the shortcut

Fullstack Web Development Tutorial Lesson 12

Today’s lesson will cover

- Events

the shortcut

JavaScript fundamentals

Browser events

- *An event* is a signal that something has happened. All DOM nodes generate such signals (but events are not limited to DOM).
- There are many supported events but below are some of the more commonly used events:
  - **Mouse events:**
    - `click` – when the mouse clicks on an element (touchscreen devices generate it on a tap).
    - `contextmenu` – when the mouse right-clicks on an element.
    - `mouseover` / `mouseout` – when the mouse cursor comes over / leaves an element.
    - `mousedown` / `mouseup` – when the mouse button is pressed / released over an element.
    - `mousemove` – when the mouse is moved.
  - **Keyboard events:**
    - `keydown` and `keyup` – when a keyboard key is pressed and released
  - **Form element events:**
    - `submit` – when the visitor submits a `<form>`.
    - `focus` – when the visitor focuses on an element, e.g. on an `<input>`.
  - **Document events:**
    - `DOMContentLoaded` – when the HTML is loaded and processed, DOM is fully built.
  - **CSS events:**
    - `transitionend` – when a CSS-animation finishes.

Event handlers

- To react on events we can assign a *handler* – a function that runs in case of an event.
- There are several ways to assign handlers
  - HTML-attribute: A handler can be set in HTML with an attribute named `on<event>`.
  - DOM property: We can assign a handler using a DOM property `on<event>`
- Accessing the element: this
  - The value of `this` inside a handler is the element. The one which has the handler on it.
- Possible mistakes
  - We can set an existing function as a handler ut be careful: the function should be assigned as `functionName`, not `functionName()`
  - On the other hand, in the markup we do need the parentheses

addEventListener

- The alternative way of managing handlers using special methods `addEventListener` and `removeEventListener` allows assigning multiple handlers to one event
- The syntax to add a handler:
  - `element.addEventListener(event, handler, [options]);`
  - **`Event`**: Event name, e.g. `"click"`.
  - **`handler`**: The handler function.
- To remove the handler, use `removeEventListener`:
  - `element.removeEventListener(event, handler, [options]);`
- Object handlers - handleEvent: We can assign not just a function, but an object as an event handler using `addEventListener`. When an event occurs, its `handleEvent` method is called

Exercise

- Create an HTML document with a button and a div with `id=”text”`
- Add a script on the button which on click will hide the `<div id=”text”>`

Exercise

- Get the Exercise files for your todos document and relevant stylesheet: https://github.com/itistheshortcut/fullstack-june-2020/tree/master/lesson%2012
- Write a script that will add a button with the `class=”remove-button”` to remove the `<div id=”pane”>` and value of `[X]`

Bubbling and Capturing

- Bubbling principle is simple: **When an event happens on an element, it first runs the handlers on it, then on its parent, <u>then all the way up on other ancestors</u>.**

- The process is called “bubbling”, because events “bubble” from the inner element up through parents like a bubble in the water.

- event.target: Note the differences from `this` (=`event.currentTarget`):
  - `event.target` – is the “target” element that initiated the event, it doesn’t change through the bubbling process.
  - `this` – is the “current” element, the one that has a currently running handler on it.

- Stopping bubbling: The method for it is `event.stopPropagation()`.
  - Don’t stop bubbling without a need! It is convenient in most cases.

- There’s another phase of event processing called “capturing”. It is rarely used in real code, but sometimes can be useful. The standard DOM Events describes 3 phases of event propagation:
  - Capturing phase – the event goes down to the element.
  - Target phase – the event reached the target element.
  - Bubbling phase – the event bubbles up from the element.

Browser default actions

- Many events automatically lead to certain actions performed by the browser. For instance:
  - `mousedown` – starts the selection (move the mouse to select).
  - `click` on `<input type="checkbox">` – checks/unchecks the `input`.
  - `submit` – clicking an `<input type="submit">` or hitting `Enter` inside a form field causes this event to happen, and the browser submits the form after it.
  - `keydown` – pressing a key may lead to adding a character into a field, or other actions.
  - `contextmenu` – the event happens on a right-click, the action is to show the browser context menu.
- Preventing browser actions: There are two ways to tell the browser we don't want it to act:
  - The main way is to use the `event` object. There's a method `event.preventDefault()`.
  - If the handler is assigned using `on<event>` (not by `addEventListener`), then returning `false` also works the same.

Dispatching custom events

- We can generate not only completely new events, that we invent for our own purposes, but also built-in ones, such as `click`, `mousedown` etc. That may be helpful for automated testing.
- Event constructor: Build-in event classes form a hierarchy, similar to DOM element classes. The root is the built-in Event class. We can create `Event` objects like this:
  - `let event = new Event(type[, options]);`
- Arguments:
  - *type* – event type, a string like `"click"` or our own like `"my-event"`.
  - *options* – the object with two optional properties:
    - `bubbles: true/false` – if `true`, then the event bubbles.
    - `cancelable: true/false` – if `true`, then the “default action” may be prevented. Later we’ll see what it means for custom events.
  - By default both are false: `{bubbles: false, cancelable: false}`.
- After an event object is created, we should “run” it on an element using the call `elem.dispatchEvent(event)`.
- For our own, completely new events types like `"hello"` we should use `new CustomEvent`. Technically CustomEvent is the same as `Event`, with one exception.

Exercise

- Write the script which adds an event listener to the button which will run `makeMadLib` function when clicked
- The function will retrieve values from the form input elements in the given HTML document and make a story from them
- The output will be shown on `<div id=”story”>` and will look like for example “Pete codes crazily with Javascript which is cool!”

```
 <body>
 <h1>Mad Libs</h1>
 <ul>
  <li>Programming language: <input type="text" id="noun">
  <li>Adjective: <input type="text" id="adjective">
  <li>Someone's Name: <input type="text" id="person">
 </ul>
 <button id="lib-button">Lib it!</button>
 <div id="story"></div>
 <script>
// Your code goes here
</script>
 </body>
```

Summary: Intro to browser events

- There are 3 ways to assign event handlers:
  - HTML attribute: `onclick="..."`.
  - DOM property: `elem.onclick = function`.
  - Methods: `elem.addEventListener(event, handler[, phase])` to add, `removeEventListener` to remove.
- HTML attributes are used sparingly, because JavaScript in the middle of an HTML tag looks a little bit odd and alien. Also can't write lots of code in there
- DOM properties are ok to use, but we can't assign more than one handler of the particular event. In many cases that limitation is not pressing.
- The last way is the most flexible, but it is also the longest to write. There are few events that only work with it, for instance `transitionend` and `DOMContentLoaded` (to be covered). Also `addEventListener` supports objects as event handlers. In that case the method `handleEvent` is called in case of the event.
- No matter how you assign the handler – it gets an event object as the first argument. That object contains the details about what's happened.

the shortcut

Self Study Assignments

To Dos

- Continue freecodecamp Javascript. Ideally finish before we resume after summer.
- Continue with FCC HTML, CSS lessons. Ideally finish all the lessons by end of this month.
- If you need help pushing your HTML CSS project on GIthub and using Github pages let me know right away.