Browser Events

What kind of events happen in the browser while our website is running?

IT TALENTS
Training Camp

What is Event?

- HTML events are "**things**" that happen to HTML elements.
- An HTML event can be something the browser does, or something a user does.
- Using JavaScript we can "**react**" on these events.
- JavaScript lets you execute code when events are detected.

Why We Need Events?

- To provide user interaction with the HTML page – capture and validate user input, detect mouse movement, keyboard events, etc.
- To detect and handle changes in the HTML page – when the page is loaded, page is closing, window is resized etc.

HTML Event Types

- Mouse events
- Keyboard events
- Drag & Drop events
- HTML Form events
- User interface events
- Mutation events
- Touch events
- Pointer events
- Refference: https://developer.mozilla.org/en-US/docs/Web/Events

Examples of HTML events

- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user presses a key[1]

How to handle HTML Events

- Event handlers are functions attached to the page elements and called when event occurs.
- They provide interface for accessing the event objects, the element to which they are attached and the element which triggered the event.
- You can handle HTML Events using one of the following approaches:
  - Event Listener as HTML attribute
  - DOM Level 1 handler as property on the DOM object
  - DOM Level 2 handler using addEventListener

Using HTML attribute

- The earliest method of registering event handlers found on the Web involved **event handler HTML attributes**
- You can attach even handlers on elements as html attributes. The format for the attribute name is “on” + event name, for example:

```
<button onclick=“sayHi()">Press me</button>
```

- The attribute value is literally the JavaScript code you want to run when the event occurs. The above example invokes a function defined inside a <script> element on the same page

```
<button onclick="alert('Hello, Pesho!');">Press me</button>
```

DOM Level 1 handler

- You can attach event handlers on elements as properties of the DOM objects.

```
<a href="http://www.ittalents.bg " id="myLink ">IT Talents</a>

<script type="text/javascript">
        document.getElementById('myLink').onclick = function() {
                return confirm('Are you sure');
        }
</script>
```

DOM Level 2 handler – addEventListener

- Using the addEventListener() function on the element you can attach an event handler to the specified element.
- Syntax:

```
element.addEventListener(event, function, useCapture);
```

- Example:

```
element.addEventListener("click",
    function() {
        alert("Hello World!");
});
```

Advantages of addEventListener()

- The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.
- You can add many event handlers to one element.
- You can add many event handlers of the same type to one element, i.e two "click" events.
- You can add event listeners to any DOM object not only HTML elements. i.e the window object.
- The addEventListener() method makes it easier to control how the event reacts to bubbling.
- When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.
- You can easily remove an event listener by using the removeEventListener() method.

Removing DOM Level 2 Handlers

```
element.removeEventListener(event, function, useCapture)
```

Anonymous handlers cannot be removed!

```
// this won't work
element.addEventListener('click', function(){ /* code*/});
element.removeEventListener('click', function(){ /* code*/});
```

```
// this will work
const someFn = function(){/* do something*/};
element.addEventListener('click', someFn, false);
element.removeEventListener('click', someFn, false);
```

Which of these should I use?

- From the three mechanisms, you shouldn't use the HTML event handler attributes — these are outdated, and bad practice.
- Using DOM Level 1 handlers is simple and easy to use. However they are way less powerful compared to the level 2 – ***addEventListener();***
- Main benefits of the ***addEventListener()*** are:
  - The event handler can be easily removed if its not used anymore.
  - You can have multiple listeners of the same type

The Event Object

- The event object contains very important information about the event itself:
  - Type: The type of the event(keydown, mouseover, click, etc)
  - keycode: The code of the key that was pressed
  - Target: The element triggered the event
  - ctrlKey, altKey, shiftKey: Indicates wherher some of this keys is pressed when event occurred
  - x, y: position of the mouse when the event happened

- The event object is passed as parameter to the event handler:

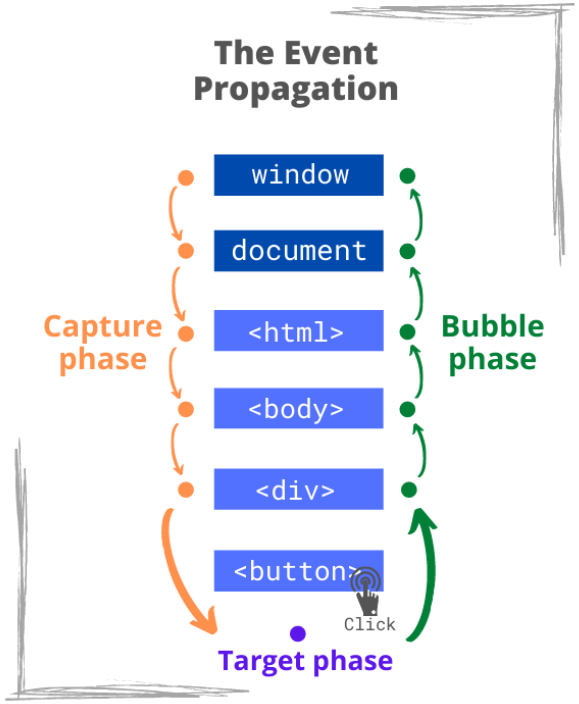
```
element.addEventListener("click", function(event){
    console.log(event);
});
```

DEMO

IT TALENTS
Training Camp

Event Propagation

- Event propagation is a mechanism that defines how events propagate or travel through the DOM tree to arrive at its target and what happens to it afterward.
- In modern browser event propagation proceeds in two phases: **capturing**, and **bubbling** phase.



IT TALENTS Training Camp

DEMO

IT TALENTS
Training Camp

Bubbling vs Capturing phase

- Using the 3rd argument of addEventListener you can choose whether the event should be executed in the capturing or in the bubbling phase.
  - If not specified, the 3rd parameter: useCapture defaults to false.

- Resource: https://www.tutorialrepublic.com/javascript-tutorial/javascript-event-propagation.php

Prevent event propagation

- We stop the event propagation using event.stopPropagation() method.
- It prevents further propagation of the current event in the capturing and bubbling phases.

```
<div id="myDiv">
  <button id="myButton">IT Talents</a>
</div>
<script>
    function onClick(e) {
        e.stopPropagation();
        console.log('first event');
    }
    const myButton = document.getElementById("myButton");
    const myDiv = document.getElementById("myDiv");
    myButton.addEventListener('click', onClick);
    myDiv.addEventListener('click', () => {
        console.log('second event') });
</script>
```

Preventing the default behavior

- We can prevent events from default behavior by calling their event.preventDefault() method.
- This can be used for preventing link to be opened, or form submission.

```
<a href="http://www.ittalents.bg" id="myLink">IT Talents</a>
<script>
   function doConfirm(e) {
       alert('Default prevented!');
       e.preventDefault();
    }
   const link = document.getElementById('myLink');
   link.addEventListener('click', doConfirm);
</script>
```

Preventing next handlers

- We stop the event propagation using event.stopPropagation() method.
- It prevents further propagation of the current event in the capturing and bubbling phases.

```
<a href="http://www.ittalents.bg" id="myLink">IT Talents</a>
<script>
    function doConfirm(e) {
        alert('No next handlers');
        e.stopImmediatePropagation();
        return false;
    }
    function doNextConfirm() {
        // this won't be executed
        return confirm('Are you really sure?');
    }
    const link = document.getElementById('myLink');
    link.addEventListener('click', doConfirm, false);
    link.addEventListener('click', doNextConfirm, false);
</script>
```