EECS1012

Net-centric Introduction to Computing

Lecture

JavaScript Events

Acknowledgements
Contents are adapted from web lectures for "Web Programming Step by Step", by M. Stepp, J. Miller, and V. Kirst.
Slides have been ported to PPT by Dr. Xenia Mountrouidou.
These slides have been edited for EECS1012, York University.
The contents of these slides may be modified and redistributed, please give appropriate credit.
(Creative Commons) Michael S. Brown, 2017.

Events and the Prototype Library

Prototype Library

- Recall last lecture we introduced the Prototype library
- For this lecture, we will often be processing JS events using this library
- This means that we will need to link in the Prototype library in our HTML files

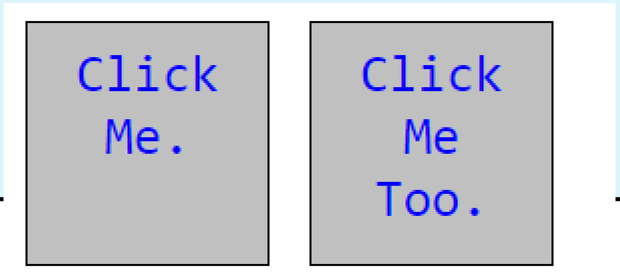
```
<script src="https://ajax.googleapis.com/ajax/libs/prototype/1.7.0.0/prototype.js"
type="text/javascript"></script>
```

NOTE: You can copy the prototype library (it is a single JS file) to your local directory and link it in too.

"This" keyword

Consider a simple example (HTML)

```
<html>
<head>
<script src="example1.js" type="text/javascript"> </script>
</head>
<body>
 <div id="box1" class="box">
  Click Me.
  </div>
  <div id="box2" class="box">
  Click Me Too.
  </div>
</body>
</html>
```

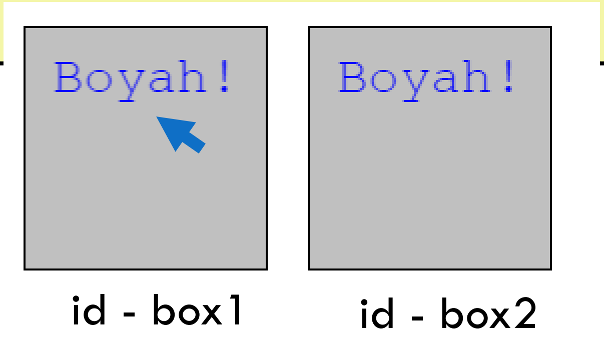


id - box1 id - box2

Consider a simple example (JS)

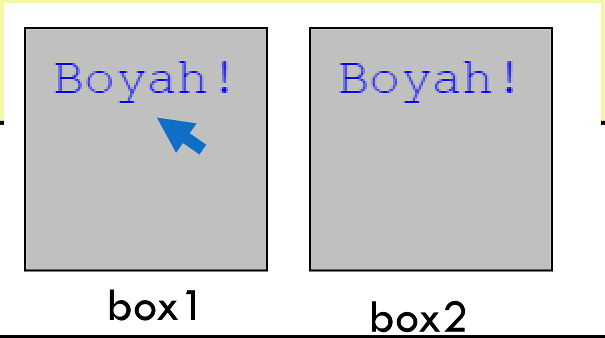
```
window.onload = function() {
   var box1 = document.getElementById("box1");
   var box2 = document.getElementById("box2");
   box1.onclick = changeText;
   box2.onclick = changeText;
}

function changeText()
{
 /* How do you know which box called this function? */
 /* use the "this" variable */
 this.innerHTML = "Boyah!";
}
```

Both elements use the same function – changeText()!
How can we know which element called the function?



The "this" keyword

```
function changeText()
{
 /* How do you know which box called this function? */
 /* use the "this" variable */
 this.innerHTML = "Boyah!";
}
                                                      JS
```



Inside the function, we can refer to the object that called it using the "this" variable.
If box1 was the one that called changeText(), the variable `this` is equal to:

```
        this = document.getElementById("box1");
```

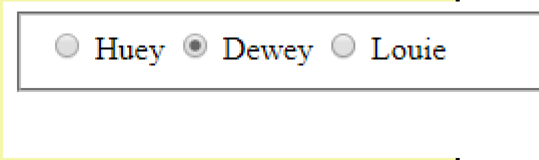
if box2 was the one that called changedText(), then the variable `this` is equal to:

```
        this = document.getElementById("box2");
```

Another example (without this)

```
<fieldset>
	<label><input id="huey" type="radio" name="ducks" value="Huey" /> Huey</label>
	<label><input id="dewey" type="radio" name="ducks" value="Dewey" /> Dewey</label>
	<label><input id="louie"type="radio" name="ducks" value="Louie" /> Louie</label>
</fieldset>
```
HTML

```
function processDucks() {
	if ($("huey").checked) {
		alert("Huey is checked!");
	} else if ($("dewey").checked) {
		alert("Dewey is checked!");
	} else {
		alert("Louie is checked!");
	}
}
```
JS



In this example, we examine each radio button to see which one is checked. Since only one radio button can be checked at a time, the element is checked must be the element that was just clicked caused this function to be called.

