

# Introduction to HTML5 canvas

By M. Barsky

# Today

- We are going to learn how to dynamically draw on the HTML page.
- In preparation to our first big assignment – creating board games in the browser.

# You plan to add

a real-time graphics into your HTML page:

- A chart based on dynamic data
- Animation
- Game rendering engine
- User-driven graphics updates

What technology to use?

# Choose the “correct” answer

- A. Just use Flash, it works on most browsers.
- B. Take a look at HTML5 and see if there are any new technologies that might help (hint: there might be one named *canvas*).
- C. Write a custom application for every device, that way you know the exact experience you’re going to get.
- D. Just compute the image on the server side and deliver a custom image back to the browser.

# Canvas

- HTML 5 defines [the <canvas> element](#) as “a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly.”
- With HTML5’s new canvas element, you’ve got the power to create, manipulate and destroy *pixels*, right in your own hands.

# Canvas as an HTML element

- A *canvas* is a rectangle in your page where you can draw anything you want using JavaScript.
- Creating a canvas element:

```
<section>
```

```
  <canvas id="boardCanvas" width="240" height="240">
```

```
  </canvas>
```

```
</section>
```

# To see canvas – add a border

```
canvas
{
    border:2px solid black;
    position:absolute;
    top:50%;
    left:50%;
    margin-left:-120px;
    margin-top:-120px;
}
```

[TTT board 0.html](#)

# Accessing canvas in JavaScript

- You can have more than one `<canvas>` element on the same page.
- Each canvas will show up in the DOM, and each canvas maintains its own state.
- If you give each canvas an *id* attribute, you can access them just like any other element:

```
<canvas id="boardCanvas" width="240" height="240">  
</canvas>
```

```
var canvasElem = document.getElementById("boardCanvas");
```



# Creating canvas element on the fly

```
var canvas = document.createElement('canvas');  
canvas.width = window.clientWidth;  
canvas.height = window.clientHeight;
```

# 2D context

```
canvasElem = document.getElementById("boardCanvas");  
context = canvasElem.getContext("2d");
```

- Every canvas has a drawing *context*. Once you've found a <canvas> element in the DOM, you call its `getContext()` method. You **must** pass the string "2d" to the `getContext()` method.
- The drawing context is where all the drawing methods and properties are defined.

---

The HTML5 specification notes, "A future version of this specification will probably define a 3d context." For now, take a look at [WebGL](#)

# Context properties for drawing rectangles

- The **fillStyle** property can be a CSS color, a pattern, or a gradient. (More on gradients shortly.) The default fillStyle is solid black, but you can set it to whatever you like. Each drawing context remembers its own properties as long as the page is open, unless you do something to reset it.
  - **fillRect** (x, y, width, height) draws a rectangle filled with the current fill style.

# Context properties for drawing rectangles

- The **strokeStyle** property is like `fillStyle` — it can be a CSS color, a pattern, or a gradient.
  - **strokeRect**(x, y, width, height) draws a rectangle with the current stroke style. `strokeRect` doesn't fill in the middle; it just draws the edges.
  - **clearRect**(x, y, width, height) clears the pixels in the specified rectangle.

