BASICS OF HTML CANVAS

By Raymond Pang

HTML5 and Canvas

- Classic HTML defines only tags for markup of different parts of a web page
  - No user-defined interactive drawing and rendering capability
- Canvas in HTML5
  - New tag to define a drawing area within a webpage
  - Supported by latest browsers
  - Draw by code in Javascript
- Following of this lecture will require knowledge of Javascript

HTML Canvas

- Canvas (畫布)
- We can create a HTML canvas on a webpage by giving width and height:

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

- At the beginning, it is an empty space
- You have to draw something on it

Drawing in HTML Canvas

- Use Javascript to draw on canvas
- First, we need to reference to the canvas's DOM object

```
var canvas = document.getElementById('tutorial');
```

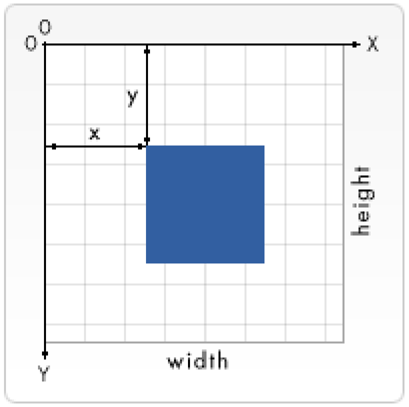
- Call getContext with '2d' as parameter to get the context of the canvas

```
var ctx = canvas.getContext('2d');
```

Drawing in HTML Canvas

- We can draw on the context by using methods like:

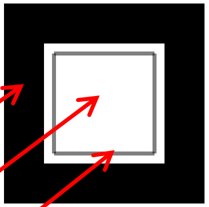
```
fillRect(x,y,width,height)
Draw a filled rectangle

strokeRect(x,y,width,height)
Draw a rectangle with only stroke

clearRect(x,y,width,height)
Clear a rectangular region
```



Drawing in HTML Canvas

- We draw 3 rectangles in different styles

```
if (canvas){
        var ctx = canvas.getContext('2d');

        ctx.fillRect(25,25,100,100);
        ctx.clearRect(45,45,60,60);
        ctx.strokeRect(50,50,50,50);
}
```

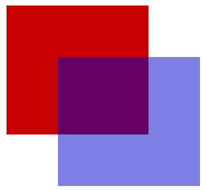


T1Example1.html

fillStyle

- We can change the color and layout by modifying the "fillStyle" property using CSS syntax

```
if (canvas.getContext) {
        var ctx = canvas.getContext("2d");

        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (10, 10, 55, 50);

        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 50);
 }
```



T1Example2.html

Drawing Arbitrary Shape

- To draw any shape you like instead of rectangles, we have to use **“Path”**
- Start drawing by calling “beginPath”

```
ctx.beginPath();
```

- Then move your “pen” to a position by “moveTo” method

```
ctx.moveTo(x,y);
```

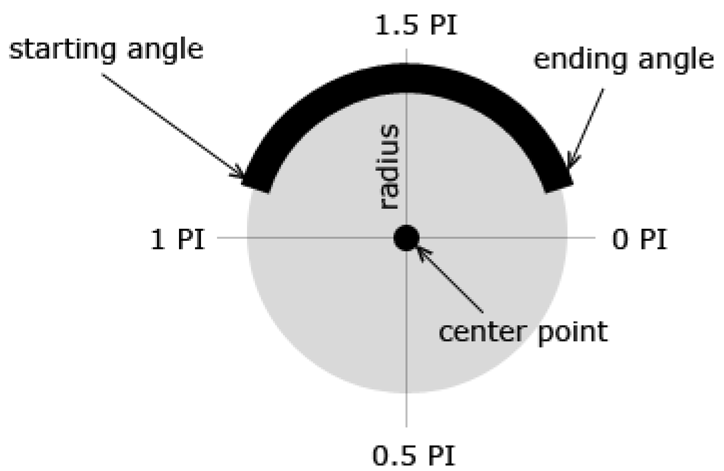
Drawing Arbitrary Shape

- Draw a line

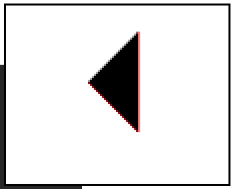
```
ctx.lineTo(x,y); (from current position to x,y )
```

- Draw an arc (弧)

```
ctx.arc(centerX, centerY, radius, startingAngle,
endingAngle, counterclockwise);
```



Example



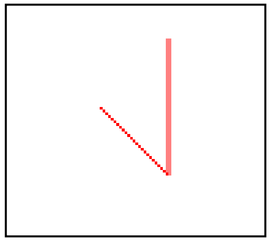
```
var ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.strokeStyle = "red";  // set the stroke to red
ctx.moveTo(75,50);
ctx.lineTo(100,75);
ctx.lineTo(100,25);
ctx.stroke();
ctx.fill();
```

