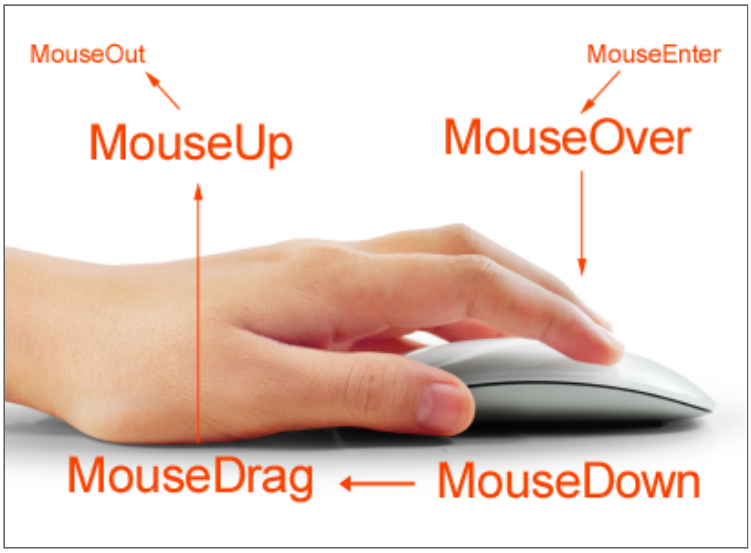
Mouse interaction, mouse events



INTRODUCTION

Detecting mouse events in a canvas is quite straightforward: you add an event listener to the canvas, and the browser invokes that listener when the event occurs.

The example below is about listening to `mouseup` and`mousedown` events (when a user presses or releases any mouse button):

```
canvas.addEventListener('mousedown',function (evt) {
 // do something with to the mousedown event
});
```

The event received by the listener function will be used for getting the button number or the coordinates of the mouse cursor. Before looking at different examples, let's look at the different event types we can listen to.

THE DIFFERENT MOUSE EVENTS

We saw in the last example how to detect the `mouseenter` and`mouseout` events.

There are other events related to the mouse:

- `mouseleave`: similar to `mouseout`, fired when the mouse leaves the surface of the element. The difference between`mouseleave` and `mouseout` is that `mouseleave` does not fire when the cursor moves over descendant elements, and `mouseout` is fired when the element moved is outside of the bounds of the original element or is a child of the original element.
- `mouseover`: the mouse cursor is moving over the element that listens to that event. A `mouseover` event occurs on an element when you are over it - coming from either its child OR parent element, but a `mouseenter` event only occurs when the mouse moves from the parent element to the child element.
- `mousedown`: fired when a mouse button is pressed.
- `mouseup`: fired when a mouse button is released.
- `mouseclick`: fired after a `mousedown` and a `mouseup` have occured.
- `mousemove`: fired while the mouse moves over the element. Each time the mouse moves, a new event is fired, unlike with `mouseover` or `mouseenter`, where only one event is fired.

THE TRICKY PART: ACCURATELY GETTING THE MOUSE POSITION RELATIVE TO THE CANVAS

When you listen to any of the above events, the event object (we call it a "DOM event"), passed to the listener function, has properties that correspond to the mouse coordinates: `clientX`and `clientY`.

However, these are what we call "window coordinates". Instead of being relative to the canvas itself, they are relative to the window (the page).

Most of the time you need to work with the mouse position relative to the canvas, not to the window, so you must convert the coordinates between the window and the canvas. This will take into account the position of the canvas, and the CSS properties that may affect the canvas position (margin, etc.).

