Programming 1

Javascript

Lecture #4: Events on the DOM and creating interactivity

CLICKER CHANNEL: 82 (Paul FM!)

The story so far

- Up until now all of our programs have a had a start, a middle, and an end
  - They start at the first line of code that isn't part of a function
  - Each line is executed in turn
  - Program flow might be re-routed because of the repetition in loops, or calls to functions…
  - …but once it reaches the final line, the program ends

What is event driven programming?

- Most Javascript programs in the real world do NOT work like the ones we've written so far
- Most Javascript programs are *event driven*
- Look at the video below:
  - http://www.youtube.com/watch?v=sCQcPEcYL9g

Cats as event driven "programs"…

- If we say the cat was a program…
  - An "event" took place
    - (the cat was surprised by the other cat)
  - The cat's "program flow" was interrupted by the event
    - (it stopped mooching about and sniffing the grass)
  - As a response to the event the cat executed a different "function"
    - (it turned around and chased the other cat away!)

What is event driven programming?

- In Javascript, events can happen on any of the HTML element in the DOM
  - A button might be **clicked**
    - (in fact most elements can be clicked)
  - An input element might **change**
- You can specify that a particular function should be run when an event occurs
  - The program will stop whatever it's doing when the event occurs and perform the specified function instead
    - (just like the cat stopped doing what it was doing and started a new "function"!)
  - When the function finishes, program flow returns to wherever it was when the event interrupted proceedings

An example of events in the DOM

HTML

```
<img id="ringo" src="http://fetlar.kingston.ac.uk/ringo.jpg" />
<button id="button1">click here for Ringo pic 1</button>
<button id="button2">click here for Ringo pic 2</button>
```

Javascript

```
function ringo1()
{
  document.getElementById("ringo").src = "http://fetlar.kingston.ac.uk/ringo.jpg";
}

function ringo2()
{
  document.getElementById("ringo").src = "http://fetlar.kingston.ac.uk/anotherRingo.jpg";
}

document.getElementById("button1").onclick = ringo1;
document.getElementById("button2").onclick = ringo2;
```

An exam

HTML

```
<img id="ringo" src="http://fetl
<button id="button1">click here
<button id="button2">click here
```

Javascript

```
function ringo1()
{
  document.getElementById("rir
}

function ringo2()
{
  document.getElementById("ringo").src = "http://fetlar.kingston.ac.uk/anotherRingo.jpg";
}

document.getElementById("button1").onclick = ringo1;
document.getElementById("button2").onclick = ringo2;
```

More on event handling

- When we assign a function to fire when an event occurs on an element, this is called *binding an event*
- We do this in Javascript by setting the *onclick* property
  - document.getElementById("button1").onclick = ringo1;

More on event handling

- When we assign a function to fire when an event occurs on an element, this is called *binding an event*
- Our example did this by setting the *onclick* property
  - `document.getElementById("button1").onclick = ringo1;`

The highlighted part is the element we're binding the event to

REMEMBER YOUR GRAMMAR! You can get the element any way you like! You might use getElementsByTagName, or some other means of navigating the DOM – or even specify a variable name here, if you've assigned an element to a variable

More on event handling

- When we assign a function to fire when an event occurs on an element, this is called *binding an event*
- Our example did this by setting the *onclick* property
  - document.getElementById("button1").onclick = ringo1;

This highlighted part is the event that we're binding – i.e. what has to happen for the function to run. So in this case, we're saying something's going to happen when this element is clicked.

More on event handling

- When we assign a function to fire when an event occurs on an element, this is called *binding an event*
- Our example did this by setting the *onclick* property
  - document.getElementById("button1").onclick = ringo1 ;

This part specifies what will happen when the event takes place. This is a function name – so we're saying that when this event is clicked, the function ringo1 should be called.

More on event handling

- When we assign a function to fire when an event occurs on an element, this is called *binding an event*
- Our example did this by setting the *onclick* property
  - document.getElementById("button1").onclick = ringo1;
- We set *onclick* on the element from Javascript code, but you can also specify it directly in the HTML markup:
  - <button id="button1“ onclick=“ringo1()”>click here for Ringo pic 1</button>
  - However, this is frowned upon by purists these days…
  - ..and you’re not allowed to alter the HTML markup in your exercises, anyway… ☺

A selected list of useful events

Event	Description
onclick	Occurs when an element is clicked
ondblclick	Occurs when an element is double-clicked
onmouseover	Occurs when the mouse pointer moves over a particular element
onmouseout	Occurs when the mouse points moves off of a particular element
onchange	Occurs when a form element (e.g. an <input> element) is changed

See http://www.w3schools.com/jsref/dom_obj_event.asp for more.

this

- The keyword `this` allows us to refer to the element on which an event occurred

HTML

```
<button id="firstbutton">First button</button>
<button id="secondbutton">Second button</button>
```