T1Example3.html

Stroke or Filled Sharp

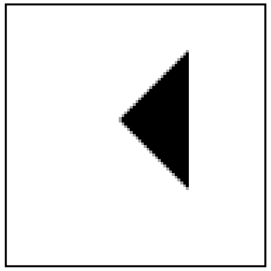
- We can choose to draw stroke only, a filled area or both
- Stroke only:

```
ctx.Stroke();
```



- Filled area:

```
ctx.fill();
```

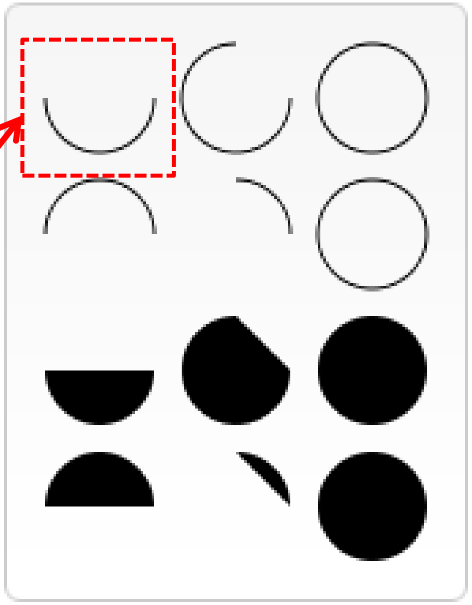


- We can find that if fill() is used, the shape will be closed automatically

Example using Arc

```
for (i=0;i<4;i++){
 for(j=0;j<3;j++){
   ctx.beginPath();
   var x             = 25+j*50;      // x coord of center
   var y             = 25+i*50;      // y coord of center
   var radius          = 20;
   var startAngle     = 0;
   var endAngle       = Math.PI+(Math.PI*j)/2;
   var anticlockwise  = i%2==0 ? false : true;
   ctx.arc(x,y,radius,startAngle,endAngle, anticlockwise);
   if (i>1){
     ctx.fill();
   } else {
     ctx.stroke();    }
 }
}
```

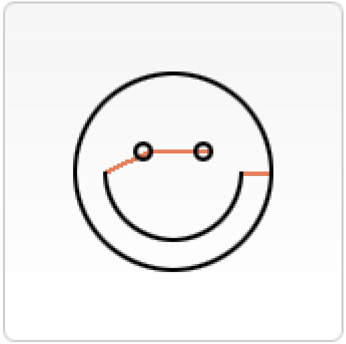
e.g.
endAngle = PI
anticlockwise= false



L1Example4.html

Example using Arc

```
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
        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();
}
```



T1Example5.html

Drawing Images

- We can draw image within the canvas
- But we need to load in the image as an Image object before we draw

```
Var image = new Image();
image.src = ‘myImage.png’;
```

- Then, set a callback function when the image is properly loaded

```
img.onload = finishedloading();
```

```
function finishedloading() { ….}
```

Drawing Images

- Inside the callback function, call “drawImage” and give the location (x,y) to draw the image on the canvas

```
ctx.drawImage(image, x, y)
```

- Or you can even scale it according to width and height given

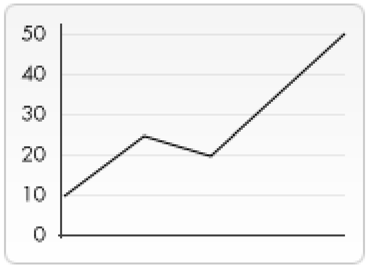
```
ctx.drawImage(image, x, y, width, height)
```

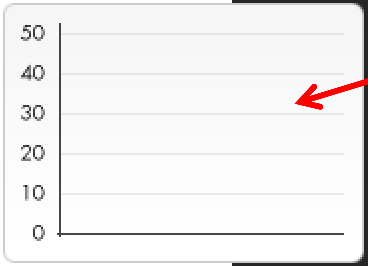
Example

```
if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        var img = new Image();
        img.src = 'backdrop.png';
        img.onload = function(){
            ctx.drawImage(img,0,0);
            ctx.beginPath();
            ctx.moveTo(30,96);
            ctx.lineTo(70,66);
            ctx.lineTo(103,76);
            ctx.lineTo(170,15);
            ctx.stroke();
        }
}
```

Draw the lines on top

T1Example6.html

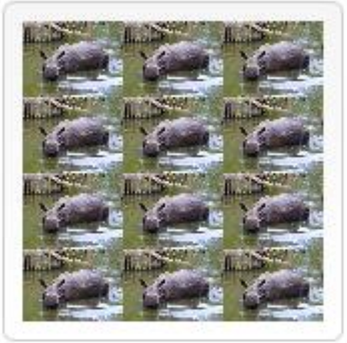




(backdrop.png)

Example

- Loop to display several images



```
if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        var img = new Image();
        img.src = 'rhino.jpg';
        img.onload = function(){
                for (i=0;i<4;i++){
                  for (j=0;j<3;j++){
                ctx.drawImage(img,j*50,i*38,50,38);
                  }
                }
        }
}
```

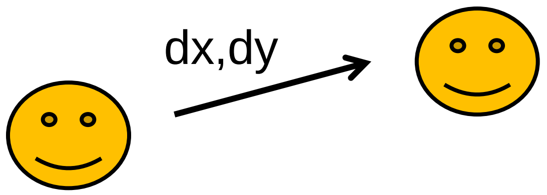
T1Example7.html

Transformation

We can do transformation to the drawings or images, e.g.