Fortunately, there exists a method for getting the position and size of any element in the

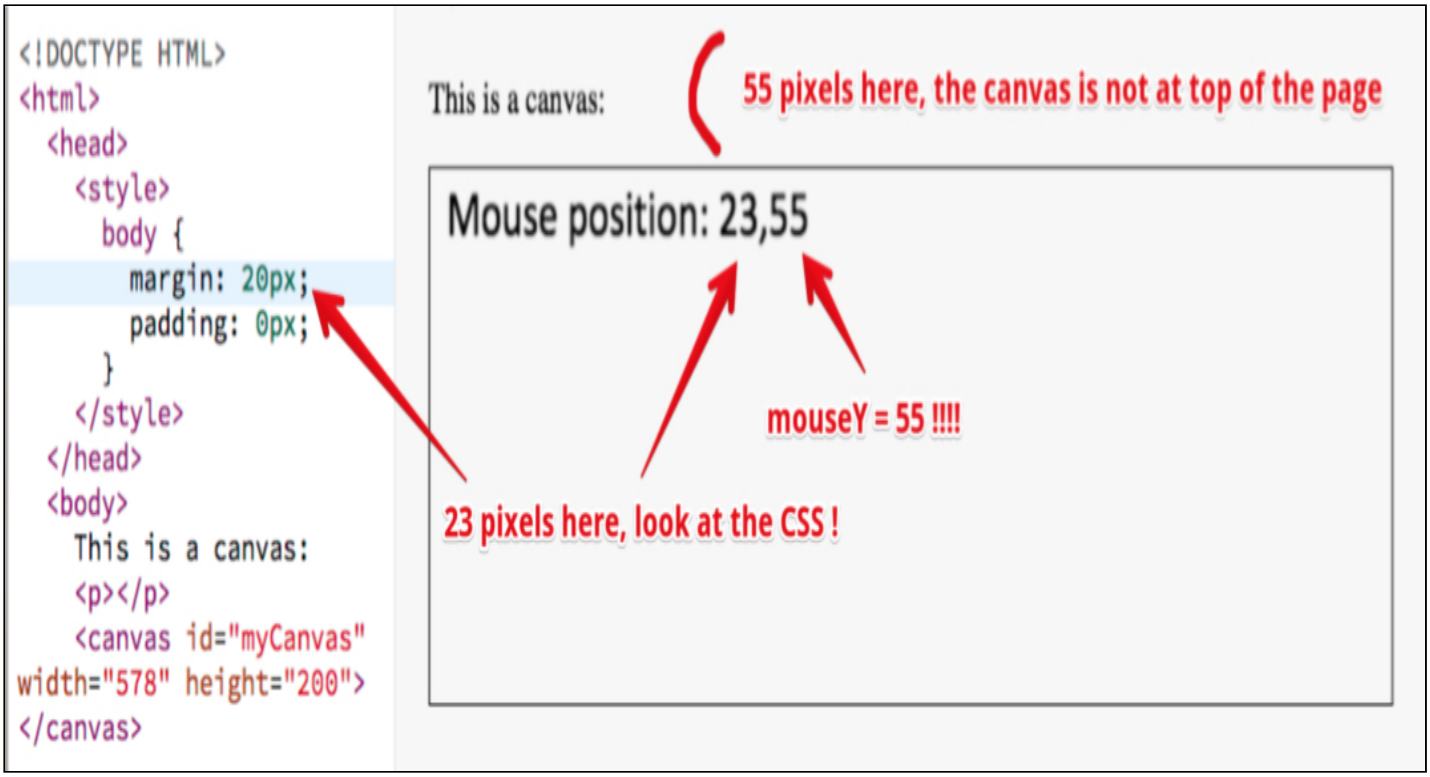
page: `getBoundingClientRect()`.