# Drawing a rectangle

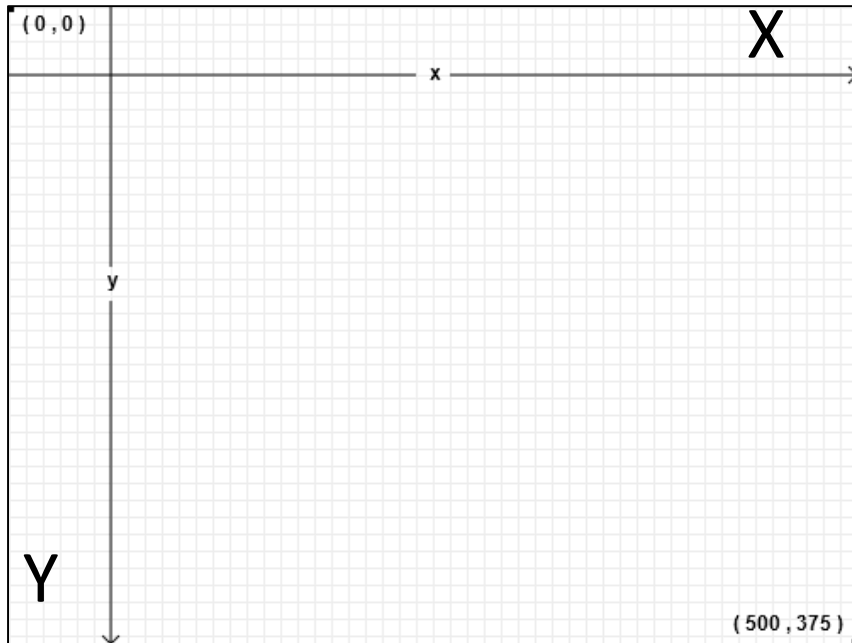
```
//draw filled rectangle at coordinate x, y, size: width, height  
context.fillRect(50, 25, 150, 100);
```

[TTT board 1.html](#)

[canvas intro 1.js](#)

- Calling the `fillRect()` method draws the rectangle and fills it with the current fill style, which is black until you change it.
- The rectangle is bounded by its upper-left corner (50, 25), its width (150), and its height (100).

# Coordinate system



- When you define width and height of a canvas element, you define the amount of pixels to be stored in this canvas
- If you define canvas size using CSS, this will stretch or compress the displayed result accordingly but will not have an effect on the number of pixels
- The default (if not defined explicitly) canvas size: 300 x 150 pixels

# Drawing lines

- Imagine you're drawing a picture in ink. You don't want to just dive-in and start drawing with ink, because you might make a mistake. Instead, you **sketch** the lines and curves with a pencil, and once you're happy with it, you **trace** over your sketch in ink.
- Each canvas has a ***path***. Defining the path is like drawing with a pencil. You can draw whatever you like, but it won't be part of the finished product until you pick up the quill and trace over your path in ink.

# Drawing lines

- To draw straight lines “in pencil”, you use the following two methods:
  - `moveTo(x, y)` moves the pencil to the specified starting point.
  - `lineTo(x, y)` draws a line to the specified ending point.
- “Inking” methods include:
  - `stroke()`
  - `fill()`



# Stroking a line

```
h = canvasElem.height;  
w = canvasElem.width;  
cellH = h/3;  
cellW = w/3;  
  
//stroke a line  
context.beginPath();  
context.moveTo(0,cellH);  
context.lineTo(w,cellH);  
context.stroke();
```

[TTT board 2.html](#)

[canvas intro 2.js](#)

# Drawing a 3 x 3 grid

The more you call `moveTo()` and `lineTo()`, the bigger the path gets. These are “pencil” methods — you can call them as often as you like, but you won’t see anything on the canvas until you call one of the “ink” methods.

```
//stroke 2 horizontal and 2 vertical lines to create a grid
context.beginPath();
for (var i=1; i< 3; i++)
{
    context.moveTo(0,i*cellH);
    context.lineTo(w,i*cellH);

    context.moveTo(i*cellW,0);
    context.lineTo(i*cellW,h);
}
context.stroke();
```

[TTT board 3.html](#)  
[canvas\\_intro\\_3.js](#)

# Drawing images

- The canvas drawing context defines a `drawImage()` method for drawing an image on a canvas.
- The method can take three, five, or nine arguments.
  - `drawImage(image, dx, dy)` takes an image and draws it on the canvas. The given coordinates (dx, dy) will be the upper-left corner of the image. Coordinates (0, 0) would draw the image at the upper-left corner of the canvas.
  - `drawImage(image, dx, dy, dw, dh)` takes an image, scales it to a width of dw and a height of dh, and draws it on the canvas at coordinates (dx, dy).