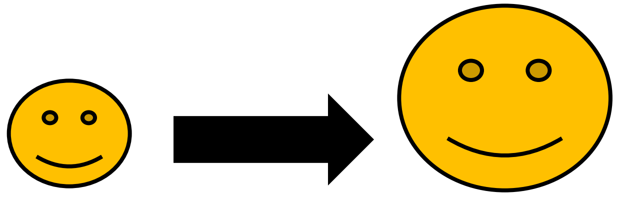
- Translate (move)

```
ctx.translate(dx, dy)
```



- Scale

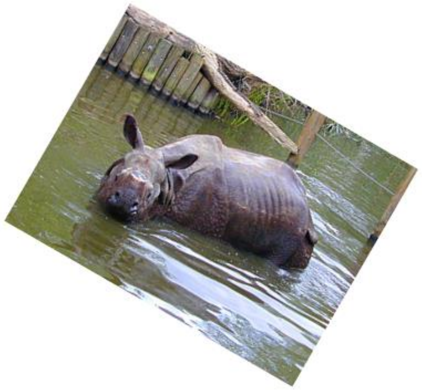
```
ctx.scale(x, y)
```



- Rotate

```
ctx.rotate(angle)
```

Example



```
if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        var img = new Image();
        img.src = 'rhino.jpg';
        img.onload = function(){
           ctx.translate(150,50);

           ctx.rotate(Math.PI/6);
           ctx.drawImage(img,0,0);
        }
}
```

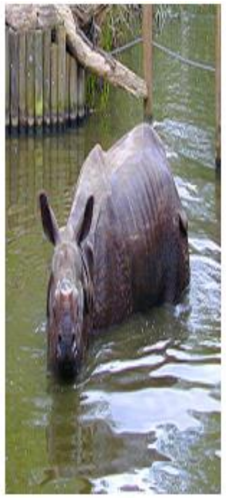
Move right 150 pixels
Move down 50 pixels

Rotate 30 degree

T1Example8.html

Example (scaling)

```
if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        var img = new Image();
        img.src = 'rhino.jpg';
        img.onload = function(){
           ctx.scale(0.5, 1.5);
           ctx.drawImage(img,0,0);
        }
}
```



T1Example9.html

Clearing Out the Canvas

- We can clear out objects drawn on a canvas by using the "clearRect" method
- To complete clear out all contents:

```
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

- Here canvas.width and canvas.height are the width and height of the canvas

Simple Animation

- Similar to the clock example in last 2 lectures
- We can refresh our drawings every **n** second or millisecond to make it an animation!
- To do constantly refreshing, we need a timer
- The most commonly used timer

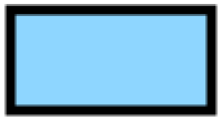
```
setInterval(callback_function,time);
setTimeout(callback_function,time);
```

Multiple time

One time

Example (T1Example10.html)

- We have a rectangle drawn in the canvas like :

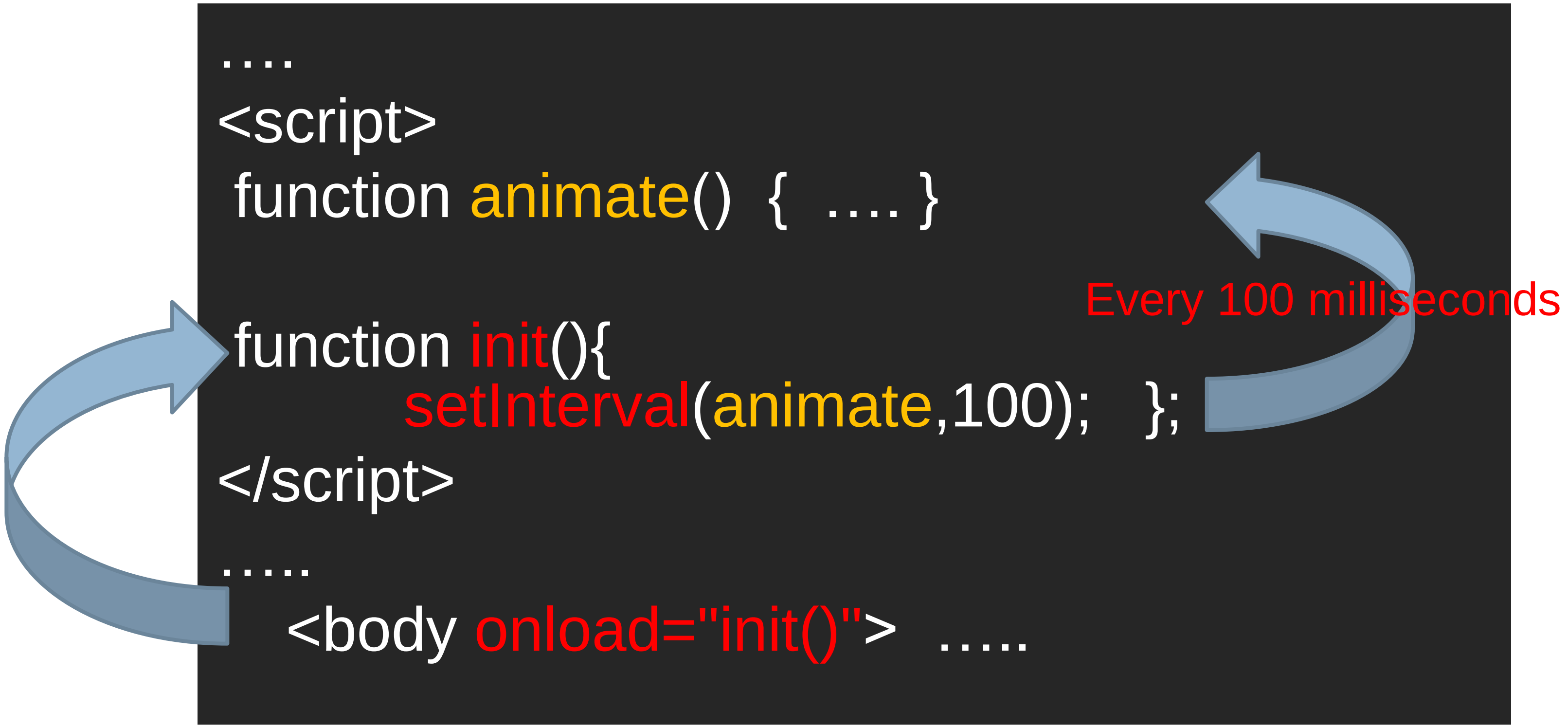


- Which is moving from the left to right
- The code of drawing the rectangle:

```
context.beginPath();
context.rect(myRectangle.x, myRectangle.y,
myRectangle.width, myRectangle.height);
context.fillStyle = "#8ED6FF"; context.fill();
context.lineWidth = myRectangle.borderWidth;
context.strokeStyle = "black";
context.stroke();
```

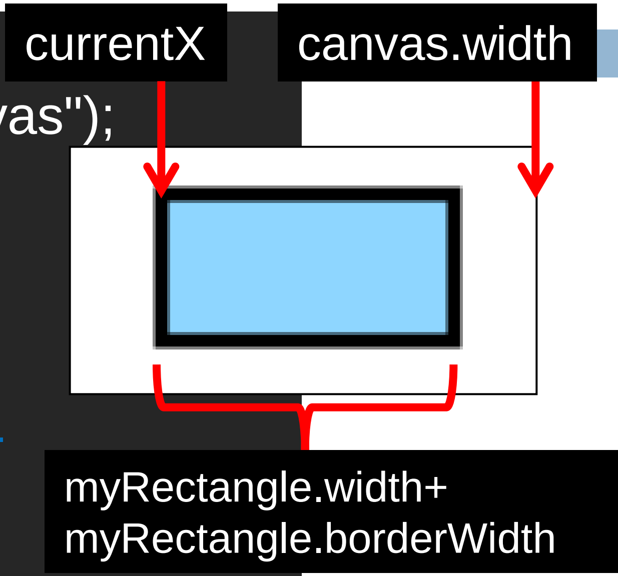
Example (T1Example10.html)

- In the initialization, we call “setInterval”, so for every 100 milliseconds, the function “animate” will be invoked


```
….
<script>
 function animate()  {  …. }

 function init(){
          setInterval(animate,100);   };
</script>
…..
   <body onload="init()">  …..
```

Example (T1Example10.html)

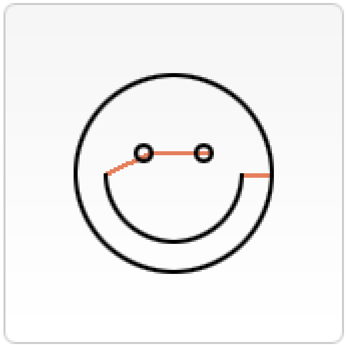
```
function animate(){
   var canvas = document.getElementById("myCanvas");
   var context = canvas.getContext("2d");
   var linearSpeed = 5; // pixels / frame
   var currentX = myRectangle.x;
    // stop when meet the border
    if (currentX < canvas.width - myRectangle.width - 
myRectangle.borderWidth) {
          // update the position
          var newX = currentX + linearSpeed;
          myRectangle.x = newX;
    }
    // clear
    context.clearRect(0, 0, canvas.width, canvas.height);

     // draw the rectangle
       ……..
}
```



More Animated Example

- Earlier, we have the following example which draws a smile face:

```
ctx.beginPath();
ctx.arc(75,75,50,0,Math.PI*2,true);
ctx.moveTo(110,75);
ctx.arc(75,75,35,0,Math.PI,false);
ctx.moveTo(65,65);
ctx.arc(60,65,5,0,Math.PI*2,true);
ctx.moveTo(95,65);
ctx.arc(90,65,5,0,Math.PI*2,true);
ctx.stroke();
```

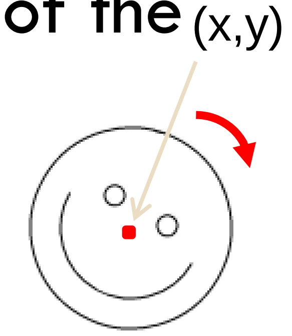


T1Example5.html

- In the coming example, we will make it rotates

More Animated Example

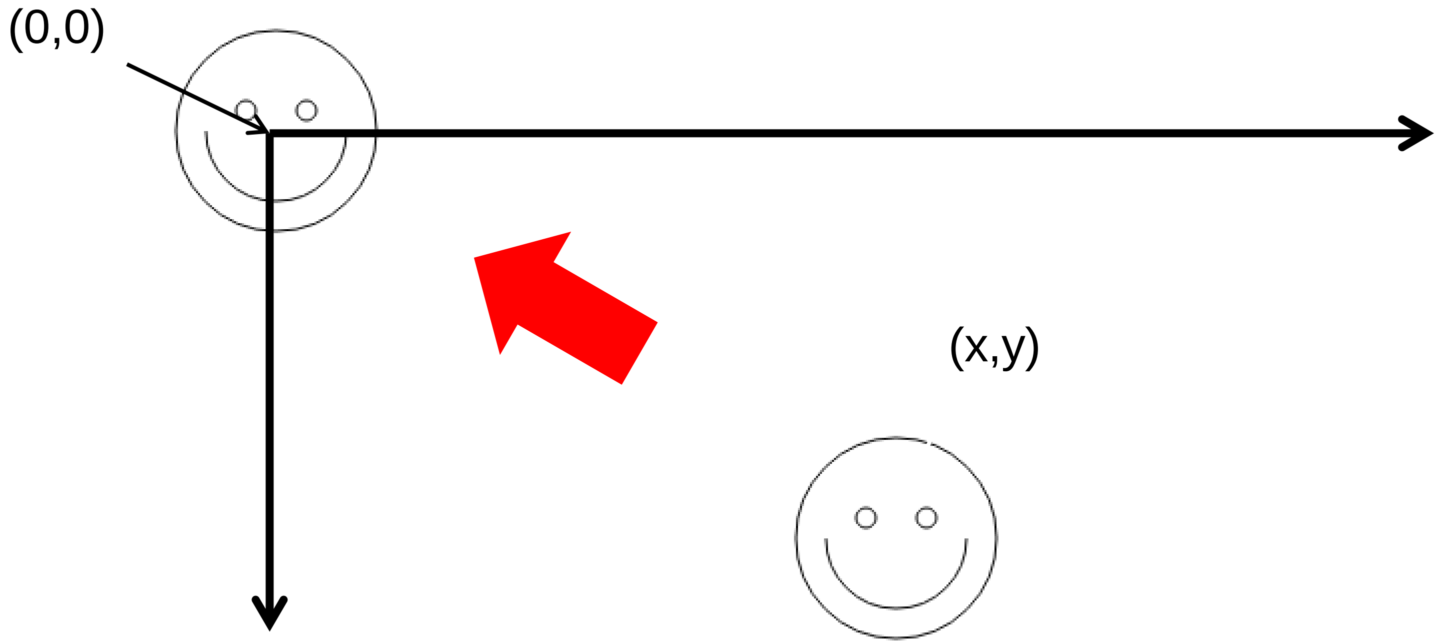
- We would like to make it rotate about center of the itself (x,y)
