defer src="Canvas Animation 3.js"></script></head><body>
<canvas id='mycanvas' width='800' height='600'></canvas></body></html>
```

defer:
run after the document's content has been completely parsed and displayed

Canvas Animation 4 XOR

```
<!DOCTYPE html><html> <head><title>Canvas Animation 4 XOR </title></head><body>
<canvas id="mycanvas" width="1200" height="1200"></canvas>
<script>
		var canvas = document.getElementById('mycanvas')
		var ctx = canvas.getContext('2d');
		ctx.globalCompositeOperation = 'xor';

		var x = 100;
		var y = 100;

		ctx.fillStyle = '#55AAFF';
		ctx.fillRect(x, y, 60, 30);

		canvas.onmousemove = function(event)
		{
			ctx.fillRect(x, y, 60, 30);
			x = event.x
			y = event.y
			ctx.fillRect(x, y, 60, 30);
		}

</script>
</body>
</html>
```

Linear Gradients

- Creates a gradient along the line connecting two given coordinates.
- The gradient must first be assigned to the **fillStyle** or **strokeStyle** properties.

- **gradient = ctx.createLinearGradient(x0, y0, x1, y1);**
- **gradient.addColorStop(0, 'white')** start position along the gradient range between 0 and 1; floating point
- **gradient.addColorStop(1, 'black')**
- **ctx.fillStyle = gradient**
- **ctx.fillRect(80, 80, 480,80)**

Radial Gradients

- Creates a radial gradient using the size and coordinates of two circles.
- The gradient must first be assigned to the **fillStyle** or **strokeStyle** properties.

- **gradient = ctx.createRadialGradient(x0,y0,r0,x1,y1,r1);**
- **gradient.addColorStop(0, 'white')**
- **gradient.addColorStop(1, 'black')**
- **ctx.fillStyle = gradient**
- **ctx.fillRect(80, 80, 480,80)**

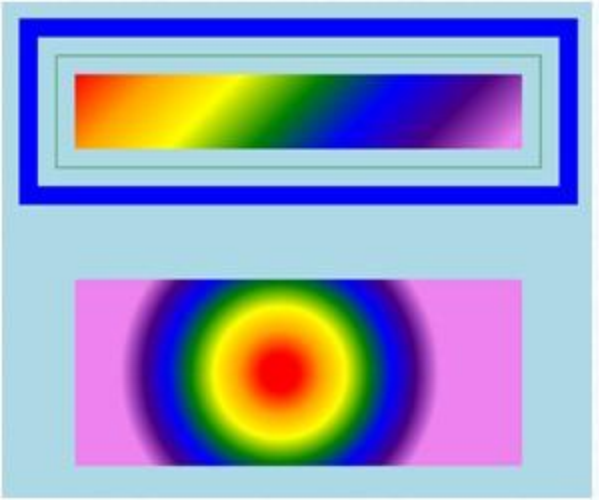
Gradient Rectangles

```
<!DOCTYPE html> <html><head><title> Gradient Rectangles</title></head><body>
<canvas id='mycanvas' width='640' height='600'> </canvas>
<script>
        var canvas = document.getElementById('mycanvas')
        canvas.style.background = 'lightblue'
        var ctx = canvas.getContext('2d')

        ctx.fillStyle = 'blue'
        ctx.strokeStyle = 'green'
        ctx.fillRect(20, 20, 600, 200)
        ctx.clearRect(40, 40, 560, 160)
        ctx.strokeRect(60, 60, 520, 120)

        var gradient = ctx.createLinearGradient(80, 80, 400, 400)
        gradient.addColorStop(0.00, 'red')
        gradient.addColorStop(0.14, 'orange')
        gradient.addColorStop(0.28, 'yellow')
        gradient.addColorStop(0.42, 'green')
        gradient.addColorStop(0.56, 'blue')
        gradient.addColorStop(0.70, 'indigo')
        gradient.addColorStop(0.84, 'violet')
        ctx.fillStyle = gradient
        ctx.fillRect(80, 80, 480, 80)
```

➔

