2015

BRAIN ACADEMY

Events

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Event definition – 1/3

An ***event*** is a signal from the browser that something has happened.

There are many **types of events**:

- DOM events, which are initiated by DOM-elements. For instance, a `click` event happens when an element is clicked, a `mouseover` - when a mouse pointer comes over an element,
- Window events. For instance, `resize` - when a browser window is resized,
- Other events, like `load`, `readystatechange`. They are used in AJAX and for other needs.

Event definition – 2/3

At a high level, everything can be modeled by the following statement:

When _____ happens, do _____.

When a page load happens, do play the video of a cat sliding into cardboard.

When a click happens, do submit my online purchase.

When a mouse release happens, do hurl the giant/not-so-happy bird.

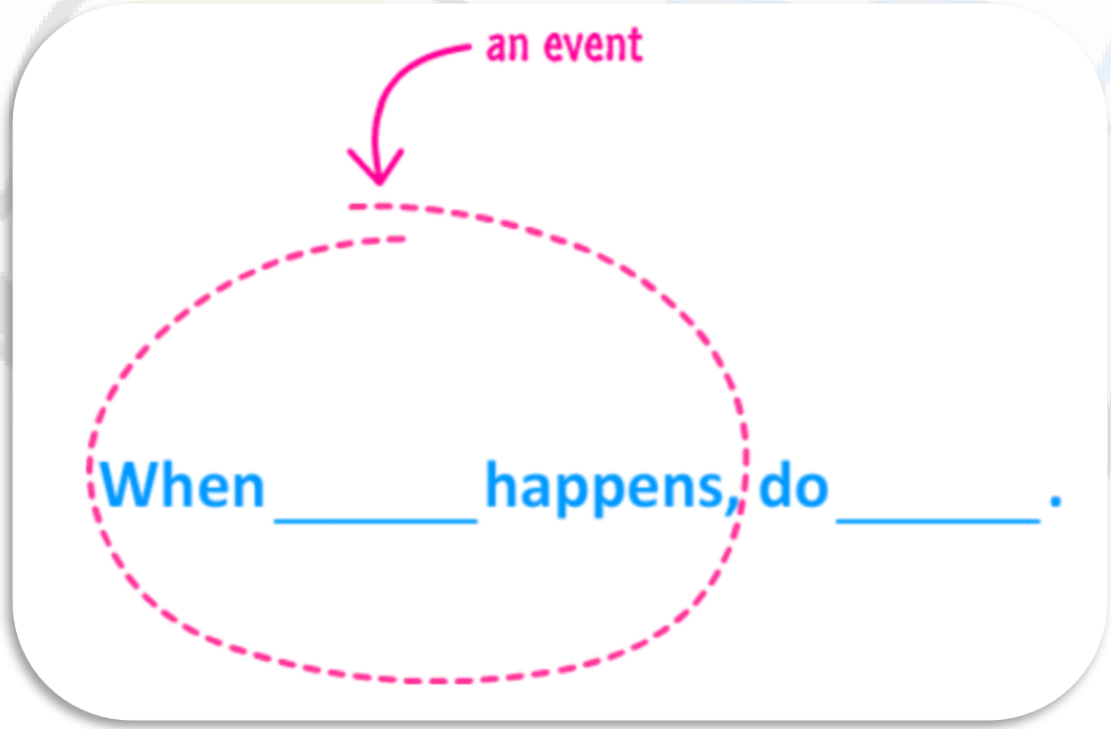
When a delete key press happens, do send this file to the Recycle Bin.

When a touch gesture happens, do apply this old timey filter to this photo.

When a file download happens, do update the progress bar.

Event definition – 3/3

Events define the thing that happens. They fire the signal. The second part of the model is defined by the reaction to the event.



Your task is to tell your application to listen only to the events you care about.