# Getting an access to an image

- To draw an image on a canvas, you need an image.
- The image can be an existing `<img>` element, or you can create an `Image()` object with JavaScript.
- Either way, you need to ensure that the image is fully loaded before you can draw it on the canvas.

# Drawing an existing image

```
  
<canvas id="e" width="177" height="113"></canvas>
```

```
<script>
```

```
    window.onload = function() {  
        var canvas = document.getElementById("e");  
        var context = canvas.getContext("2d");  
        var cat = document.getElementById("cat");  
        context.drawImage(cat, 0, 0);  
    };
```

```
</script>
```

# Using an Image() object

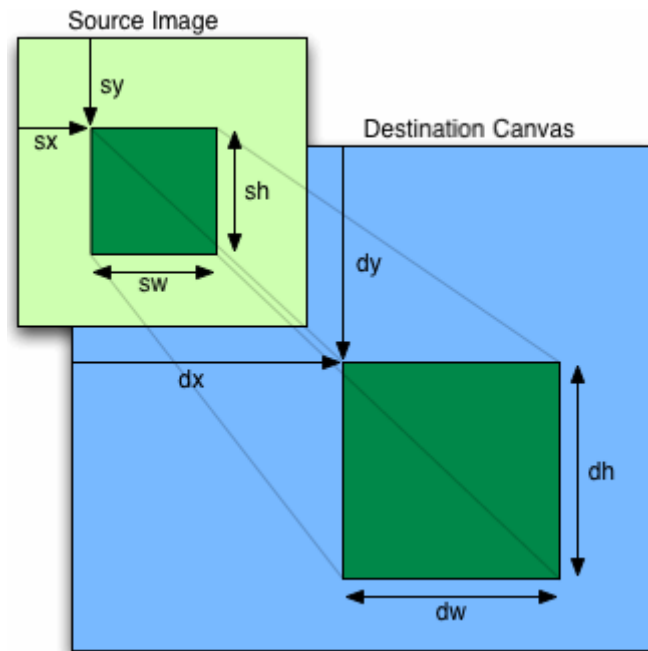
```
var bgImage = new Image();
bgImage.src = 'images/bg.png'; //path starting from html file, not js file

bgImage.onload = function ()      {
    context.drawImage(bgImage,0,0,297,297);
    //stroke a grid over an image
    context.beginPath();
    for (var i=1; i< 3; i++)      {
        context.moveTo(0,i*cellH);
        context.lineTo(w,i*cellH);
        context.moveTo(i*cellW,0);
        context.lineTo(i*cellW,h);
    }
    context.stroke();
}
```

# Drawing clipped images

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

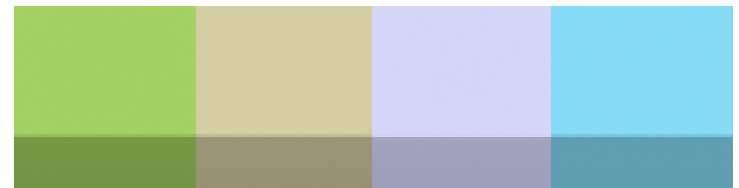
takes an image, clips it to the rectangle  $(sx, sy, sw, sh)$ , scales it to dimensions  $(dw, dh)$ , and draws it on the canvas at coordinates  $(dx, dy)$ .



