# Part 6

## Events

Events are a way of linking HTML to JS code.

Events let you trigger JS when something happens on the page, such as:

* Clicking on a button
* Page load
* Key presses
* Though there are many more

Using an event on the DOM is easy. If you want to call a function just call it like this:

<html>

<head>

<script>

function sayHello() {

alert("Hello World");

}

</script>

</head>

<body>

<input type="button" onclick="sayHello()" value="Say Hello" />

</body>

</html>

In this example the script in part of the HTML. This works the same if you’ve got your JavaScript in a separate file.

Events can also provide context using the **this** property. **this** is the element that emitted the event. Context information can be useful if you have more than one element calling a single function.

<html>

<body>

<p id="output">0</p>

<button onclick="clicked(this)">1</button>

<button onclick="clicked(this)">2</button>

<script src="index.js"></script>

</body>

</html>

// index.js

function clicked(el) {

document.getElementById('output').innerText = el.innerText;

}

### EventListener

Another way to handle events on the DOM with **EventListeners**. These let you bind to DOM events directly in JavaScript.

* Multiple listeners can be used on a single element
* Multiple listeners can be bound to a single event type on an element
  + EG: having two click listeners for a single button

<html>

<body>

<button id="myButton">Test Button</button>

<script src="index.js"></script>

</body>

</html>

// index.js

document.getElementById('myButton').addEventListener(

'click',

(event) => console.log('Button ' + event.srcElement.id)

);

In this example it’s important you have the script below the button! If the script was before the button it’d get executed before the button present on the page. That’d mean when we try to add an event listener to it, we’d get an error.

Just like the previous method, you can provide context to the handler. In the above example ‘event’ is a **MouseEvent**. This contains a lot of extra information on what just happened to trigger the event, such as:

* where the mouse is on the page
* if the user was holding any modifier keys (CTRL, SHIFT, etc.)
* and much more

In this case we’re just getting the element the event is from and logging the ID ('myButton‘).

## Manipulating the DOM

The HTML DOM can be accessed via JavaScript Methods

**All elements are defined as Objects**

This means that we can reference every element on the DOM just like any other object with properties

<html>

<body>

<p id="demo"></p>

<script>

document.getElementById("demo")

.innerHTML = "Hello World!";

</script>

</body>

</html>

This code changes the content of the **<p>** element with the id of **demo** to **Hello World!**

* **getElementById** is a method that returns an Object
* **innerHTML** is a property of this Object.

### Adding an Element

Using **createElement** we can create new HTML elements in JavaScript.

Using **appendChild**, we can append the new element to an existing element on the DOM. By using appendChild you’re providing some context on where on the page you want the element to go.

<html>

<body>

<div id="header-box">

</div>

</body>

</html>

let myH1 = document.createElement('h1');

myH1.innerText = 'Page Header';

myH1.id = 'header-text';

document.getElementByID('header-box').appendChild(myH1);

### Deleting an Element

Using **removeChild** we can remove elements from the DOM.

An element can’t remove itself, but the parent element can remove a child.

<html>

<body>

<div id="header-box">

<h1 id="header-text">Page Header</h1>

</div>

</body>

</html>

document.getElementById('header-box')

.removeChild(document.getElementById('header-text'));

Elements can’t remove themselves directly but they can go the parent element and remove themselves there. <https://stackoverflow.com/questions/3387427/remove-element-by-id>

In the above example we’re using document.getElementById() to get the parent and child elements. If we were to combine this and the previous example, we could use the reference we already had for the h1 to remove it.

So instead of the above we’d have:

document.getElementById('header-box').removeChild(myH1);