9. JavaScript events

9.1 Introduction

- Events in JavaScript are what happens on the browser when a user browses/manipulates any website. Your job as a JavaScript developer is to write a script that responds to an event that happens on the browser. This is what makes websites interactive.
- Under this section, we will learn what JavaScript events mean, types of JavaScript events and what happens when an event (mouse movement) fires in JavaScript (such as a keystroke or mouse movement).

9.2 What are JS events and JS event types?

- **JavaScript events, definition**
    - Change in the state of any object is known as event, i.e., event describes the change in the state of the source. HTML/browser events are "changes" that happen to HTML elements when a user or the browser manipulates the website. When the page loads, it is called an event. When the user clicks a button, presses any key or closes/resizes a window, these too are events.
    - Your job as a JavaScript developer is to write a script that responds to an event that happens on the browser. This is what makes websites interactive. When one interacts with a web page (like clicking a button), that interaction is registered as an event on the browser. In response to that event, the script that is bound with that event changes something on the web page.
    - Event in JavaScript are also called HTML or browser events.
    - **How do these events (like clicking button or pressing arrow key) change the HTML elements?**
        - Developers, like yourselves, write JavaScript code that will execute/react in response to events. Meaning, you will write a JavaScript code that will run in response to a user's button clicks. You can write a JavaScript code that will display "password needs to include special characters" when a user submits (onsubmit event) a form.
        - **Note**: One key concept to understand in here is that you, as a web developer, will be responsible to prepare the scripts in response to potential events that could happen on your website.

9.3 Event types

- The browser triggers many events, and our JavaScript code reacts/gets executed in response to these events. Below, please find some of the most common event types and event names: **UI/Window events**: These events occur as the result of any interaction with the browser window rather than the HTML page. The various UI events are as follows.
        - In these events, we attach the event listener to the window object, not the document object
        - **Load event**: The event fires when the browser finishes loading a page

- **Unload event**: This event fires when This event fires before the users leave the page, probably because user requested a new page
- **error event**: This event fires when the browser encounters a JavaScript error or a value that doesn't exist
- **resize event**: This event occurs when a user changes/resizes the current size of the browser window
- **scroll event**: This event fires when the user scrolls up/down on the browser window. It can relate to the entire page or a specific element on the page.

- **Key board events**: These events fire when a user interacts with a keyboard
  - **keydown event**: This event fires when user first presses any key on the keyboard. **Note**: If the user holds down the key, this event fires repeatedly.
  - **keyup event**: This event fires when a user releases any key on the keyboard.
  - **keypress event**: Just like keydown event, this event fires when a user presses any key on the keyboard. However, the pressing of key results in printing a character on the screen. The keypress event will not fire for enter, tab or arrow keys as pressing them will not result in printing characters on the screen, but the keydown event will fire if a user presses these keys.
  - **Note**: The keydown and keypress events fire before a character appears on the screen, the keyup fires after it shows.
- **Mouse events**: These events fire when the mouse moves, or the user clicks a button. All the elements of the page support these events.
  - **Click event**: This event fires when the user presses on and releases the primary mouse button (usually the left button).
    - **Note 1**: This event also fires if the user presses the "Enter" key on the keyboard when an element has focus.
    - **Note 2**: For touchscreen devises, a tap on the screen acts like a single left-click
    - **Note 3**: We can add the above two events to any element, but it's better to apply it only on the items that are usually clicked (like button element)
  - **dbclick event**: This event fires when user quickly presses and releases a button twice
    - **Note**: For touchscreen devises, a double-tap on the screen acts like a double left-click.
    - **Note 2**: We can add the above two events to any element, but it's better to apply it only on the items that are usually clicked (like button element)
  - **mouseover event:** It fires when the user moves the cursor, which was outside an element before, inside the element, in short, when a user moves the mouse over an element.

