CAPTURING EVENTS WITH JAVASCRIPT

HTML events are **"things"** that happen to HTML elements. When JavaScript is used in HTML pages, JavaScript can **"react"** on these events. An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

When the user interacts with the HTML on a web page, there are three steps involved in getting it to trigger some JS code. Together, there is three main steps how JS handle events

Ways to bind an event to an element

HTML event handler attributes

Early versions of HTML included a set of attributes that could respond to events on the element they were added to. This approach is now considered bad practice; however, you need to be aware of it because you may see it at older code.

Inline events

Inline events happen when we place the event inline in the open tag of the element in the HTML code. It is not recommended but it exist and it is possible

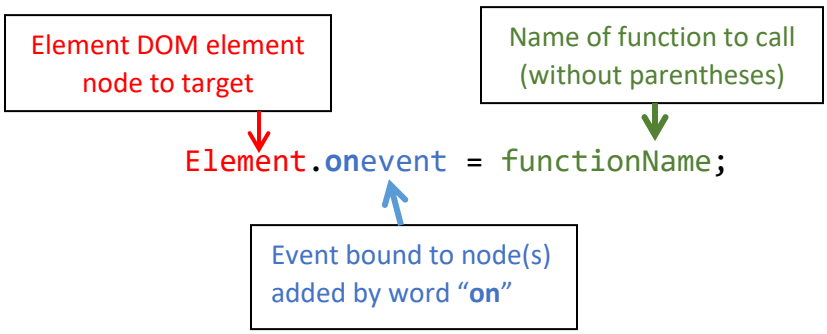
Example 1) apply inline event to a button that when it is clicked it opens an alert window.

```
<button onclick="alert('You clicked!')">CLICK HERE</button>
```

Traditional DOM event handlers

All modern browser understand this way of creating an event handler, but you can only attach one function to each event handler.

Syntax:



Example 2) use even handler to a button that displays an alert message when is clicked.

```
<button>PRESS ME!</button>

<script type="text/javascript">
    let btn = document.querySelector('button');
    btn.onclick = function(){ alert("Hi There!");}
      // anonymous function- good for only one run
</script>
```

```
//named function: use for multiple runs
 function clickedBtn(){
      alert("Hi There!");
    }
    btn.onclick = touch;
```

The most common HTML event handlers are:

Event handler	Description
`onchange`	An HTML element has been changed
`onclick`	The user clicks an HTML element
`onmouseover`	The user moves the mouse over an HTML element
`onmouseout`	The user moves the mouse away from an HTML element
`onkeydown`	The user pushes a keyboard key
`onload`	The browser has finished loading the page

Example 3) create a link that when a mouse goes over it or when it is clicked, a message will display in the console.

```
<a href="https://www.qcc.cuny.edu/" target="_blank" id='qccLink'>QCC</a>

<script type="text/javascript">
    const aLink = document.querySelector('#qccLink');
    aLink.onclick = function(){
    console.log('I WAS CLICKED!')
    }
    function testing(){
    console.log('HOVER!')
    }
    aLink.onmouseover = testing;
</script>
```

We can also add an event directly to an element

Example 4) create an event to `<h1 class="title">` that when a mouse leaves the element, it will display a message in the console.

```
<h1 class="title">Events</h1>

<script>
  document.querySelector('.title').onmouseout = function(){alert('MOUSEOUT!')}
```

