JavaScript Events

Springboard

Download Demo Code

Goals

- Explain what an event is in JavaScript
- Add event listeners and prevent default actions using JavaScript
- Access information about the event using a special object
- Add event listeners efficiently
- Explore data attributes

Events

What is an event?

DOM events are "actions" that occur as a result of something the user or the browser does.

We can use JavaScript to execute code when these kinds of "events" happen.

What Kinds Of Events Do We Have?

- clicking on something
- hovering over something with the mouse
- pressing certain keys
- when the DOM has loaded
- when a form is submitted

This Is Really What Most Browser Based JS Is About!

We write code that listens for events - and react accordingly!

This is commonly called Event Driven Programming

What kinds of web pages have you used recently that have JavaScript that listens for events?

So How Do We Do It?

We have three options!

1. Attach the name of the function to the element in HTML
2. Attach the name of the function to an element in JavaScript
3. Use the addEventListener method

Attaching The Name Of The Function

```
<h1 onclick="runClickHandler()"> Hello World </h1>
In HTML

function runClickHandler(){
  console.log("You just clicked the h1 element");
}
```

Adding In Javascript

```
const h1 = document.querySelector("h1");

h1.onclick = function(){
  console.log("You just clicked the h1 element");
}
```

Using addEventListener

```
const h1 = document.querySelector("h1");

h1.addEventListener("click", function(){
  console.log("You just clicked the h1 element");
})
```

Which One Should We Use?

We're going to go with addEventListener - here's why:

- It gives us the most flexibility around our event listeners
- It avoids writing any inline code in our HTML and keeps our HTML and JS seperate
- This is commonly called "Separation of Concerns"

A Gotcha - Waiting For The Dom To Load

If you are trying to access properties in the DOM, before the web page has loaded - it won't work! This becomes an issue if you put <script> tags in the <head> before the DOM has loaded

Thankfully we have an event we can listen to handle this! It's called DOMContentLoaded

```
document.addEventListener("DOMContentLoaded", function(){
  // place your code inside here
})
```

Accessing The Event Object

Inside of the callback to addEventListener, we get access to a special object as a parameter - the event object

```
const h1 = document.querySelector("h1");

// we can call this parameter whatever we want - event is very common
h1.addEventListener("click", function(event){
  console.log(event) // let's take a look!
})
```

What Is Inside Of The Event Object?

- target - what element is the target of the event
- pageX / pageY - where on the page did this event occur?
- key - what key was pressed that triggered this event?
- preventDefault() - a function used to prevent the default behavior of the event.
  - This is **very useful** for stopping form submissions from refreshing the page which is their default behavior

Let's see this in action

```
<form>
  Name: <input id="firstName" type="text">
  <button>Add your name!</button>
</form>

const formElement = document.querySelector("form");

formElement.addEventListener("submit", function(event){
  console.log("You just submitted the form!")
})
```

Unfortunately this will not work! The default behavior of a form is to trigger a refreshing of the page.

Solving our problem

If we want to stop the default behavior of an event, we need to use the special **event.preventDefault()** method.

```
const formElement = document.querySelector("form");

formElement.addEventListener("submit", function(event){
  event.preventDefault();
  console.log("You just submitted the form!");
})
```

Another example

So far we've seen click and submit, events - let's take a look at another one, keyPress!

demo/keypress-demo/index.html

```
<!DOCTYPE html>
<html>
<body>
  <h1>Press the "a" key!</h1>
  <script src="script.js"></script>
</body>
</html>
```

demo/keypress-demo/script.js

```
// listen for the keypress everywhere
document.addEventListener("keypress", function(event) {
  if (event.key === "a") {
    alert("You just pressed the 'a' key!");
  }
});
```

Adding Multiple Event Listeners

It's very common that you will want to add multiple event listeners on elements

Let's see an example:

```
  <body>
    <h1>See your friend list!</h1>
    <ul id="friend-list">
      <li>Michelle <button>Remove</button></li>
      <li>Juan <button>Remove</button></li>
      <li>Emma <button>Remove</button></li>
    </ul>
    <script src="script-list.js"></script>
  </body>

const buttons = document.querySelectorAll("button");

for (let button of buttons) {
  button.addEventListener("click", function(event) {
    event.target.parentElement.remove();
  });
}
```

Everything seems to be working!