        - **mouseout event:** This event fires when the user moves the cursor, which was inside an element before, outside the element, in short, when users move the mouse from an element
    - **Form Events**: These events are common while using forms on a webpage. We find the submit event mostly in form of validation (Ex: if a user misses a required information or enters an incorrect input, submit event fires to check the form values before the data is sent/submitted to the server).
        - **change event**: This event occurs when the value of an element has been changed. Example, the change event fires when the checked value in check box, or radio button changes
        - **submit event**: This event fires when a user submits a form. The change occurs on the <form> element when user submits a form using a button or a key.
        - **cut event**: This event fires when a user cuts a content from any HTML element.
            - **Note**: The cut event is mostly used on <input> elements with type="text"
            - **Note 2**: Although the cut event is supported by all HTML elements, it is not possible to cut the content of a <p> element, unless we add the "contenteditable" attribute and give it a value of "true".
        - **paste event**: This event fires when a user pastes content in an element
            - **Note 1**: The **paste** event is mostly used on <input> elements with type="text"
            - **Note 2**: Although the paste event is supported by all HTML elements, it is not possible to cut the content of a <p> element, unless we add the "contenteditable" attribute and give it a value of "true".
        - **copy event**: This event happens when a user copies the content of an element (ex: copying content from a form field)
            - Tip: The copy event is mostly used on <input> elements with type="text".
        - **select event**: This event fires when a user selects some text in an element.
            - **Note**: The onselect event is mostly used on <input type="text"> or <textarea> elements.
    - **Focus and blur events**: These events fire when the HTML elements you can interact with gain/ lose focus. They are most commonly used in forms and especially helpful when you want to do the following tasks:
        - The tips are usually shown in the elements other than the one the user is interacting with.
        - To trigger form validation as a user moves from one control to the next without waiting to submit the form.
        - **Here are some of the focus and blur events**:
            - **Focus/focusin event**: This event fires, when an element gains focus.
            - **Blur/focusout event**: This event fires, when an element loses focus.
- **Visit this website for list of all JavaScript events**:
    - https://www.w3schools.com/tags/ref_eventattributes.asp

9.4 Event handling in HTML

- **Events**: We have said earlier those events are generated as result of user interaction with the HTML elements on the browser (such as pressing keys or clicking a button).
- **Event handling**: is a mechanism that controls these events and decides what should happen when an event happens.
- **Event handlers/listeners**:
    - An event handler is also known as an event listener. An event handler/listener is a specific script (JavaScript code) that gets executed when a particular event (such as clicking of a button) happens on an HTML element.
    - An event handler is a script/function that listens to the event and responds accordingly to the event.
    - **Note 1**: An event listener/handler is a function with a name if is we want to reuse it for various events.
    - **Note 2**: An event listener/handler is an anonymous function if it is going to be used only once.
    - **Note 3**: An event can be handled by one or multiple event handlers. If an event has multiple event handlers, all the event handlers will be executed when the event is fired. For example, if there are multiple scripts that will be executed when a button is clicked, all the scripts/functions will be executed when the button is clicked.
- **Event binding:** The event binding allows you to add an event handler (JavaScript function) for a specified event so that your chosen JavaScript function will be executed when that event is triggered.
- **What happens during event handling?** When a user interacts with any HTML element on a web page, there are three steps involved in triggering a script associated with an event:
    - **1. Selecting an element:** The element is selected to bind it with the event handler (JavaScript function) when an event occurs on it
    - **2. Binding an event**: This is to identify the specific event that will trigger the event handler to execute.
    - **3. Attach a script:** This is to instruct the web browser which specific event handler to execute when a specific event happens
- **Note**: The script that you want to bind with an event needs to be written first

9.5 Ways to bind an event (HTML event handler attributes/inline event handlers)

- **There are three ways to assign/bind event handlers to an event**: HTML event handler attributes/inline event handlers, traditional/DOM Level 0 event handlers and DOM Level 2 event handlers. We will discuss the HTML event handlers attribute in this section and the other 2 will be discussed in the following sections.