# Example: tile set

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

```
tilesetImage = new Image();
tilesetImage.src = "images/tileset.png";
tilesetImage.onload = function () {
  for(var i=0;i<3;i++) {
    for(j=0; j<3; j++)
      context.drawImage(tilesetImage,
        i*63,0,
        63,64,
        i*cellW,j*cellH,
        cellW,cellH);
  }
}
```



252 x 64 pixels

Result: [TTT board 5.html](#)  
[canvas intro 5.js](#)



# Example: random map from a tile set

[TTT board 6.html](#)

[canvas intro 6.js](#)


# Adding canvas onclick event

//on click - draws rectangle of the corresponding player's color  
canvasElem.addEventListener("click",changeCell, false);

# Determining the tile that was clicked

```
function changeCell (e)  
{  
  var x;  
  var y;  
  
  if (e.pageX || e.pageY) {  
    x = e.pageX;  
    y = e.pageY;  
  }  
  else {  
    x = e.clientX + document.body.scrollLeft + document.documentElement.scrollLeft;  
    y = e.clientY + document.body.scrollTop + document.documentElement.scrollTop;  
  }  
  
  var clickedCellX = Math.floor ( (x - canvasElem.offsetLeft) / cellW );  
  var clickedCellY = Math.floor ( (y - canvasElem.offsetTop) / cellH );  
}
```

Event object that called the function

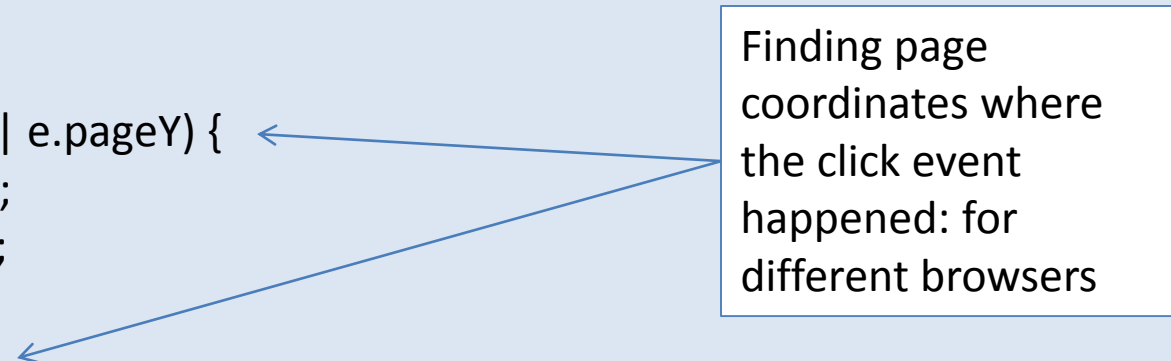


# Determining the tile that was clicked

```
function changeCell (e)
{
  var x;
  var y;

  if (e.pageX || e.pageY) {
    x = e.pageX;
    y = e.pageY;
  }
  else {
    x = e.clientX + document.body.scrollLeft + document.documentElement.scrollLeft;
    y = e.clientY + document.body.scrollTop + document.documentElement.scrollTop;
  }

  var clickedCellX = Math.floor ( (x - canvasElem.offsetLeft) / cellW );
  var clickedCellY = Math.floor ( (y - canvasElem.offsetTop) / cellH );
}
```



Finding page coordinates where the click event happened: for different browsers

# Determining the tile that was clicked

```
function changeCell (e)
{
  var x;
  var y;

  if (e.pageX || e.pageY) {
    x = e.pageX;
    y = e.pageY;
  }
  else {
    x = e.clientX + document.body.scrollLeft + document.documentElement.scrollLeft;
    y = e.clientY + document.body.scrollTop + document.documentElement.scrollTop;
  }

  var clickedCellX = Math.floor ( (x - canvasElem.offsetLeft) / cellW );
  var clickedCellY = Math.floor ( (y - canvasElem.offsetTop) / cellH );
}
```