The example that shows the problem is at: http://jsbin.com/bekeso/2/edit

WRONG code:

```
...
canvas.addEventListener('mousemove',function (evt) {
    mousePos = getMousePos(canvas, evt);
    var message = 'Mouse position:
' +mousePos.x + ',' + mousePos.y;
    writeMessage(canvas, message);
 }, false);

...
function getMousePos(canvas, evt) {
   // WRONG!!!
   return {
      x: evt.clientX,
      y: evt.clientY
   };
}
```

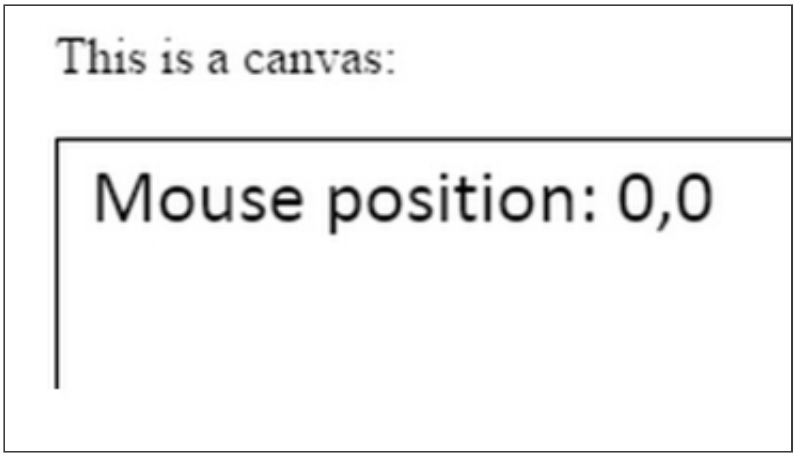
Here is the result, when the mouse is approximately at the top left corner of the canvas:



Good version of the code: http://jsbin.com/miduqu/3/edit

```
function getMousePos(canvas, evt) {
    // necessary to take into account CSS boundaries
    var rect =canvas.getBoundingClientRect();
    return {
        x: evt.clientX - rect.left,
        y: evt.clientY - rect.top
    };
}
```

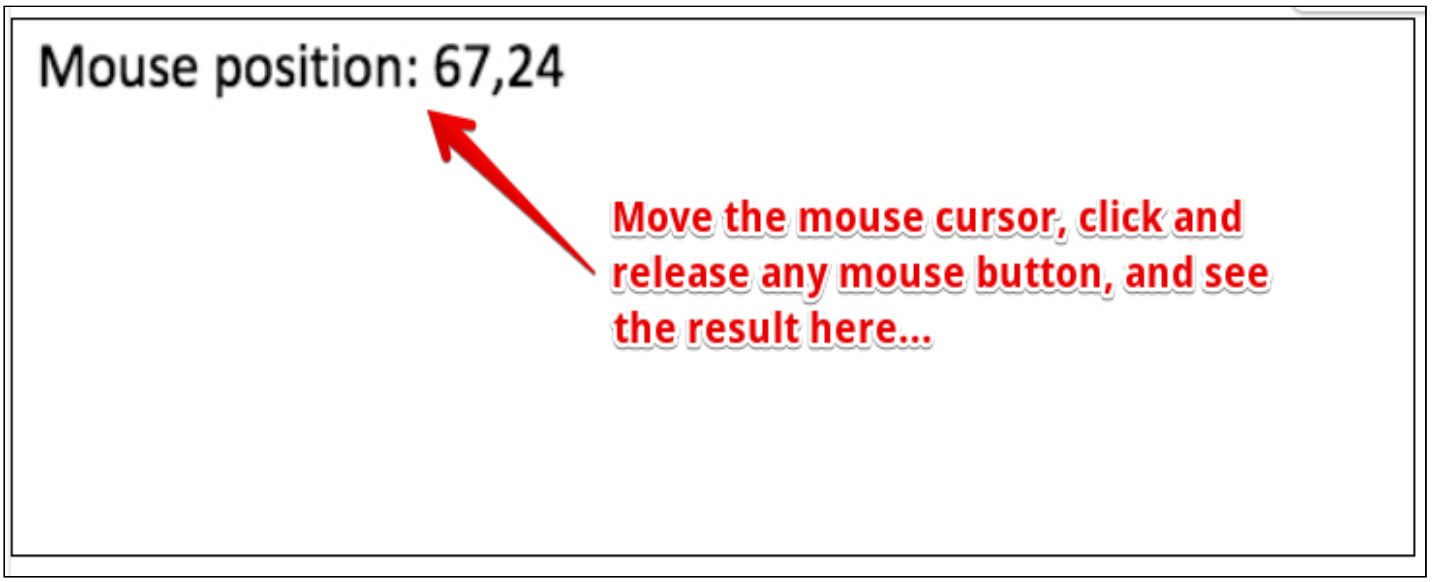
Result (the cursor is approximately at the top left corner):



