Web Programming (CSci 130)

Department of Computer Science

College of Science and Mathematics

California State University Fresno

H. Cecotti

Learning outcomes

- The **canvas** in Javascript
- Some remarks about Javascript:
    - When you use a variable, ask yourself :
        - What is its type?
            - Object, symbol, string, number …
        - To what element is it connected in the whole structure?
            - Simple variable
            - Object
            - Property of an object
        - What do you want to do with it?
            - Access the value, update the value?
        - HTML (ID) ←→ DOM Object ←→ variable to access/modify

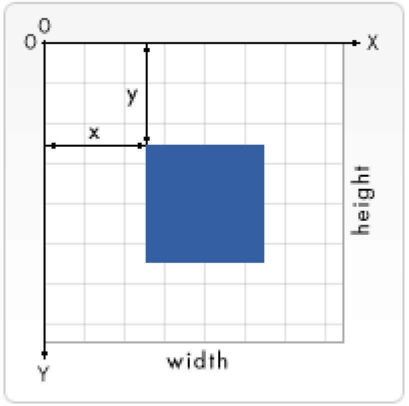
Canvas API

- Added to HTML5
  - <canvas>
    - To draw graphics via scripts in Javascript
      - Graphs, photo compositions, animations, real time video processing, video games…
  - HTML
    - <canvas id="Mycanvas"></canvas>
  - JavaScript
    - var canvas = document.getElementById('Mycanvas');
    - var ctx = canvas.getContext('2d');
    - ctx.fillStyle = 'blue';
    - ctx.fillRect(20, 20, 100, 100); // position and size

Canvas

- "Similar" to an image (<img>)
- HTML
  - <canvas id="MyCanvas" width="150" height="150"></canvas>
    - Default size 300px wide x 150px height
    - Styled with
      - Margin, border, background…
  - Check for support

```
if (canvas.getContext) {
  var ctx = canvas.getContext('2d');
  // drawing code here
} else {
  // canvas-unsupported code here
}
```



Canvas: Draw rectangles

- Inputs
  - position (x,y) + size
- fillRect(x, y, width, height)
  - Draws a filled rectangle.
- strokeRect(x, y, width, height)
  - Draws a rectangular outline.
- clearRect(x, y, width, height)
  - Clears the specified rectangular area, making it fully transparent.

Canvas: Draw paths

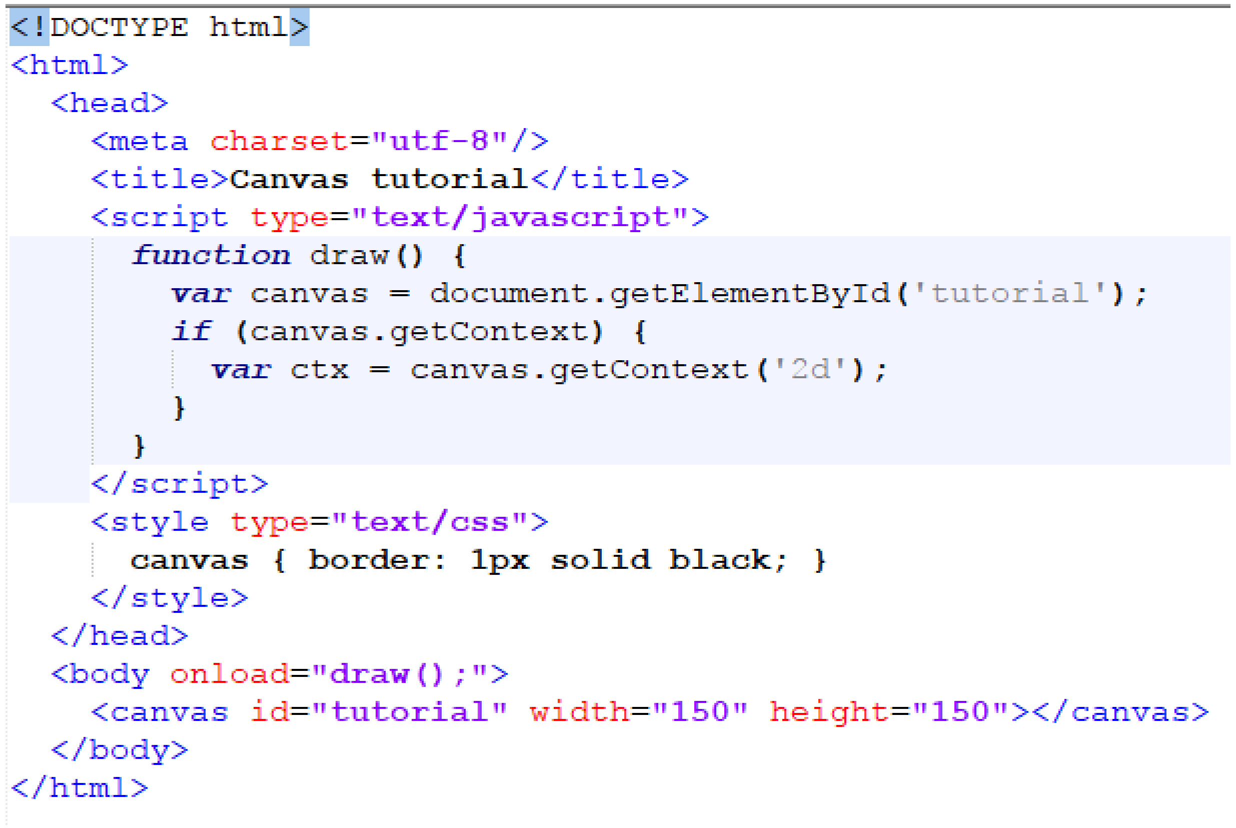
- CanvasRenderingContext2D.beginPath()
  - Starts a new path by emptying the list of sub-paths.
    - → Call this method when you want to create a new path.
- CanvasRenderingContext2D.closePath()
  - Causes the point of the pen to move **back** to the start of the current sub-path.
  - It tries to draw a straight line from the current point to the start.
  - If the shape has already been closed or has only one point, this function does nothing.
- CanvasRenderingContext2D.moveTo()
  - Moves the starting point of a new sub-path to the (x, y) coordinates.
- CanvasRenderingContext2D.lineTo()
  - Connects the last point in the subpath to the x, y coordinates with a straight line.
- CanvasRenderingContext2D.bezierCurveTo()
  - Adds a **cubic Bézier curve** to the path. (3 points).
    - The first 2 points are control points and the third one is the end point.
    - The starting point is the last point in the current path, which can be changed using moveTo() before creating the Bézier curve.

