CITY UNIVERSITY
LONDON

Event Handlers bubbling

Aris Markogiannakis
City University - Short Courses
CS3606 : JavaScript 2: Advanced Javascript for Websites and Web

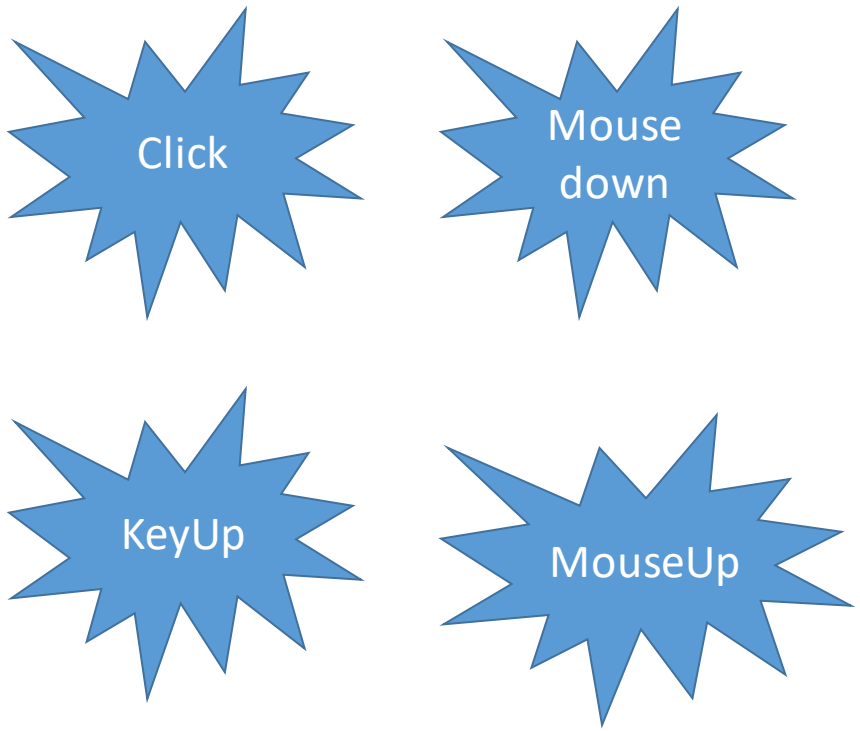
This week

We will learn about:

- DOM Events
- Custom Events

JavaScript an event-driven Language

- JavaScript an event driven language
- You can interact with the browser with some of the following events
- We can write code that is executed when those events are fired



Event Handlers

Event Handlers

- Specific Event
- Attached to a specific element in the DOM

Many handlers can be attached to a specific event/element

Use **addEventListener()** to attach handlers

el.addEventListener(event_type, func_to_run);

Or, use an anonymous function to handle events

el.addEventListener(eventtype. function() { });

Event Handlers

```
// The handler function
function listenerTest() {
  console.log('I am handling an Event!');
}

// The element to attach handler to
var myEl = document.getElementById("headline");

// Adding the event handler to myEl...
myEl.addEventListener("click", listenerTest);
```

```
// The element to attach handler to
var myEl2 = document.getElementById("headline2");
// Adding the event handler to myEl...
myEl2.addEventListener("click", function () {
console.log('I am handling an Event too!');
});
```

Click event information

Depending on the type of event, the event object:

- Contains x and y co-ordinates of the click
- Key pressed of the key up event
- All events contain target and currentTarget properties

```
var myEl = document.getElementById("headline");

myEl.addEventListener("click", function (event) {
// "event.target" is a DOM node...
// ... so DOM methods are available
    console.log(event.target.textContent);
});
```

Events default functionality

- The browser has built-in functionality for some of the events.
- When the click event fires from an anchor <a> the browser opens the page referenced in the href attribute.
- This default behavior occurs after any event handlers attached to the element have completed.
- If we dont want the default behavior to occure we can stop it inside our listener function.
- We can do this with the preventDefault() method.

```
// The element to attach handler to
var myEl2 = document.getElementById("headline2");
// Adding the event handler to myEl...
myEl2.addEventListener("click", function () {
  event.preventDefault();
console.log('I am handling an Event too!');
});
```

Event bubbling

For some event types, when they are triggered:

- They are fired on the element that triggered them: handlers attached to that element run.
- Then the event is fired for each ancestor of the trigger element.
- If any of the ancestor elements have handlers attached to the same event, they will also run.

```
var inner = document.getElementById("headline"),
outer = document.getElementById("content");

inner.addEventListener("click", function (event) {
    console.log('I am the inner element');
});

outer.addEventListener("click", function (event) {
    console.log('I am the outer element');
});
```

```
<div id="content">
  <p id="headline">This is my headline</p>
</div>
```

Event Bubbling

If the user clicks on the “headline” element:

1. First, the event handler attached to the “headline” element will run
2. Then, the handler attached to the “content” element will run

Both handlers will receive an event object, but the object will be slightly different

For every handler in the chain, the event.target property will always be a **reference** to the **element that initially fired the event**.