- By default, the "rotate" function is about the world coordinate (0,0)
- Steps to rotation around the center (x,y)
  - Translate to (0,0)
  - Rotate
  - Translate back to original position (x,y)



More Animated Example

- Translate to (0,0)

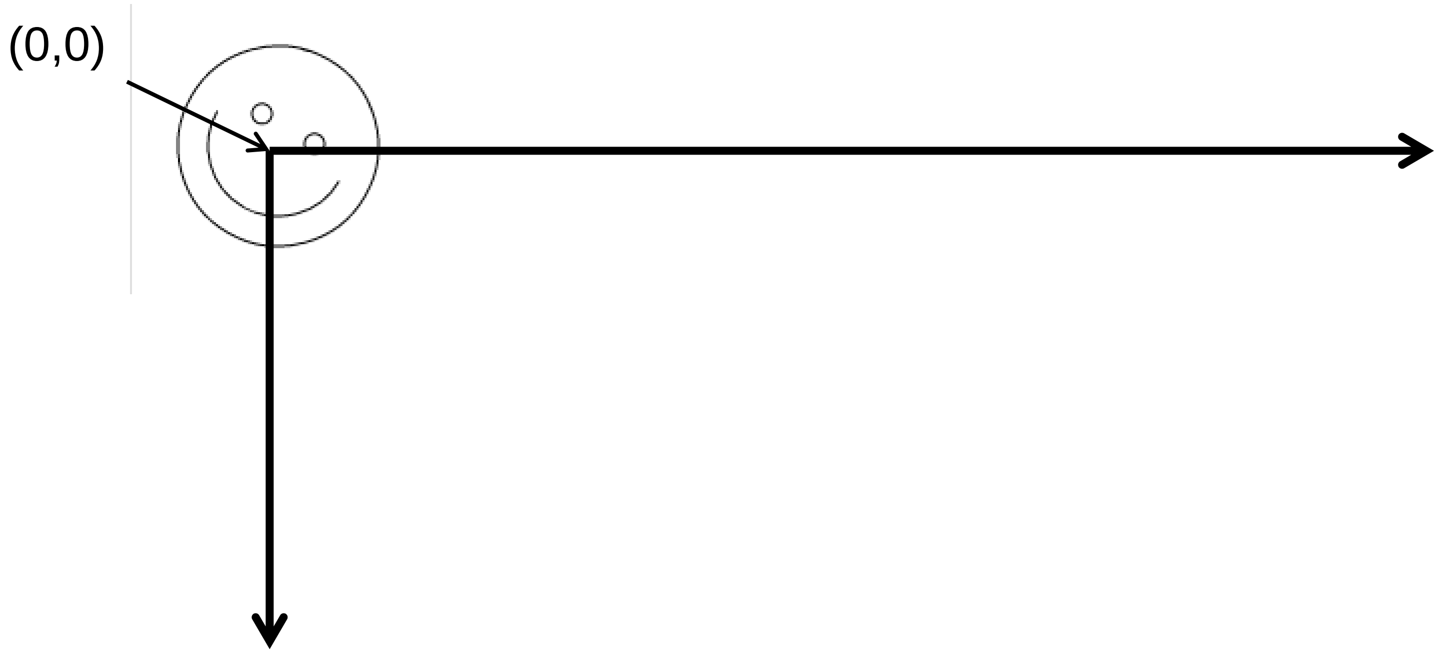
```
ctx.translate(-centerx,-centery);
```



More Animated Example

- Rotate (here we rotate 10 degree)

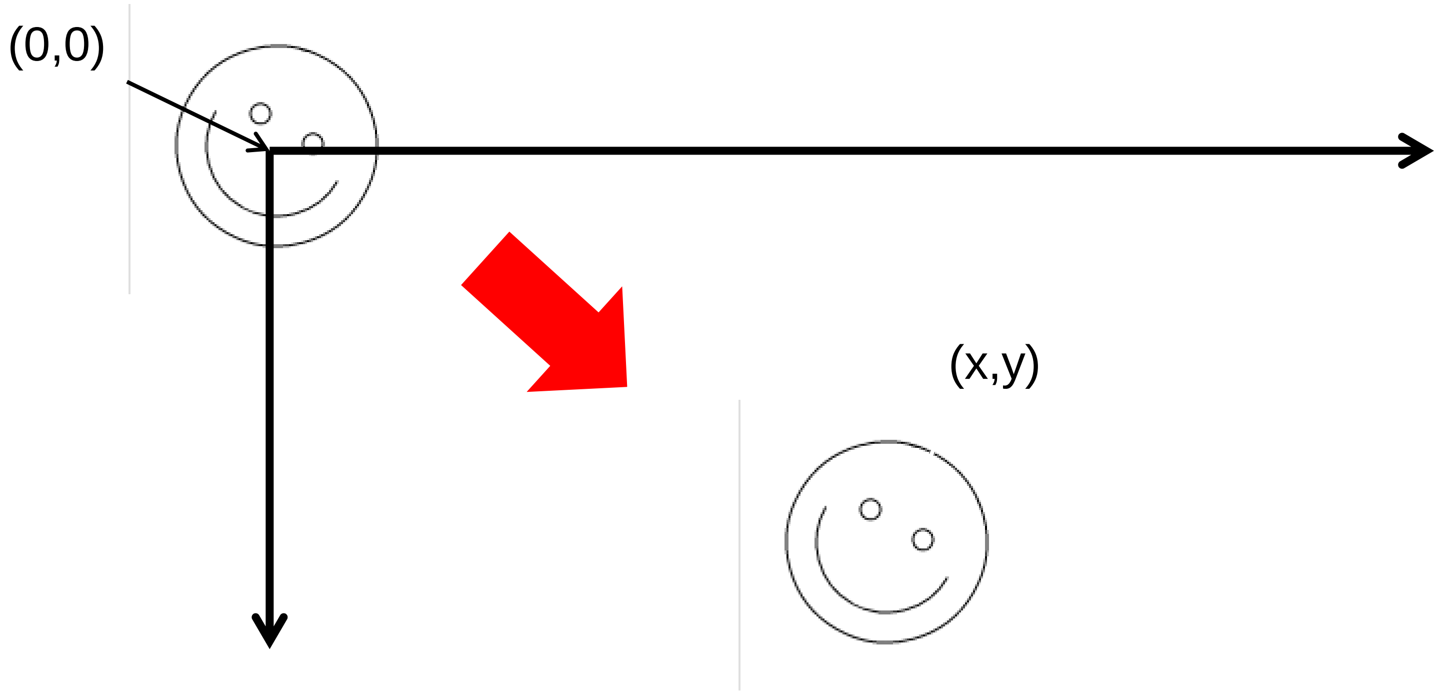
```
ctx.rotate(Math.PI*2/36);
```



More Animated Example

- Finally translate back to original center