```
gradient = ctx.createRadialGradient(300,400,20, 300,400,200)
gradient.addColorStop(0.00, 'red')
gradient.addColorStop(0.14, 'orange')
gradient.addColorStop(0.28, 'yellow')
gradient.addColorStop(0.42, 'green')
gradient.addColorStop(0.56, 'blue')
gradient.addColorStop(0.70, 'indigo')
gradient.addColorStop(0.84, 'violet')
ctx.fillStyle = gradient
ctx.fillRect(80, 300 , 480, 200)
```

```
</script></body></html>
```



Patterns

Using Patterns for Fills

var pattern = **ctx.createPattern(image, 'repeat')**

- **repeat**: Repeat the image both vertically and horizontally.
- **repeat-x**: Repeat the image horizontally.
- **repeat-y**: Repeat the image vertically.
- **no-repeat**: Do not repeat the image.

Image Pattern

```
<!DOCTYPE html> <html><head><title> Image Pattern</title></head><body>
<canvas id='mycanvas' width='640' height='600'> This text is visible only in non-HTML5  </canvas>
<script>

    var canvas = document.getElementById('mycanvas')
    canvas.style.background = 'lightblue'
    var ctx = canvas.getContext('2d')

    var image = new Image()
    image.src = 'image.png'

    image.onload = function()
    {
       var pattern = ctx.createPattern(image, 'repeat')
       ctx.fillStyle = pattern
       ctx.fillRect(0, 0, 800, 400)
    }


</script>
</body>
</html>
```

Text

Writing Text to a Canvas:

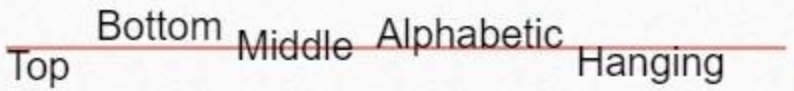
ctx.strokeText(text, x, y [, maxWidth in pixels]);

draws the outlines of the characters of a text at the specified coordinates.

ctx.fillText(text, x, y [, maxWidth in pixels]);

fills the characters of a text at the specified coordinates.

- ctx.textBaseline = 'top'



- ctx.textAlign

Iamge Pattern Text

```
<!DOCTYPE html> <html><head><title>Iamge Pattern Text </title></head><body>
<canvas id='mycanvas' width='640' height='600'> This text is visible only in non-HTML5  </canvas>
 <script>
      var canvas = document.getElementById('mycanvas')
      canvas.style.border = '1px solid black'
      var ctx = canvas.getContext('2d')

      ctx.font = 'bold 140px Times New Roman'
      ctx.textBaseline = 'top'
      ctx.textAlign = 'left'

      ctx.lineWidth = 16
      ctx.strokeStyle = 'red'
      ctx.strokeText('Yanbu', canvas.width / 4, canvas.height / 4)

      ctx.fillStyle = 'yellow'
      ctx.fillText('Yanbu', canvas.width / 4, canvas.height / 4)
```

➔

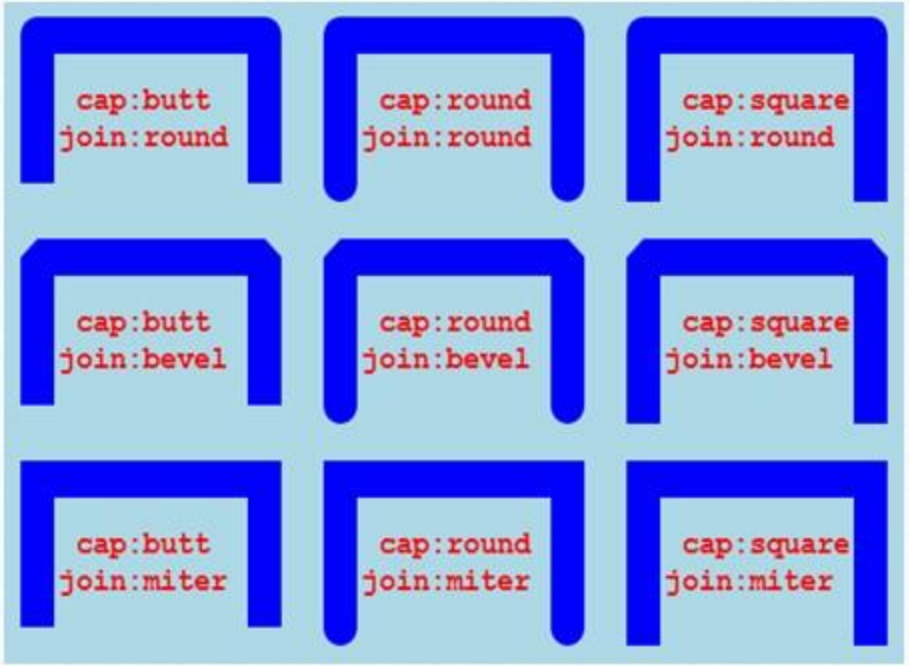

