Session 5 exercises

Advanced JavaScript for Web Sites and Web Applications

Exercise 1

Download the workshop 5 files from Moodle.

Extract them and open `exercise1.html` in your browser and editor.

Part A: a bubbling event:

In `exercise1.js`, add event handlers for these elements:

- `#child`
- `#parent`
- `body`

All 3 handlers should listen for the *click* event on the respective element. In each handler, `console.log` a message stating which handler it is. E.g. :

```
console.log('I am the child');
console.log('I am the parent');
console.log('I am the body');
```

View the page and:

- click the *child* element...
- ... then click the *parent* element...
- ... then click the page heading: *Exercise 1*.

What do you see in the console?

Now amend your code so that you also `console.log` the `event.target` element in each handler.

Test your page again. What do you see in the console?

Now amend the code so that the event does not bubble up from the child (with `stopPropagation`).

Test your page again. Notice how clicking on the parent still causes a bubble?

Part B: a non bubbling event:

Add event handlers to `#child` and `#parent` that listen for the `mouseleave` event, which does not *bubble*

In each handler, `console.log` a message stating which handler it is, as you did in part A of the exercise.

View the page and:

- place your mouse over the *child*...

- ... then move your mouse out of *child* so it is over *parent*...
- ... then move your mouse out of *parent*

What do you see in the console?

Exercise 2

Open `exercise2.html` in your browser and editor. You are going to use *event delegation* to react to clicks on the buttons.

In `exercise2.js`, declare an event handler that listens for clicks on the `div` with the `id`: *button-wrapper*.

In the handler function, simply `console.log` the `currentTarget` property of the *event object*

Test the page by clicking the buttons. What do you see in the console?

Clearly, the `currentTarget` property will not help us here as it always points to the button-wrapper!

Amend your code so that you `console.log` the `target` property of the *event object*.

Test your code again. What do you see in the console?

Using `event.target`, we can determine which button was clicked from inside our event handler (which is triggered for all buttons) and perform the necessary action.

Take a look at the button HTML in `exercise2.html`. Notice how each button has a *data-action* attribute? This is a common way of storing custom data with an element in HTML code.

Remember, Javascript can access an element's attributes with `getAttribute`:

```
var attr = element.getAttribute('data-action');
```

Inside your event handler:

- Use `getAttribute` to retrieve the value stored in the *data-action* attribute for the button that was clicked
- `console.log` the value of the *data-action* attribute

Exercise 3

Open `exercise3.html` in your browser and editor.

On the page is a form and a list of elements, each of which has a text description (*Element with some description*, etc.).

The list of elements are contained within a `<ul>` that has the `id`: *element-wrapper*

Write a small program that:

- adds a list item element to `#element-wrapper` when the **Add new element** button is clicked.
  - The description text for the element should be the text from the `#element-description` input field.
- removes a list item element from the DOM when that element is clicked.

Bonus points: use the module or revealing module pattern!

Exercise 4

Using your completed code from exercise 3, you will add some custom events.

1. When an element is added to the group, fire an event
   - When this event fires, you should return the *description* of the element as part of the *detail* object.
2. When an element is removed from the DOM, fire another event

Attach event handlers to listen for both of these events and `console.log` appropriate messages:

- The *add* event: log the element description
- The *remove* event: log a message: "Element Removed"

Note, the event handlers need to be attached to the event **after** the event has been created, but **before** it has been *dispatched*.

If you have time, add a paragraph to the HTML page where the total number of items in the group is displayed:

```
<p>Total: <span id="total-items">X</span></p>
```

When the custom events fire, update the value in the total-items `<span>` element.

Bonus points: add the paragraph dynamically with JavaScript as opposed to manually adding it to the HTML code.