Another example (with this)

```
<fieldset>
        <label><input id="huey" type="radio" name="ducks"
value="Huey" /> Huey</label>
        <label><input id="dewey" type="radio" name="ducks"
value="Dewey" /> Dewey</label>
        <label><input id="louie"type="radio" name="ducks"
value="Louie" /> Louie</label>
</fieldset>                                            HTML
```

```
function processDucks() {
        alert(this.value + " is checked!");
}                                                         JS
```

We have replaced the previous slide's code with this more compact code that uses the "this" variable.

We can instead use the "this" keyword, because it corresponds to the object that just called the event. In this case, we use the "value" attribute of the element in our alert.

Events

Lots of events! Prototype style

abort	blur	change	click	dblclick	error	focus
keydown	keypress	keyup	load	mousedown	mousemove	mouseout
mouseover	mouseup	reset	resize	select	submit	unload

- the click event (onclick) is just one of many events that can be handled
- **problem**: events are tricky and have incompatibilities across browsers
  - reasons: fuzzy W3C event specs; IE disobeying web standards; etc.
- **solution**: Prototype includes many event-related features and fixes

Event handlers the Prototype way

```
~~element.onevent = function;~~
element.observe("event", "function");
                                                              JS
```

```
// call the playNewGame function when the Play button
$("play").observe("click", playNewGame);
                                                              JS
```

- to use Prototype's event features, you must attach the handler using the DOM element
- object's observe method (added by Prototype)
- pass the event of interest and the function to use as the handler
- handlers must be attached this way for Prototype's event features to work

The Event object

```
function name(event) {
// an event handler function ...
}                                                                    JS
```

- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

method / property name	**description**
type	what kind of event, such as "click" or "mousedown"
element()	the element on which the event occurred
stop()	cancels an event
stopObserving()	removes an event handler

Mouse events

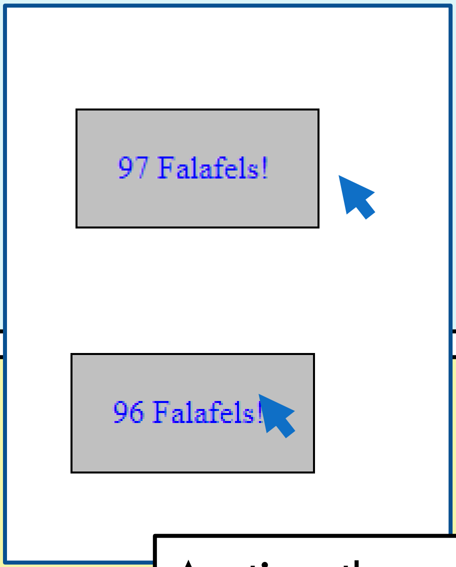
click	user presses/releases mouse button on this element (***note: click is used instead of onclick***)
dblclick	user presses/releases mouse button twice on this element
mousedown	user presses down mouse button on this element
mouseup	user releases mouse button on this element

Mouse events

mouseover	mouse cursor enters this element's box
mouseout	mouse cursor exits this element's box
mousemove	mouse cursor moves around within this element's box

Example – mouse over

```
</head>
<body>
  <div id="counter">
    99 Falafels!
  </div>
</body>
</html>
```



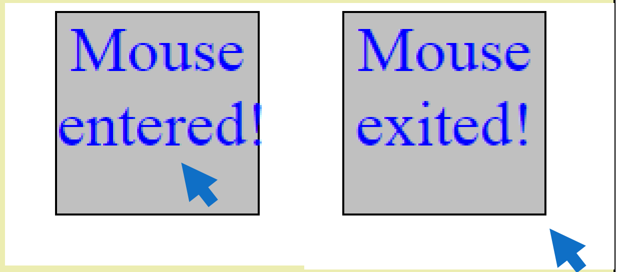
```
var count = 99;  /* global variable */

window.onload = function() {  /* set event to observe */
  $("counter").observe("mouseover", countDown);
}

function countDown () {  /* function to call */
    count = count - 1;
  $("counter").innerHTML = Count + " Falafels!";
}
```

Anytime the mouse moves over the div, the countDown() function is called.

Example 2 (mouse in and out)

```
<body>
  <div id="region">
    Mouse here!
  </div>
</body>
```

```
window.onload = function() {
  $("region").observe("mouseover", enterRegion);
  $("region").observe("mouseout", exitRegion);
}

function enterRegion () {
  $("region").innerHTML = " Mouse entered! ";
}

function exitRegion () {
  $("region").innerHTML = " Mouse exited! ";
}
```

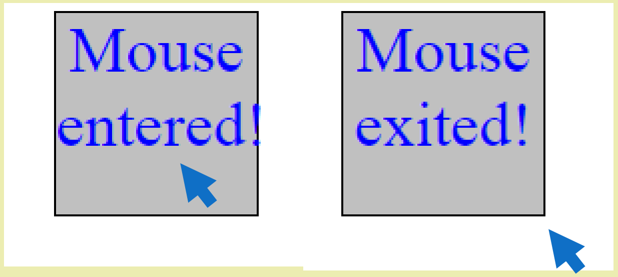


We are observing two events – mouseover (enter) and mouseexit.

Example 3 (using event and this)

```
<body>
  <div id="region">
    Mouse here!
  </div>
</body>
```

```
window.onload = function() {
  $("region").observe("mouseover", mouseEvent );
  $("region").observe("mouseout",  mouseEvent );
}

function mouseEvent (event) {
  if (event.type == "mouseover")   {
     this.innerHTML = " Mouse entered! ";
  }
  else {
     this.innerHTML = "Mouse exited! ";
  }
}
```



We can also do this using the event object. In this example, both events call the mouseEvent function. We can check the event.type to see what event was called.