```
var image = new Image()
 image.src = 'Car.png'

 image.onload = function ()
 {   ctx.lineWidth = 1
     ctx.fillStyle = ctx.createPattern(image, 'repeat')
     ctx.strokeText('Yanbu', 200, 250)
     ctx.fillText('Yanbu', 200, 250)
 }
```

```
   </script>
<body>
</html>
```

Line Properties

Line Cap to set the line end style.

- **butt** a flat end. **default**.
- **round** a rounded end.
- **square** a square end.



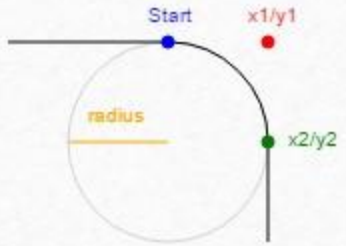
Line Join to set the corner style.

- **miter** a sharp corner. **default**.
- **round** a rounded corner.
- **bevel** an oblique corner.

Drawing Lines and Arcs

Drawing Lines:

- ctx.lineCap = ‘round’
- ctx.lineJoin = ‘bevel’
- ctx.beginPath()
  - ctx.moveTo(200, 300)
  - ctx.lineTo(200, 200)
  - ctx.stroke()
- ctx.closePath()

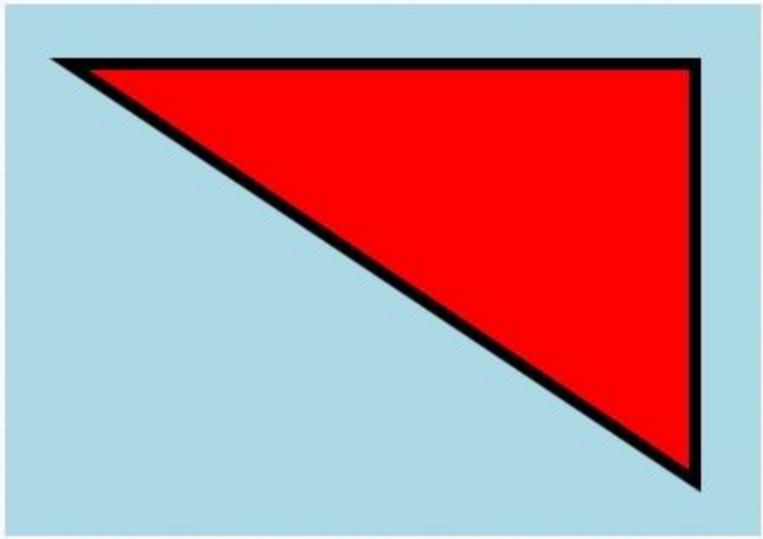


The arc Method:

- ctx.arc(x, y, radius, startAngle, endAngle [, anticlockwise]) default: clockwise (false)
- ctx.arcTo(x1, y1, x2, y2, radius) to create an arc/curve between two tangents.

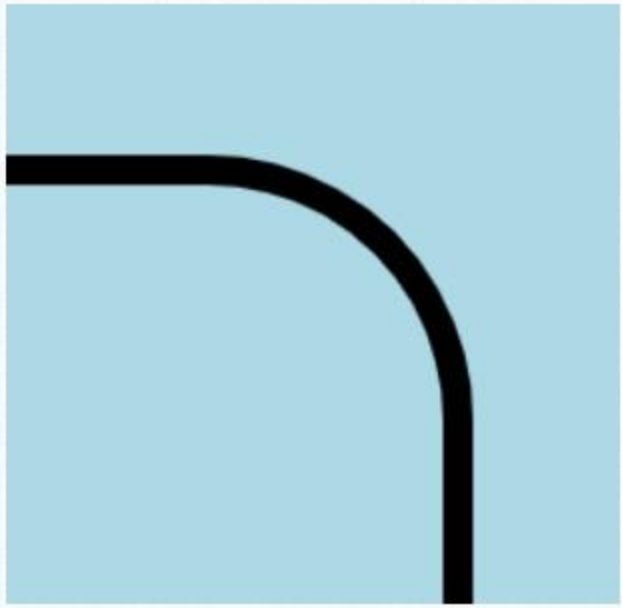
lineTo