Canvas: Draw paths

- CanvasRenderingContext2D.quadraticCurveTo()
  - Adds a quadratic Bézier curve to the current path.
- CanvasRenderingContext2D.arc()
  - Adds an arc to the path which is centered at (x, y) position with
    - radius **r** starting at **startAngle** and ending at **endAngle** going in the given direction by **anticlockwise** (defaulting to clockwise).
- CanvasRenderingContext2D.arcTo()
  - Adds an arc to the path with the given control points and radius
  - connected to the previous point by a straight line.
- CanvasRenderingContext2D.ellipse()
  - Adds an ellipse to the path which is centered at (x, y) position with the radii **radiusX** and **radiusY** starting at **startAngle** and ending at **endAngle** going in the given direction by anticlockwise (defaulting to clockwise).
- CanvasRenderingContext2D.rect()
  - Creates a path for a rectangle at position (x, y) with a size that is determined by width and height.

Canvas: Example

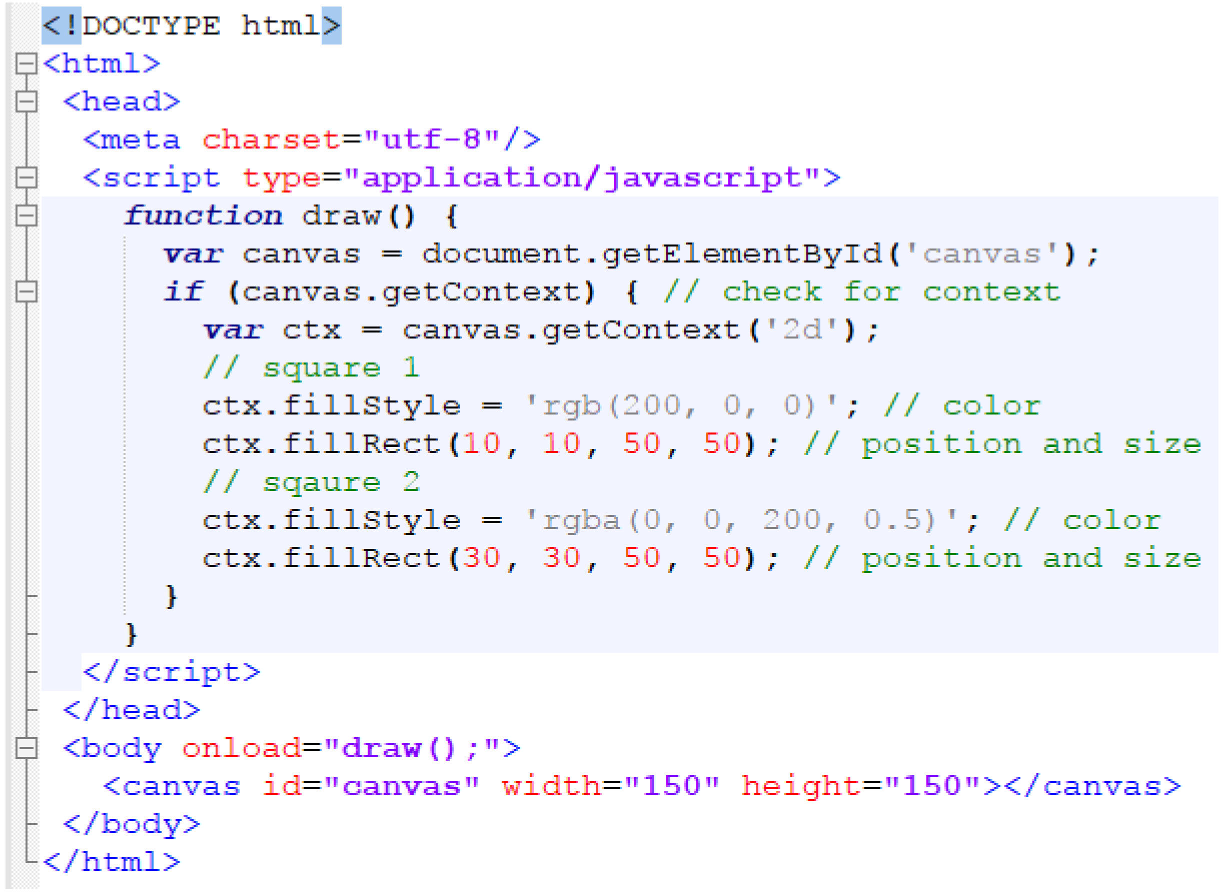
- The main structure:



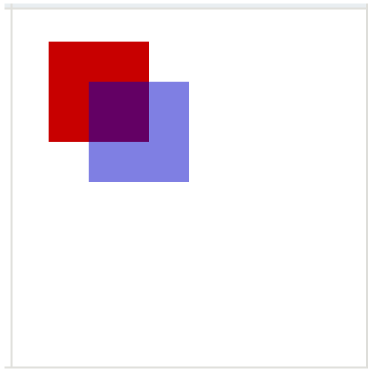
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Canvas tutorial</title>
    <script type="text/javascript">
      function draw() {
        var canvas = document.getElementById('tutorial');
        if (canvas.getContext) {
          var ctx = canvas.getContext('2d');
        }
      }
    </script>
    <style type="text/css">
      canvas { border: 1px solid black; }
    </style>
  </head>
  <body onload="draw();">
    <canvas id="tutorial" width="150" height="150"></canvas>
  </body>
</html>
```

Canvas: Example

- Example:
  - ➢Rectangle

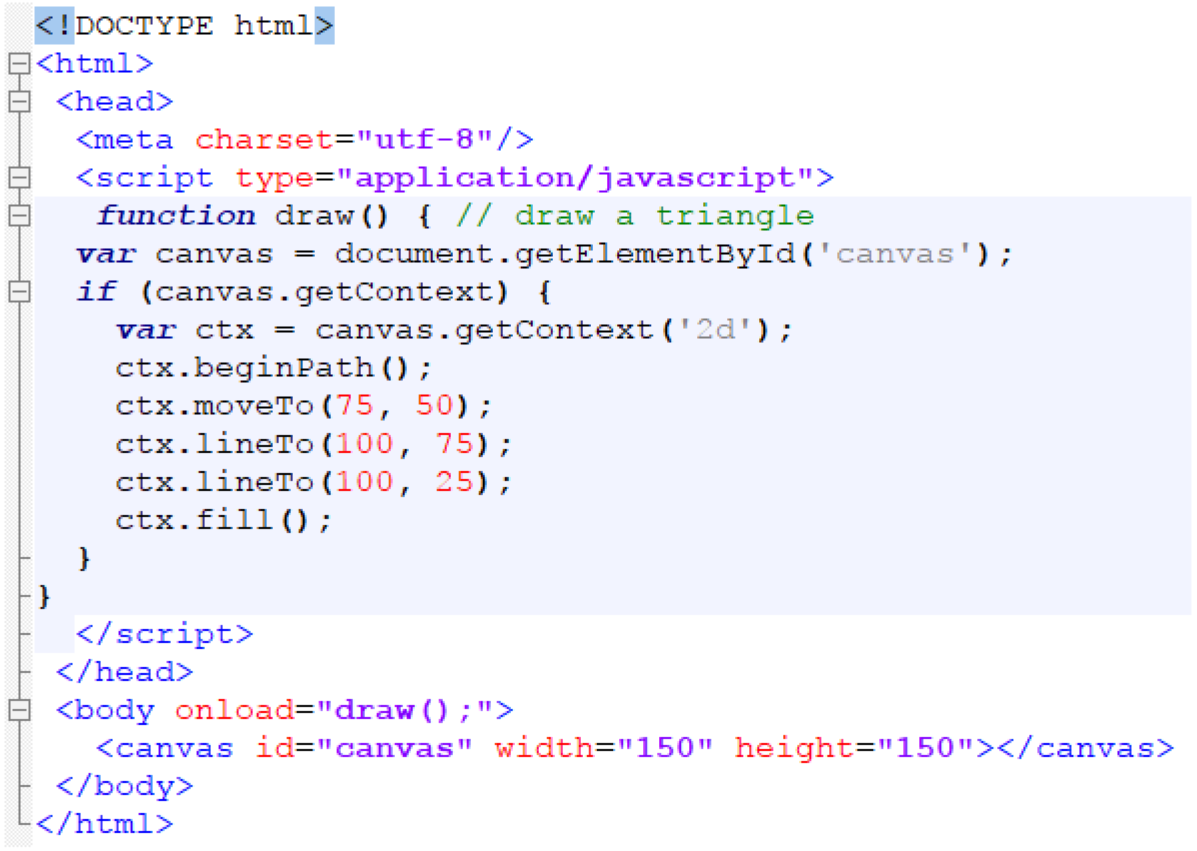


```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
    function draw() {
      var canvas = document.getElementById('canvas');
      if (canvas.getContext) { // check for context
        var ctx = canvas.getContext('2d');
        // square 1
        ctx.fillStyle = 'rgb(200, 0, 0)'; // color
        ctx.fillRect(10, 10, 50, 50); // position and size
        // sqaure 2
        ctx.fillStyle = 'rgba(0, 0, 200, 0.5)'; // color
        ctx.fillRect(30, 30, 50, 50); // position and size
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```



