The Mouse and the Canvas

WE'LL COVER THE FOLLOWING

- Meet the Events
- Listening to and Handling Mouse Events
- The Global Mouse Position
- The Mouse Position Inside the Browser
- Detecting Which Button was Clicked

Often times, the canvas seems like its own mysterious beast. It is not until you run into areas like events that aren't specific to the `canvas` where you realize that all of this is just a small part of this larger JavaScript enclosure. For the next few sections, you are going to be learning general JavaScript 101 stuff about events, so feel free to skip or gloss over the material if you are intimately familiar with dealing with mouse events.

Meet the Events

Our `canvas` element works with all the mouse events that your browser exposes:

- `click`
- `dblclick`
- `mouseover`
- `mouseout`
- `mouseenter`
- `mouseleave`
- `mousedown`
- `mouseup`

- `mousemove`
- `contextmenu`
- `mousewheel and DOMMouseScroll`

What these events do should be somewhat easy to figure out given their name, so I won't bore you with details about that. Let's just move on to the next section where we learn how to listen to them.

Listening to and Handling Mouse Events

To listen for these events on your `canvas` element, use the `addEventListener` method and specify the event you are listening for and the event handler to call when that event is overheard:

```
var canvas = document.querySelector("#myCanvas");
var context = canvas.getContext("2d");

canvas.addEventListener("mousemove", doSomething, false);
```

That is pretty straightforward. In the highlighted line, we are listening for the **mousemove** event. When that event gets overheard, the `doSomething` function (aka the event handler) gets called. The last argument specifies whether we want our event to be captured in the bubbling phase or not. We won't focus on this argument. Instead, let's take another look at the event handler...

The event handler is responsible for reacting to an event once it is overheard. Basically, it's pretty important...and totally a big deal in the scene that is your code. Under the covers, an event handler is nothing more than a function with an argument for getting at what is known as the **event arguments**:

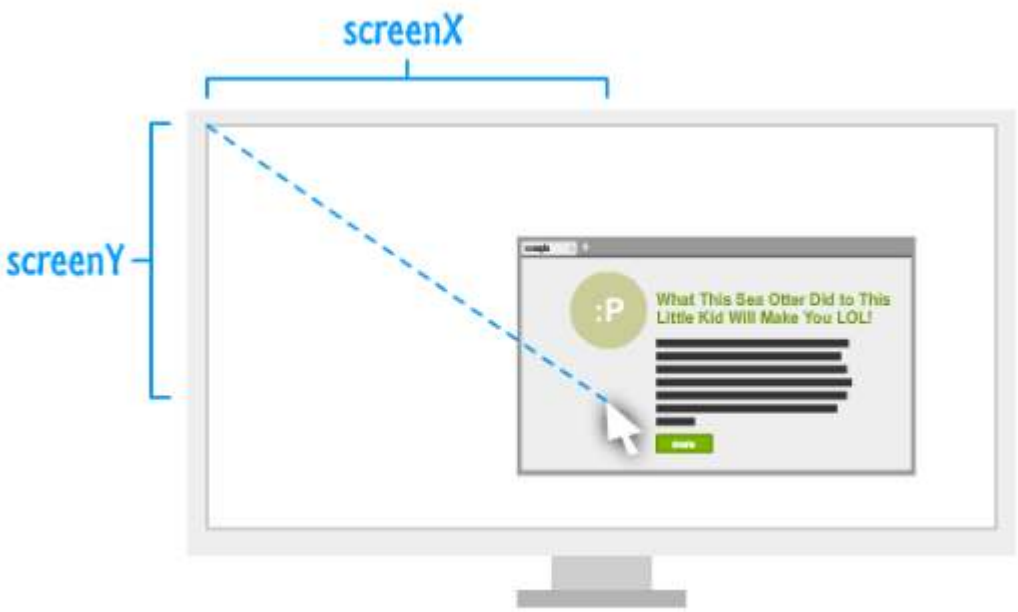
```
function doSomething(e) {
  // do something interesting

}
```

That detail is important. This event argument is the extremely awesome `MouseEvent` object that gives you access to a bunch of useful properties about the event that you just fired - properties that contains information such as the mouse's position, the button that was pressed, and more. We are going to devote the next few sections looking deeper into some of these useful properties.

The Global Mouse Position

The first `MouseEvent` properties we will look at are the `screenX` and `screenY` properties that return the distance your mouse cursor is from the top-left location of your primary monitor:



Here is a very simple example of the `screenX` and `screenY` properties at work:

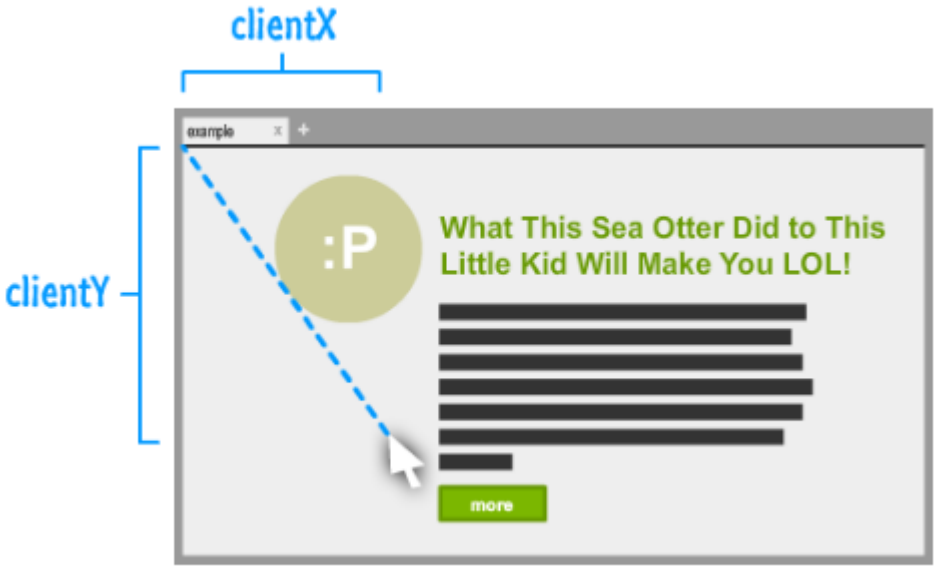
```
canvas.addEventListener("mousemove", mouseMoving, false);

function mouseMoving(e) {
    console.log(e.screenX + " " + e.screenY);
}
```

It doesn't matter what other margin/padding/offset/layout craziness you may have going on in your page. The values returned are always going to be the distance between where your mouse is now and where the top-left corner of your primary monitor is.

The Mouse Position Inside the Browser

This is probably the part of the `MouseEvent` object that you will run into over and over again. The `clientX` and `clientY` properties return the x and y position of the mouse relative to your browser's (technically, the browser viewport's) top-left corner:



The code for this is nothing exciting:

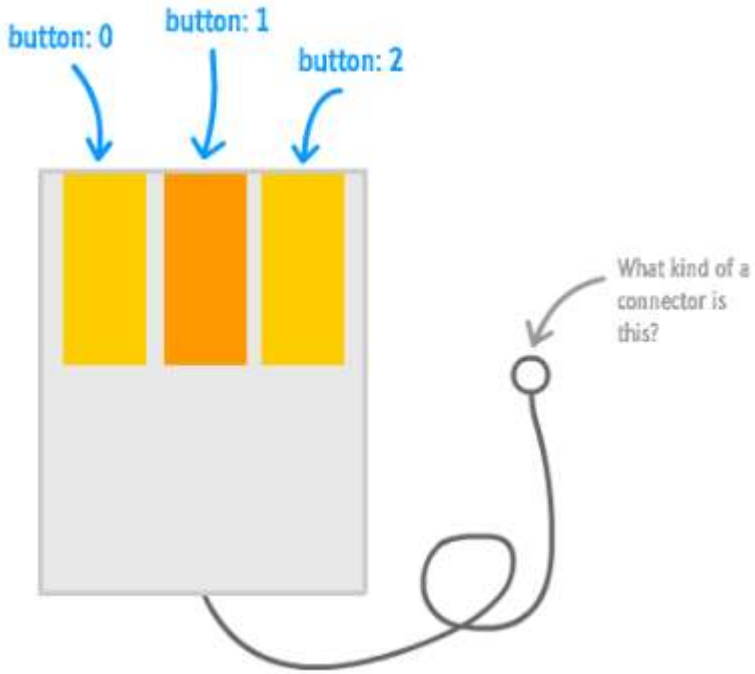
```
var canvas = document.querySelector("#myCanvas");
canvas.addEventListener("mousemove", mouseMoving, false);

function mouseMoving(e) {
    console.log(e.clientX + " " + e.clientY);
}
```

We just call the `clientX` and `clientY` properties via the `MouseEvent` argument to see some sweet pixel values returned as a result. What these values don't advertise is that you often need to do a little extra bit of additional calculation to get the exact mouse position inside our `canvas`. That calculation isn't hard, and we will look at that in a few sections!

Detecting Which Button was Clicked

Your mice often have multiple buttons or ways to simulate multiple buttons. The most common button configuration involves a left button, a right button, and a middle button (often a click on your mouse wheel). To figure out which mouse button was pressed, you have the `button` property. This property returns a **0** if the left mouse button was pressed, a **1** if the middle button was pressed, and a **2** if the right mouse button was pressed:



The code for using the `button` property to check for which button was pressed looks exactly as you would expect:

```
canvas.addEventListener("mousedown", buttonPress, false);

function buttonPress(e) {
    if (e.button == 0) {
        console.log("Left mouse button pressed!");
    } else if (e.button == 1) {
        console.log("Middle mouse button pressed!");
    } else if (e.button == 2) {
        console.log("Right mouse button pressed!");
    } else {
        console.log("Things be crazy up in here!!!");
    }
}
```

In addition to the `button` property, you also have the `buttons` and `which` properties that sorta do similar things to help you figure out which button was pressed. I'm not going to talk too much about those two properties, but just know that they exist.