In the example, this will be the “headline” element

But the object also has a currentTarget property which contains a reference to the element who’s handler is currently being executed event.currentTarget;

```
// Assuming user clicks on the "inner" element
inner.addEventListener("click", function (event) {
    console.log(event.target); // inner
    console.log(event.currentTarget); // inner
});

outer.addEventListener("click", function (event) {
    console.log(event.target); // inner
    console.log(event.currentTarget); // outer
});
```

Event Bubbling

The event object has a method we can use to prevent the event from bubbling up through the DOM

event.stopPropagation();

Any function in the propagation chain can call this method, and it will cause the event to stop bubbling at that point in the DOM

Event Delegation

A pattern which takes advantages of bubbling in order to write more efficient and flexible event handlers.

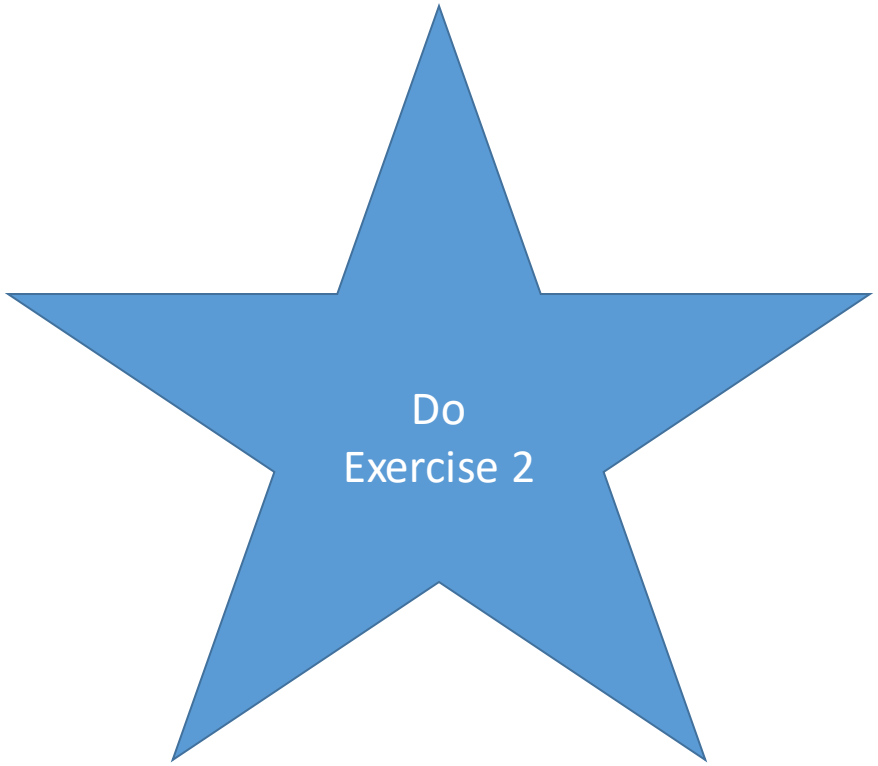
- The event is fired on each ancestor of the element that triggered the event
- Each handler in this chain receives the event object as an argument.
- The event object has a target property which is a reference to the element that triggered the event.
- In other words, at any point in the chain, we can determine which element initially triggered the event

```
var w = document.getElementById('wrapper');
w.addEventListener('click', function (event) {
// Get textContent of clicked element
var cText = event.target.textContent;
    console.log(cText);
});
```

```
<body>
    <div id="content">
        <div id="wrapper">
            <p class="button">One</p>
            <p class="button">Two</p>
        </div>
    </div>
    <script src="demo4.js"></script>
```

Delegation is useful

- We can write less code and declare fewer event handlers
- If we add elements to the DOM dynamically, we do not have to attach new event handlers to them.
- Their events will still bubble up the DOM



But there is a problem….

If you click inside the button-wrapper element, but not on an actual button, the event is still fired!

We need to perform some checks within our handler that stop this from happening.

In other words, we need to implement a **filter**

We can do this by examining the **target element** and ensuring it is one we are interested in before continuing. Depending on our needs, we could:

- Check if the element is a “p” tag (use nodeName)
- Check if the element has the class button (use classList.contains)
- Check if the element has a data-action attribute (use getAttribute)

Delegation Filtering

Task: Implement one of these tests within your event handler from **exercise 2 and then do exercise 3.**

When choosing which method to use, consider that other elements, which we do not want our code to respond to clicks on, may get added to the wrapper