← Converting to canvas coordinates

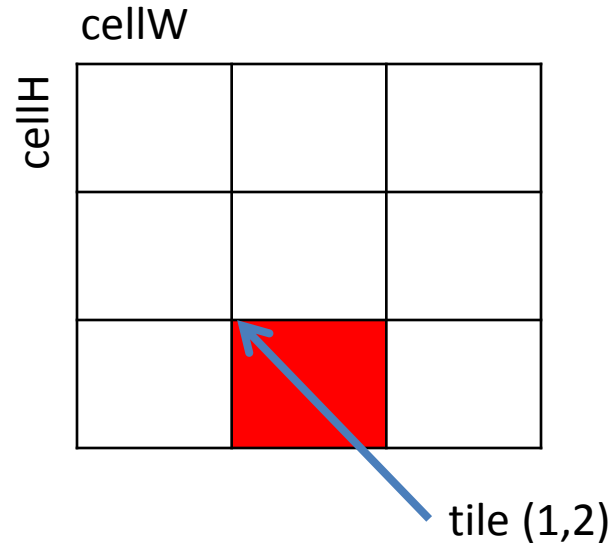
# Drawing a rectangle of a player's color – on click

```
var playerColors = ["rgb(255,0,0)","rgb(0,0,255)"];  
context.fillStyle=playerColors[playerIndex];  
context.fillRect(clickedCellX*cellW, clickedCellY*cellH, cellW, cellH);  
nextPlayer();
```

Result:

[TTT board 7.html](#)

[canvas intro 7.js](#)



# Drawing triangles

```
context.save();
```

Store context state to restore it later to its original state

```
context.fillStyle=playerColors[playerIndex];
```

```
context.beginPath();
```

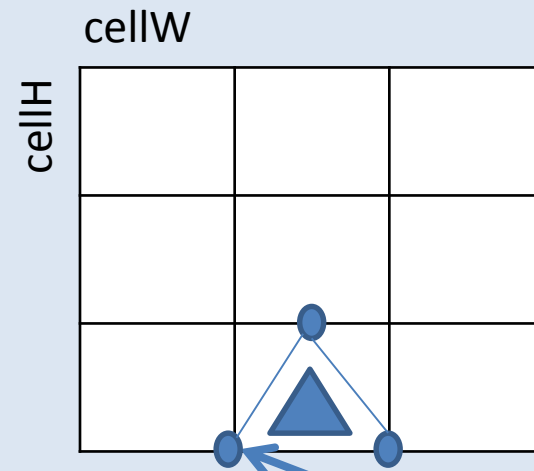
```
context.moveTo((clickedCellX)*cellW,(clickedCellY+1)*cellH);
```

```
context.lineTo(clickedCellX*cellW + cellW/2,clickedCellY*cellH );
```

```
context.lineTo((clickedCellX+1)*cellW,(clickedCellY+1)*cellH);
```

```
context.fill();
```