```
<!DOCTYPE html> <html><head><title>ArcTo</title></head><body>
<canvas id='mycanvas' width='640' height='600'> This text is visible only in non-HTML5  </canvas>
 <script>

 var canvas = document.getElementById('mycanvas')
       canvas.style.border = '1px solid black'
       var ctx = canvas.getContext('2d')

       canvas.style.background = 'lightblue'

       ctx.lineWidth = 12
       ctx.fillStyle = 'Red'

       ctx.beginPath()
          ctx.moveTo(100, 100)
          ctx.lineTo(400, 100)
          ctx.lineTo(400, 300)
       ctx.closePath()
       ctx.stroke()
       ctx.fill()

    </script>
<body>
</html>
```



arcTo

```
<!DOCTYPE html> <html><head><title>ArcTo</title></head><body>
<canvas id='mycanvas' width='640' height='600'> This text is visible only in non-HTML5  </canvas>
 <script>

      var canvas = document.getElementById('mycanvas')
      canvas.style.border = '1px solid black'
      var ctx = canvas.getContext('2d')

       canvas.style.background = 'lightblue'

      ctx.lineWidth = 12

      ctx.beginPath()
         ctx.moveTo(100, 100)
         ctx.lineTo(400, 100)
         ctx.arcTo(600, 100, 600, 200, 100)
         ctx.lineTo(600, 600)
      ctx.stroke()

   </script>
<body>
</html>
```



Images and Shadow

drawImage

- **ctx.drawImage(image, dx, dy)**
- **ctx.drawImage(image, dx, dy, dWidth, dHeight)**
- **ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)**

Adding Shadows

- **ctx.shadowOffsetX** the shadow should be shifted to the right (negative: left).
- **ctx.shadowOffsetY** the shadow should be shifted down (negative: up).
- **ctx.shadowBlur** the number of pixels over which to blur the shadow's outline.
- **ctx.shadowColor** if a blur is in use, this color will blend with the background in the blurred area.

Image Draw Shadow

```
<!DOCTYPE html> <html><head><title>Image Draw Shadow</title></head><body>
<canvas id='mycanvas' width='1200' height='600'> This text is visible only in non-HTML5 </canvas>
   <script>
      var canvas = document.getElementById('mycanvas')
      canvas.style.border = '1px solid black'
      var ctx = canvas.getContext('2d')

      var myimage = new Image()
      myimage.src = 'Apple.png'

      myimage.onload = function ()
      {

         ctx.drawImage(myimage, 20, 20, myimage.width/4, myimage.height/4)

         ctx.shadowOffsetX = 10
         ctx.shadowOffsetY = 10
         ctx.shadowBlur = 5
         ctx.shadowColor = 'black'
         ctx.drawImage(myimage, 600, 20, myimage.width/4, myimage.height/4)

