HTML5 & CSS3

A chance to Do things Differently

Eng. Niveen Nasr El-Den
iTi

Day 3

GeoLocation

Geolocation

- The Geolocation API is one of the most exciting features of the new web standard.
- Geolocation is the art of figuring out where you are in the world and (optionally) sharing that information with people you trust.
- The ability to get device’s geographic location.
- It is set to request location once or continually.

Geolocation Facts

- HTML5 uses this API for working with maps.
- It is a new property that is added to the existing DOM browser object **navigator**
- The user must agree to share their location, and can tell the browser to remember his choice.

Geolocation Requesting Pattern

- To get user's current location (once)
  - navigator.geolocation.getCurrentPosition(x[,y,z])
    - x: is the onSuccess callback function where a Position object is passed in as the only invocation argument. This Position object contains a coords object which, in turn, contains our latitude and longitude, etc.. values.
    - y: is the errorHandler callback function where the object passed to this handler has code and message properties as follows:
      - 0: UNKNOWN_ERROR
      - 1: PERMISSION_DENIED
      - 2: POSITION_UNAVAILABLE
      - 3: TIMEOUT
    - z: is the options object

Location Option

- enableHighAccuracy (Boolean)
  - Attempt to gather more accurate location coordinates
  - May not do anything and cause request to take longer
  - Default **false**
- timeout (msec)
  - Determines max time allowed to calculate location
  - Default is **no limit**
- maximumAge (msec)
  - Determines how old location value may be before an attempt to refresh coordinates
  - Default is **0** (immediate recalc.)

Example

```
var options = {
  enableHighAccuracy: true,  //boolean (default: false)
  timeout: 10000,//00       // in ms (default: no limit)
  maximumAge: 1000         //  in ms (default: 0)
};

navigator.geolocation.getCurrentPosition(showPosition, positionError, options);

  function showPosition(position) {
    var coords = position.coords;
    console.log(coords.latitude);
    console.log(coords.longitude);
  }
```

```
function positionError(e){//error has code and message properties
  switch (e.code) {
    case 0: // e.UNKNOWN_ERROR -->error.UNKNOWN_ERROR
      console.log("The application has encountered an unknown error while trying\
      to determine your current location. Details: ")
      console.log(e.message);
      break;
    case 1: // e.PERMISSION_DENIED-->error.PERMISSION_DENIED
    //Permission denied - The user did not allow Geolocation
      console.log("You chose not to allow this application access to your location.");
      break;
    case 2: // e.POSITION_UNAVAILABLE--error.POSITION_UNAVAILABLE
    //Position unavailable - It is not possible to get the current location
      console.log("The application was unable to determine your location.");
      break;
    case 3: // e.TIMEOUT-->error.TIMEOUT
    //Timeout - The operation timed out
      console.log("The request to determine your location has timed out.");
      break;
  }
}
```

Geolocation Requesting Pattern

- To watch location change (continual)
  - navigator.geolocation.watchPosition(x[,y,z])
    - gets the user's current position and continually returns updated position.
  - navigator.geolocation.clearWatch()
    - used to stop "watchPosition()" running & execution.

https://www.sitepoint.com/html5-geolocation/

Web Storage APIs

Web Storage APIs

- Sometimes called DOM Storage
- Similar to http-cookies, for storing name-value pairs on the client side; but can store much larger amount of data.
- Two kinds for storing data on the client
  - localStorage
    - stores data with no expiration date
  - sessionStorage
    - stores data for one session

Web Storage APIs

- Web Storage APIs are instance of storage object, and can only store strings.
- It provide up to 5Mbytes per origin
- Same Origin Restrictions
- Stored as key/value pairs, and can only store strings
- We may need to check browser support before using Web Storage APIs & add its polyfill if needed

Storage Object Methods & Properties

- Methods
  - clear()
  - getItem(‘key’)
  - setItem(‘key’,’value’)
  - removeItem(‘key’)
  - key(idx)

- Properties
  - length

localStorage

window.localStorage

- Persistent on page reloads
- Data stored locally with no expiration date.
- Avoids HTTP overhead of cookies
- Great for storing user preferences

sessionStorage

window.sessionStorage

- Data stored for only one session
- Lasts as long as browser is open
- Opening page in new window or tab starts new session
- Good for sensitive data

https://html.spec.whatwg.org/multipage/webstorage.html

Cookies
Vs.
Web Storage
?

https://gist.github.com/remy/350433

New Element Enable & Feature Detection

New Element Enable

- Earlier IE doesn’t know how to render CSS on elements that it doesn’t recognize