GOOD EXAMPLE THAT SHOWS HOW TO DISPLAY THE MOUSE POSITION, AND THE MOUSE BUTTON THAT HAS BEEN PRESSED OR RELEASED

This example uses the previous function for computing the mouse position correctly. It listens to `mousemove`, `mousedown`and `mouseup` events, and shows how to get the mouse button number using the `evt.button` property.

Online example: http://jsbin.com/miduqu/2/edit



Extract from source code:

```
var canvas, ctx, mousePos, mouseButton;
```

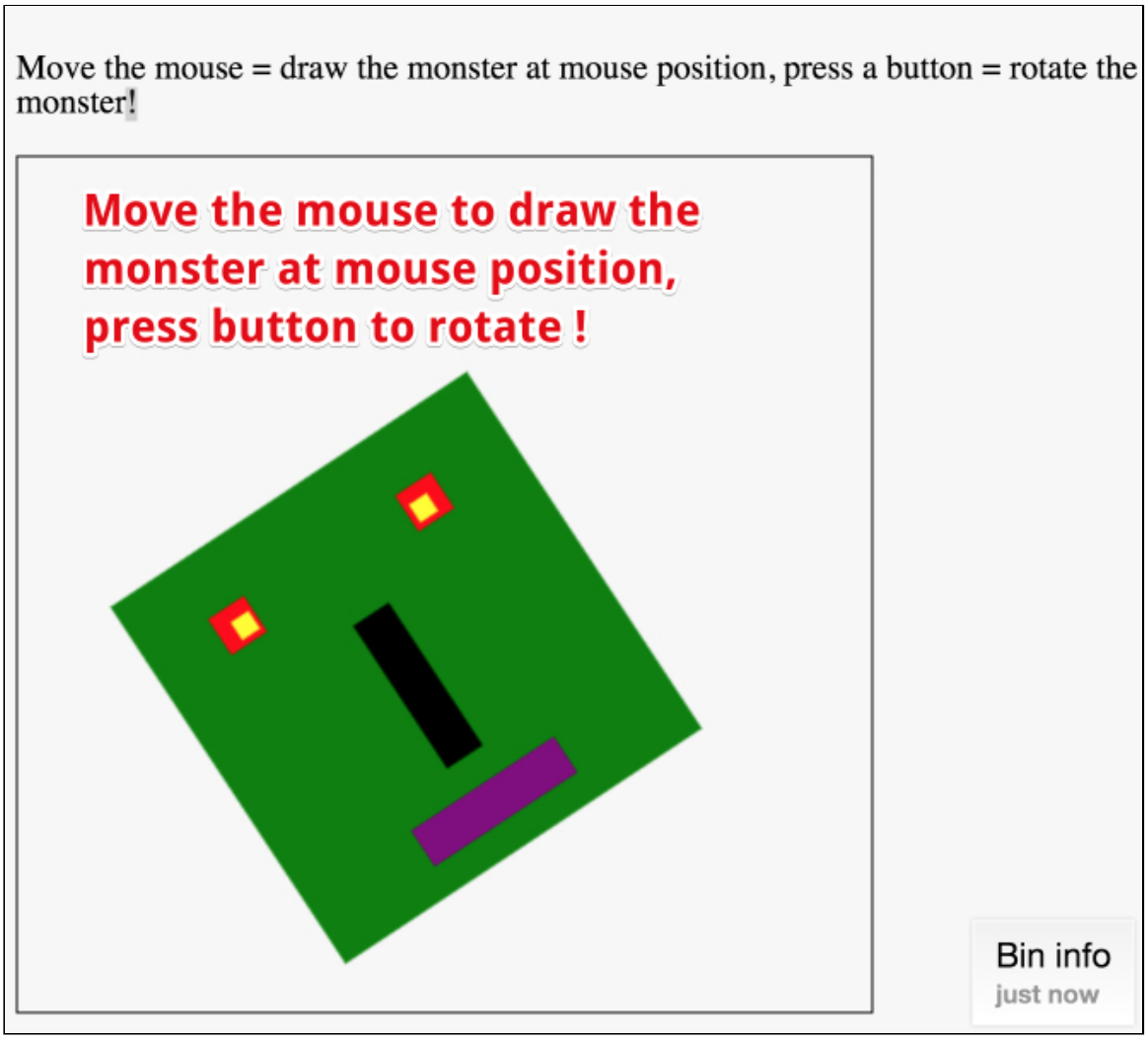
```
window.onload = function init() {
    canvas =document.getElementById('myCanvas');
    ctx = canvas.getContext('2d');

    canvas.addEventListener('mousemove',function (evt) {
        mousePos = getMousePos(canvas,evt);
        var message = 'Mouse position:
'+ mousePos.x + ',' + mousePos.y;
        writeMessage(canvas, message);
    }, false);

    canvas.addEventListener('mousedown',function (evt) {
        mouseButton = evt.button;
        var message = "Mouse button " +evt.button + " down at
position: " +mousePos.x + ',' + mousePos.y;
        writeMessage(canvas, message);
    }, false);

    canvas.addEventListener('mouseup',function (evt) {
        var message = "Mouse up at position:
" + mousePos.x + ',' +mousePos.y;
        writeMessage(canvas, message);
    }, false);
};

function writeMessage(canvas, message) {
   ctx.save();
   ctx.clearRect(0, 0, canvas.width,canvas.height);
   ctx.font = '18pt Calibri';
   ctx.fillStyle = 'black';
   ctx.fillText(message, 10, 25);
   ctx.restore();
}

function getMousePos(canvas, evt) {
   // necessary to take into account CSS boudaries
   var rect =canvas.getBoundingClientRect();
   return {
      x: evt.clientX - rect.left,
      y: evt.clientY - rect.top
```

```
    };
}
```