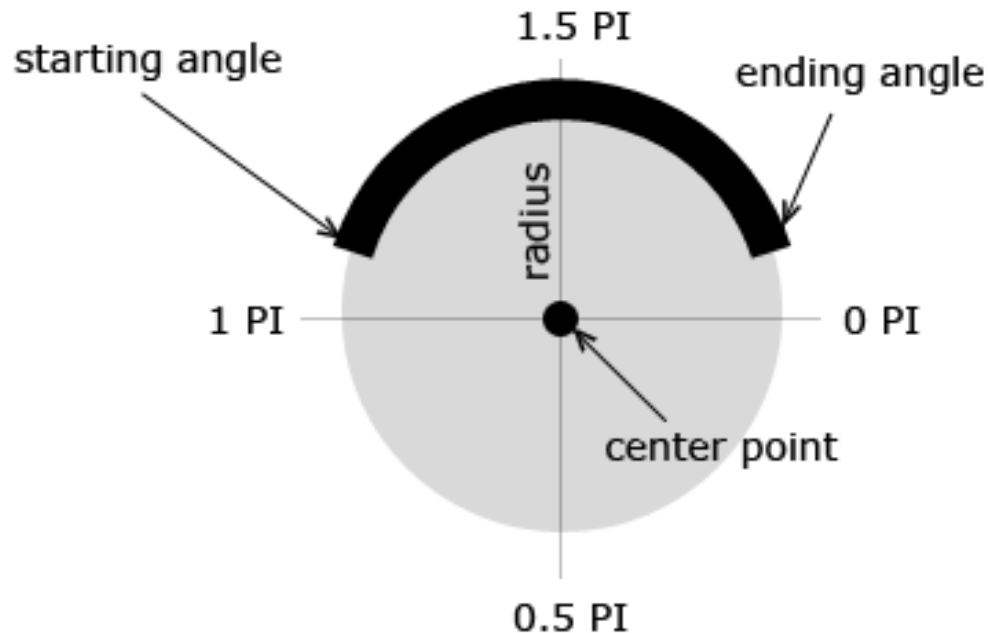
```
context.restore();
```



tile (1,2)

# Drawing an arc, circle, oval

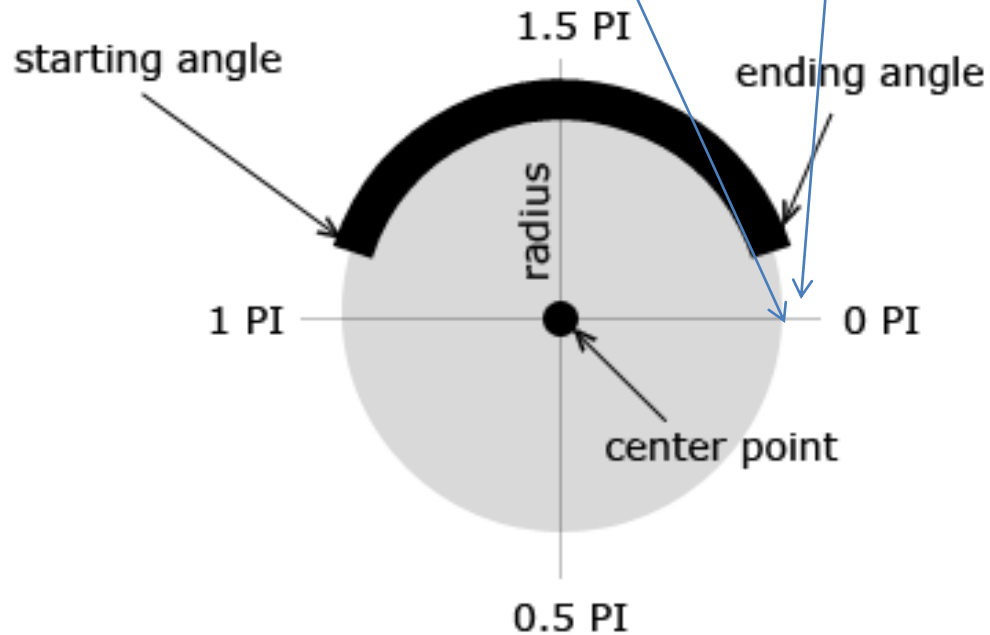
```
context.arc(x, y, radius, startAngle, endAngle, antiClockwise);
```





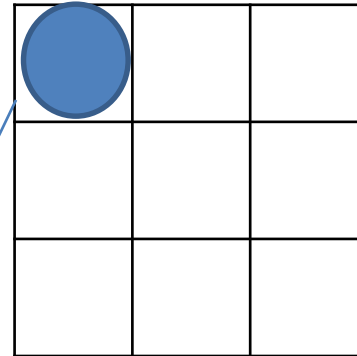
# Drawing a circle

```
context.arc(200, 200, 30, 0, 2 * Math.PI, false);
```