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

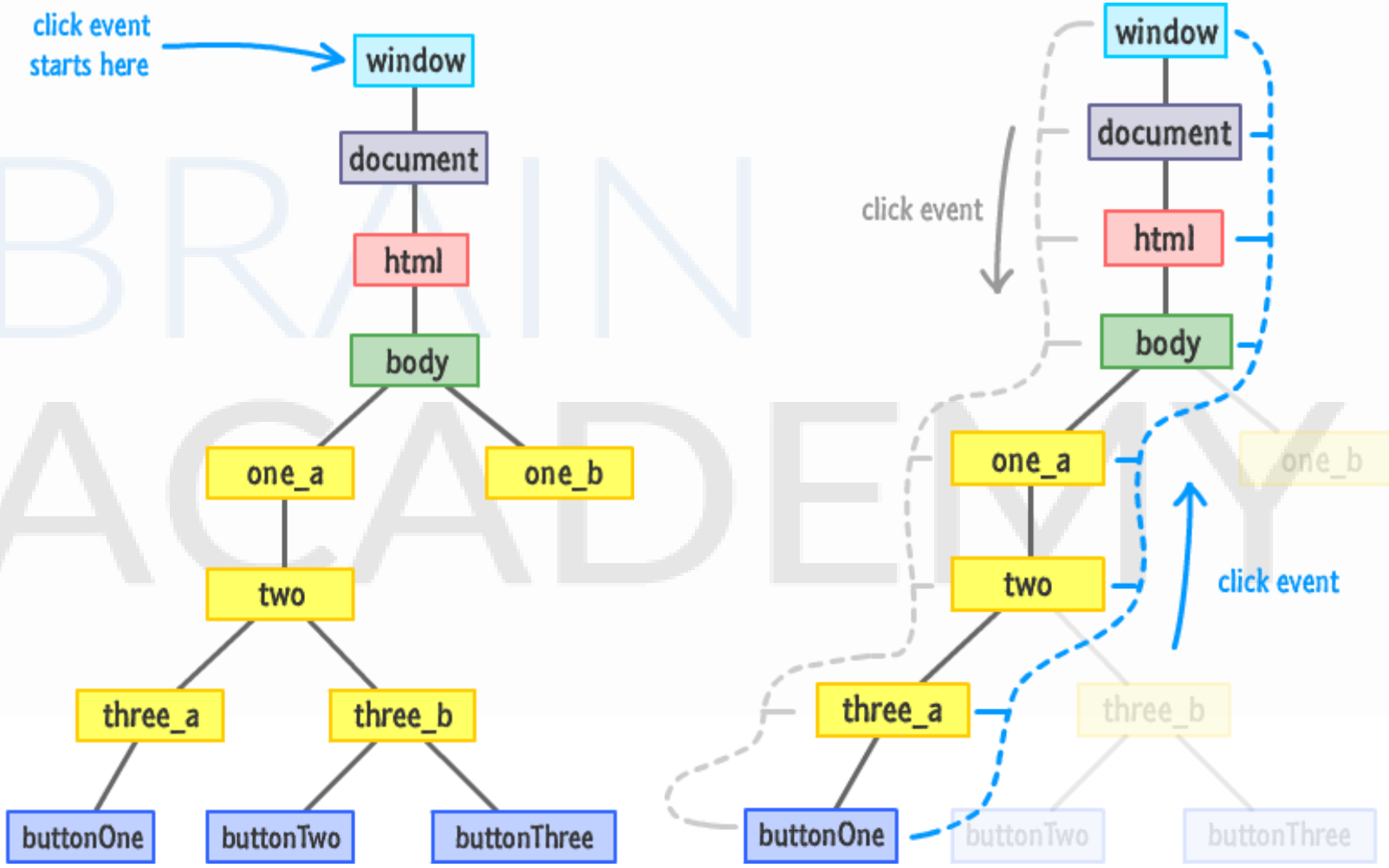
Event phases - 1/4

1. Let's say that we click on the **buttonOne** element.

2. an event starts at the root of your document

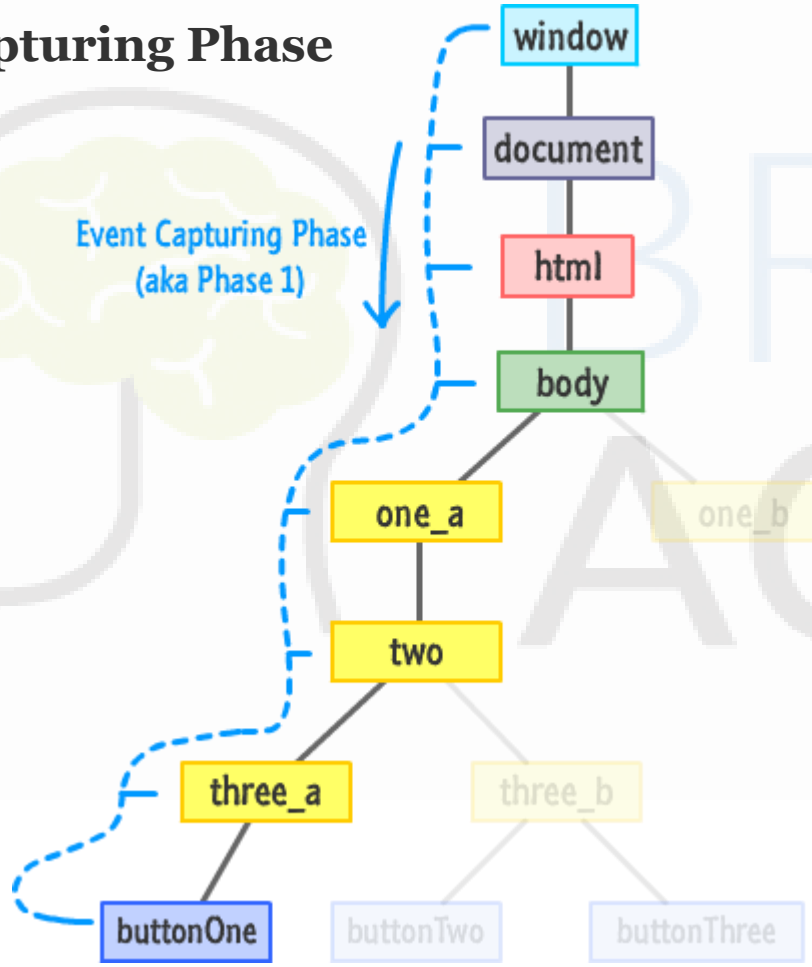
3. From the root, the event makes its way through the narrow pathways of the DOM and stops at the element that triggered the event, **buttonOne** (also more formally known as the event **target**)