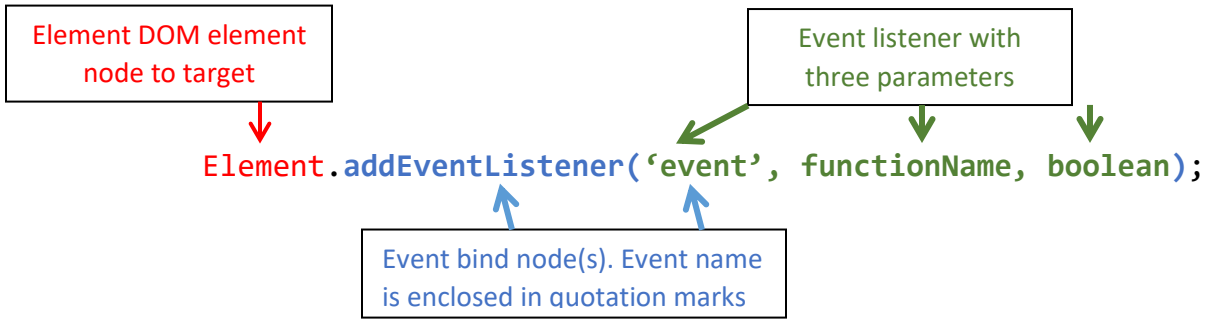
You can also use arrow function

```
  document.querySelector('.title').onmouseout = () =>{alert('MOUSEOUT!')}
```

Event listeners

Event listeners are a more recent approach to handling events. They can deal with more than one function at a time but they are not supported in older browser.

Syntax:



The **`addEventListener()`** method takes three parameters:

- Event you want to listen for
- The code that you want it to run when the event fires
- A Boolean indicating how events flow. Usually is set to **`false.`**

The browser can trigger many different types of events on the DOM(Document Object Model). The full list of all DOM event types are located here: MDN.

Here are some of the most common event types and event names:

- Mouse Events: click, dblclick, mousedown, mouseup, contextmenu, mouseout, mousewheel, mouseover
- Touch Events: touchstart, touchend, touchmove, touchcancel
- Keyboard Events: keydown, keyup, keypress
- Form Events: focus, blur, change, submit
- Window Events: resize, scroll, load, unload, hashchange

Touch events are triggered on touch-enabled devices such as smartphones, tablets, and touch-screen laptops. Mouse events are triggered on the majority of all browsers and devices. The MouseEvent interface represents events that occur due to the user interacting with a pointing device.

The click event: The onclick event occurs when the user clicks on an element.
The dblclick event: ondblclick event occurs when the user double-clicks on an element.
Mousedown & Mouseup events: A pointing device button is pressed/released on an element.

Mouseout event: A pointing device is moved off the element that has the listener attached.

Keyboard events(keydown, keyup, keypress):

- Keydown: Any key is pressed.
- keyup: Any key is released.
- keypress: Any key (except shift, Fn, or capslock) is in the pressed position(fired continuously.)

Form Events(focus, blur, change, submit):

- focus: An element that has received focus.
- blur: An element that has lost focus.
- change: Event that is fired for input, select, and textarea elements when an alteration to the element's value is done by the user.
- submit: The submit button is pressed.

Window Events(resize, scroll, load, unload, hashchange):

- resize: This event fires when the document view(window) has been resized.
- scroll: Event fires when the document view or an element has been scrolled.
- load/unload: The load event is fired when the whole page has loaded, including all resources such as css and images. Unload is when the document or child resource is being unloaded.
- hashchange: This event is fired when the identifier of the URL has changed(the part of the URL that begins with a # symbol).

Some other common DOM events that are used are:

- error: A resource has failed to load.
- abort: The loading fo a resource was aborted.
- online: The browser has gained access to the network.
- animationstart: This event fires when a CSS animation has started.

Event listeners are what make Javascript dynamic and fun. It allows the developer to be creative and experiment with their application to make it interesting and expressive. There are all kinds of keyword events that just about allow the user to do anything to interact with the browser. Knowing more event listener types will allow you(the developer) to become a Javascript wizard! [1]

[1] *Add Event Listener DOM Event Types,* retrieved from https://dbchung3.medium.com/add-event-listener-dom-event-types-6c10a844c9d8, 7/19/2020

Mouse event

Mouse events are event that happen with the features of a mouse. Some of the mouse events are:
`click, dblclick, mousedown, mouseup, contextmenu, mouseout, mousewheel, mouseover`

Click event

The **`click`** event occurs when the user clicks on an element. **`dblclick`** event occurs when the user double-clicks on an element.

