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`

Note, variables holding references to these element have already been defined for you (`docBody`, `parentElement`, `childElement`)

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. :

```
childElement.addEventListener('click', function (event) {
    console.log('I am the child');
});
```

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` and `event.currentTarget` elements 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* element (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`, add an event listener for clicks on the `div` that has the `id`: *button-wrapper*.

In the listener 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

Extra

Review your completed code from exercise 2 in the session 1 exercises. Similar to the code you have just been working with, there is a collection of buttons in the `#menu` container, and each of them has a `data-style` attribute which is used by the event listener code.

See if you can re-implement this exercise, using event delegation.

Exercise 3

Open `exercise3.html` in your browser and editor. You should also open `exercise3.js` in your editor. The code is a variation of the script we wrote in week 1.

The main addition to the script is that a *delete* button has been added to each of the paragraphs that get added to the TODO list.

Your task: Using event delegation, write an event listener that removes a paragraph from the "TODO list" when the corresponding delete button is clicked.

You will have to:

- Add the listener to a common ancestor of each button (i.e. the `resultEl` element)
- Inside the listener function:
  - Make sure the thing that was clicked was actually a delete button. Note, the delete buttons all have the class: **item-delete**, so you can simply check the `classList` of the `event.target` object . E.g.:

    ```
    event.target.classList.contains('item-delete')
    ```

  - If the thing that was clicked is a delete button, you should prevent the default behaviour (they are a elements)
  - If the thing that was clicked is a delete button, you will need to delete its *parent* element (i.e. the *p* element). You can use a combination of `parentNode` and `removeChild` for this. To do this you will need a reference to the *p* element (`event.target.parentNode`) and to the div in which the *p* elements are contained (`resultEl`). E.g.:

    ```
    resultEl.removeChild(event.target.parentNode);
    ```

When done, you should be able to add items to the TODO list, and then remove them by clicking the **X**. Clicking elsewhere on the TODO item or elsewhere in the list container will do nothing.

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 text
- 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.