3. the event keeps going by retracing its steps and returning back to the root

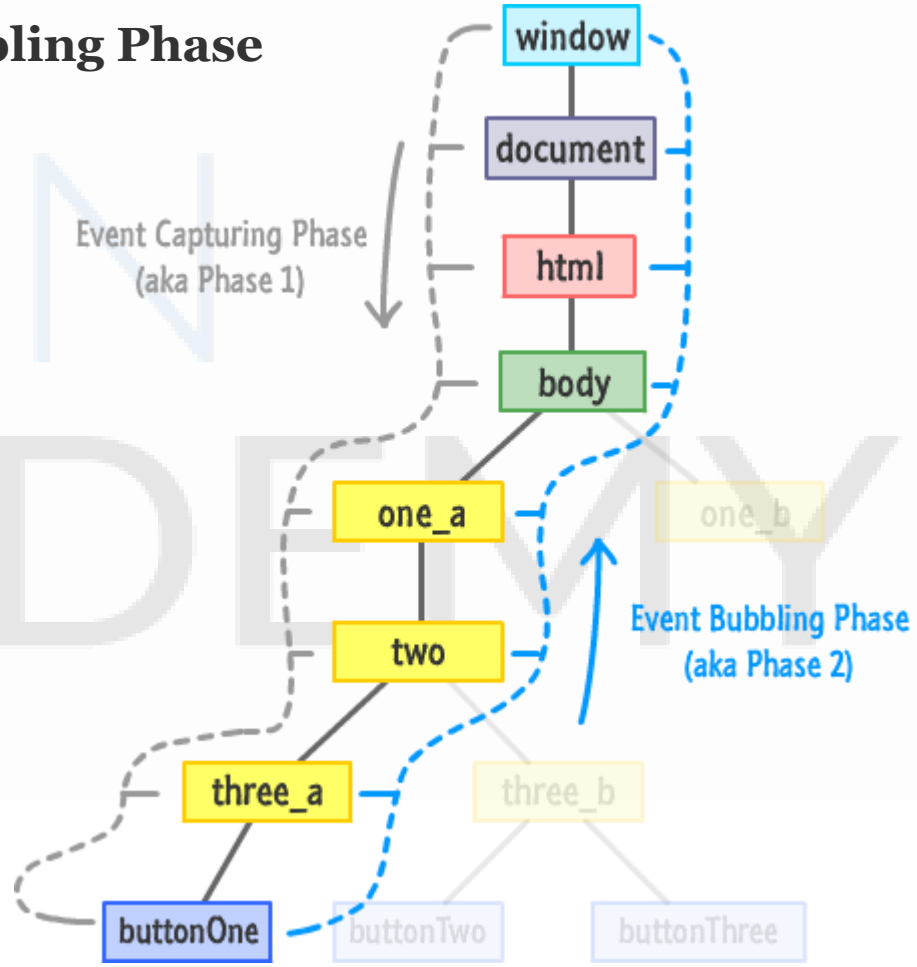


Event phases – 2/4

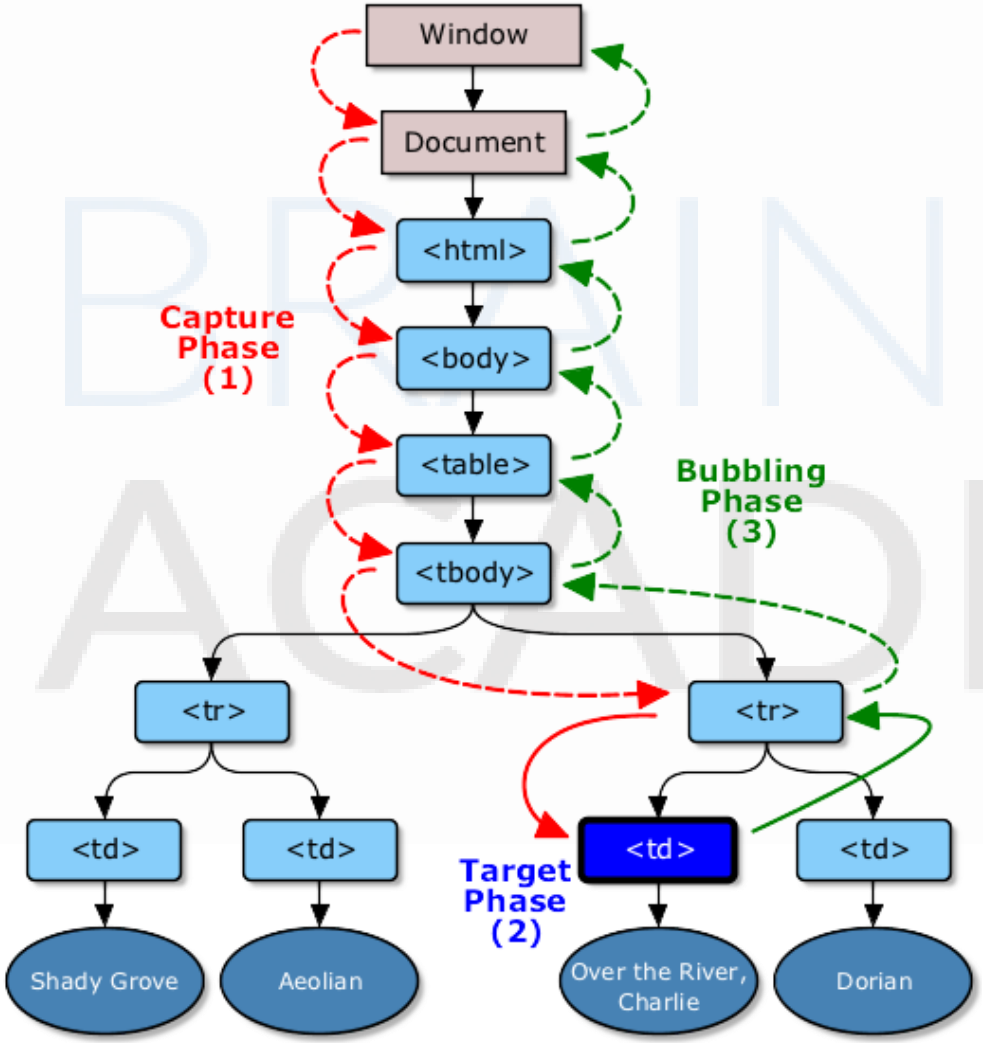
Event Capturing Phase



Event Bubbling Phase



Event phases – 3/4



Event phases – 4/4

- There are the situations where you have to consciously be aware of which phase of event's life you are watching for:
  - Dragging an element around the screen and ensuring the drag still happens even if my mouse cursor slips out from under the cursor
  - Nested menus that reveal sub-menus when you hover over them
  - You have multiple event handlers on both phases, and you want to focus only on the capturing or bubbling phase event handlers exclusively
  - A third party component/control library has its own eventing logic and you want to circumvent it for your own custom behavior
  - You want to override some built-in/default browser behavior such as when you click on the scrollbar or give focus to a text field

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Listening to an event

Functions which react on events are called ***event handlers***.

There are several ways of assigning an event handler:

- Using a **attribute** of HTML-tag
- Using a DOM-object **property**
- Special methods

Attribute of HTML-tag – 1/2

A handler can be set directly in the **markup**, right into the **attribute** named ***onevent***.

```
<input id="b1" value="Click me" onclick="alert('Thanks!');" type="button"/>
```

It is also possible to call a function for the event handling:

```
<input type="button" onclick="count_rabbits()" value="Count rabbits!"/>

<script type="text/javascript">
    function count_rabbits() {
        for(var i=1; i<=3; i++) {
            alert("Rabbit “ + i + " out of the hat!");
        }
    }
</script>
```

Attribute of HTML-tag – 2/2

Please recall that HTML-tag attribute names are case-insensitive, so **`oNcLiCk`** will work same as **`onClick`** or **`onclick`**. But it is generally considered a good style to use lowercase.

When to use this method:

This way of assigning handlers is very **convenient** - it's simple and all-inline, that's why it is sometimes used for really simple tasks.

There are certain drawbacks of this method. When a handler becomes longer than one line - **readability suffers** greatly.