```
ctx.translate(centerx,centery);
```



More Animated Example

- So the code will look like:

```
function animate()
{
  canvas = document.getElementById("canvas");
  if (canvas.getContext) {
     ctx = canvas.getContext('2d');
     // clear
     ctx.clearRect(0, 0, canvas.width, canvas.height);
     ctx.translate(centerx,centery);
     ctx.rotate(Math.PI*2/36);
     ctx.translate(-centerx,-centery);
     draw();
  }
}
```

T1Example11.html

More Animated Example

- In the draw function, we now draw the face relative to the center position

```
function draw() {
  ctx.beginPath();
  ctx.arc(centerx,centery,50,0,Math.PI*2,true); // outer circle
  ctx.moveTo(centerx+35,centery);
  ctx.arc(centerx,centery,35,0,Math.PI,false);  // mouth（clockwise）
  ctx.moveTo(centerx-10,centery-10);
  ctx.arc(centerx-15,centery-10,5,0,Math.PI*2,true);  // left eye
  ctx.moveTo(centerx+20,centery-10);
  ctx.arc(centerx+15,centery-10,5,0,Math.PI*2,true);  // right eye
  ctx.stroke();
}
```

T1Example11.html

Mouse Event in Javascript

- We have learnt events in previous lectures…
- Here are some events specific for mouse
- **mousedown**
  - Triggered when a mouse button is pressed down over an element
- **mouseup**
  - Triggered when a mouse button is released over an element
- **mouseover**
  - Triggered when the mouse comes over an element
- **mouseout**
  - Triggered when the mouse goes out of an element
- **mousemove**
  - Triggered on every mouse move over an element

Mouse Position

- One important information we want from the mouse is the position of the cursor within the window or element
- We can get them from the event object
  - Two attribute called: **clientX, clientY**
- First, we need to register the callback of mouse event, e.g. onmousemove

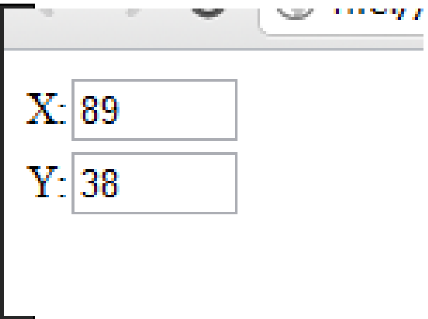
```
document.onmousemove = getMouseXY;
```

Name of callback function

Example

- The input parameter of the callback function is an event object
- The mouse positions are stored in the event object

```
function getMouseXY(event) {
  document.Show.MouseX.value = event.clientX;
  document.Show.MouseY.value = event.clientY;
}
```


X: 89
Y: 38

T1Example12.html

```
<form name="Show">
X:<input type="text" name="MouseX" value="0" size="4"> <br>
Y:<input type="text" name="MouseY" value="0" size="4"> <br>
</form>
```

Mouse Event with Canvas

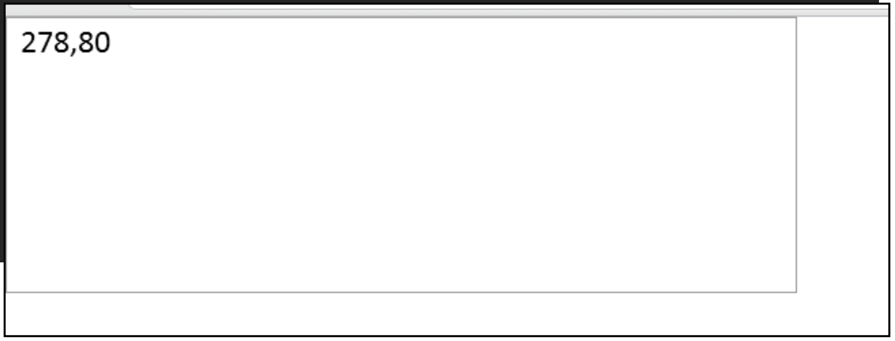
- In the last example, we add a mouse move event to the document object which represent the whole web page
- Actually, we can add mouse event to any DOM object, including the Canvas!
- E.g. we have a canvas with id='myCanvas'