- HTML5 Shiv or Shim by John Resig document.createElement(“….”) for all of the used tag

https://github.com/aFarkas/html5shiv/blob/master/src/html5shiv.js

API Feature Detection

- Modernizr.js
  - Implement HTML5 Shim
  - Apply classes to <html> based on what the browser support
  - Better place its script within <head> and after<style>
  - if(!Modernizr.localstorage){
    //provide polyfill
    }

http://html5please.com/#polyfill

https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-browser-Polyfills

API Feature Detection

- Modernizr.js
  - Runs automatically, creating a ***global*** object called ***Modernizr*** that contains a set of Boolean properties for each feature it can detect.
    - Example:
      if your browser supports the video API , the Modernizr.video property will be true.
      else, the Modernizr.video property will be false
  - By default, ***Modernizr*** sets classes for all of tests on the root element.
    - i.e. adding the class for each feature when it is supported, and adding it with a no- prefix when it is not.
  - It is recommended to add no-js class to root element

API Feature Detection

http://caniuse.com/

- Conditionally loading .js file
  - Conditionizr library
    - https://conditionizr.github.io/
    - https://github.com/conditionizr/conditionizr
  - Conditionize jQuery Plugin
    - https://github.com/renvrant/conditionize.js/tree/master
    - https://www.jqueryscript.net/form/jQuery-Plugin-For-Conditional-Form-Fields-conditionize-js.html

Loading Polyfills and/or shim|shiv files is no longer a common practice to provide compatibility

MathML

MathML

- MathML is an XML vocabulary for representing mathematical expressions
- The HTML5 specification provides native support for MathML in HTML documents
- MathML provides both Presentation and Content Markup models.
  - Presentation markup tags math expressions based on how they should be displayed
    - e.g., “superscripted 2”
  - Content markup tags expressions based on the mathematical operations performed
    - e.g., “taken to the 2nd power”

MathML Presentation Markup Glossary

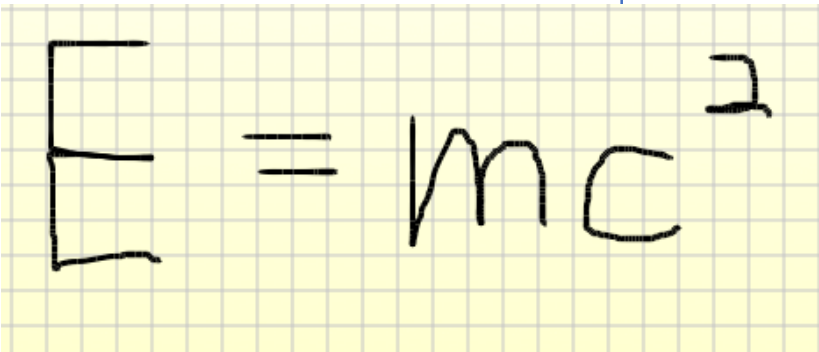
- <math> -- Root element for a mathematical expression
- <mrow> -- Element for grouping subexpressions
- <mo> -- Math operator (e.g., +, -)
- <mi> -- Math identifier (e.g., variable or constant)
- <mn> -- Number
- <mfrac> -- Fraction
- <msqrt> -- Square root
- <msup> -- Superscript
- <msub> -- Subscript
- etc..