Canvas: Example

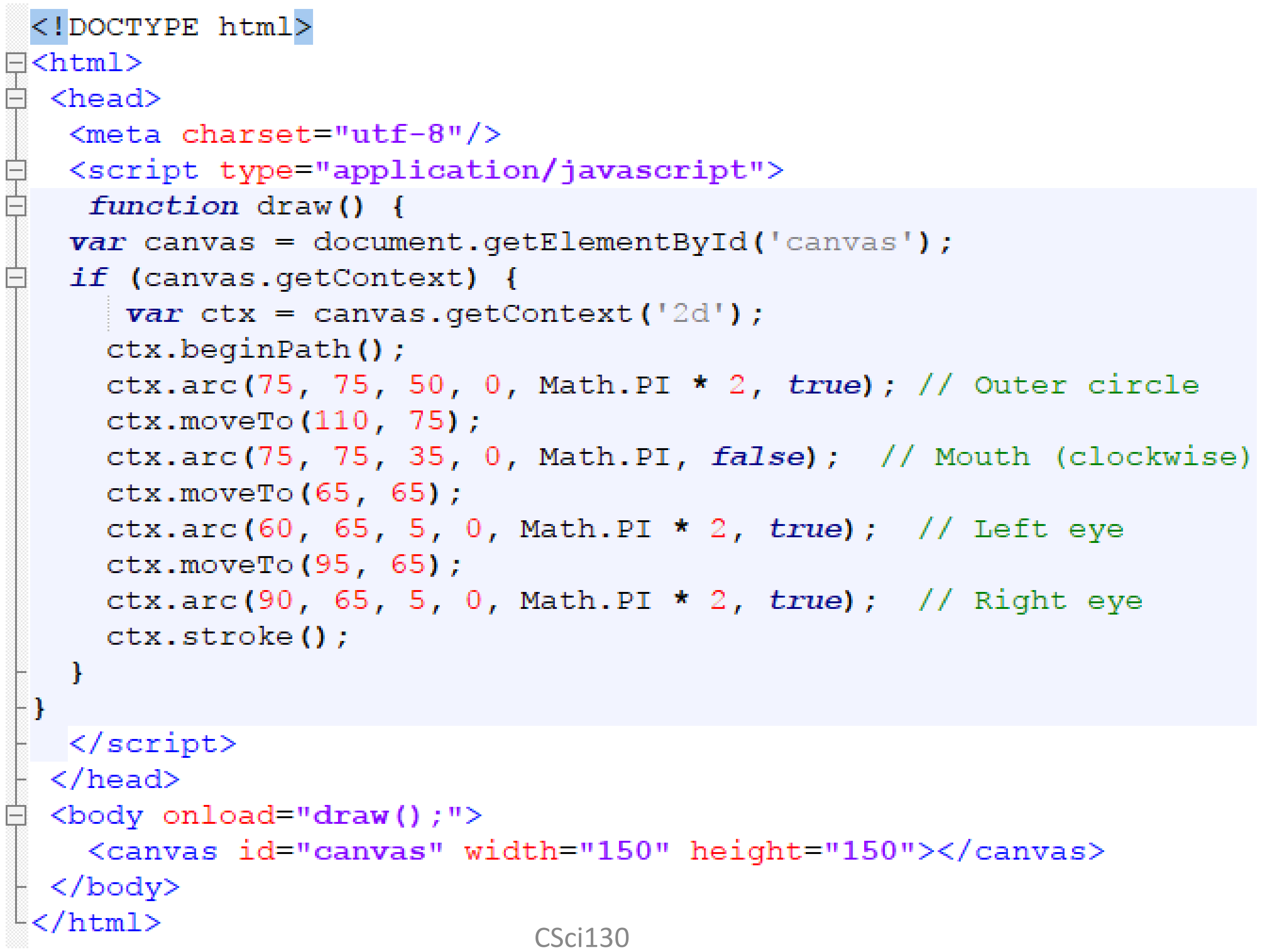
- Example:
  - Triangle



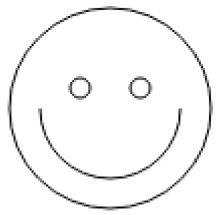
```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
   function draw() { // draw a triangle
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
    ctx.beginPath();
    ctx.moveTo(75, 50);
    ctx.lineTo(100, 75);
    ctx.lineTo(100, 25);
    ctx.fill();
  }
}
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

Canvas: Example

- Example
  - ➢Smiley