E

Previous example details

```
function mouseEvent (event) {
  if (event.type == "mouseover") {
     this.innerHTML = " Mouse entered! ";
  }
  else {
     this.innerHTML = "Mouse exited! ";
  }
}
```

Notice that we have added an "event" in the function parameter list.

The event object is passed automatically. The event object (see slide 13) has several attributes, one is "type" that returns the type of event this is (e.g. click, mouseover, etc).

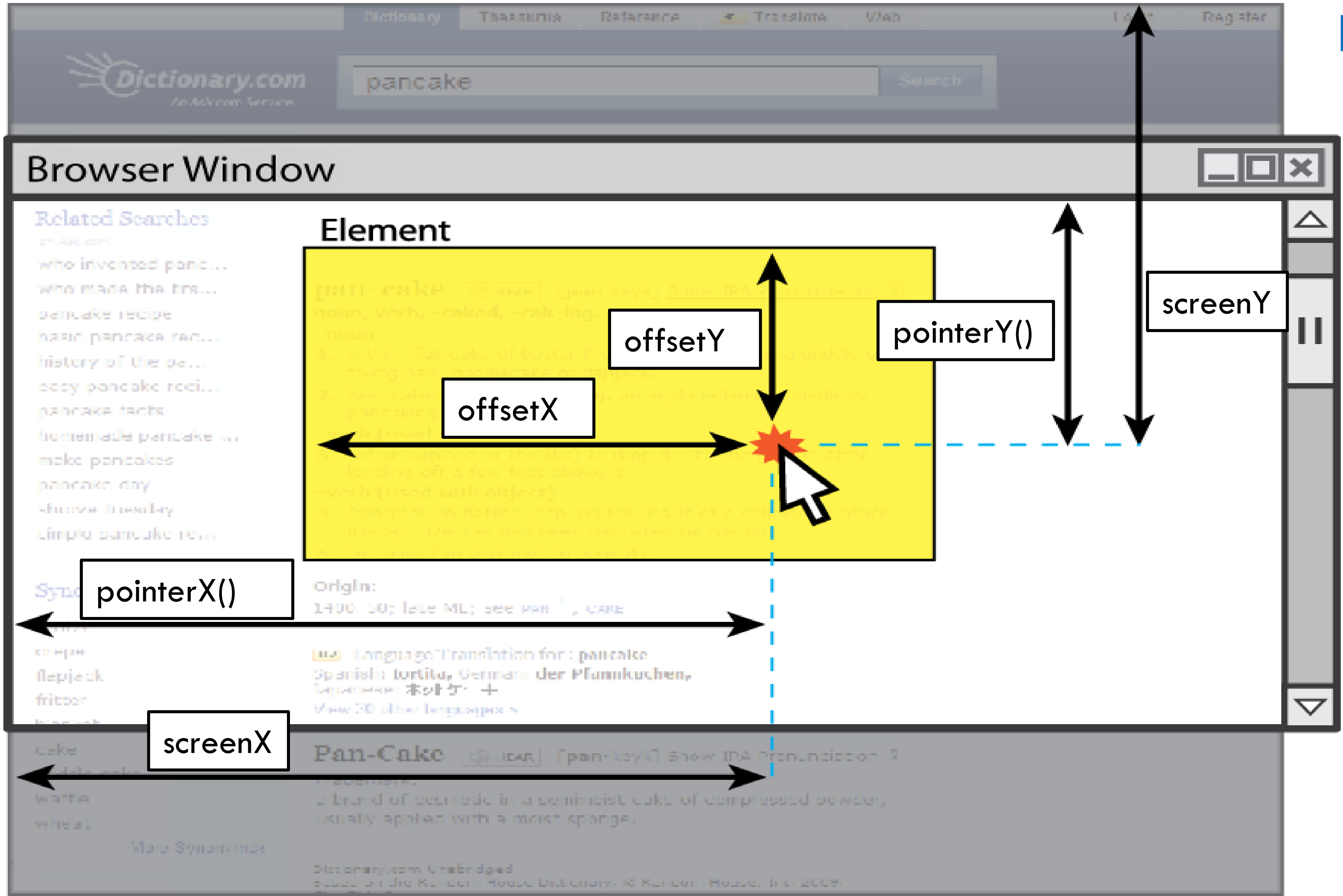
The example also uses the "this" variable which access the element that called the function.

Mouse event object*

property/method	**description**
clientX, clientY	coordinates in *browser window*
screenX, screenY	coordinates in *screen*
offsetX, offsetY	coordinates in *element* (non-standard)
pointerX(), pointerY()	coordinates in *entire web page*
isLeftClick()	true if left button was pressed

*If an event object is created by a mouse event object, there are additional properties in the event object as shown above.