Example 5) Use `addEventListener()` method that when a button is clicked, it will display an alert window.

```
<button id='btn5'>PRESS ME!</button>

<script type="text/javascript">
    let btn5 = document.querySelector('btn5');
      btn.addEventListener('click', function(){
        alert("Hi There!");}
     );
```

Example 6) create an event that when a button is clicked, it will randomly select a background color a container.

HTML

```
<div class="color">
      <button id='btnColor'>CHANGE COLOR</button><p class="displayRGB"></p>
</div>
```

CSS

```
.color{width: 50%; height: 200px; margin: 2em auto; border: ridge 10px;}
.displayRGB{text-align: center; font-size:1.5em}
#btnColor{margin: 2em 30%; color: gray; width: 40%; padding:1.2em; }
```

JS

```
  const divCon = document.querySelectorAll('.divColor')
  const btnColor = document.querySelector('.btnColor')
  btnColor.addEventListener('click', function(){
      divColor.style.backgroundColor = randomRGB();
  })
  const  randomRGB = function(){
        let randomR = Math.floor(Math.random()*255)
        let randomG = Math.floor(Math.random()*255)
        let randomB = Math.floor(Math.random()*255)
        return `rgb(${randomR}, ${randomG}, ${randomB})`
      }
```

mouseove *and* **mouseout** *event*

The `mouseover` event takes place when the pointer of the mouse comes over an element. On the contrary, the `mouseout` event occurs when it leaves.

Example 7) Use **`addEventListener()`** method to add a **`mouseover`** and **`click`** event to the same button

```
<button id="btn2">Try it</button>
<p class="display">MESSAGE WILL ADD HERE<br></p>

<script>
      let button2 = document.querySelector("#btn2");
      button2.addEventListener("mouseover", hover1);
      button2.addEventListener("click", alert1);

      function hover1() {
        document.querySelector(".display").innerHTML += "Moused over!<br>";
      }

      function alert1() {
        document.querySelector(".display").innerHTML += "Clicked!<br>";
      }
</script>
```

Example 8) Use **`addEventListener()`** method to add two click events to the same button

```
<button id="btn6">CLICK ME!</button>

<script>
    let btn6 = document.querySelector("#btn6");
    btn6.addEventListener("mouseout", click1);
    btn6.addEventListener("dblclick", click2);

    function click1() {
      console.log('BUTTON 6 = mouseout');
    }

    function click2() {
      alert ("It's me again! Double click");
    }
</script>
```

Alternate way to write the script

```
    <script>
    let button1 = document.querySelector("#btn");
    button1.addEventListener("mouseout", function(){
        console.log('BUTTON 6 = mouseout');
    });
    button1.addEventListener("dblclick", function(){
        alert ("It's me again! Double click");
    });
    </script>
```

Example 9) using example 7, create 12 `<div>` elements that when a mouse moves out from a `<div>` its background color changes.

```
<section style="display:inline-block">
    <div class="divColor"></div> <div class="divColor"></div>
    <div class="divColor"></div> <div class="divColor"></div>
    <div class="divColor"></div> <div class="divColor"></div>
    <div class="divColor"></div> <div class="divColor"></div>
  </section>

<script type="text/javascript">
   const  randomRGB = function(){
        let randomR = Math.floor(Math.random()*255)
        let randomG = Math.floor(Math.random()*255)
        let randomB = Math.floor(Math.random()*255)
        return `rgb(${randomR}, ${randomG}, ${randomB})`
      }

    const divColor = document.querySelectorAll('.divColor');
      for(let divEach of divColor){
         divEach.addEventListener('mouseout', function(){
          divEach.style.backgroundColor = randomRGB();
        })
      }
</script>
```

Keyboard events

Keyboard event happens when interacting with keys in the keyword. Keyboard events are:

- **`keydown`**: Any key is pressed.
- **`keyup`**: Any key is released.
- **`keypress`**: Any key (except shift, Fn, or capslock) is in the pressed position(fired continuously.)