Javascript

```
function click()
{
  alert(this.innerHTML+” was clicked”);
}

document.getElementById("firstbutton").onclick = click;
document.getElementById("secondbutton").onclick = click;
```

this

- The keyword `this` allows us to refer to the element on which an event occurred

HTML

```
<button id="firstbutton">First button</button>
<button id="secondbutton">Second button</button>
```

Javascript

```
function click()
{
  alert(this.innerHTML+” was clicked”);
}

document.getElementById("firstbutton").onclick = click;
document.getElementById("secondbutton").onclick = click;
```

this

- The keyword `this` allows us to refer to the element on which an event occurred

HTML

```
<button id="firstbutton">First button</button>
<button id="secondbutton">Second button</button>
```

Javascript

```
function click()
{
  alert( this .innerHTML+" was clicked");
}

document.getElementById("firstbutton").onclick = click;
document.getElementById("secondbutton").onclick = click;
```

- If it was the button with ID firstbutton that was clicked, then `this` will contain that HTML element
- If it was the button with ID secondbutton that was clicked, then `this` will contain that HTML element

this

- The keyword `this` allows us to refer to the element on which an event occurred

HTML

```
<button id="firstbutton">First button</button>
<button id="secondbutton">Second button</button>
```

Javascript

```
function click()
{
  alert(this.innerHTML +" was clicked");
}

document.getElementById("firstbutton").onclick = click;
document.getElementById("secondbutton").onclick = click;
```

- Because `this` will contain an HTML element from the DOM, we can use any properties – e.g. innerHTML - that we would normally be able to use with our elements

Getting more information about the event

- A Javascript event is an object in its own right
- This object has properties that we can read to find out more about the event, e.g.
  - Was the SHIFT key pressed?
  - Was the ALT key pressed?
  - Was the CONTROL key pressed?
  - Was the META key pressed?
    - (COMMAND or ⌘ on a Mac)
    - (the Windows key on PCs)
  - Where was the mouse on the browser page (X position/Y position?)
  - Where was the mouse on the screen?

Reading the event properties

- If we are going to need to read the event properties from the function that gets called, we need to do our binding slightly differently:

```
document.getElementById("someElement").onclick = function(event)
{
  handleClick(event);
}

function handleClick(event)
{
  if (event.shiftKey == 1)
  {
     alert("The shift key was pressed when you clicked!");
  }
  else
  {
    alert("The shift key was NOT pressed when you clicked!");
  }
}
```

Reading the event properties

- If we are going to need to read the event properties from the function that gets called, we need to do our binding slightly differently:

```
document.getElementById("someElement").onclick = function(event)
{
  handleClick(event);
}

function handleClick(ev
{
  if (event.shiftKey ==
  {
     alert("The shift k
  }
  else
  {
    alert("The shift ke
  }
}
```

- If we want to read the event in the function that gets called when the event takes place, we need to pass it as a parameter.
- Because we can't specify parameters when we use the shorter way to bind an event, we need to use this longer form
- The instructions within the code block (between the curly brackets) are what gets run when the event takes place
- So in this case, when the onclick event happens, it runs the code handleClick(event)
  - This calls our function handleClick with a single parameter – the event itself

Reading the event properties

- If we are going to need to read the event properties from the function that gets called, we need to do our binding slightly differently:

```
document.getElementById("someElement").onclick = function(event)
{
  handleClick(event);
}

function handleClick(event)
{
  if (event.shiftKey == 1)
  {
     alert("The shift key was pressed when you clicked!");
  }
  else
  {
    alert("The shift key was NOT pressed when you clicked!");
  }
}
```

- The event is passed as a parameter to our function

Reading the event properties

- If we are going to need to read the event properties from the function that gets called, we need to do our binding slightly differently:

```
document.getElementById("someElement").onclick = function(event)
{
  handleClick(event);
}

function handleClick(ev
{
  if (event.shiftKey)
  {
     alert("The shift key was pressed when you clicked!");
  }
  else
  {
    alert("The shift key was NOT pressed when you clicked!");
  }
}
```

- We can then read the event properties from within our function

- event.shiftKey returns a boolean – true or false – depending on whether the shift key is pressed at the time of the event

Useful event properties

Property name	Description
altKey	Gives true if the ALT key is pressed, false if it is not
shiftKey	Gives true if the SHIFT key is pressed, false if it is not
ctrlKey	Gives true if the CONTROL key is pressed, false if not
metaKey	Gives true if the meta key is pressed, false if it is not (see a couple of slides back for what this key is)
button	Returns which mouse button was pressed: • 0 for the default button (usually the left) • 1 for the middle button (if your mouse has three buttons – the scroll wheel button can count here) • 2 for the secondary button (usually the right)
screenX	Gives the X coordinate on the screen where the mouse was when the event took place
screenY	Gives the Y coordinate on the screen where the mouse was when the event took place
clientX	Gives the X coordinate within the browser window where the mouse was when the event took place
clientY	Gives the Y coordinate within the browser window where the mouse was when the event took place