But, after all, no one writes somewhat **complex handlers in HTML**. Instead of it, use JavaScript-only ways which are described in the next subsection.

- A simple way for simple tasks
- Mixed JavaScript-code and HTML-markup
- Difficult to write complex handlers

DOM-object **property** – 1/2

A closest relative of the way described above - is an assignment using the **property** named ***onevent***.

All you need is:

1. To get an element
2. To assign a handler to the property *onevent*

```
<input id="myElement" type="button" value="Press me"/>

<script>
    document.getElementById('myElement').onclick = function() {
        alert('Thanks');
    }
</script>
```

DOM-object **property** – 2/2

Please, note the two details:

1. It is a **property**, not an attribute. The name of the property is **on*event***, **case-sensitive** and must be ***lowercased***. `onClick` won't work.
2. The handler must be a **function**, not a string.

- When the browser meets an `on...` attribute in HTML-markup - it basically creates a function from its contents and assigns it to the property.
- If there is a handler set in markup, the script overwrites it.
- Of course, it is possible to use an existing function.

When to use:

- Assigning handlers using a property is a very **simple** and popular way.
- It has a problem: **only one handler** for a certain event type can be set.

1. A convenient and reliable way, works in JavaScript

2. A single handler per event

addEventListener – 1/3

```
source.addEventListener(eventName, eventHandler, false);
```

source - You call addEventListener via an element or object that you want to listen for events on. Typically, that will be a DOM element, but it can also be you document, window, or any other object that just happens to fire events.

eventName - the event name goes without the “on” prefix

eventHandler – function that should be assigned as event handler

phase - the third parameter , which is usually not used and set to false

* W3C or official event handler assignment works in all modern browsers and for IE9

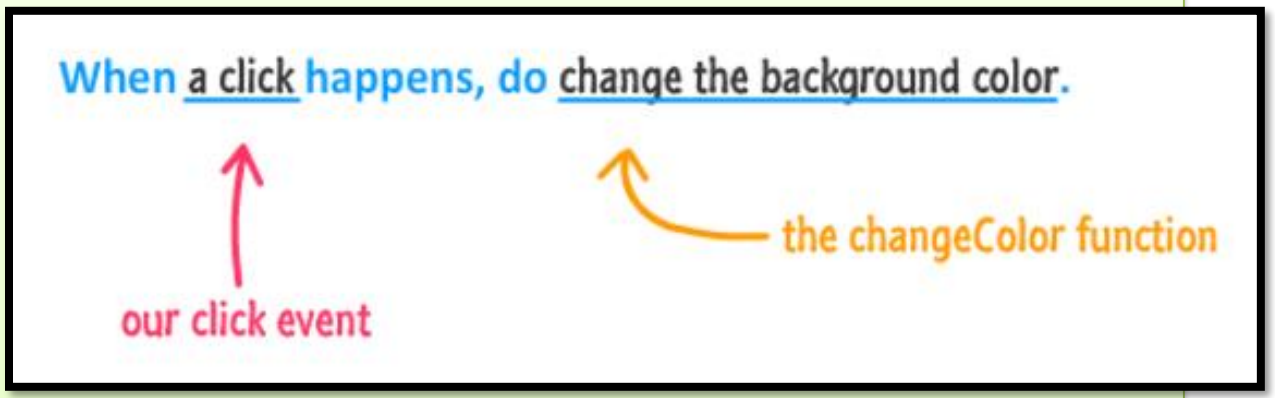
```
// ... declare a function called handler ...

elem.addEventListener( "click" , handler, false) // assign the handler
```

addEventListener – 2/3

```
<!DOCTYPE html>
<html>
<head>
    <title>Click Anywhere!</title>
</head>
<body>
    <script>
        document.addEventListener("click", changeColor, false);

        function changeColor() {
            document.body.style.backgroundColor = "#FFC926";
        }
    </script>
</body>
</html>
```



addEventListener – 3/3

So, there is a one big plus and one minus of the special method:

1. As many handlers as you want (Browser does not guarantee the order in which they execute. Generally, the order of assignment is not related with the order of execution. The order may happen to be same, or inversed or random.)

2. Cross-browser incompatibilities (The incompatibilities is not just different syntax, but there are few other differences. We'll return to it in the next sections and discuss a cross-browser method of event handling)

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Removing event handlers – 1/3

- DOM-object **property**

```
<input id="myElement" type="button" value="Press me"/>

<script>
    document.getElementById('myElement').onclick = function() {
        alert('Thanks');
    }
    document.getElementById('myElement').onclick = ‘’;
</script>
```

Removing event handlers – 2/3

- Event handler added by **addEventListener**:

```
source.removeEventListener(eventName, eventHandler, false);
```

source - an element or object that was assigned an event handler before

eventName - the event name goes without the “on” prefix

eventHandler – function that should be removed as event handler

phase - the third parameter , which is usually not used and set to false

Removing event handlers – 3/3

```
<!DOCTYPE html>
<html>
<head>
    <title>Click Anywhere!</title>
</head>
<body>
    <script>
        function changeColor() {
            document.body.style.backgroundColor = "#FFC926";
        }

        document.addEventListener("click", changeColor, false);

        document.removeEventListener("click", changeColor, false);
    </script>
</body>
</html>
```

If the `removeEventListener` call used any argument that was different than what was specified with the corresponding `addEventListener` call, then its impact would be ignored and the event listening will continue.

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

‘this’ keyword inside handler – 1/2

In JavaScript ‘this’ always refers to the “owner” of the function we're executing, or rather, to the object that a function is a method of.

- **in-line handler –** ‘this’ is referring to Html element of handler (but ‘this’ inside inner function of handler will reference the window object)
- **property handler** - ‘this’ is referring to Html element that is currently handling the event.
- **addEventListener -** ‘this’ is referring to Html element that is currently handling the event.