      }
   </script><body>
</html>
```

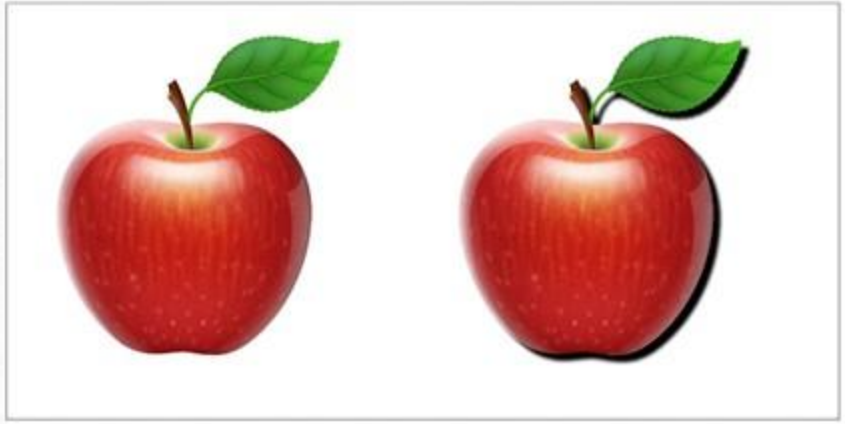


Image Manipulation

Editing at the Pixel Level

- ctx.getImageData(sx, sy, sw, sh)

returns an array containing all the pixel data for the selected area.

- ctx.putImageData(imageData, dx, dy)

paints data from the given ImageData object onto the canvas.

The scale Method:

- ctx.scale(x, y)

Scale factors. A negative value flips pixels across the axis. A value of 1 results in no scaling.

Image Pixle Manipulating

```
<!DOCTYPE html> <html><head><title>Image  Pixle Manipulating</title></head><body><canvas  id='mycanvas' width='640' height='600'>
</canvas><script>
      var canvas = document.getElementById('mycanvas')
      canvas.style.border  = '1px solid black'
      var ctx = canvas.getContext('2d')

      var myimage = new Image()
      myimage.src = 'House.jpg'
      myimage.onload = function ()
      {
         ctx.drawImage(myimage,  0, 0)
         var idata = ctx.getImageData(0,  0, myimage.width,  myimage.height)

         for (y = 0; y < myimage.height;  y++)
         {    var pos = y * myimage.width  * 4

              for (x = 0; x < myimage.width;  x++)
              {    var avg = (idata.data[pos] + idata.data[pos + 1] + idata.data[pos + 2]) / 3
                   idata.data[pos] = avg
                   idata.data[pos + 1] = avg
                   idata.data[pos + 2] = avg
                   pos += 4
              }
         }
         ctx.putImageData(idata,  400, 0)
      }

</script><body></html>
```

pixel at location (0,0): 4 values
idata.data[0] // Red level
idata.data[1] // Green level
idata.data[2] // Blue level
idata.data[3] // Alpha level

Location (1,0) then follows,
idata.data[4] // Red level
idata.data[5] // Green level
idata.data[6] // Blue level
idata.data[7] // Alpha level

Image Transformations

```
<!DOCTYPE html> <html><head><title>Image Transformations</title></head><body>
<canvas id='mycanvas' width='1400' height='800'> This text is visible only in non-HTML5  </canvas>
 <script>
      var canvas = document.getElementById('mycanvas')
      canvas.style.border = '1px solid black
      var ctx = canvas.getContext('2d')

      var myimage = new Image()
      myimage.src = 'House.jpg'

      myimage.onload = function ()
      {   ctx.save()                    // Save the current state
          ctx.translate(400, 0)         //  moving the canvas and its origin x &y on the grid.
          ctx.scale(-1, 1)
          ctx.drawImage(myimage, 0, 0)
          ctx.restore()

          ctx.scale(3, 2)
          ctx.drawImage(myimage, 100, 100)

          ctx.scale(.2, 1)
          ctx.drawImage(myimage, 300, 200)
      }
</script><body></html>
```

Questions

Questions

1. How do you create a canvas element in HTML?

2. How do you give JavaScript access to a canvas element?

3. How do you start and finish the creation of a canvas path?

4. What method can you use to extract data from a canvas into an image?

5. How can you create gradient fills of more than two colors?

6. How can you adjust the width of lines when drawing?

7. How many items of data per pixel are returned by the getImageData method

1. <canvas id='mycanvas'> </canvas>

2. canvas = document.getElementById('mycanvas')
   ctx = canvas.getContext('2d')

3. ctx.beginPath() & ctx.closePath()

4. image.src = canvas.toDataURL()

5. gradient = ctx.createLinearGradient(x0, y0, x1, y1);
   gradient.addColorStop(0, 'green')
   gradient.addColorStop(0.79, 'orange')

6. ctx.lineWidth = 5

7. 4

Audio

Codecs

stands for encoder/decoder. It describes the functionality provided by software that encodes and decodes media such as audio and video.

Audio