- **HTML event handlers attribute**: HTML allows event handler attributes, with JavaScript code, to be added to HTML elements. To handle events using this method, all you need to do is to use an HTML attribute with the name of the event handler function.

```
Syntax:  onclick="changeBackground()"
```

  - **Example 1**: Let's try to show an alert text on browser that says, "button clicked!!" when a user clicks on a button
    - **Our code in the JavaScript file**:

```
function showClickedAlert() {
alert("button clicked!!");
}
```

    - **Attaching the event handler function when there is a click event on our <button> element**:

```
<button onclick="showClickedAlert()">Save</button>
```

  - **Example 2**: Let's try to change the color of our button's text to red when the <button> is clicked
    - **Our code in the JavaScript file**:

```
function changeButtonColor() {
  var myButton = document.getElementById("buttonId");
     myButton.style.color = "red";
}
```

    - **Attaching the event handler function to when there is a click event on our <button> element**:

```
<button id="buttonId"
onclick="changeButtonColor()">Change  Color</button>
```

  - **Note**: This approach (using inline event handlers) is considered a bad practice for the following reasons:

- The event handler code is mixed with the HTML code, which will make the code more difficult to maintain, especially on a larger scale it can get quite messy. Look at this example:

```
<button onclick="changeButtonColor()"
ondblclick="showClickedAlert()"
onfocus="changeBackgroundColor()" >Change Color</button>
```

    - Look how unclean it will be to have all these event handlers in our <button>
- The approach won't allow us bind more than one event handler for an event. Meaning, if we use the inline event handling approach, we cannot assign a text color changer function and a background color changer function for a single <button> click event.
- In inline HTML event handling, the scope of the event handler functions cannot be controlled. This approach requires the event to be global, meaning, the event handler functions need to be globally accessible
- If the element is loaded fully before the JavaScript code, users can start interacting with the element on the webpage which will cause an error. To use the above showClickedAlert() function, if the HTML page is loaded fully and the JavaScript has not been loaded, the showClickedAlert () function is undefined. Meaning, if a user clicks the button at this moment, an error will occur.
- **Priority order**: If the same event, that had an event handler attribute within the HTML, is assigned DOM level event handler in the external JavaScript file, the event handler assigned in the HTML will be replaced by DOM level event handler. Let's assume we have an event handler attribute that changes the text color of a <button> to red when it is clicked. We also have a DOM level event handler that changes the text color of the same <button> to green when it is clicked.

```
var myButton = document.getElementById("buttonId");
//function changing button text to green
function changeTextToGreen() {
myButton.style.color = "green";
}
//function changing button text to red
function changeTextTored() {
myButton.style.color = "red";
}
```

```
// Attribute event handler (for the <button> click
event)
<button id="buttonId" onclick="changeButtonColor()"
ondblclick="showClickedAlert()">Change Color</button>
// DOM level event handler (for the <button> click
event)
myButton.onclick = changeTextToGreen;
```

- The <button> text color will change to green, not red, when clicked as the event handler in the external JavaScript gets priority over the attribute handler

- Here is a list of event attributes (inline event handlers) that can be added to HTML elements:
  - https://www.w3schools.com/tags/ref_eventattributes.asp

9.6 Ways to bind an event (traditional DOM event handlers/ DOM Level 0 event handlers)

- **Traditional DOM event handlers/ DOM Level 0 event handler**: Each element has event handler properties (or onevent properties) such as onclick, ondblclick or onfocus. Under this event handling approach, you just need to set/assign the event handler property to a function (event handler function).
- **Syntax**:
  - element.onevent = functionName;
    - Notice that the event name is preceded by "on"
- **Steps involved under traditional DOM event handling**:
  - First, select the element you want to bind an event with
  - Then bind the element with the event handler
  - Finally attach the event handler function on the event (onevent property)
- **Example 1**: Let's try to change the text color of a <button> to red when a user clicks on the <button>
  - **Method 1: It is fine to make the handler function a separate named function, like below:**
    - **Selecting the <button> we want to bind the event (click) with**

```
let myButton = document.getElementById("buttonId");
```

- **Binding an event handler with the element selected using a named function**

```
var changeToRed = function () {
myButton.style.color = "red";
};
```

- **Attaching the event handler function with the event handler property (onclick)**

```
myButton.onclick = changeToRed;
```

  - **Method 2: We can use anonymous handler function like below**

```
var myButton = document.getElementById("buttonId");
 myButton.onclick = function () {
myButton.style.color = "red";
};
```

- **Note**:
  - Under traditional DOM event handling, we can only attach a single function to an event
  - Under traditional DOM event handling, the event handler can be replaced by assigning another function to the same property.