‘this’ keyword inside handler – 2/2

- in-line handler

```
<button onclick="alert(this.innerHTML)">
        Click me
</button>
```

alerts ‘Click me’

```
<button onclick=“clickHandler()">Click me</button>

clickHandler(){
    alert(this.innerHTML);
}
```

Error – this is window

- property handler

```
<button id= “btn”>Click me</button>

btn.onclick = function(){
    alert(this.innerHTML);
}
```

alerts ‘Click me’

- addEventListener

```
<button id= “btn”>Click me</button>

btn.addEventListener(‘click’, function(){
    alert(this.innerHTML);
})
```

alerts ‘Click me’

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Event object – 1/4

- How to get:

```
element.addEventListener("click", myEvent, false);

function myEvent(event) {
    //event variable will contain Event object
}
```

- Or

```
function myEvent(e) {
    //e variable will contain Event object
}
```

Or any other name could be provided for event object

Event object – 2/4

General properties of event:

- **type** – string with event name (‘click’, ‘focus’, ‘change’ etc.).
- **target** – DOM-element which have triggered an event.
- **currentTarget** – DOM-element which have called event handler.
- **eventPhase** – returns which phase of the event flow is currently being evaluated (1 – capture phase (or intercept phase), 2 – target phase, 3 – bubbling phase (or pop-up phase)).
- **timestamp** – date when event happened.
- **bubbles** – returns whether or not a specific event is a bubbling event.
- **cancelable -** returns whether or not an event can have its default action prevented.
- **defaultPrevented** – defines whether or not the preventDefault() method was called for the event.
- **view** – reference to window object in which event have happened.

Event object – 3/4

Methods of event object:

- **preventDefault()** – discards default browser handler if possible.
- **stopPropagation()** – prevents further propagation of an event during event flow.
- **stopImmediatePropagation()** – prevents other listeners of the same event from being called.

Event object – 4/4

```
divB.onclick = function(e) {
        if (e && e.stopPropagation) { //if stopPropagation method supported
                e.stopPropagation();
        }
        else {
                event.cancelBubble = true; //IE variant
        }
         console.log('Event type = ' + e.type);
}
```

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

attachEvent – IE lte 8

```
element.attachEvent( "on"+eventName, handler);
```

element - The DOM element to listen for the event on.

eventNameWithOn - The name of the event to listen for, prefixed with "on", as if it were an event handler attribute. For example, you would use `"onclick"` to listen for the `click` event.

handler - Callback function to call when the event is triggered on this target. The function will be called with no arguments, and with the `this` reference set to the `window` object.

detachEvent – IE lte 8

```
element.detachEvent( "on"+eventName, handler);
```

element - The DOM element to listen for the event on.

eventNameWithOn - The name of the event to listen for, prefixed with "on", as if it were an event handler attribute. For example, you would use `"onclick"` to listen for the `click` event.

handler - Callback function to call when the event is triggered on this target.

Events handling – IE lte 8

```
<button id="myButton">Button</button>
<button onclick="function3();">Apply an event handler "Button"</button>
<button onclick="function2();">Detach</button>

<script>
    var btn = document.getElementById(‘myButton’);
    function function3() {
            btn.attachEvent("onclick", function1);
    }
    function function1() {
            document.body.style.backgroundColor = 'red';
    }
    function function2() {
            document.body.style.backgroundColor = 'white';
            btn.detachEvent('onclick', function1);
    }
</script>
```

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Cross browser event handling – 1/4

```
function addEvent (elem, type, handler){
        if (elem.addEventListener){
                elem.addEventListener(type, handler, false);
        } else {
                elem.attachEvent("on"+type, handler);
        }
}
```

```
function removeEvent (elem, type, handler){
        if (elem.removeEventListener){
                elem.removeEventListener(type, handler, false);
        } else {
                elem.detachEvent("on"+type, handler);
        }
}
```

It works good in most cases, but the handler will lack ‘this’ in IE, because attachEvent doesn’t provide ‘this’.

Cross browser event handling – 2/4

Generic way of getting the event object:

```
element.onclick = function(event) {
        event = event || window.event; // Now event is the event object in all browsers.
}
```

The deepest element which triggered the event is called the **target** or, the originating element.

IE has the **srcElement** property for it, all W3C-compliant browsers use **event.target**.

The cross-browser code is usually like this:

```
var target = event.target || event.srcElement;
```

Cross browser event handling – 3/4

```
event.preventDefault = event.preventDefault || function(){
        this.returnValue = false
};
event.stopPropagation = event.stopPropagaton || function(){
        this.cancelBubble = true
};
```

```
if (!event.relatedTarget && event.fromElement) {
        event.relatedTarget = event.fromElement == event.target
                ? event.toElement
                : event.fromElement;
}
```

Cross browser event handling – 4/4

```
if ( event.pageX == null && event.clientX != null ) {
        var html = document.documentElement, body = document.body;
        event.pageX = event.clientX +
                      (html && html.scrollLeft || body && body.scrollLeft || 0) -
                      (html.clientLeft || 0);
        event.pageY = event.clientY +
                      (html && html.scrollTop || body && body.scrollTop || 0) -
                      (html.clientTop || 0);
}
```

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Mouse events 1/5

- **click**

```
var button = document.querySelector("#myButton");
button.addEventListener("click", doSomething, false);

function doSomething(e) {
    console.log("Mouse clicked on something!");
}
```

the **click** event is fired when you use your mouse to press down on an element and then release the press while still over that same element.

- **dblClick**

```
var button = document.querySelector("#myButton");
button.addEventListener("dblclick", doSomething, false);

function doSomething(e) {
    console.log("Mouse clicked on something...twice!");
}
```

The **dblclick** event is fired when you basically quickly repeat a click action a double number of times

Don't Overdo It

If you happen to listen to both the `click` and `dblclick` event on an element, your event handlers will get called three times when you double click. You will get two `click` events to correspond to each time you clicked. After your second click, you will also get a `dblclick` event.

Mouse events 2/5

The classic hover over and hover out scenarios are handled by the appropriately titled **`mouseover`** and **`mouseout`** events respectively:


```
var button = document.querySelector("#myButton");
button.addEventListener("mouseover", hovered, false);
button.addEventListener("mouseout", hoveredOut, false);

function hovered(e) {
    console.log("Hovered!");
}

function hoveredOut(e) {
    console.log("Hovered Away!");
}
```

Mouse events 3/5

There are two more events that pretty much do the exact same thing. These are **`mouseenter`** and **`mouseleave`** events. **They do not bubble**.