https://developer.mozilla.org/en-US/docs/Web/MathML/Element

Converting Famous Eqn. to MathML

https://github.com/fred-wang/mathml.css

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mi> E </mi>
  <mo> = </mo>
  <mi> m </mi>
  <msup>
   <mrow>
    <mi> c </mi>
   </mrow>
   <mrow>
    <mn> 2 </mn>
   </mrow>
  </msup></math>
```



SVG

SVG

- SVG stands for **S**calable **V**ector **G**raphics and it is a language for describing 2D-graphics and graphical applications in XML

- SVG is W3C standard

- HTML5 allows embedding SVG directly using **<svg>...</svg>**

SVG

- SVG would draw

https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial

  - rectangle using
    - <rect x="" y="" width=“" height=“" style=“”>
  - line using
    - <line x1=“" y1=“” x2=“" y2=““ style=“”>
  - circle using
    - <circle cx="" cy=“" r=“" stroke=““ stroke-width=“" fill=““>
  - ellipse using
    - <ellipse cx=“" cy=“" rx=“" ry="“ style=“”>

SVG

- SVG would draw
  - path
    - <path d=“”>
  - polygon using
    - <polygon points=“”> tag
  - polyline using
    - <polyline points=“”> tag

http://tutorials.jenkov.com/svg/index.html

Canvas

Canvas

- Canvas is a new HTML element
- A canvas is a rectangular area, that you control every pixel of it.
- The canvas element has several methods for drawing paths, boxes, circles, characters, and adding images...

Canvas

- <canvas> element is an HTML tag, with the exception that its contents are rendered with JavaScript.

- It creates a fixed size drawing surface that exposes one or more *rendering contexts* using canvas context object.

- Each canvas element can only have one context that can be “2d”.

Canvas

- Draw dynamic and interactive graphics
- Draw images using 2D drawing API
  - Lines, curves, paths, shapes, fill styles, etc.
- Useful for:
  - Graphs
  - Applications
  - Games and Puzzles
  - And more…

Steps to follow

- Place the canvas tag somewhere inside the HTML document,
- Access the canvas tag with JavaScript,
- Create a 2D context, and then
- Utilize the HTML5 Canvas API to draw visualizations.

```
<canvas id="myCanvas" width="300" height="150"></canvas>

<script>
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');