EXAMPLE: MOVE THE MONSTER WITH THE MOUSE, ROTATE IT WHEN A MOUSE BUTTON IS PRESSED.

This example shows an animation at 60 frames/s using`requestAnimationFrame`, were the monster is drawn at the mouse position, and if a mouse button is pressed, the monster starts rotating around its center. If we release the mouse button, the rotation stops.

Online example: http://jsbin.com/pedihokoyu/1/edit



Extract from source code:

```
var canvas, ctx;
var monsterX=100, monsterY=100,monsterAngle=0;
var incrementX = 0;
var incrementAngle =0;
var mousePos;
function init() {
    ...
    // 3bis - Add mouse listeners

 canvas.addEventListener('mousemove',handleMousemove, false);

 canvas.addEventListener('mousedown',handleMousedown, false);
    canvas.addEventListener('mouseup',handleMouseup, false);

    // 4 - Start the animation
    requestId =requestAnimationFrame(animationLoop);
}
function handleMousemove(evt) {
    // The mousePos will be taken into account in the
animationLoop
    mousePos = getMousePos(canvas, evt);
}
function handleMousedown(evt) {
   // the increment on the angle will be
   // taken into account in the animationLoop
   incrementAngle = 0.1;
}
function handleMouseup(evt) {
    incrementAngle = 0;  // stops the rotation
}
function getMousePos(canvas, evt) {
... // same as before
}
...
function animationLoop() {
   // 1 - Clear
   ctx.clearRect(0, 0, canvas.width,canvas.height);

   // 2 - Draw
```

```
    drawMonster(monsterX, monsterY,monsterAngle, 'green', 'yellow');

    // 3 - Move
    if(mousePos !== undefined) { // test necessary, maybe the
mouse is not yet on canvas
46.      monsterX = mousePos.x;
        monsterY = mousePos.y;
        monsterAngle += incrementAngle;
    }
    ...

    // call again mainloop after 16.6 ms (60 frames/s)
    requestId =requestAnimationFrame(animationLoop);
}
```