This detail only matters if the element you are interested in hovering over or out from has child elements. All four of these events behave identically when there are no child elements at play. If there are child elements at play:

- **mouseover** and **mouseout** will get fired **each time you move the mouse over and around a child element**.
- **mouseenter** and **mouseleave** will get fired **only once**. It doesn't matter how many child elements your mouse moves through.

For 90% of what you will do, mouseover and mouseout will be good enough.

Mouse events 4/5

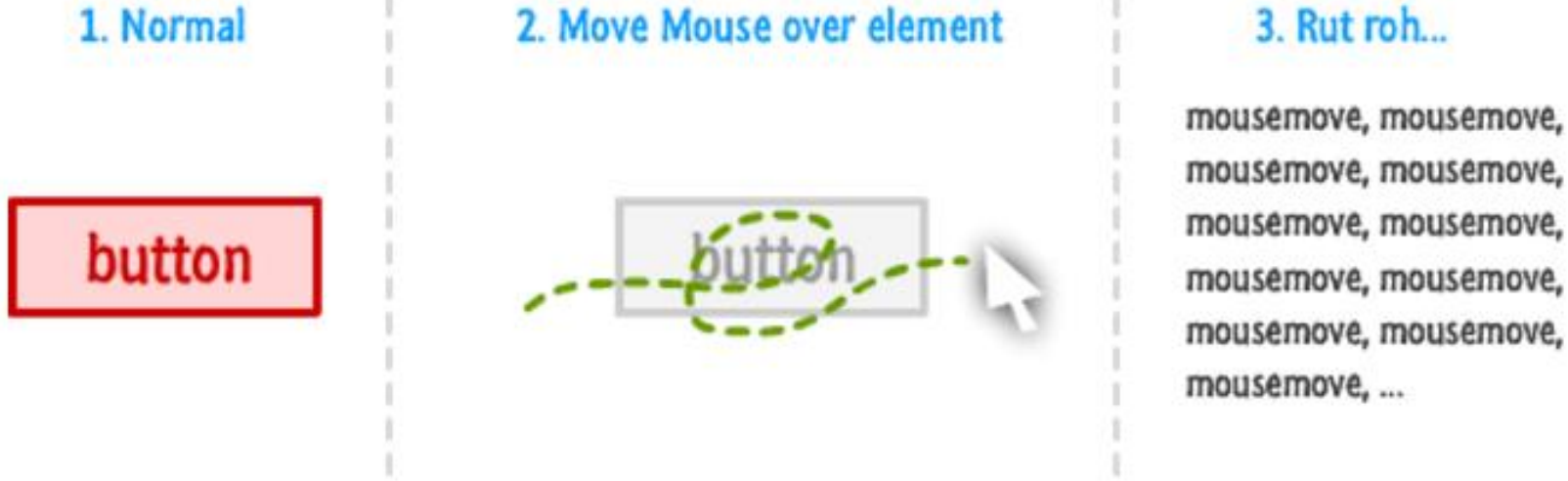
Two events that are almost sub-components of the `click` event are the **`mousedown`** and **`mouseup`** ones.

If the element you pressed down on and released from are the same element, the `click` event will also fire.

Mouse events 5/5

One of the most chatty events that you'll ever encounter is the **`mousemove`** event.

This event fires a whole lotta times as your mouse moves over the element you listening for the **`mousemove`** event on.



Your browser controls the rate at which the `mousemove` event gets fired, and this event gets fired if your mouse moves even a single pixel.

Mouse events summary

There are following simplest mouse events:

`mousedown` - Triggered by an element when a mouse button is pressed down over it

`mouseup` - Triggered by an element when a mouse button is released over it

`mouseover` - Triggered by an element when the mouse comes over it

`mouseout` - Triggered by an element when the mouse goes out of it

`mousemove` - Triggered by an element on every mouse move over it.

Also browser provides the following more complex events for convenience:

`click` - Triggered by a mouse click: mousedown and then mouseup over an element

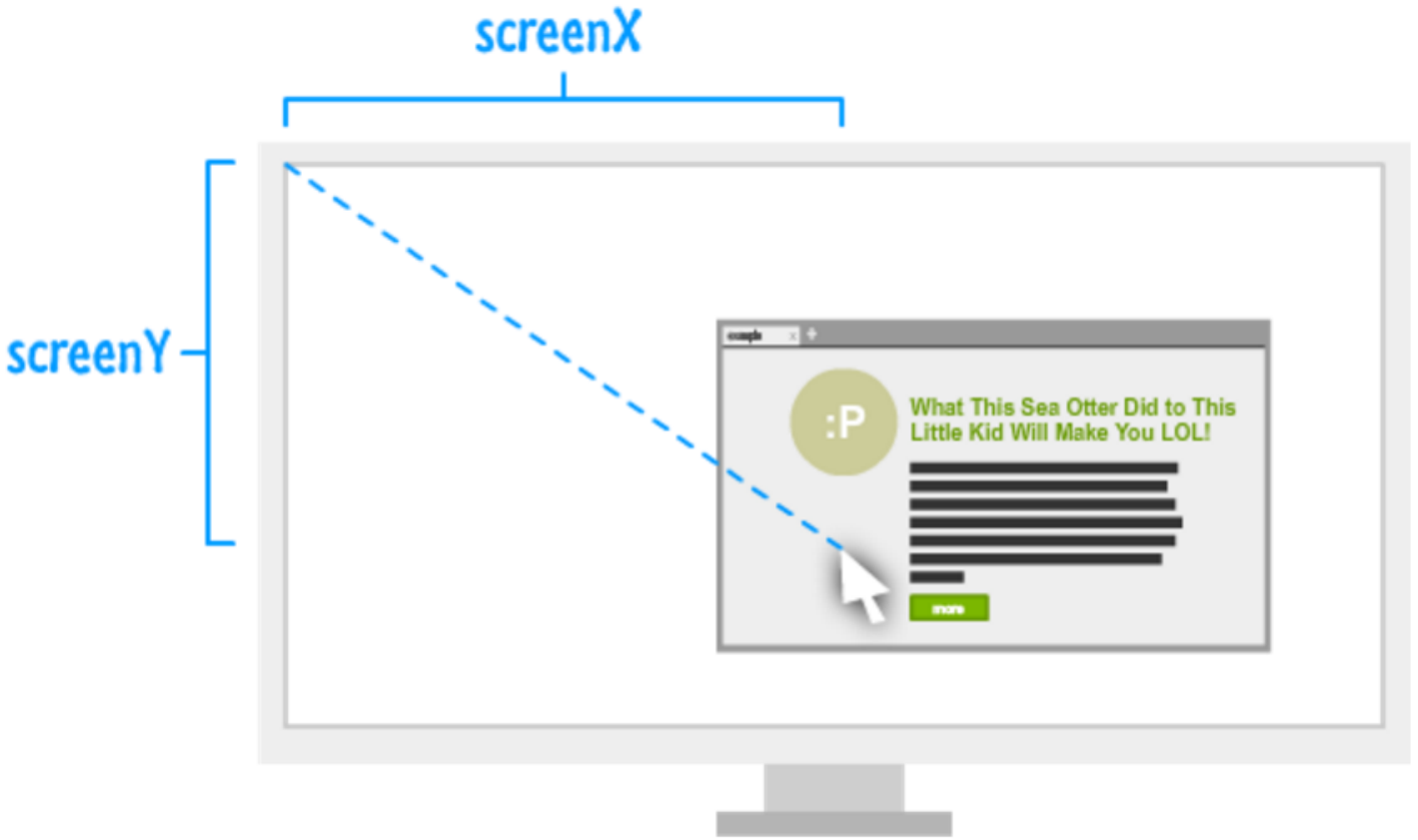
`contextmenu` - Triggered by a right-button mouse click over an element.