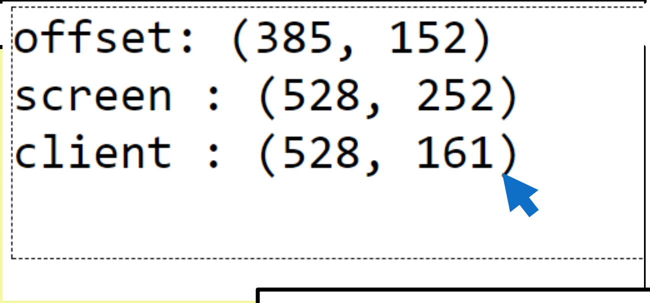
Mouse event object



Example

```
<body>
  <pre id="target">
  </pre>
</body>
```

```
window.onload = function() {
  $("target").observe("mousemove", showCoords);
};

function showCoords(event) {
  $("target").innerHTML =
      "offset: (" + event.offsetX + ", " + event.offsetY + ")\n"
    + "screen : (" + event.screenX + ", " + event.screenY + ")\n"
    + "client : (" + event.clientX + ", " + event.clientY + ")";
}
```



```
offset: (385, 152)
screen : (528, 252)
client : (528, 161)
```

We can examine the "event" object to see information about the mouse position.

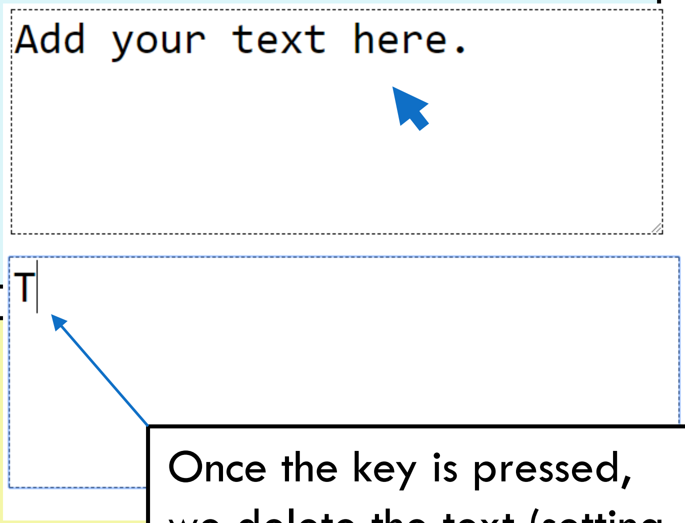
Keyboard events

These events are generally used with HTML elements that lets the user type.
For example <input type="text"> elements </input>.
Other elements may not respond to keyboard events.

name	**description**
keydown	user presses a key while this element has keyboard focus
keyup	user releases a key while this element has keyboard focus
keypress	user presses and releases a key while this element has keyboard focus

Example with keypress

```
<body>
<textarea id="target">
Add your text here.
</textarea>
</body>
```



Add your text here.

T

```
window.onload = function() {
  $("target").observe("keydown", keyPress);
};

function keyPress(event) {
  $("target").innerHTML = "";
  $("target").stopObserving("keydown");
}
```

Once the key is pressed, we delete the text (setting it to empty string "") and then stop observing this event.

Form Submit

We can even submit using JS!

```
<body>
<form id="myForm" action="http://www.google.com" method="get">
  <input id="query" type="text" name="q" maxlength="10">
  <button id="mybutton" type="button"> Submit JS </button>
</form>
</body>
```

IMPORTANT: In this case, it is necessary to tell HTML the button type is "button", otherwise the default type will be "submit". If it is submit, the button itself will submit the document, however, we want to perform the submit using JS.

```
window.onload = function() {
  $("mybutton").onclick = submitCheck;  };

function submitCheck() {
  if ($("query").value == "") {
    alert("Please enter text for pressing submit!");
  }
  else {
    $("myForm").submit();
  }
}
```

Once the button is clicked, we can check conditions before submitted (e.g. is the "query" value not an empty string "".

We use the form element to perform the **submit().**

Timers

Timers

After 5 minutes, call function X.

Only calls this function one time.

We can set a "timer" object to call a function after waiting a specified amount of time.

setTimeout(..)

Keep calling function X every 5 minutes.

We can set a "timer" object to repeatedly call a function after waiting a specified amount of time.

setInterval(..)

Timers can set to call functions

method	**description**
setTimeout(*function*, *delayMS*);	arranges to call given function after given delay in ms
setInterval(*function*, *delayMS*);	arranges to call function repeatedly every *delayMS* ms

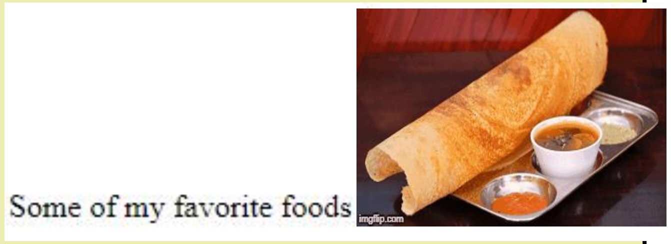
clearTimeout(*timerID*);
clearInterval(*timerID*);
	stops the given timer so it will not call its function

Example – change images

```
<body>
  <p>  Some of my favorite foods  <img src="dosa.jpg" height="100" id="food"></p>
</body>
```

```
var i=0;
timerId = 0;
window.onload = function() {
  timerId = setInterval(changeImage, 1000);
}

function changeImage()  {
  var images = ["dosa.jpg", "falafel.jpg", "pide.jpg", "malaxiangguo.jpg"];
$("food").src = images[i]; /* change the source of the image to a new image */
   i++;
   if (i > 3) {
      i = 0;
   }
}
```

Set interval timer to call "changeImage" every 1000 milliseconds (which is 1second)