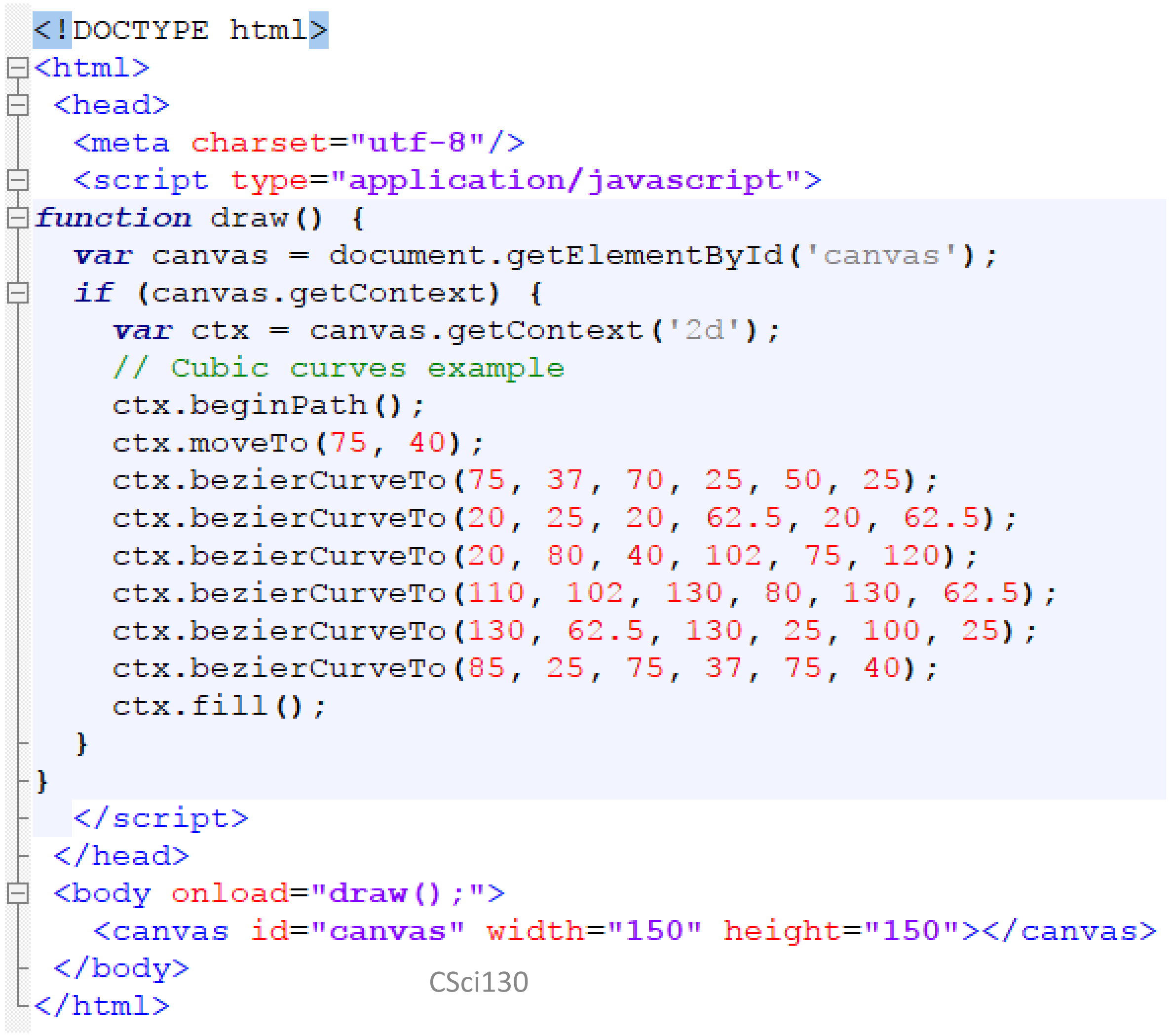


```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
   function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
     var ctx = canvas.getContext('2d');
    ctx.beginPath();
    ctx.arc(75, 75, 50, 0, Math.PI * 2, true); // Outer circle
    ctx.moveTo(110, 75);
    ctx.arc(75, 75, 35, 0, Math.PI, false);  // Mouth (clockwise)
    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();
  }
}
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