This example shows one very important good practice when doing animation and interaction: if you want to achieve a smooth animation, set the state variables 60 times/s inside the animation loop (lines 45-49), depending on increments you set in event listeners (lines 23-31).

EXAMPLE: DRAW IN A CANVAS AS IF YOU WERE USING A PENCIL

Online example: http://jsbin.com/bijusa/3/edit

Source code:

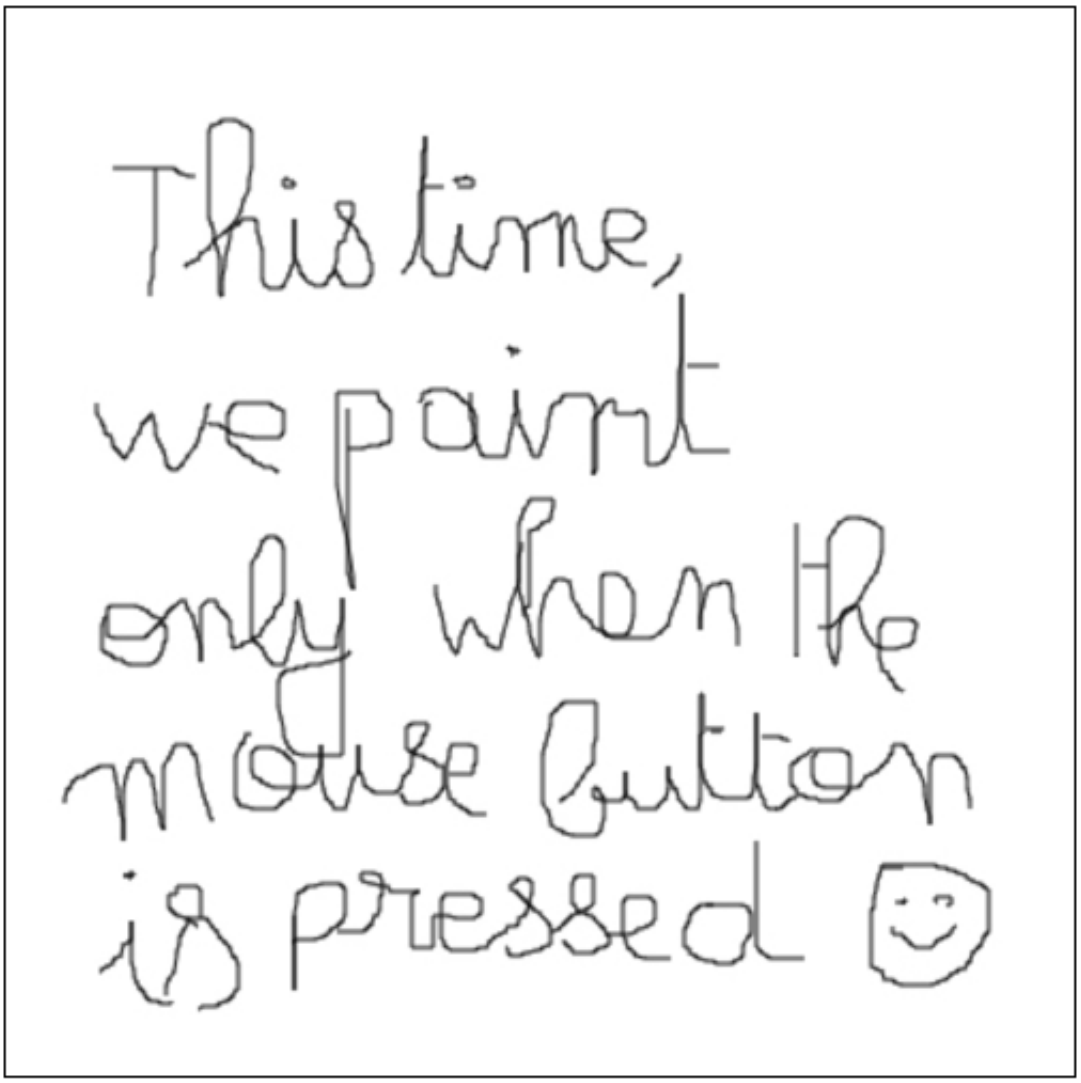
```
...
<script>
var canvas, ctx, previousMousePos;
...
function drawLineImmediate(x1, y1, x2,y2) {
    // a line is a path with a single draw order
    // we need to do this in this example otherwise
    // at each mouse event we would draw the whole path
    // from the beginning. Remember that lines
    // normally are only usable in path mode
    ctx.beginPath();
    ctx.moveTo(x1, y1);
    ctx.lineTo(x2, y2);
    ctx.stroke();
}

function handleMouseMove(evt) {
```

```
        var mousePos = getMousePos(canvas,evt);

        // Let's draw some lines that follow the mouse pos
        if (!started) {
            previousMousePos = mousePos; // get the current
    mouse position
23.         started = true;
        } else {
            // We need to have two consecutive mouse positions
    before drawing a line

    drawLineImmediate(previousMousePos.x,previousMousePos.y,
                                mousePos.x,          mousePos.y);
            previousMousePos = mousePos;
        }
    }
    window.onload = function () {
      ...
      started = false;


    canvas.addEventListener('mousemove',handleMouseMove, false);
    };
    </script>
```