Some of my favorite foods

Version 2 - Adding in mouse events.

```
<body>
  <p>  Some of my favorite foods  <img src="dosa.jpg" height="100" id="food"></p>
</body>
```

```
var i=0;
timerId = 0;
window.onload = function() /* when the timer is set, it returns and ID */
  timerId = setInterval(changeImage, 1000);  /* save id to var */
 $("food").observe("mouseover", enterRegion);/* register mouse events – enter */
 $("food").observe("mouseout", exitRegion);  /* mouse - exit */
 $("food").observe("click", changeImage);    /* mouse - click */
}

function enterRegion ()  {  /* if we enter the image, clear the event */
   clearTimeout(timerId);   /* this stops it from changing */
}                           /* timerID variable is passed to the clear func */

function exitRegion () {
   timerId = setInterval(changeImage, 100);  /*if we exit, set the timer! */
}                                           /*record the new timers id */
```

More complex example

.. continue from previous slide ..

```
/* NOTICE – we set the timer to automatically call this function, we also set the "click" event
    to call this function */
function changeImage()  {
  var images = ["dosa.jpg", "falafel.jpg", "pide.jpg", "malaxiangguo.jpg"];
$("food").src = images[i]; /* change the source of the image to a new image */
   i++;
   if (i > 3) {
      i = 0;
   }
}
```

This JS program stops the timer when the mouse enters the element.   If the user clicks, it changes the image. When the mouse leaves the element, the timer is set again.

A global variable is used to keep track of the timers ID. This ID is necessary when we remove the timer.

Recap

- The power of JavaScript is reacting to events
- There are many events beyond just "click"
- Putting these together we can make very interactive webpages
- We can also validate data before "submit" via forms
- Timers can be used to create events

Cool example

Note that the position attribute has been set for both divs.

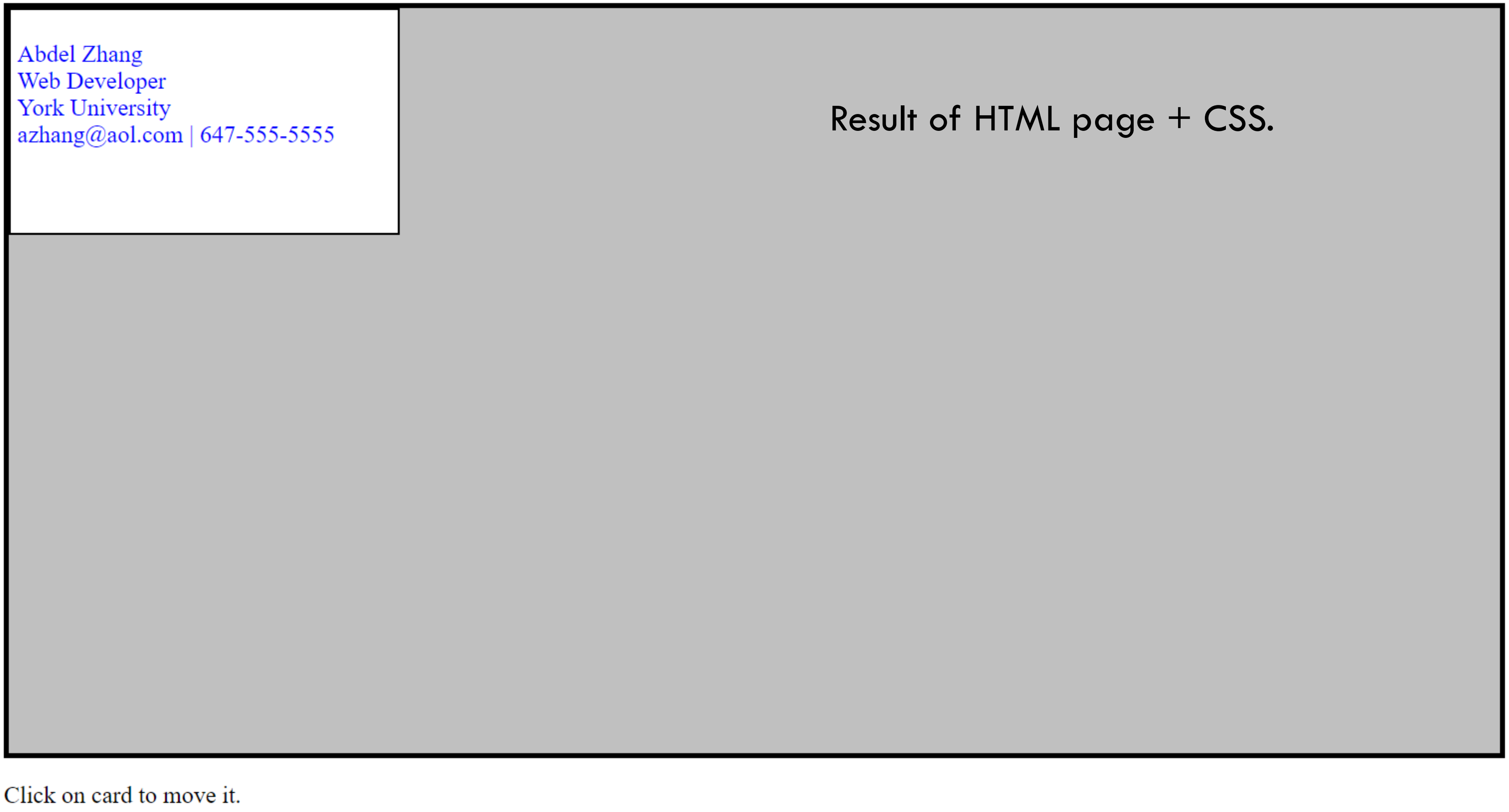
HTML

```
<html>
<head>
<script src="prototype.js" type="text/javascript"> </script>
<script src="example8.js" type="text/javascript"> </script>
<link rel="stylesheet" type="text/css" href="example8.css">
</head>
<body>
<div id="cardarea">
<div id="card">
  <p> Abdel Zhang <br>
   Web Developer <br>
  York University <br>
  azhang@aol.com | 647-555-5555 </p>
</div>
</div>
<p id="hi">Click on card to move it.</p>
</form>
</body>
</html>
```