```
<audio controls src="audio.mp3"> browser does not support the audio element </audio>
<audio controls>
        <source src='audio.mp3' type='audio/mpeg'>
        <source src='audio.ogg' type='audio/ogg'>
        Your browser does not support the audio element.
</audio>
```

<source> element specifies multiple media resources

Attributes

The <audio>, <video> elements and <source> tag support several attributes:

autoplay: start playing as soon as it is ready.

controls: display the control panel.

loop: play over and over.

src: specifies the source location of a file.

type: specifies the codec used in creating the file.

JavaScript must be added if no controls and no autoplay.

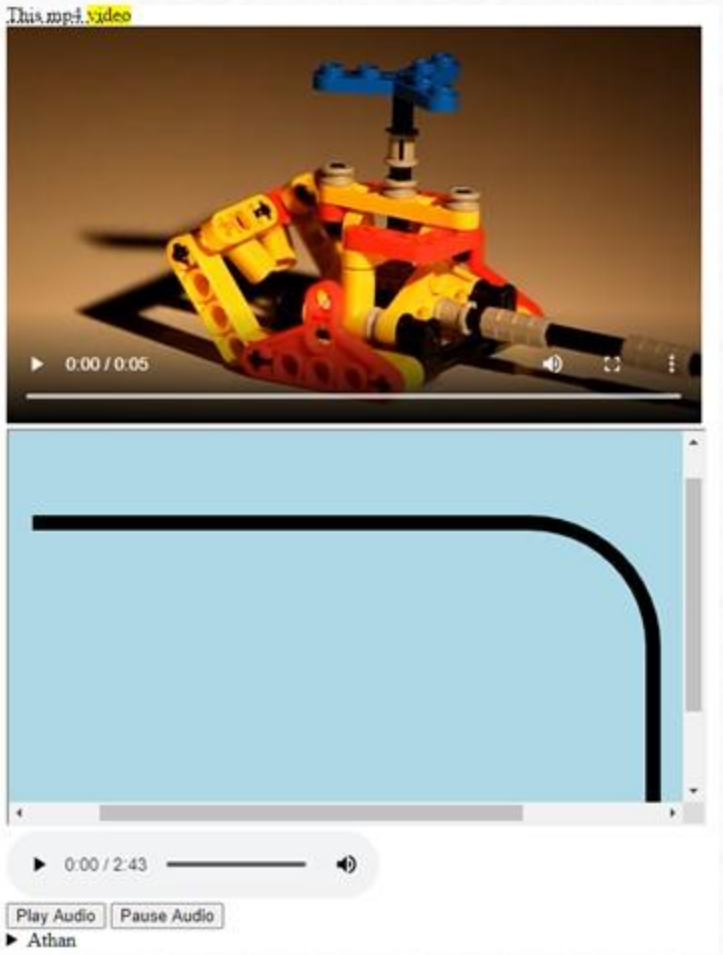
<iframe> an inline frame; an HTML document embedded inside another HTML document.

<details> specifies some details that the user can show and hide.

<abbr> abbreviation or short form of an element.

Audio n Video

```
<!DOCTYPE html> <html><head><title> Audio n Video </title></head><body>
   <abbr title="MPEG-4 Part 14 (MP4 file format), Second edition">This mp4 <mark>video</mark></abbr><br>

   <video controls width='560' height='320'>
      <source src='small.mp4' type='video/mp4'>
      <source src='VID_1.3gp' type='video/3gp'>
   </video><br>

   <iframe width='560' height ='316' src="ArcTo.html">not supported</iframe>

   <section style="width:560px; ">
                  <audio id='myaudio' controls autoplay loop>
                        <source src='Athan.m4a' type='audio/aac'>
                        <source src='Athan.mp3' type='audio/mp3'>
                        <source src='Athan.ogg' type='audio/ogg'>
                  </audio> <br>

                  <button onclick='playAudio()'>Play Audio</button>
                  <button onclick='pauseAudio()'>Pause Audio</button>