    // do stuff here
</script>
```

Canvas Element & Canvas Context

- The canvas element is an actual DOM node that's embedded in the HTML page.
- The canvas context is an object with properties and methods that you can use to render graphics inside the canvas element.
- The context is *2d.*

Canvas Context Properties & Methods

- Color &Fill Styles
- Line
- Path
- Curve
  - Besier
  - Quadratic
- Shapes
  - Rectangle
  - Circle
  - Custom Shapes
- Text
- Shadows
- Images/Videos
- Clipping
- Transforms
  - Scale
  - Translate
  - Rotate
- Patterns
- Gradients
  - Linear
  - Radial

Line using HTML5 Canvas

- To draw a line using HTML5 Canvas

http://www.w3.org/TR/2dcontext/#building-paths

  - First, use the *beginPath()*
    - method to declare that we are about to draw a new path.
  - Next, use the *moveTo()*
    - method to position the context point (i.e. drawing cursor
  - Then, use the *lineTo()*
    - method to draw a straight line from the starting position to a new position.
  - Finally, to make the line visible, we can apply a stroke to the line using *stroke()*.
  - Note:
    - without declaring strokeStyle property before using stroke(), the stroke default color is black

Line useful Properties & Methods

- lineWidth
  - used to define width of the required line to be drawn in px,
  - should be declared before strokeStyle property.
- lineCap = square | round | butt
  - declares how the drawn line ends look
- lineJoin = bevel | round | miter
  - declares how two lines are joined together

Curves & Arcs Using HTML5 Canvas

arc(x, y, radius, startAngle, endAngle, antiClockwise);

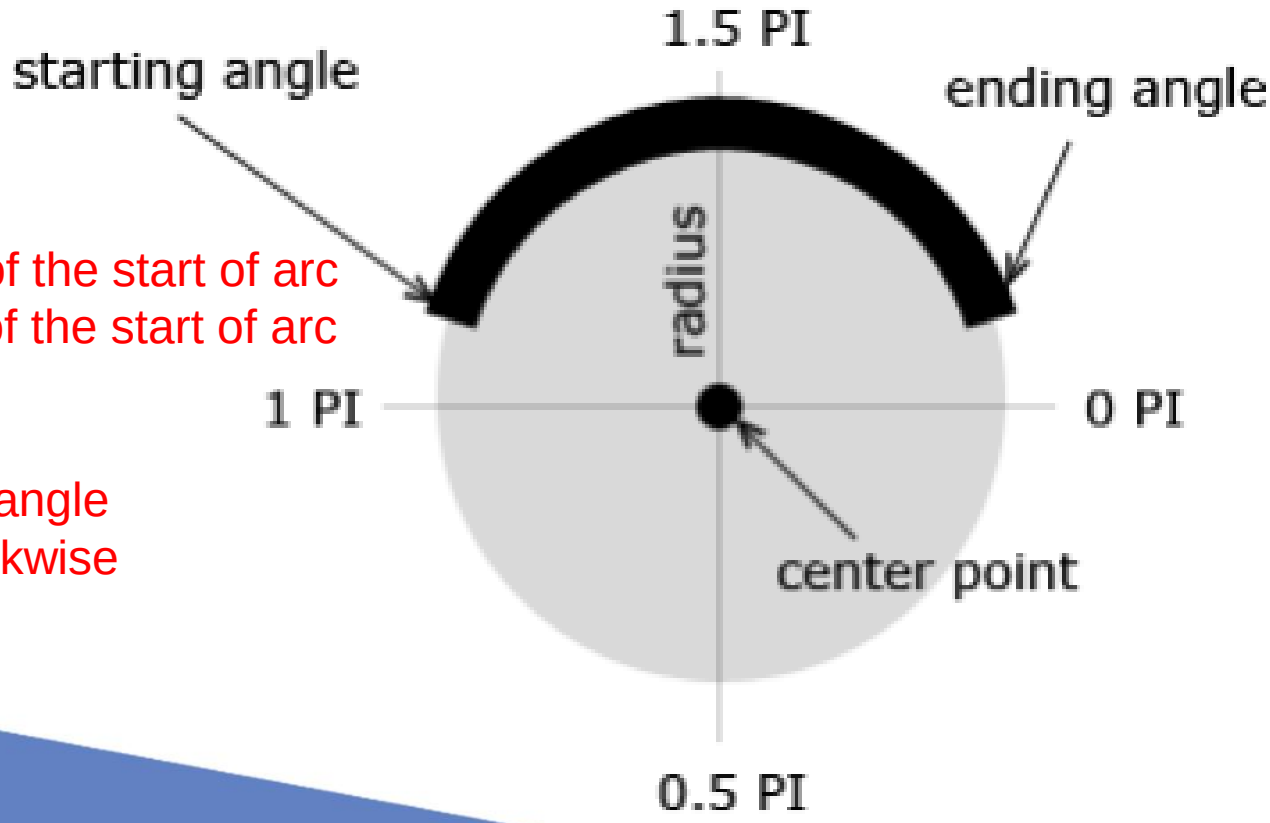
- An arc is nothing more than a section of the circumference of an imaginary circle that can be defined by *x*, *y*, and *radius*.
- *startAngle* and *endAngle*. These two angles are defined in radians.
- *antiClockwise* boolean value which defines the direction of the arc path between its two ending points, its default is false
  - i.e. the arc to be drawn is clockwise

Curves & Arcs Using HTML5 Canvas

- arc(x, y, radius, startAngle, endAngle, antiClockwise);
- arcTo(controlX,controlY,endX,endY,radius);

Example:

```
arc(
        210, // X coordinate of the start of arc
        210, // Y coordinate of the start of arc
        200,  // Radius
        0,     // Start angle
        Math.PI * 2,    // End angle
        true);          // Anticlockwise
```



Circle & Semi-Circle using HTML5 Canvas

- To draw a circle
  - Use *arc()* method and define its starting angle as 0 and the ending angle as 2 * PI.

```
arc(x, y, radius, 0, 2*Math.PI, anticlk);
```

- To draw a semi-circle
  - Use *arc()* method and define its ending angle has *startAngle* + PI.

```
arc(x, y, radius, sAngle,  sAngel+Math.PI, anticlk);
```

Rectangle using HTML5 Canvas

rect(x, y, width, height)

fillRect(x, y, width, height)

strokeRect(x, y, width, height)

clearRect(x, y, width, height)

roundRect(x, y, width, height, radii)

- An HTML5 Canvas rectangle is positioned with *x* and *y* parameters, and is sized with *width* and *height* parameters.
- Radii parameter is similar to border-radius of CSS property
- The rectangle is positioned about its top left corner.

Paths & shapes using HTML5 Canvas

- To create a path with HTML5 Canvas, connect multiple subpaths using
  - *lineTo()*,
  - *arcTo()*,
  - *quadraticCurveTo()*, and
  - *bezierCurveTo()*

- To create a custom shape
  - First create a path and mentioned above
  - Then, close it using the *closePath()*

- *Note:*
  - *beginPath()* is used in the beginning to start drawing a new path.
  - fillStyle property & fill() can be used to fill in color within drawn shape.

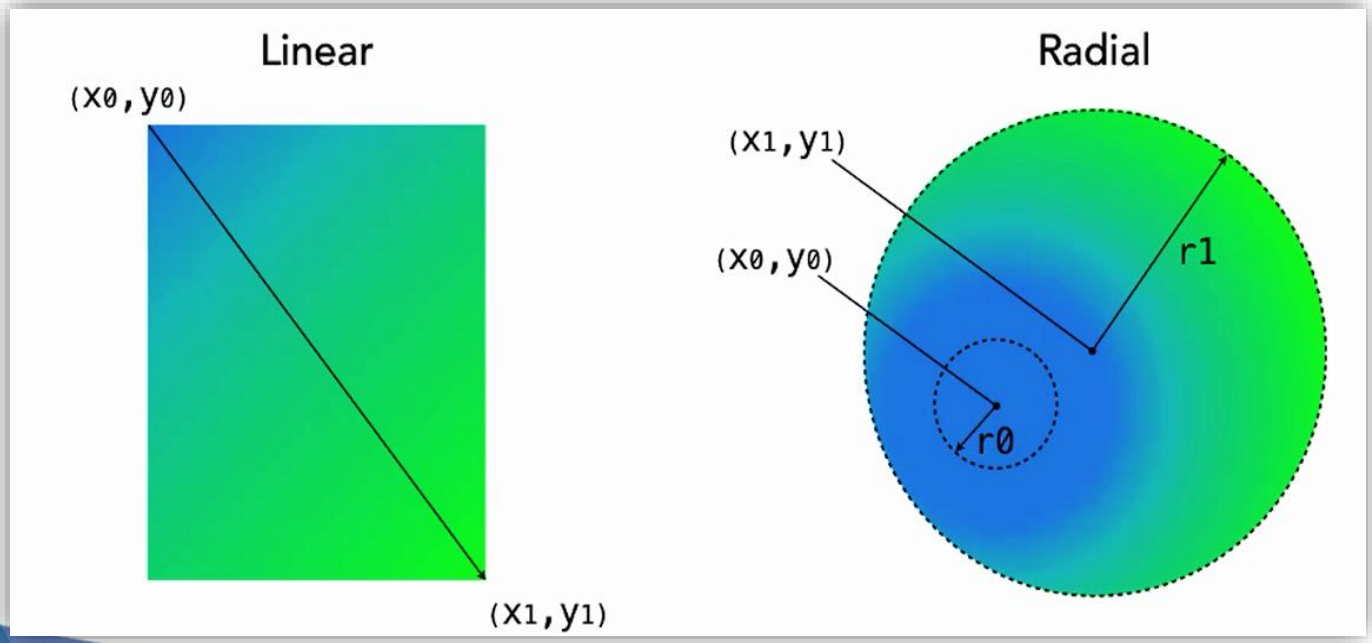
Text Properties & Methods

- font
  - style, size, font family
- fillStyle
  - color or rgb()
- fillText(txt, x, y)
- strokeStyle
  - color or rgb()
- strokeText(txt, x, y)
- textAlien, textBaseline, measureText(txt)…

http://diveintohtml5.info/canvas.html#text

Gradient

- Gradient can be used to fill rectangles, circles, lines, text, etc.. it can be used anywhere a stroke or fill is used
- Two types of gradient
  - Linear Gradient
  - Radial Gradient



Gradient

- Linear Gradient
  - ▻ `createLinearGradient(startX, startY, endX, endY);`
- Radial Gradient
  - ▻ `createRadialGradient(startX, startY, startRadius, endX, endY, endRadius);`
- Note:
  - ▻ Add color stops to create color transitions using `addColorStop(offset, color);`
    - It can be called multiple times to change a gradient
    - Its offset value between 0.0 and 1.0

Dealing with Image

- To draw an image on canvas area we use
  - *drawImage(imgObj, x, y [, width, height])*
    - *imgObj* defines image required to be displayed, it must be created first and wait for being loaded befor instantiating drawImage().
    - *x,y defines* top left corner of the image relative to the top left corner of the canvas (0,0)
    - *width, height define width, height of the displayed image*
  - Note:
    - Construct your image object using “new Image()”

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage

Transformation

- Transformation affects all drawing operations that come after it
- 3 basic transformation
  - Translate
  - Scale
  - Rotate
- Transformation is additive
- Its good using save() & restore() for the context state

Scaling, Rotating & Translating

- scale(x, y)
  - resize current drawing either bigger or smaller
- rotate(angle)
  - rotate the current context around the origin within the canvas area
- translate(x, y)
  - move current context within the canvas area into a different point

Saving & Restoring Canvas State

- Every canvas object contains a stack of drawing states.
- The canvas state can store:
  - strokeStyle
  - fillStyle
  - font
  - globalAlpha
  - lineWidth
  - lineCap
  - lineJoin
  - miterLimit
  - The current transformation matrix (rotation, scaling, translation)
  - shadowOffsetX
  - shadowOffsetY
  - shadowBlur
  - shadowColor
  - globalCompositeOperation
  - textAlign
  - textBaseline
  - The current clipping region

Assignment