What happens when we want to add new elements?

```
  <body>
    <h1>See your friend list!</h1>
    <ul id="friend-list">
    <form>
      <label for="first-name">*</label>
      <input type="text" name="first-name" />
      <button>Add a friend!</button>
    </form>
    <script src="script-form.js"></script>
  </body>

const form = document.querySelector("form");
const friendList = document.querySelector("#friend-list");

const buttons = document.querySelectorAll("li button");

for (let button of buttons) {
  button.addEventListener("click", function(event) {
    event.target.parentElement.remove();
  });
}

form.addEventListener("submit", function(event) {
  event.preventDefault();
  const newFriendInput = document.querySelector("#first-name");
  const newLi = document.createElement("li");
  const newButton = document.createElement("button");
  newLi.innerText = newFriendInput.value;
  newButton.innerText = "Remove";

  newLi.append(newButton);
  friendList.append(newLi);
  form.reset();
});
```

It doesn't work!

The issue here is that our event listener only works for elements **currently** on the page

There are two ways we can fix this

1. Adding the event listener when we create elements
2. Event Delegation

Let's start with adding an event listener when we create

Adding the listener when we create

```
const form = document.querySelector("form");
const friendList = document.querySelector("#friend-list");

form.addEventListener("submit", function(event) {
  event.preventDefault();
  const newFriendInput = document.querySelector("#first-name");
  const newLi = document.createElement("li");
  const newButton = document.createElement("button");
  newLi.innerText = newFriendInput.value;
  newButton.innerText = "Remove";

  newButton.addEventListener("click", function(event) {
    event.target.parentElement.remove();
  });

  newLi.append(newButton);
  friendList.append(newLi);
  form.reset();
});
```

This will work, but it's not the most efficient approach

We're adding an event listener for every single button inside of each

This means if we had 1,000,000 friends, we'd have 1,000,000 listeners!

We can fix this using event delegation

Event delegation

The idea behind event delegation is that we make a parent element the "delegate"

In our case, the parent element is the <ul> element

We attach a single event listener on the parent or delegate element and then if the event happens inside a certain child element, we can access that child element using event.target

How to do better using event delegation

```
friendList.addEventListener("click", function(event) {
  if (event.target.tagName === "BUTTON") {
    event.target.parentElement.remove();
  }
});
```

Exact same behavior with only one event listener!

Event Bubbling And Capturing

The process of an event moving from the place it is clicked to its target is called capturing

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.

This is called bubbling.

Removing Event Listeners

Sometimes after you add an event listener, you will want to remove it.

Some examples of this include:

- A game is over and no more events should be registered
- You can no longer drag and drop something into a part of the page
- You do not want the user submitting any more information

How it works

We can use the removeEventListener function to remove any event listeners

This **can not be** called on multiple elements

```
let buttons = document.getElementsByTagName("button");

buttons.removeEventListener("click", function(){
  alert("You just clicked a button");
});
```

The code above will not work!

Maybe we can just add it to an element individually like this:

```
for(let button of buttons){
  button.removeEventListener("click", function(){
    alert("You just clicked on a button!");
  });
}
```

But this won't work either!

Removing Event Listeners Correctly

removeEventListener needs a reference to the name of the function to remove

```
function alertData(){
  alert("You just clicked a button");
}

for(let button of buttons){
  button.removeEventListener("click", alertData);
}
```

annonymous functions will not work here!

Data attributes

When creating elements and HTML pages, it's very common that you might want to add some metadata or additional information about elements

These are not things that the user should see, but accessible information in CSS and JavaScript

Instead of placing this in an **id** or **class**, we can create custom attributes called data attributes

These attributes start with data- and then anything you would like. You can read them easily in CSS and JavaScript

An example

demo/data-attributes/index.html

```
<!DOCTYPE html>
<html lang="en">
<body>
  <ul id="cars">
    <li data-model="model 3" data-year="2016">Tesla</li>
    <li data-model="f-150" data-year="2017">Vendor</li>
    <li data-model="focus" data-year="2011">Ford</li>
    <li data-model="bronco" data-year="2013">Toyotaa</li>
  </ul>
  <script src="script.js"></script>
</body>
</html>
```

demo/data-attributes/script.js

```
const ul = document.querySelector("ul");

ul.addEventListener("click", function(event) {
  const selectedElement = event.target;
  console.log("Here are all data attributes", selectedElement.dataset);
  console.log(
    "Use one data attribute",
    selectedElement.getAttribute("data-model")
  );
});
```

Recap

- We can add event listeners using addEventListener and remove them using removeEventListener
- Using the event object, we can gather useful information about the target, tagName and much more
- To add element metadata, we can use data attributes and read them using getAttribute or dataset