CSS

```
#cardarea {
  width: 1000px;
  height: 500px;
  border: 3px solid black;
  background-color: silver;
  position: relative;
}

#card {
  border: 1px black solid;
  width: 250px;
  height: 140px;
  background-color: white;
  color: blue;
  padding: 5px;
  position: absolute;
}
```

Cool example



Abdel Zhang
Web Developer
York University
azhang@aol.com | 647-555-5555

Result of HTML page + CSS.

Click on card to move it.

Final (awesome) example

```
/* Global variables */
var moving = false; /* is the card moving or not? */

window.onload = function() {
  $("card").observe("mousedown", cardMouseDown);
  $("card").observe("mousemove", cardMouseMove);
  $("card").observe("mouseup", cardMouseUp);
  $("card").style.left = 0;       /* sets the position of the card
  $("card").style.top = 0;        /* this is part of its CSS */
                                  /* the position is "absolute" wit
                                  /* container div  - see CSS on
};

....
```

Create some global variables.
We will use these later.

Set up event functions (using Prototype library).

Set the left and top attributes. These correspond to the position attribute in CSS.

Final (awesome) example

```
/* Called when the user presses down the mouse button.
   Moves the clicked square to the top and starts moving it. */
function cardMouseDown(event) {
   moving = true;
  this.style.backgroundColor = "lightgrey";
  $("card").style.left = (event.pointerX() - 125) + "px";
  $("card").style.top = (event.pointerY() - 70)  + "px";
}

/* Called when the user lifts the mouse button.  Stops dragging. */
function cardMouseUp(event) {
  moving = false;
  this.style.backgroundColor = "white";
}
....
```

When the mouse is clicked on the card, set moving to true and change the cards background (using "this"). Sets the card's left and top position to be the current mouse position minus 125 and 70 because the card is 300x140. This will center the card.

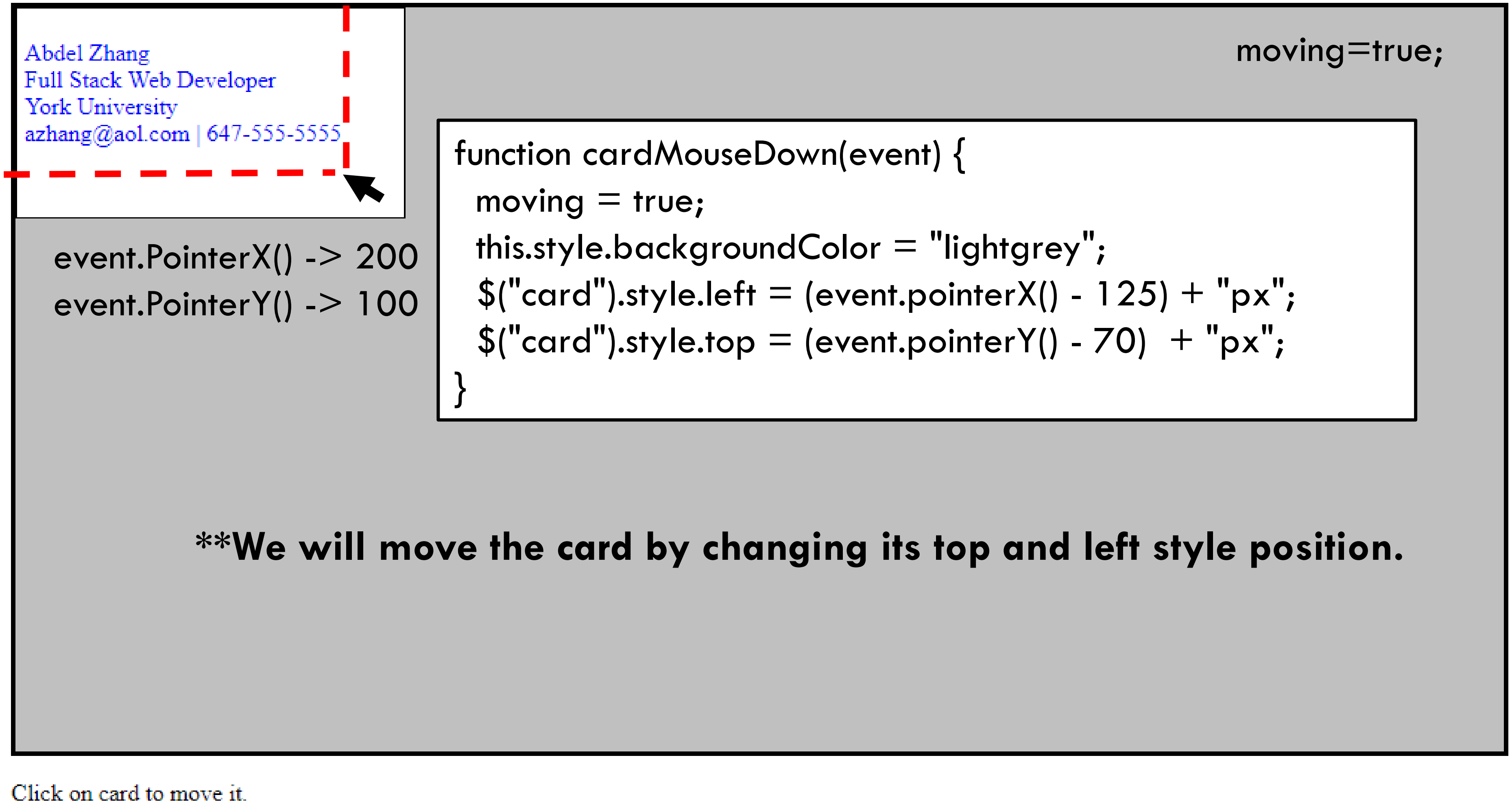
When the button is released, it will call this mouseUp event. Set moving to false, and reset the background to white.