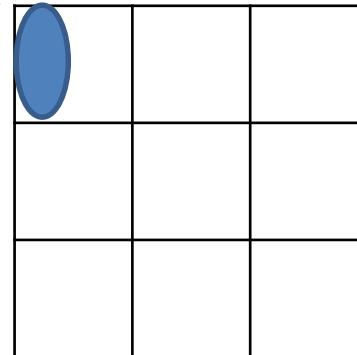


# Drawing an oval

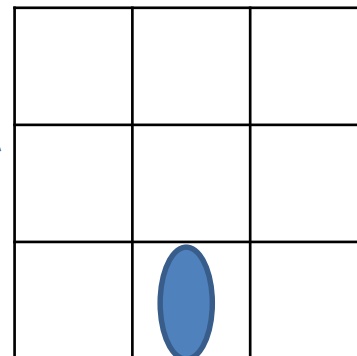
```
function drawEllipse(context,
cx, cy, radius, rx, ry, tx, ty){
...
    context.scale(rx, ry);
    context.translate(tx, ty/2);
    context.arc(cx, cy, radius,
        0, 2 * Math.PI, false);
...
}
```



Draw at  
(0,0)



Scale  
0.5 on  
X axis



Move to the  
corresponding  
cell

More on canvas  
[transformations](#)

# TTT with shapes

[TTT board 8.html](#)

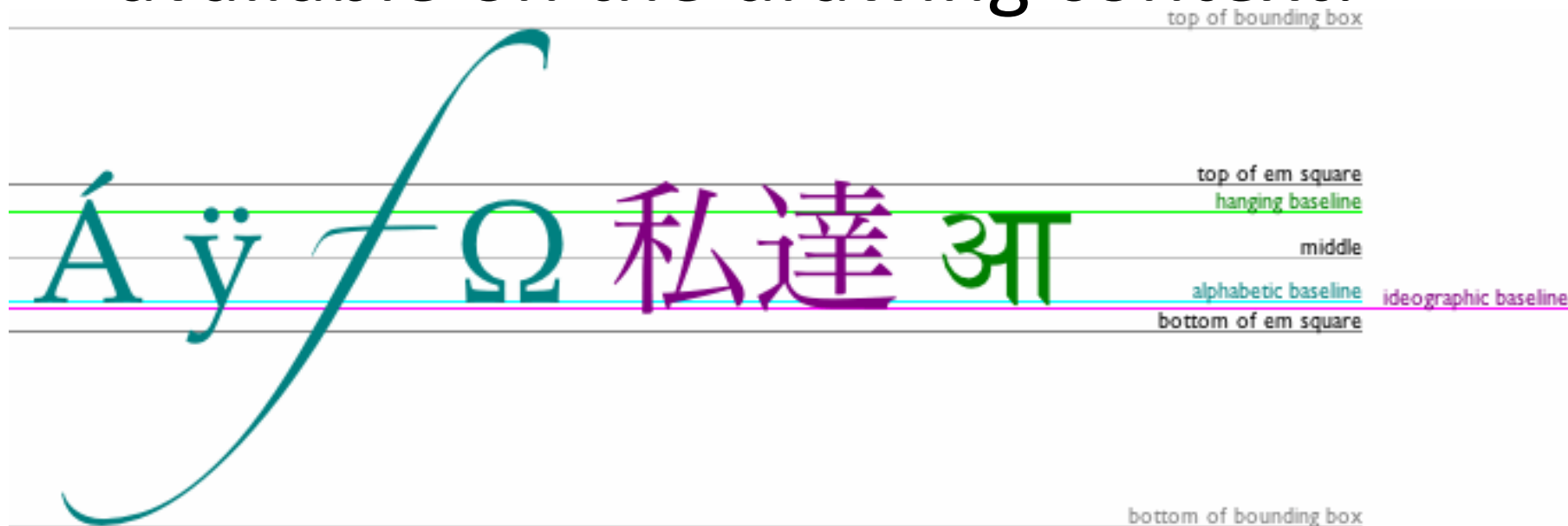
[canvas intro 8.js](#)

# Drawing text

- To draw text on canvas, you set font attributes and call **fillText**:

```
context.textBaseline = "top";  
context.fillText("( 0 , 0 )", 8, 5);
```