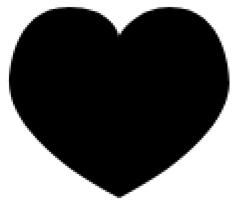


Canvas: Example

- Example
  - Bezier curve



```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
    // Cubic curves 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();
  }
}
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

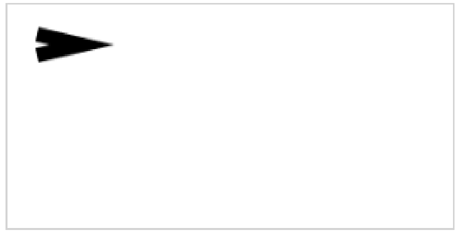


Canvas: Colors

- fillStyle = color
  - Sets the style used when filling shapes.
- strokeStyle = color
  - Sets the style for shapes' outlines.
- Format
  - ctx.fillStyle = 'orange'; // string (predefined colors)
  - ctx.fillStyle = '#FFA500'; // Hexadecimal : one channel = 2 bytes
  - ctx.fillStyle = 'rgb(255, 165, 0)'; // RGB
  - ctx.fillStyle = 'rgba(255, 165, 0, 1)'; // RGBA

Canvas: line styles

- lineWidth = value
  - ➢ Sets the width of lines drawn in the future.
- lineCap = type
  - ➢ Sets the appearance of the ends of lines.
- lineJoin = type
  - ➢ Sets the appearance of the "corners" where lines meet.
- miterLimit = value
  - ➢ Establishes a limit on the miter when two lines join at a sharp angle, to let you control how thick the junction becomes.
- getLineDash()
  - ➢ Returns the current line dash pattern array containing an even number of non-negative numbers.
- setLineDash(segments)
  - ➢ Sets the current line dash pattern.
- lineDashOffset = value
  - ➢ Specifies where to start a dash array on a line



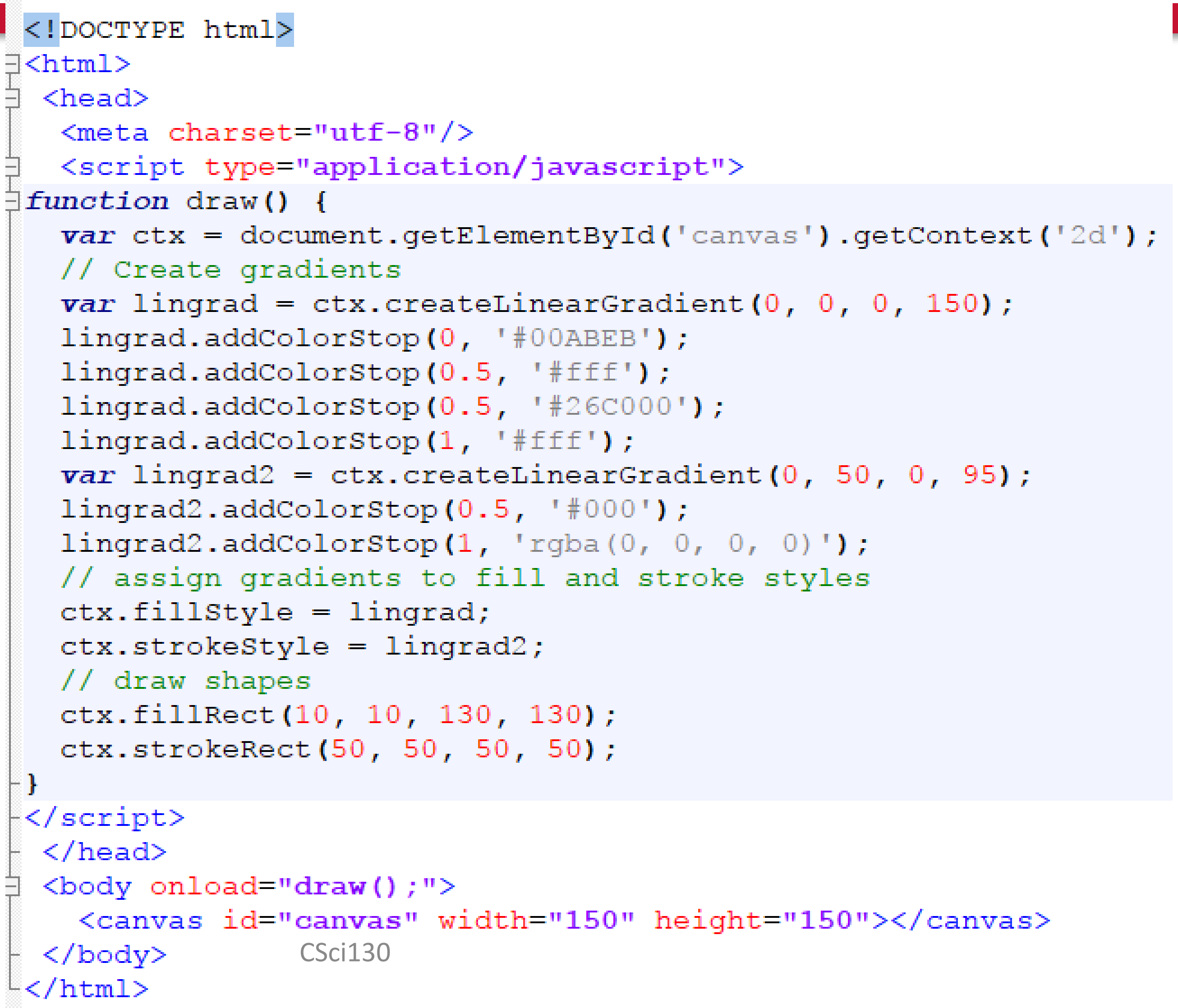
```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.lineWidth = 10;
ctx.lineJoin = "miter";
ctx.miterLimit = 8;
ctx.moveTo(20, 20);
ctx.lineTo(50, 27);
ctx.lineTo(20, 34);
ctx.stroke();
</script>
```

Canvas: special effects

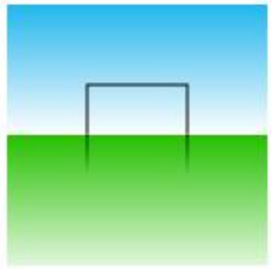
- Gradients (same effect as with CSS)
  - Linear, gradial gradients
    - createLinearGradient(x1, y1, x2, y2)
      - Creates a linear gradient object with a starting point of (x1, y1) and an end point of (x2, y2).
    - createRadialGradient(x1, y1, r1, x2, y2, r2)
      - Creates a radial gradient.
      - Parameters: 2 circles
        - one with its center at (x1, y1) and a radius of r1
        - the other with its center at (x2, y2) with a radius of r2
    - gradient.addColorStop(position, color)
      - Creates a new color stop on the gradient object.
      - Position: number between 0.0 and 1.0
        - defines the relative position of the color in the gradient
          - the color argument must be a string representing a CSS <color>
            - indicating the color the gradient should reach at that offset into the transition.

Canvas: special effects

- Example
  - Gradient