Final (awesome) example

```
// Called when the user moves the mouse.  Drags a square if
// the mouse button is being held down.
function cardMouseMove(event) {
  if (moving)
  {
     $("card").style.left = (event.pointerX() - 125) + "px";
     $("card").style.top = (event.pointerY() -   70)  + "px";
  }
}
```

If we are "moving", then set the card's left and top to be the current mouse (X,Y). We subtract 125 and 70 to shift the box to be centered around the mouse.
The card size is 300x140 pixels.

Final (awesome) example



Abdel Zhang
Full Stack Web Developer
York University
azhang@aol.com | 647-555-5555

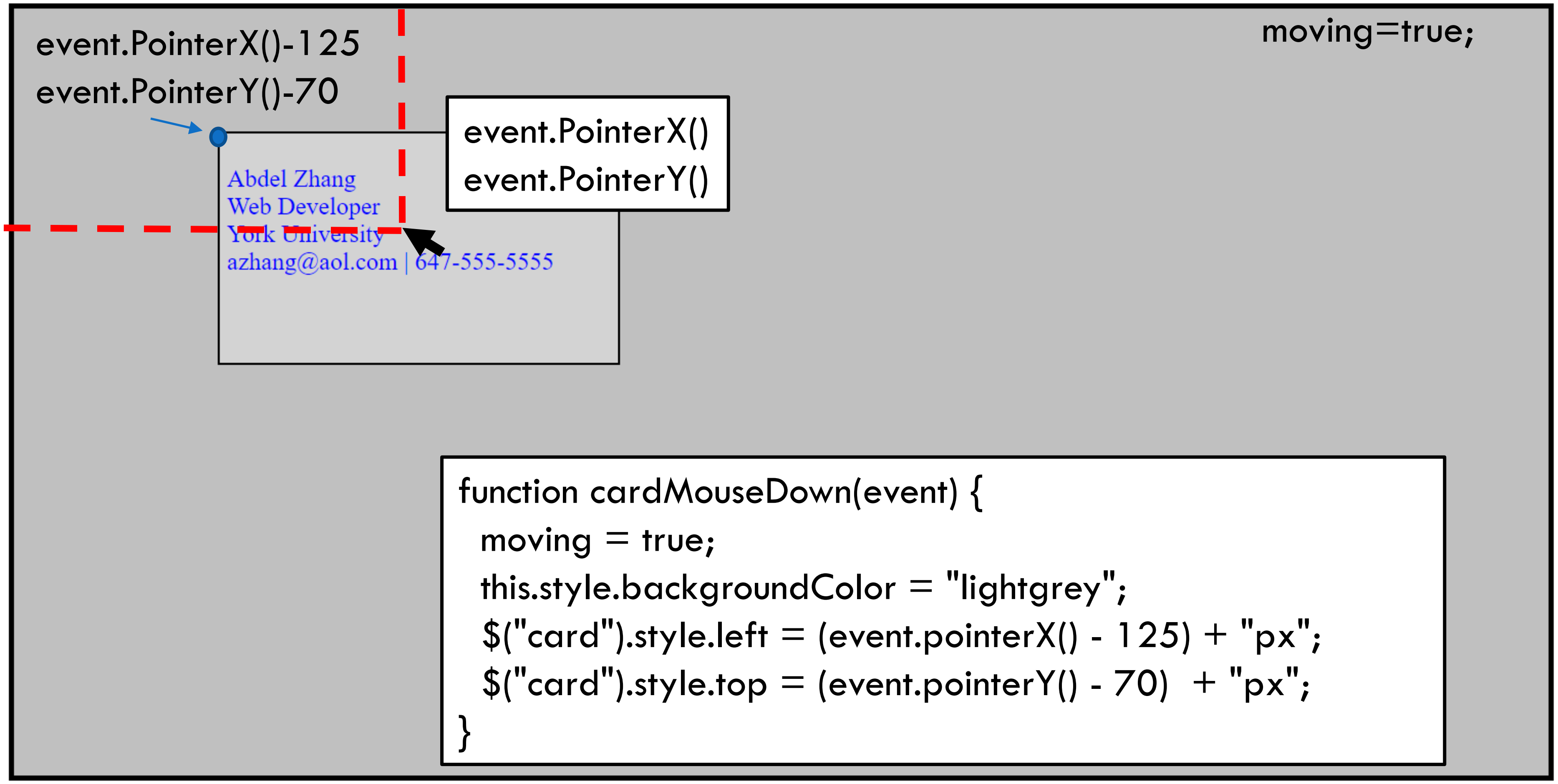
moving=true;

event.PointerX() -> 200
event.PointerY() -> 100

```
function cardMouseDown(event) {
  moving = true;
  this.style.backgroundColor = "lightgrey";
  $("card").style.left = (event.pointerX() - 125) + "px";
  $("card").style.top = (event.pointerY() - 70)  + "px";
}
```