`dblclick` - Triggered by two clicks within a short time over an element

There is also a mousewheel event, but it's not used. The `scroll` event is used to track scrolling instead. It occurs on any scroll, including keyboard scroll.

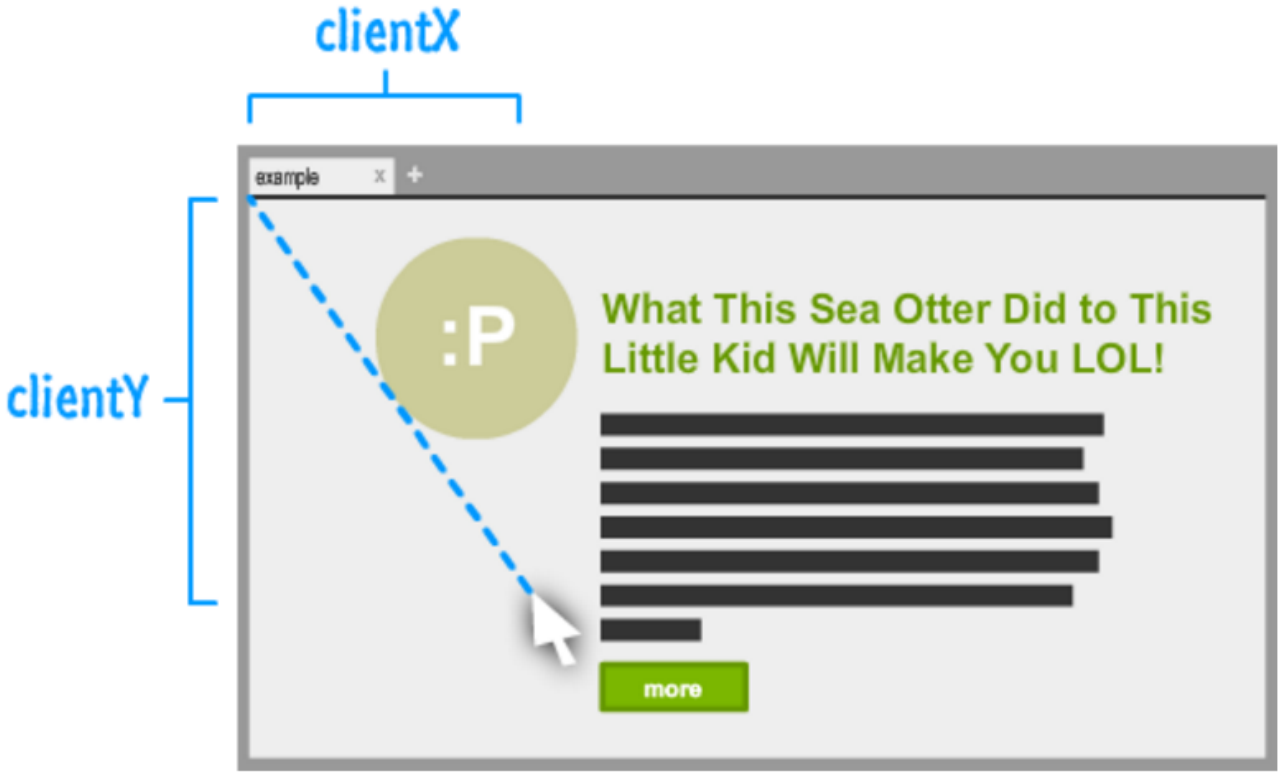
Mouse event properties – 1/3

- **Global mouse position** - The **screenX** and **screenY** properties return the distance your mouse cursor is from the top-left location of your primary monitor



Mouse event properties – 2/3

- **Mouse Position Inside the Browser -**
  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



Mouse events properties – 3/3

- **Detecting Which Button was Clicked –** the **button** property
  - **0** if the left mouse button was pressed
  - **1** if the middle button was pressed
  - **2** if the right mouse button was pressed

```
document.addEventListener("mousedown", buttonPress);

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!!!");
    }
}
```

Cross-browser:

```
function fixWhich(e) {
    if (!e.which && e.button) {
        if (e.button & 1) e.which = 1      // Left
        else if (e.button & 4) e.which = 2 // Middle
        else if (e.button & 2) e.which = 3 // Right
    }
}
```

Mouse event properties summary

altKey, ctrlKey, shiftKey Returns true if the alt/ctrl/shift key was down when the mouse event was fired.

button The button number that was pressed when the mouse event was fired.

buttons The buttons being pressed when the mouse event was fired

metaKey Returns true if the meta key was down when the mouse event was fired.

which The button being pressed when the mouse event was fired.

relatedTarget The secondary target for the event, if there is one.

clientX, clientY The X/Y coordinate of the mouse pointer in local (DOM content) coordinates.

movementX, movementY The X/Y coordinate of the mouse pointer relative to the position of the last mousemove event.

offsetX, offsetY The X/Y coordinate of the mouse pointer relative to the position of the padding edge of the target node.

pageX, pageY The X/Y coordinate of the mouse pointer relative to the whole document.

screenX, screenY The X/Y coordinate of the mouse pointer in global (screen) coordinates.