We had to define a variable `started=false;` as we cannot draw any line before the mouse moved (we need at least two consecutive positions). This is done in the test at line 21.

SAME EXAMPLE BUT WE DRAW ONLY WHEN A MOUSE BUTTON IS PRESSED

Online example: http://jsbin.com/lavexi/3/edit



We just added `mouseup` and `mousedown` listeners, extract from the source code:

```
    function handleMouseMove(evt) {
        var mousePos = getMousePos(canvas,evt);

        // Let's draw some lines that follow the mouse pos
        if (painting) {

    drawLineImmediate(previousMousePos.x,previousMousePos.y,
                                mousePos.x,          mousePos.y);
            previousMousePos = mousePos;
9.      }
    }
    function clicked(evt) {
```

```
    previousMousePos =getMousePos(canvas, evt);
    painting = true;
}

function released(evt) {
    painting = false;
}

window.onload = function () {
    canvas =document.getElementById('myCanvas');
    ctx = canvas.getContext('2d');
    painting = false;


canvas.addEventListener('mousemove',handleMouseMove, false);
    canvas.addEventListener('mousedown',clicked);
    canvas.addEventListener('mouseup',released);
};
```

KNOWLEDGE CHECK 4.3.3 (NOT GRADED)

What is the **correct** way to get the mouse coordinates in a mousemove listener attached to a canvas, in the canvas coordinate system?

- There are multiple ways to get the mouse cursor coordinates: using the `clientX` and `clientY` properties of the event passed to the listener, using the `event.pageX` and `event.pageY` properties works too...

- Getting the mouse coordinate in the canvas coordinate system is not straightforward: we must take into account the position of the canvas into the page, the different CSS margins, etc.

- [ ] No problem: use the `event.mouseX` and `event.mouseY` properties.