# The following font attributes are available on the drawing context:



- **font** can be anything you would put in a CSS font rule. That includes font style, font variant, font weight, font size, line height, and font family.
- **textAlign** controls text alignment. It is similar (but not identical) to a CSS text-align rule. Possible values are start, end, left, right, and center.
- **textBaseline** controls where the text is drawn relative to the starting point. Possible values are top, hanging, middle, alphabetic, ideographic, or bottom.
- HTML5 [spec](#)

# TTT with text

```
var playerLetters = ["X","O"];
context.font = "bold 2em sans-serif";
context.textBaseline = "middle";
context.textAlign = "center";
//fillText(str, x, y);
context.fillText(playerLetters[playerIndex],
    clickedCellX*cellW+cellW/2, clickedCellY*cellH+cellH/2);
```

[TTT board.html](#)

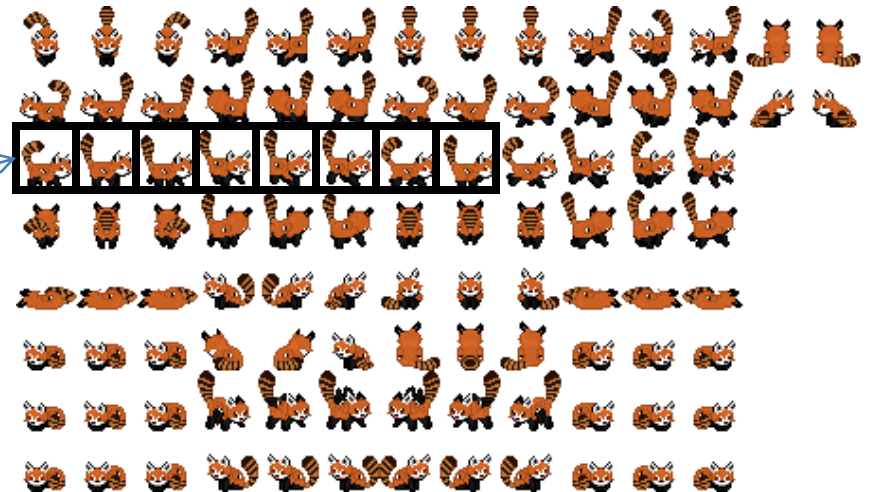
[canvas intro.js](#)

# Combining timer events with image drawing to produce an effect of animation

- **Animation** is the rapid display of a sequence of images to create an illusion of movement.
- We can draw clipped images at different times to produce an animation effect on canvas

# Spritesheet-based animation

- To create a cat moving across canvas we will use images sliced out at some cells of this sprite sheet:
  - To move left: 8 first images at row 2
  - To move right: 8 first images at row 1





# Reminder: JS timer loop

- To start a timer loop with 30 frames per second =  $1000/30$  milliseconds between timer ticks:

```
var fps = 30;  
var loopHandler=setInterval(draw, 1000/FPS);
```

- To stop this animation:

```
clearInterval(loopHandler)
```

# Drawing sprites at each timer tick

```
context.drawImage(spriteImage, currentSpriteX, currentSpriteY,  
    spriteCellW, spriteCellH, currentX, h/2,  
    spriteCellW*1.5, spriteCellH*1.5);  
  
//changes source image by slicing different pieces of a sprite  
sheet  
currentSpriteX += 1;  
  
//moves an image 3 pixels to the left  
currentX += 3;
```

Result:

[spriteanimation.html](#)

[canvas\\_animation.js](#)

# Only with canvas: procedural animation

- Randomly varying one coordinate in an array of points outlining the shape

Example: upset lonely iRock

[proceduralAnimation.html](#)

[irock\\_animation.js](#)

# Further reading: canvas tutorials

- [Mozilla developer network](#)
- [Safari developer library](#)