9.7 Ways to bind an event (DOM level 2 event handlers/listeners)

- **DOM level 2 event handlers/listeners**: Under this event handling approach, we will add/remove the event handler as a listener for an element using the addEventListener() method.
  - This is the most favored way of handling events as the approach allows adding (removing) of multiple event handlers for a single event using the event listener methods.
- **DOM Level 2 Event Handlers provide two main methods to register/deregister event listeners**:
  - **addEventListener():** registers an event handler
  - **removeEventListener():** removes an event handler
- **Syntax**:
  - element.addEventListener(event, function)
  - element. removeEventListener(event, function)
- **addEventListene()**: This is the recommended mechanism for adding event handlers in web pages. Inside the addEventListener() function, we specify two parameters: the name of the event and event handler function we want to run in response to the event.
  - **Example 1**: Let's change the text color of a <button> to red when <button> is clicked

```
var myButton = document.getElementById("buttonId");
myButton.addEventListener("click", function () {
myButton.style.color = "red";
});
```

- **Exmaple 2**: We can use a named function to do the same thing as example 1 above

```
var myButton = document.getElementById("buttonId");
let changeToRed = function () {
myButton.style.color = "red";
};
myButton.addEventListener("click", changeToRed);
```

- **removeEventListener ()**: The removeEventListener() removes an event listener that was added via the addEventListener(). However, we need to pass the same parameters that were passed to the addEventListener()
  - **Exmaple**: Let's remove the above "addEventListener".

```
var myButton = document.getElementById("buttonId");
let changeToRed = function () {
myButton.style.color = "red";
};
myButton.addEventListener("click", changeToRed);
myButton.removeEventListener("click", changeToRed);
```

- **Adding multiple listeners for a single event**: Through DOM Level 2 event model, it is possible for one event to trigger multiple event handler functions. Meaning, a specific object (ex: document.getElementById("elementId")), with a specific event ((ex: click or keypress) can be assigned with any number of event-handler functions.
  - Let's add different event handler functions and change the text color of button to red, change the button's background to yellow and alert a message that says, "button clicked!!!"

```
var myButton = document.getElementById("buttonId");
myButton.addEventListener("click", function () {
myButton.style.color = "red";
});
myButton.addEventListener("click", function () {
alert("button clicked!!!");
});
myButton.addEventListener("click", function () {
myButton.style.backgroundColor = "yellow";
});
```

9.8 Halting default behaviors

- There are scenarios where you want the default HTML behavior not to take effect on the browser. For example, you don't want the submit button to submit the form when required filled are not filled out. Let's see the preventDefault() method and try to halt default HTML behaviors.
- **The preventDefault() method**: It tells the user agent that if the event does not get explicitly handled, its default action should not be taken as it normally would be.
    - **Syntax**: event.preventDefault();
    - **Example 1**: A checkbox has a default behavior of toggling a checkbox (showing the check mark and hiding the check mark) upon clicking the <input> element. Using preventDefault() method, let's prevent the checkbox toggling and instead change the background color of the HTML <body> to yellow.

```
var myCheckbox = document.querySelector("#id-checkbox");
var myeventHandler = function (event) {
document.body.style.backgroundColor = "yellow";
event.preventDefault();
};
myCheckbox.addEventListener("click", myeventHandler);
```

    - **Example 2**: By default, the submit event fires when the user clicks a submit button (<button> or <input type="submit">) or presses Enter while editing a field. Now let's prevents the default action of submitting the form.
        - **Here is your HTML code for your form**

```
<form id="formID">
<button id="buttonId" type="submit">Submit here</button>
</form>
```

        - **Now let's prevent the form from being submitted and change the <button>'s text to "Form submission prevented" on submit event**

```
var myForm = document.getElementById("formID");
var myButton = document.getElementById("buttonId");
function functionToPrevent(event) {
```

```
myButton.innerHTML = "Form submission prevented";
event.preventDefault();
}
myForm.addEventListener("submit", functionToPrevent);
```

9.9 Explaining how to approach the form validation exercise

- Please watch the class demo by pausing and coding along