Example 10) create a text input that when any key is pressed from the keyboard into the text field, an alert window will pop

```
<input type="text" class="inputTxt">

<script type="text/javascript">
  const inputTxt = document.querySelector('.inputTxt')
  inputTxt.addEventListener('keydown', function(){
        alert('KEYDOWN!')
      })
</script>
```

We can also create a parameter for the event function. The parameter **`e`** is by default an event parameter and it has different properties. Some of those properties are **`key`** and **`code`**. **`key`** returns the key that was pressed and `code` returns the key code. More examples are posted at event object.

Window event

Window events happen when user interact with the browser window. Window events include:

- load: when the browser finish the loading of the page.
- unload: when the visitor leaves the current webpage, the browser unloads it.
- resize: when the visitor resizes the window of the browser.
- scroll: Event fires when the document view or an element has been scrolled.

`Scroll` *event*

When you scroll a document or an element, the scroll events fire. `Scroll` events on the window object to handle the scroll of the whole webpage.

To register a scroll event handler, you call the `addEventListener()` method on the target element, like this:

Syntax:

```
targetElement.addEventListener('scroll', (event) => {
    // handle the scroll event
});
```

or assign an event handler to the `onscroll` property of the target element:

```
targetElement.onscroll = (event) => {
    // handle the scroll event
};
```

Example 11) Show the pixels from the top of the window browser.

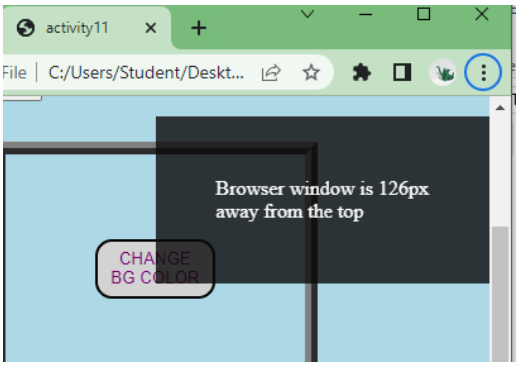
```
<p class="display9"></p>
```

CSS

```
.display9{
  position: fixed; right: 0%; top: 50%; padding: 3em; background-color:
rgba(10,10,10,0.8); width: 50%; color: white;
}
```

```
<script type="text/javascript">
    const display9 = document.querySelector('.display9')
    window.addEventListener('scroll', function(){
    let pxTop = window.pageYOffset;
    display9.innerHTML =`Browser window is ${pxTop}px away from the top `});
</script>
```

When we scroll, it will show the pixes height of the box.



Example 12) Using the window property `window.pageYOffset;` create a link that appears when the window is scrolled 120 px away from window, and disappears when the window is closer to the top by less than 120 px.

HTML code

```
<div class="toTop">
  <a href="#">&#x270C;<br> Top </a>
</div>
```

JS code

```
const toTop = document.querySelector('.toTop')
window.addEventListener('scroll', function(){
  let pxTop = window.pageYOffset;
  if (pxTop>80){ toTop.style.display ='block';}
  else{ toTop.style.display ='none'; }
})
```

CSS

```
html{scroll-behavior: smooth;}
.toTop{
  position: fixed; bottom: 10%; right: 10%;
  width: 60px; height: 60px; border-radius: 50%;
  background-color: rgba(20,20,20,0.9);
  text-align: center;
  display: none;
}
.toTop a{color: white; text-decoration: none; }
.toTop:hover{ box-shadow: 0px 0px 5px 2px black; opacity: 0.8; }
```

Window resizing

The **`resize`** event fires when the document view (window) has been resized. In some earlier browsers it was possible to register resize event handlers on any HTML element. It is still possible to set **`onresize`** attributes or use `addEventListener()` to set a handler on any element. However, resize events are only fired on the window object (i.e. returned by `document.defaultView`). Only handlers registered on the window object will receive resize events.