```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <script type="application/javascript">
function draw() {
  var ctx = document.getElementById('canvas').getContext('2d');
  // Create gradients
  var lingrad = ctx.createLinearGradient(0, 0, 0, 150);
  lingrad.addColorStop(0, '#00ABEB');
  lingrad.addColorStop(0.5, '#fff');
  lingrad.addColorStop(0.5, '#26C000');
  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);
}
</script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```



Canvas: special effects

- Shadows
  - shadowOffsetX = float
    - Indicates the horizontal distance the shadow should extend from the object.
    - Not affected by the transformation matrix.
      - Default value = 0.
  - shadowOffsetY = float
    - Indicates the vertical distance the shadow should extend from the object.
    - Not affected by the transformation matrix.
      - Default value = 0.
  - shadowBlur = float
    - Indicates the size of the blurring effect
    - No correspondence to a number of pixels
    - Not affected by the current transformation matrix.
      - Default value = 0.
  - shadowColor = color
    - A standard CSS color value indicating the color of the shadow effect
      - Default: it is fully-transparent black.

Canvas: draw text

- fillText(text, x, y [, maxWidth])
  - Fills a given text at the given (x,y) position.
    - Optionally with a maximum width to draw.
- strokeText(text, x, y [, maxWidth])
  - Strokes a given text at the given (x,y) position.
    - Optionally with a maximum width to draw.
- font = value
  - The current text style being used when drawing text.
  - Same syntax as the CSS font property.
    - Default font = 10px sans-serif.
- textAlign = value
  - Text alignment setting. Possible values: start, end, left, right or center.
    - Default value = start.
- textBaseline = value
  - Baseline alignment setting.
  - Possible values: top, hanging, middle, alphabetic, ideographic, bottom.
    - Default value = alphabetic.
- direction = value
  - Directionality.
  - Possible values: ltr, rtl, inherit.
    - Default value = inherit

Canvas: draw existing images

- drawImage(image, x, y)
  - Draws the CanvasImageSource
    - specified by the image parameter at the coordinates (x, y).
- drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
  - Given an image (image),
    - the area of the source image specified by the rectangle whose **top-left** corner is (sx, sy)
    - width and height : sWidth and sHeight
    - → draws it into the canvas
      - Placing it on the canvas at (dx, dy)
      - Scaling it to the size specified by dWidth and dHeight.

Canvas: geometric transformation

- translate(x, y)
  - Moves the canvas and its origin on the grid.
    - x: how far to move the grid horizontal
    - y: how far to move the grid vertically
- rotate(angle)
  - Rotates the canvas **clockwise** around the current origin by the angle number of **radians**.
- scale(x, y)
  - Scales the canvas units by x horizontally and by y vertically.
  - Both parameters are **real numbers**.
    - Values < 1.0 → reduce the unit size
    - Values > 1.0 → increase the unit size.
    - Values == 1.0 → leave the units the same size.
- transform(a, b, c, d, e, f)
  - a (m11) : Horizontal scaling.
  - b (m12): Horizontal skewing.
  - c (m21): Vertical skewing.
  - d (m22): Vertical scaling.
  - e (dx): Horizontal moving.
  - f (dy): Vertical moving.

Canvas: saving states

- save()
  - Saves the entire state of the canvas.
- restore()
  - Restores the most recently saved canvas state.
- Canvas states
  - stored on a stack.
  - Every time the save() method is called → the current drawing state is pushed onto the stack.

Canvas: animation

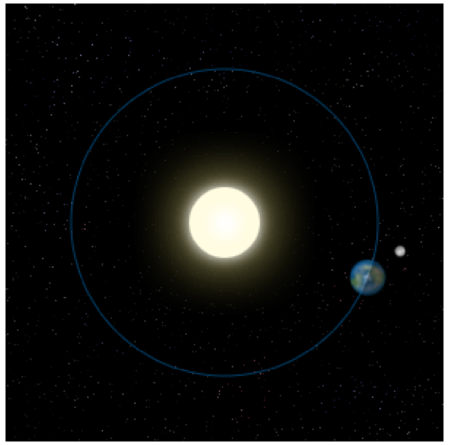
- Animation frame by frame
  - Clear the canvas
    - Unless the shapes you'll be drawing fill the complete canvas
      - Example: a backdrop image
    - you need to clear any shapes that have been drawn previously!
      - The easiest way: clearRect() method.
  - Save the canvas state
    - If you're changing any setting
      - styles, transformations, etc.
    - which affect the canvas state and you want to make sure the original state is used each time a frame is drawn
      - → you need to save that original state.
  - Draw animated shapes
    - The step where you do the actual frame rendering.
  - Restore the canvas state
    - If you've saved the state, restore it before drawing a new frame!

Canvas: animation

- Control
  - setInterval(function, delay)
    - Starts repeatedly executing the function specified by function every delay milliseconds (ms)
  - setTimeout(function, delay)
    - Executes the function specified by function in delay milliseconds (ms).
  - requestAnimationFrame(callback)
    - Tells the browser that:
      - you wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint.