A trick to get your head around event driven programs

- In many respects, you can think of an event driven program as being like an infinite loop

```
document.getElementById("someElement").onclick = sayHello;
function sayHello()
{
   alert("Hello!");
}
```

- This program would run forever
- Each time the user clicked, it'd say "Hello"
  - so the user is the thing that makes the "loop" repeat

A worked example

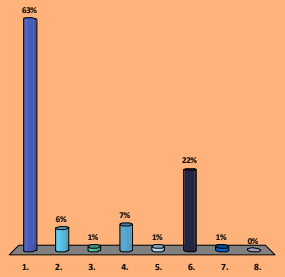
- We are now going to work through a simple example
- We will look at
  - How a combination of an HTML page and Javascript code is actually constructed in a real-world scenario
  - How the HTML page specifies the associated Javascript
  - Different approaches for binding events
  - Different approaches for handling events
  - Using "this"
- Available at
  - http://nooblab.kingston.ac.uk/NoobLab/contents/paulneve.com/programming1/js4-supplement

What would be the contents of the DIV element after running this code?

```
<div id="output">Waiting…</div>
<button id="input">Click me</a>

function handleClick()
{
  var a = document.getElementById("output");
  var b = document.getElementById("input");
  a.innerHTML = a.innerHTML + "Ouch!";
}

document.getElementById("input").onclick = handleClick;
```



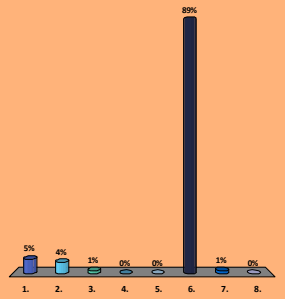
1. Waiting…
2. Click me
3. Ouch!
4. Nothing
5. Waiting…Click me
6. Waiting…Ouch!
7. Click meOuch!
8. Something else or there would be an error

OK then, what would be the contents of the DIV element after running this code and clicking the button?

```
<div id="output">Waiting…</div>
<button id="input">Click me</a>

function handleClick()
{
  var a = document.getElementById("output");
  var b = document.getElementById("input");
  a.innerHTML = a.innerHTML + "Ouch!";
}

document.getElementById("input").onclick = handleClick;
```



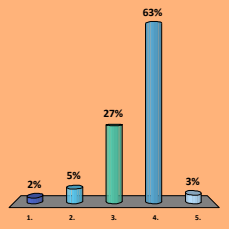
1. Waiting…
2. Click me
3. Ouch!
4. Nothing
5. Waiting…Click me
6. Waiting…Ouch!
7. Click meOuch!
8. Something else or there would be an error

Which button would need to be clicked in order for the *You Guessed Correctly* text to appear?

```
<div id="output">Waiting…</div>
<button id="input1">17</a>
<button id="input2">31</a>
<button id="input3">22</a>

function handleClick()
{
  var a = document.getElementById("output");
  if (this.innerHTML.indexOf("2") != -1)
  {
     a.innerHTML = "You guessed correctly!";
  }
}

for (var count = 1; count <= 3; count++)
{
  var b = document.getElementById("input"+count);
  b.onclick = handleClick;
}
```



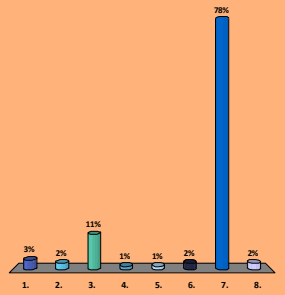
1. The button labelled 17
2. The button labelled 31
3. The button labelled 22
4. Any of them
5. You could click them all until the cows came home and that text would never appear

How about now; what would be the contents of the DIV element after running this code and clicking the button?

```
<div id="output">Waiting…</div>
<button id="input">Click me</a>

function handleClick()
{
  var a = document.getElementById("output");
  var b = document.getElementById("input");
  a.innerHTML = this.innerHTML + "Ouch!";
}

document.getElementById("input").onclick = handleClick;
```



1. Waiting…
2. Click me
3. Ouch!
4. Nothing
5. Waiting…Click me
6. Waiting…Ouch!
7. Click meOuch!
8. Something else or there would be an error

Summary

- Event driven programming involves defining functions that run when and only when a certain *event* happens
  - e.g. click, mouseDown, mouseUp, etc
- An event driven program might run forever
  - It will sit there and potentially do nothing until an event takes place
  - When an event takes place on an element in the DOM, whatever function has been *bound* to the event/element will run
- `this` allow you to reference the element upon which the event took place from within the bound function