Example 13) Use the resize event to display the height and width of the window.

HTML code

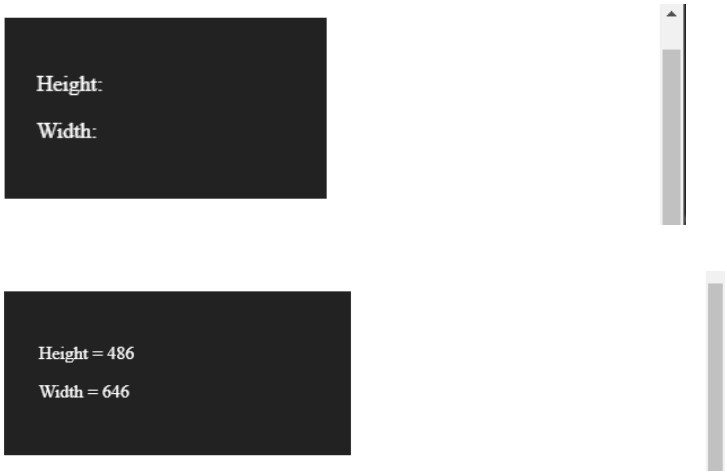
```
<div class="resizeInfo">
  <p class="rHeight">Height: </p>
  <p class="rWidth">Width: </p>
</div>
```

CSS code

```
*{box-sizing:border-box;}
.resizeInfo{
   padding: 5%; width: 50%;
   position: sticky; top: 0px;
   background-color: rgba(10,10,10,0.9); color: white;
}
```

JS code

```
let windowH = document.querySelector('.rHeight');
let windowL = document.querySelector('.rWidth');

window.addEventListener('resize', function(){
  let readH = window.innerHeight;
  let readW = window.innerWidth;
  windowH.innerHTML = `Height = ${readH}`;
  windowL.innerHTML = `Width = ${readW}`;
});
```



The event object

There is an event object that can be passed into event handlers, and sometimes you need to access it specifically. When an event occurs, the event object tells the information about the event and the element it happened upon.

Every time an event fires, the event object contains helpful data about the event, such as:

- Which element the event happened on
- Which key was pressed for a `keypress` event
- What part of the viewport the user clicked for a `click` event.

The event object is passed to any function that is the event handler or listener. If you need to pass arguments to a named function, the event object will first be passed to the anonymous wrapper function, it happens automatically. Then it must specify it as a parameter of the named function.

When the event object is passed into a function, it is often given the parameter name **`e`**, for event. It is a widely used shorthand. However, some programmer use the parameter name **`e`** to refer to error object.

Some of the event objects properties and methods are:

Event object Property	IE5 to IE8 equivalent	Purpose
`target`	`srcElement`	The target of the event (most specific element interacted with)
`type`	`type`	Type of event that was fired
`cancelable`	Not supported	Whether you can cancel the default behavior of an element
Method		
`preventDefault()`	`returnValue`	Cancel default behavior of the event (if it can be canceled)
`stopPropagation()`	`cancelBubble`	Stops the event from bubbling or capturing any further

Example 14) add an event listener to example 8 to print in the alert window the key that was pressed and its key code.

We can also check which key was pressed by given an argument to the anonymous function:

- *which* method shows the ASCII code of the selected key
- *key* method shows the key character that you have pressed
- *code* method returns the key code of the pressed key