Keyboard events – 1/4

To work with keyboards in a HTML document, there are **three events** that you will need to familiarize yourself with. Those events are:

- **keydown -** is fired when you press down on a key on your keyboard
- **keypress -** is fired only when you press down on a key that displays a character (letter, number, etc.).
- **keyup -** is fired when you release a key that you just pressed

Keyboard events – 2/4

```
window.addEventListener("keydown", dealWithKeyboard, false);
window.addEventListener("keypress", dealWithKeyboard, false);
window.addEventListener("keyup", dealWithKeyboard, false);

function dealWithKeyboard(e) {
    // gets called when any of the keyboard events are overheard
}
```

Keyboard events – 3/4

Properties of event object:

- **keyCode** - Every key you press on your keyboard has a number associated with it. This read-only property returns that number.
- **charCode** - This property only exists on event arguments returned by the keypressevent, and it contains the ASCII code for whatever character key you pressed.
- **ctrlKey**, **altKey**, **shiftKey** - These three properties return a **true** if the Ctrl key, Alt key, or Shift key are pressed.
- **metaKey** - The metaKey property is similar to the ctrlKey, altKey, and shiftKey properties in that it returns a **true** if the Meta key is pressed. The Meta key is the Windows key on Windows keyboards and the Command key on Apple keyboards.

Keyboard events – 4/4

The following example shows how to use the keyCode property to check if a particular key was pressed:

```
window.addEventListener("keydown", checkKeyPressed, false);

function checkKeyPressed(e) {
    if (e.keyCode == "65") {
        alert("The 'a' key is pressed.");
    }
}
```

Window events – 1/3

- **LOAD**

The load event fires on any resource that has finished loading (including any dependent resources). This could be an image, style sheet, script, video, audio file, **document** or **window**.

```
image.addEventListener('load', function(event) {
        image.classList.add('has-loaded');
});
```

Window events – 2/3

- **ONBEFOREUNLOAD**

window.onbeforeunload enables developers to ask the user to confirm that they want to leave the page. This can be useful in applications that require the user to save changes that would get lost if the browser's tab were to be accidentally closed.

```
window.onbeforeunload = function() {
    if (textarea.value != textarea.defaultValue) {
        return 'Do you want to leave the page and discard changes?';
    }
};
```

Note: assigning an onbeforeunload handler prevents the browser from caching the page, thus making return visits a lot slower. Also, onbeforeunload handlers must be synchronous.

Window events – 3/3

- **RESIZE**

- is super-useful for complex responsive layouts. Achieving a layout with CSS alone is not always possible. Sometimes JavaScript has to help us calculate and set the size of elements. When the window is resized or the device's orientation changes, then we would likely need to readjust these sizes.

```
window.addEventListener('resize', function() {
    // update the layout
});
```

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Prevent default

- **The browser has default behaviors** that will respond when certain events occur in the document (for example, link click will navigate to new page)
- To prevent it there is special method **event.preventDefault()** for W3C-compliant browsers and **event.returnValue = false** for IE<9.

Cross-browser code:

```
element.onclick = function(event) {
        event = event || window.event;
        if (event.preventDefault) {
                event.preventDefault(); // W3C variant
        } else {
                event.returnValue = false; // IE<9 variant
        }
}
```

or

```
element.onclick = function(event) {
        ...  return false;
}
```

Returning false is simpler and used in most cases, but `preventDefault()` approach does not finish the handling, so it also has it's usage.

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Stop propagation

Interrupting the path of the event at any point on its journey (i.e. in the capture or bubbling phase) is possible simply by calling the **stopPropagation** method on the event object – the event will no longer call any listeners on nodes that it travels through on its way to the target and back to the document.

```
child.addEventListener('click', function(event) {
    event.stopPropagation();
});

parent.addEventListener('click', function(event) {
    // If the child element is clicked
    // this callback will not fire
});
```

Stop immediate propagation

Calling **event.stopPropagation()** will **not prevent any additional event listeners** from being called on the current target if multiple listeners for the same event exist. If you wish to prevent any additional listeners from being called on the current node, you can use the more aggressive **event.stopImmediatePropagation()** method.

```
child.addEventListener('click', function(event) {
    event.stopPropagation();
});
child.addEventListener('click', function(event) {
    // If the child element is clicked this callback will not fire
});
parent.addEventListener('click', function(event) {
    // If the child element is clicked this callback will not fire
});
```

Events

- Definition and fire process
- Event bubbling and interception
- Assigning handlers – using attributes, properties and special methods
- Removing handlers
- ‘this’ keyword inside handler
- Event object and it’s properties
- Assigning and removing handlers in IE 8 and less
- Cross browser events
- Most common events for mouse, keyboard, window…
- Prevent default browser behavior
- Stopping event propagation (and stopping immediate propagation)
- Event delegation

Event delegation – 1/2

The delegation concept

- If there are many element inside one parent, and you want to handle events on each of them - don’t bind handlers to each element.
- Instead, bind the single handler to their parent, and get the child from event.target.

Event delegation – 2/2

```
<ul id="parent-list">
   <li id="post-1">Item 1</li>
   <li id="post-2">Item 2</li>
   <li id="post-3">Item 3</li>
   <li id="post-4">Item 4</li>
   <li id="post-5">Item 5</li>
   <li id="post-6">Item 6</li>
</ul>
```

```
// Get the element, add a click listener...

var list = document.getElementById("parent-list");

list.addEventListener("click", function(e) {

        // e.target is the clicked element!
        // If it was a list item

        if(e.target && e.target.nodeName == "LI") {
                // List item found!  Output the ID!
                console.log("List item ",
                        e.target.id.replace("post-"),
                        " was clicked!");
        }
});
```