****We will move the card by changing its top and left style position.**

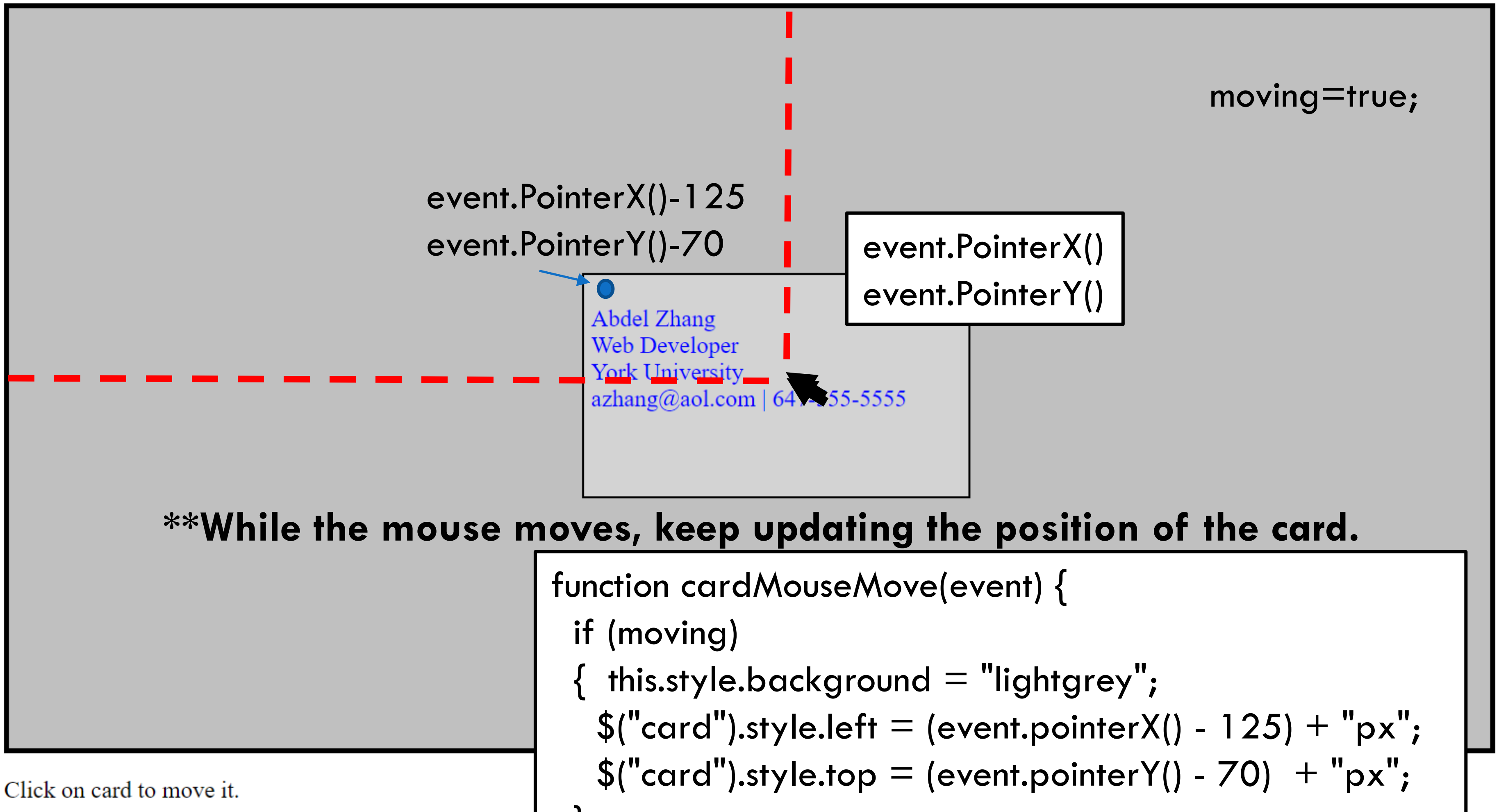
Click on card to move it.



```
function cardMouseDown(event) {
  moving = true;
  this.style.backgroundColor = "lightgrey";
  $("card").style.left = (event.pointerX() - 125) + "px";
  $("card").style.top = (event.pointerY() - 70)  + "px";
}
```

Click on card to move it.

Final (awesome) example



While the mouse moves, keep updating the position of the card.

```
function cardMouseMove(event) {
  if (moving)
  {  this.style.background = "lightgrey";
    $("card").style.left = (event.pointerX() - 125) + "px";
    $("card").style.top = (event.pointerY() - 70)  + "px";
  }
}
```

Recap

- This last example draws from knowledge on . .
  - how HTML works
  - how CSS works (e.g. position attributes)
  - how JavaScript works
  - cleverness

- Now . . . do your own awesome stuff!