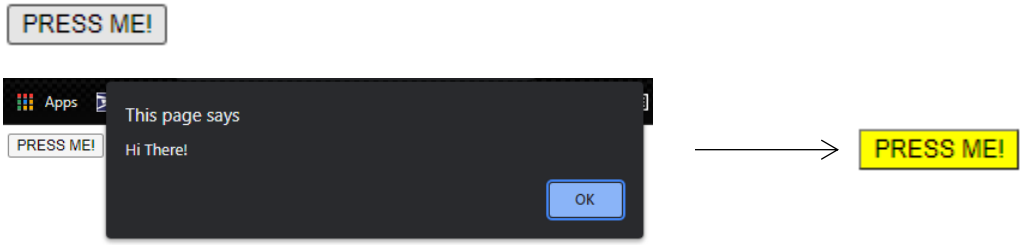
```
<input type="text" class="inputTxt">

const getInput = document.querySelector('.inputTxt')
      getInput.addEventListener('keydown', function(e){
        alert(`KEYDOWN. Key pressed = ${e.key}.\nKey code = ${e.code}`)
      })
```

Example 15) we can also key event to a window

```
window.addEventListener('keypress', function(e){
        console.log(e.key)
        if (e.key==='a'){
          alert(`You pressed ${e.key}`)
        }
      })
```

Example 16) Use event object to change the button's background color and display an alert message when it is clicked

```
<button>PRESS ME!</button>
  <script type="text/javascript">
    var btn = document.querySelector('button');
    btn.addEventListener('click', function(event){
    event.target.style.backgroundColor = "yellow";
    alert('Hi There!');
    });
</script>
```



preventDefault()

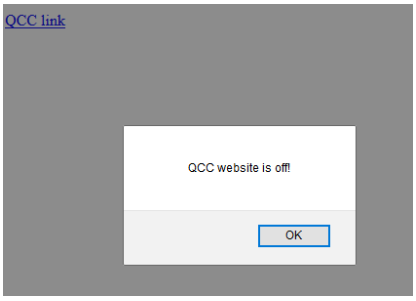
The `preventDefault()` method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

- Clicking on a "Submit" button, prevent it from submitting a form
- Clicking on a link, prevent the link from following the URL

Example 17) Use **`preventDefault()`** method to prevent the link to QCC to open when is clicked

```
<a href=www.qcc.cuny.edu class>QCC link</a>

<script type="text/javascript">
    let link = document.querySelector('a');
    link.addEventListener('click',function(e){
      e.preventDefault();
      alert('QCC website is off!');
    });
</script>
```



If we remove `e.preventDefault();` line from the code, when we click on the QCC link, the alert message will appear and the load www.qcc.cuny.edu website.

Event Bubbling

When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.

Example 18) Use three nesting elements with click event on it. When an element is clicked, it opens an alert window.

```
<section class="section"> SECTION
   <p class="pClick">PARAGRAPH
      <button class="button">CLICK BUTTON</button>
   </p>
</section>

<script>
  const p1 = document.querySelector('.pClick')
    const s1 = document.querySelector('.section')
    const b1 = document.querySelector('.button')
    s1.addEventListener('click', function(){
      alert('Section clicked!')
    })
    p1.addEventListener('click', function(){
      alert('Paragraph clicked!')
    })
    b1.addEventListener('click', function(e){
      e.stopPropagation();
      alert('Button clicked!')
    })
</script>
```

In order to stop bubbling, there is an event method called **`stopPropagation()`** **`stopPropagation()`** will run the current event and prevent propagation to the any parent or sibling events.

Event Delegation

Event delegation is applied when we have a lot element s handled in a similar way, then instead of assigning a handler to each of them – we put a single handler on their common ancestor.

In the handler, we use **`event.target()`** to see where the event actually happened and handle it. [2]

Example 19) use event delegation to remove an item in the list when it is clicked.

```
<form action="/nowhere">
       <label for="item">Enter A Product</label>
       <input type="text" id="product" name="product">
       <label for="item">Enter A Quantity</label>
       <input type="number" id="qty" name="qty">
       <button>Submit</button>
       <ul id='listProduct'></ul>
   </form>
```

READING

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

https://developer.mozilla.org/en-US/docs/Web/Events

[2] *Event Delegation,* JavaScript.info, https://javascript.info/event-delegation, retrieved on July 2022