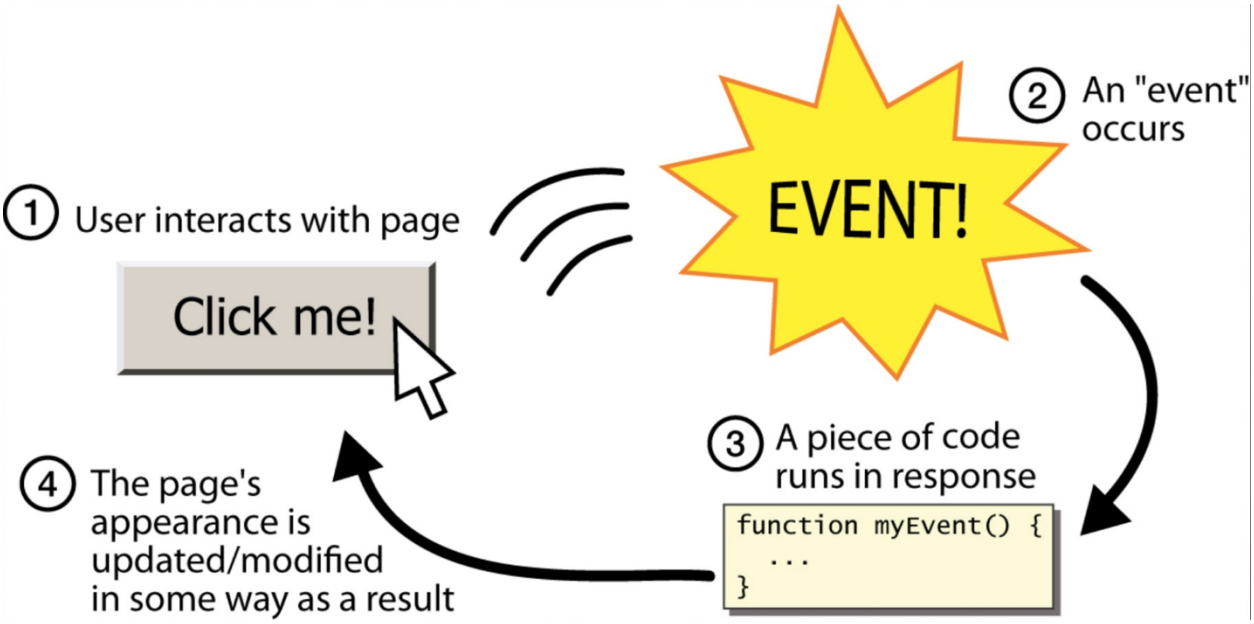
Events

Events are actions or occurrences that trigger JavaScript code to run, making the web page interactive. For example, when the user clicks a button, an event is triggered. JavaScript lets us listen for these events and respond with custom functionality.



Some other examples include events like moving a mouse, pressing a key, or submitting a form. Each event has a keyword that will be used as a parameter to listen for that event. Here is a brief list of some.

Mouse Events
- click
- mouseover
- mouseout

Form Events
- submit
- input

Keyboard Events
- keydown
- keyup

Document Events
- scroll
- DOMContentLoaded

The following is an example of listening for a 'click' event. Once the event is triggered, a function runs that puts a message on the screen that says 'Button was clicked!'.

```html
<button id="myButton">Click Me!</button>
<p id="message"></p>

<script>
  // Get the button element
  const button = document.getElementById('myButton');

  // Add an event listener for the 'click' event
  button.addEventListener('click', function () {
    // Change the text of the paragraph
    document.getElementById('message').textContent = 'Button was clicked!';
  });
</script>
```

Notice the syntax of the addEventListener method.

```
button.addEventListener('click', functioncall)
```

The addEventListener() method expects two parameters, separated by a comma. The first parameter is the event you are listening for. In this case, it's listening for the user to click the button. The second parameter is the function that will run the code—or what we want to happen—in response to the event.

The example shows the second parameter as an anonymous function. The code could have also called a named function instead. The function call showMessage is different from other function calls in that you don't include the empty parentheses after the function name.

```
button.addEventListener('click', showMessage);

function showMessage() {
```

```
        document.getElementById('message').textContent =
    'Button was clicked!';
    }
```

You may run across code that triggers an event within HTML. Such as:

```
<p>Click the button to display the date.</p>
<button onclick="displayDate()">The time is?</button>

<script>
    function displayDate() {
      document.getElementById("demo").innerHTML = Date();
    }
</script>
```

It is not best practice to handle events this way. JavaScript and HTML should be separate. The proper way to handle an event is using the **.addEventListener()** method within JavaScript. Just as we don't like to use inline or embedded CSS, we also don't like to embed our JavaScript inside of HTML. It is harder to follow and maintain our code when it is in separate files like this. Having it in the HTML also limits you to one event per element. With the **.addEventListener()** method, you can have multiple events listened for on one element.

You can also remove the event listener if needed using the **.removeEventListener()** method. You cannot use this if the event listener was added within the HTML.

```
    button.removeEventListener('click', showMessage);
```

Passing parameters with an event function call is a little different since the .addEventListener() method only provides the event object by default. The most common way to pass a parameter is to use an anonymous function or arrow function to call your function. Here is an example using an arrow function where the string 'Alice' is sent as a parameter to the greet function.

```
    const btn = document.getElementById('myBtn');

    function greet(name) {
```

```
  console.log(`Hello, ${name}!`);
}

// Using an arrow function to pass the parameter
btn.addEventListener('click', () => greet('Alice'));
```

So, now we know the basics of how to handle user events to make our webpages more interactive.