```
canvas = document.getElementById('myCanvas');

canvas.onmousemove = getMouseXY;
```

Example

T1Example13.html

```
function init(){
    canvas = document.getElementById('myCanvas');
    ctx = canvas.getContext('2d');
    canvas.onmousemove = getMouseXY;
}
// Main function to retrieve mouse x-y pos.s within Canvas
function getMouseXY(event) {
  // display the coordinates in the canvas
  writeMessage(event.clientX + "," + event.clientY );
}
function writeMessage(message){
                ctx.clearRect(0, 0, canvas.width, canvas.height);
                ctx.font = '18pt Calibri';
                ctx.fillStyle = 'black';
                ctx.fillText(message, 10, 25);
}
```



```
278,80
```

Drag and Drop a Drawn Shape

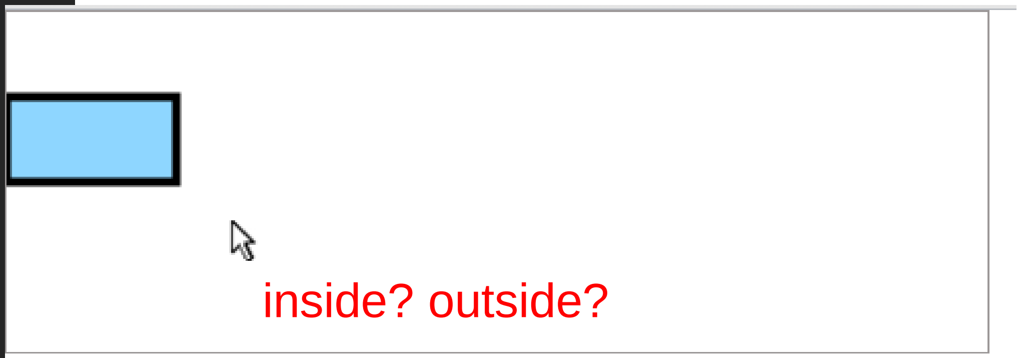
- We can detect position of mouse in the canvas, but how to select a shape?
- Or even Drag and Drop a shape?

- One simple method is to check if the mouse is inside the shape when the mouse button is down

Example of DnD a rectangle

- Using our previous lecture's example in which there is a rectangle drawn
- All its info. like position and dimensions are stated as :

```
var myRectangle = {
                x: 0,
                y: 50,
                width: 100,
                height: 50,
                borderWidth: 5
};
```



- Our first task is therefore to check if the mouse is inside this rectangle or not!

Example of DnD a rectangle

- As the checking is done at the time when the mouse button is down
- So, we need to listen to the onmousedown event first (just similar to the case of onmousemove)

Name of callback function

```
canvas.onmousedown = checkDrag;
```

Example of DnD a rectangle

- Inside the callback function:

T1Example14.html

```
function checkDrag(event) {
   // check if the rectangle is being dragged

   if (event.clientX > myRectangle.x &&
event.clientX < (myRectangle.x + myRectangle.width) &&
      event.clientY > myRectangle.y &&
event.clientY < (myRectangle.y + myRectangle.height) )
   {
            isDrag = true;
            dragoffsetx = myRectangle.x- event.clientX;
            dragoffsety = myRectangle.y- event.clientY;
   }
}
```

Check horizontally (i.e. in x)

Check vertically (i.e. in y)

If it is inside, record it as start dragging

Example of DnD a rectangle

- In the callback of onmousemove

T1Example14.html

```
function getMouseXY(event) {
  if (isDrag == true)
  {
    myRectangle.x = event.clientX + dragoffsetx;
    myRectangle.y = event.clientY + dragoffsety;

    // redraw
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawRect();
  }
}
```

Check if being dragged

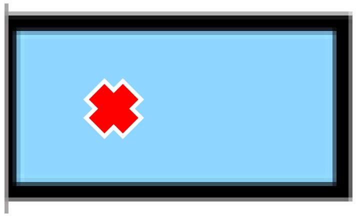
Move the box based on current mouse position

Redraw the box

Example of DnD a rectangle

- dragoffsetx and dragoffsety
- Use to record the position of the cursor within the box when dragging starts

```
dragoffsetx = myRectangle.x- event.clientX;
dragoffsety = myRectangle.y- event.clientY;
```



- So, we can keep this relative distance when we move together with the mouse curser

```
myRectangle.x = event.clientX + dragoffsetx;
myRectangle.y = event.clientY + dragoffsety;
```

Example of DnD a rectangle

- Finally, we need to reset the drag when mouse button is up (onmouseup event)

```
function finishDrag(event)
{
  isDrag = false;
}
function init()
{
    isDrag = false;
    ……
    canvas.onmouseup = finishDrag;
    ……
}
```

Stop dragging

T1Example14.html

Summary

- Using Canvas in HTML5 to draw simple 2D shapes and loading of images
- Ways to translation and rotation of shapes in 2D
- Simple animation in Canvas
- Mouse actions and interactivities with Javascript to the shapes in Canvas