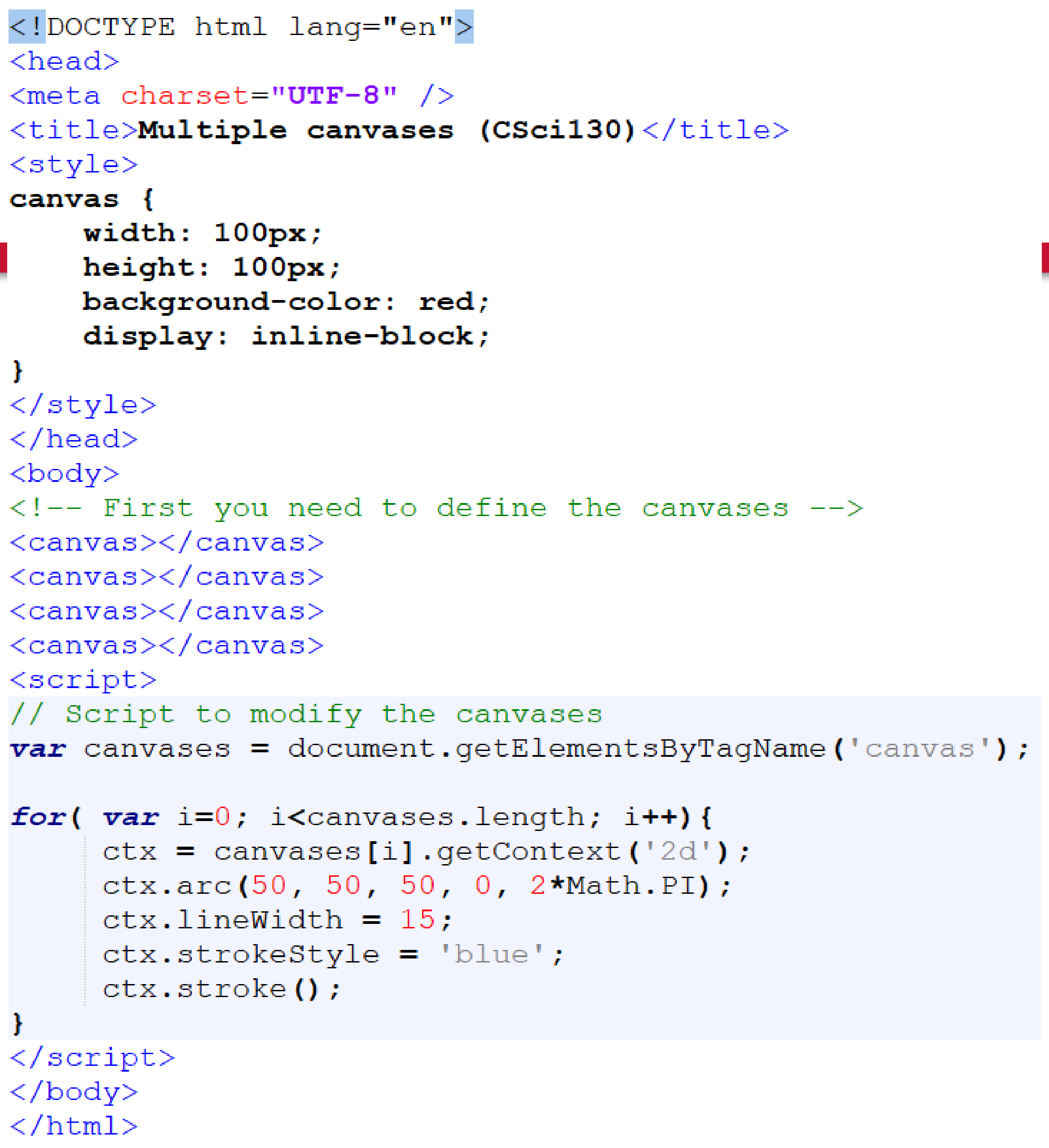
Canvas: animation

- See examples on blackboard:
  - class_javascript_canvas_01.html
  - class_javascript_canvas_02.html
    - Moon, Earth, Sun animation
  - class_javascript_canvas_03.html
    - Mouse event + animation
    - More about the mouse:
      - https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent



Canvases

- Example
  - ➢CSS+HTML+JS



```
<!DOCTYPE html lang="en">
<head>
<meta charset="UTF-8" />
<title>Multiple canvases (CSci130)</title>
<style>
canvas {
    width: 100px;
    height: 100px;
    background-color: red;
    display: inline-block;
}
</style>
</head>
<body>
<!-- First you need to define the canvases -->
<canvas></canvas>
<canvas></canvas>
<canvas></canvas>
<canvas></canvas>
<script>
// Script to modify the canvases
var canvases = document.getElementsByTagName('canvas');

for( var i=0; i<canvases.length; i++){
     ctx = canvases[i].getContext('2d');
     ctx.arc(50, 50, 50, 0, 2*Math.PI);
     ctx.lineWidth = 15;
     ctx.strokeStyle = 'blue';
     ctx.stroke();
}
</script>
</body>
</html>
```

Conclusion

- Animation in Web Programming
  - Several approaches, depending on the complexity of the scene to display:
    - CSS, JS, multimedia files embedded in HTML
- JS animation
  - The canvas for drawing elements
    - A powerful tool to create animations
      - Use it wisely for the project – you may not need it !
    - To create interactions between the user and the website
      - 2D Video games
  - Events (mouse)
    - To interact with graphical elements inside the canvas
- Midterm/Final type of question
  - Given an animation or an image, and partially completed code: complete the code
  - Set the right parameters for a desired effect