                  <details>         <summary>Athan</summary>
                                    <p>Madina</p>
                                    <p>2016</p>
                  </details>
   </section>
   <script>
        function playAudio()
        {       document.getElementById('myaudio').play()
        }

        function pauseAudio()
        {       document.getElementById('myaudio').pause()
        }
   </script></body></html>
```



Client-Side Storage

Enables

- Persisting previous site activity.
- Personalizing site preferences.
- Save disk space on servers.
- Load document up locally (faster) than from across the web.
- Saving web application generated documents locally for offline use.

Cookies

- Save only a maximum of 4 KB of data.
- Passed back and forth on every page reload.
- Transmitted in the clear unless the server uses encryption.

Client-Side Storage

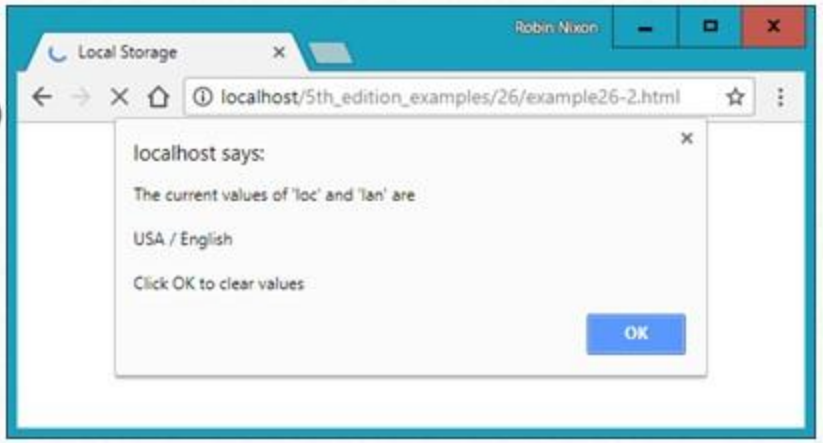
Local Storage

- Access to a much larger space between 5 and 10 MB per domain.
- Data is not sent to the server on each page load.
- Data is not accessible by any domain other than the one that stored it.

- **localStorage** stores data with no expiration date
- **sessionStorage** stores data for one session (data is lost when the browser is closed)
  - .**setItem**('loc', 'USA')
  - .**removeItem**('loc')
  - .**clear**() totally wipe the storage for the current domain
  - loc = .**getItem**('loc')

Local Storage

```
<!DOCTYPE html> <html><head><title> Local Storage </title></head><body>
<script>

    if (typeof localStorage == 'undefined') alert("Local storage is not supported.")
    else
    {
       loc = localStorage.getItem('loc')
       lan = localStorage.getItem('lan')
       alert("Current 'loc' and 'lan' are " + loc + " - " + lan)

       localStorage.setItem('loc', 'USA')
       localStorage.setItem('lan', 'English')

       loc = localStorage.getItem('loc')
       lan = localStorage.getItem('lan')
       alert("Current 'loc' and 'lan' are " + loc + " - " + lan)

       // localStorage.clear()
       localStorage.removeItem('loc')
       localStorage.removeItem('lan')
    }

</script>
</body></html>
```



Server Side Events

Server-Sent Events (SSE)

Web server automatically sends updates to web browsers called **server push**.

A client subscribes to an event from a server; whenever a new event occurs, the server sends it to the client.

Examples: Facebook / Twitter updates

Message Format

ID: ID for each event-stream; to track lost messages.

Event: event's type; to send different contents.

Data: message; can be more than one line

Retry: time in milliseconds - to use when resending the event.

Server Side Events

```
<!DOCTYPE html><html><head><title>SSE</title></head><body>
<button onclick="evtSource.close()"> End </button>
<div id="display"></div>
<script>
    var evtSource = new EventSource("sse.php");

    evtSource.onmessage = function(event) {
            document.getElementById("display").innerHTML += event.data + " (by onmessage) <br><br> ";
    }

    evtSource.addEventListener("news", function(event) {
            document.getElementById("display").innerHTML += event.data + " (by news) <br>";
    });

    evtSource.onerror = function(err) {
            console.error("EventSource failed:", err);
    };
</script>
</body></html>
```