Lesson | Wednesday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Forms, Hiding and Showing Elements, and the Event Object

Text | Cheat sheet

So far, the only way we've been able to capture user input is by using `window.confirm()` and `window.prompt()`. Let's learn about forms so that we can build more interesting pages. In this lesson, we'll learn about:

- HTML forms
- Hiding and showing HTML elements.
- Printing text to the DOM.
- The `event` object for 'submit' events and preventing default behavior

You are welcome to code along with this lesson or just read it. In the next practice prompt, we'll ask you to recreate the project in this lesson, so you'll be creating this project either way!

Exploring HTML Forms with Mad Libs

Have you ever played Mad Libs? You're prompted to fill out a list of nouns, verbs, adjectives, etc., and then copy them onto another piece of paper that contains a story, missing those crucial words that you are now providing. The idea is to pick bizarre words without knowing what the story is, and then when you fill them in, the results can be hilarious.

Let's make a page that mimics the Mad Libs format:

mad-libs.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <link rel="stylesheet" href="css/styles.css" type="text/c
ss">
  <script src="js/scripts.js"></script>
  <title>A fantastical adventure</title>
</head>
<body>
  <h1>Fill in the blanks to write your story!</h1>
  <form>
    <label for="person1Input">A name</label>
    <input id="person1Input" type="text" name="person1Inpu
t">
    <label for="person2Input">Another name</label>
    <input id="person2Input" type="text" name="person2Inpu
t">
    <label for="animalInput">An animal</label>
    <input id="animalInput" type="text" name="animalInput">
    <label for="exclamationInput">An exclamation</label>
    <input id="exclamationInput" type="text" name="exclamat
ionInput">
    <label for="verbInput">A past tense verb</label>
    <input id="verbInput" type="text" name="verbInput">
    <label for="nounInput">A noun</label>
    <input id="nounInput" type="text" name="nounInput">
    <button type="submit">Show me the story!</button>
  </form>

  <div id="story" class="hidden">
    <h1>A Fantastical Adventure</h1>
    <p>
      One day, <span id="person1a">_________</span> and <sp
an id="person2a">_________</span> were walking through the
woods, when suddenly a giant <span id="animal">_________</s
pan> appeared. "<span id="exclamation">_________</span>", <
span id="person1b">_________</span> cried. The two of them
<span id="verb">_________</span> as quickly as possible, an
d when they were safe, <span id="person1c">_________</span>
and <span id="person2b">_________</span> gave each other a
giant <span id="noun">_________</span>.
```

```
    </p>
  </div>
</body>
</html>
```

In our HTML we have a form that gets different words from the user: a person's name, another person's name, a verb, an exclamation, etc. In the second half of our HTML we have a story area where the form data from the user will be added to create the Mad Lib story.

A form in HTML uses the `<form>` element. Inside of it goes any combination of form inputs and labels (we'll see more examples of this later in this section) and a `<button>` element that users can click to submit the form.

To create form fields with labels and inputs, we use `<label>` and `<input>` HTML elements. Each of these has a series of attributes that make the form function properly:

- The `id` attribute is the same `id` attribute that we already learned about. In this case, it helps us target and access that input's value.
- The `type` attribute in any `<input>` allows you to select the type of content that will go into that input. We've selected `text` because we're only inputting text. There are many values we can set for the `type` attribute, all of which change the look of the input element. We'll learn more about these in an upcoming lesson.
- The `for` attribute on the `<label>` element should have the same value as the `name` attribute on the corresponding `<input>`. This allows us to connect the label and input together in code. Functionally, this enables a user to click on the label to select the corresponding input (a text cursor will appear in the input). The `name` attribute on the `<input>` has other functions, too, which we won't get into now.

It's standard practice to include all of the above attributes on form `<label>` and `<input>` elements, in their respective locations. You shouldn't worry about memorizing this information. Instead, just reference this lesson as needed to review how to create a form.

Hiding and Showing the Story

Let's make our Mad Libs site show the story only after the user has submitted the form. This means we'll have to hide and show the form with CSS. Create a `css` directory and add a `styles.css` file to it with the following CSS rule.

css/styles.css

```
.hidden {
  display: none;
}
```

The `display` property lets us define the layout and visibility of an element on the page. This includes hiding elements by giving `display` a value of `none`. There are a lot of values to use with `display`, and you can read about them all on MDN documentation (https://developer.mozilla.org/en-US/docs/Web/CSS/display) or your preferred documentation for CSS.

With the above CSS rule, we've created a class called `hidden` that hides any element that we add the class to. We've already added this class to the div containing the story:

```
<div id="story" class="hidden">
  ...
</div>
```

When we want to show our story, all we'll have to do is remove the `hidden` class from the div element. We'll remove the `hidden` class with the `Element.removeAttribute()` method in our scripts file.

As a refresher, when we write `Element.removeAttribute()`, we're referring to the `removeAttribute()` method that belongs to the `Element` object. The `Element` object represents an HTML element and it has properties and methods that let us interact with the HTML elements on our webpage. If you want to review the `Element` object along with the topics of inheritance and object types, visit this lesson (https://www.learnhowtoprogram.com/introduction-to-programming/javascript-and-web-browsers/understanding-web-apis-interfaces-object-types-and-inheritance).

Adding an `onsubmit` Event Handler

Next, our JavaScript needs to get the value from the form inputs, insert them into the `<span>`s where the information should go, and then show the hidden story itself. We know how to insert text into our pages, so let's get that working before we try to get the data from the form:

js/scripts.js

```
window.onload = function() {
  // first we set up an event handler for the form submissi
on
  let form = document.querySelector("form");
  form.onsubmit = function(event) {
    // then we print values to the story area;
    // we're hardcoding these values for now
    document.querySelector("span#person1a").innerText = "pe
rson 1";
    document.querySelector("span#person1b").innerText = "pe
rson 1";
    document.querySelector("span#person1c").innerText = "pe
rson 1";
    document.querySelector("span#person2a").innerText = "pe
rson 2";
    document.querySelector("span#person2b").innerText = "pe
rson 2";
    document.querySelector("span#animal").innerText = "an a
nimal";
    document.querySelector("span#verb").innerText = "verb";
    document.querySelector("span#noun").innerText = "noun";
    document.querySelector("span#exclamation").innerText =
"exclamation";

    // then we show the story by removing the class attribu
te
    document.querySelector("div#story").removeAttribute("cl
ass");
  };
};
```

The `onsubmit` property lets us create an event handler for when a form is submitted. It's different from the `onclick` event handler property because a form can be submitted by clicking on the submit button as well as by pressing *Enter* while a form field is selected.

For the `onsubmit` event handler to work, we need an HTML button element with its `type` attribute set to `submit`:

```
<button type="submit">Show me the story!</button>
```

Also, the `<button>` element needs to be located within the `<form>` tags:

```
<form>
  <label for="person1Input">A name</label>
  <input id="person1Input" type="text" name="person1Input">
  ...
  <button type="submit">Show me the story!</button>
</form>
```

Going forward, when you create an HTML form, you should always use a button element with the `type` attribute set to `submit` and the `<button>` should be inside of the `<form>` tags. With HTML like this in place, you'll be able to use the `onsubmit` event handler property to react to the form submission.

Let's submit our form and see what happens! When we do this we should see the story to show with the hardcoded values we've added to our scripts (for ex, "person 1", "person 2", "an animal", etc). However ...

Preventing Default Behavior with `event.preventDefault()`

... When we submit the form, the story briefly flashes and then disappears! This is caused by the default behavior of the 'submit' event. It's typical for a form submission to be sent to a server (which might hold a database) where the information on the form can be processed and stored. And that's exactly what the default behavior for the submit event is — to try to submit the information to a server and then refresh the page with any information from the server.

But we aren't submitting our form to a server and this default behavior is causing the page to refresh and erase the information submitted in the form.

If you look in the URL bar of your browser, you should see that there's a ? within the address now. This is your clue that the form has been submitted to nowhere and the page refreshed. So what do we do about this? We need to prevent the default action for the form:

js/scripts.js

```
window.onload = function() {
  let form = document.querySelector("form");
  form.onsubmit = function(event) {  // there's a new param
eter
    document.querySelector("span#person1a").innerText = "pe
rson 1";
    document.querySelector("span#person1b").innerText = "pe
rson 1";
    document.querySelector("span#person1c").innerText = "pe
rson 1";
    document.querySelector("span#person2a").innerText = "pe
rson 2";
    document.querySelector("span#person2b").innerText = "pe
rson 2";
    document.querySelector("span#animal").innerText = "an a
nimal";
    document.querySelector("span#verb").innerText = "verb";
    document.querySelector("span#noun").innerText = "noun";
    document.querySelector("span#exclamation").innerText =
"exclamation";

    document.querySelector("div#story").removeAttribute("cl
ass");

    // we prevent the default refresh action for the submit
event
    event.preventDefault();
  };
};
```

Notice that we've added a parameter `event` to the function expression that we've set for the `onsubmit` event handler property. We've also called `event.preventDefault();` at the bottom of the form submit event handler. We can place `event.preventDefault();` anywhere in the form submit event handler, not just the bottom.

Also, developers often use `e` for short, instead of `event`:

```
form.onsubmit = function(e) {  // there's a new parameter
  ...
  e.preventDefault();
}
```

The `event` (or `e`) parameter is a placeholder for an object that represents the event we're dealing with. Since we're working with the `onsubmit` event handler property that means the event we're dealing with is a 'submit' event. The `event` object is a Web API, and it has handy information and tools like:

- What type of event just happened.
- Where in the webpage the event originated from.
- Stopping the default behavior for the event with the `event.preventDefault()` method.

An `event` object is automatically created for every event that happens in the web browser. Typically, we can only access the `event` object from within the event handler that targets the event. Taking our Mad Libs form for an example, when the form submission event happens, a submission event object gets created and passed in as the value of the `event` parameter in the function expression.

The `event` object is typically confusing for new developers, because the creation of the `event` object happens behind the scenes. It feels odd to create a parameter for an object that we can't see get created. However, this is how events are set up. In a future course section, we'll learn more about the `event` object, how to access it, and where we can learn more information about it.

For now, let's focus on the important piece of functionality that we need from the `event` object for our form: preventing the submit event's default behavior to refresh the page. **Going forward when we're working with forms, we always need to remember to call `event.preventDefault();` within the handler function set to the**

`onsubmit` property and include a parameter for `event`. When we get to the backend course on C# and .NET or Ruby and Rails, we'll learn how to submit forms values to servers so that we don't need to prevent this default behavior.

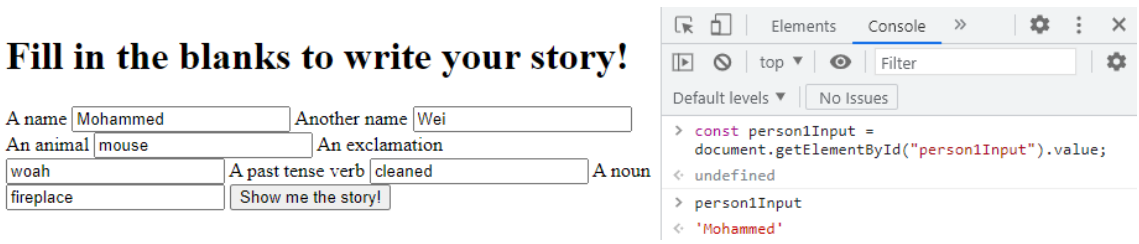
Anytime you have a `?` within the browser URL, follow these steps to fix the issue:

1. Add `event.preventDefault()` into the `onsubmit` event handler. Don't forget to add the `event` parameter in the function.
2. Remove the `?` and anything after it from your browser URL. For example if you are serving the project with LiveServer, your URL might look like `http://127.0.0.1:5501/index.html?person1Input=Mohammed`, and you'll want to reset it to `http://127.0.0.1:5501/index.html`. If you forget to do this, there shouldn't be an issue, but you should know that your URL won't automatically reset with a page refresh.
3. Refresh the page before proceeding.

Getting User Inputted Form Values

Now, we need to actually get the values from the form. With the Mad Libs project open in the browser, we can try this out in the DevTools console first:

```
> const person1Input = document.getElementById("person1Inpu
t").value;
> person1Input;
"Mohammed"
```



In the above example, I've targeted the first HTML input element with `document.getElementById("person1Input")` and then I've accessed the `value` property of the returned `input` object. Let's break that down into steps.

First we get the DOM input element:

```
// getting the input element
> const person1Input = document.getElementById("person1Inpu
t");
> person1Input;
<input id="person1Input" name="person1Input" type="text">
```

Then we check the exact type of the input element:

```
// checking the exact type of the input element
// which is HTMLInputElement
> Object.prototype.toString.call(person1Input);
'[object HTMLInputElement]'
```

And then we get the value of the input element:

```
// getting the value from the HTMLInputElement
> const person1Value = person1Input.value;
> person1Value;
"Mohammed"
```

Just like how the object `HTMLHeadingElement` represents an H1 element (or any of the others H2, H3, etc), the `HTMLInputElement` represents the HTML `<input>` element. We can also see that the `value` property of the `HTMLInputElement` gives us the value that the user typed into the form input.

Let's update our code to gather the values from each form `<input>` element. As you read through the updated scripts below, take note of the comments.

js/scripts.js

```
window.onload = function() {
  let form = document.querySelector("form");
  form.onsubmit = function(event) {
    // in this section we get the value for each form input
    const person1Input = document.getElementById("person1In
put").value;
    const person2Input = document.getElementById("person2In
put").value;
    const animalInput= document.getElementById("animalInpu
t").value;
    const exclamationInput = document.getElementById("excla
mationInput").value;
    const verbInput = document.getElementById("verbInput").
value;
    const nounInput = document.getElementById("nounInput").
value;

    // then we set the story variables to the values we got
from the form
    document.querySelector("span#person1a").innerText = per
son1Input;
    document.querySelector("span#person1b").innerText = per
son1Input;
    document.querySelector("span#person1c").innerText = per
son1Input;
    document.querySelector("span#person2a").innerText = per
son2Input;
    document.querySelector("span#person2b").innerText = per
son2Input;
    document.querySelector("span#animal").innerText = anima
lInput;
    document.querySelector("span#verb").innerText = verbInp
ut;
    document.querySelector("span#noun").innerText = nounInp
ut;
    document.querySelector("span#exclamation").innerText =
exclamationInput;

    document.querySelector("div#story").removeAttribute("cl
ass");
```

```
    event.preventDefault();
  };
};
```

When we refresh the page and submit the form, we can see our Mad Libs website work. Hooray! We've created our first project that gathers user input through a form!

Styling Forms with Bootstrap

Bootstrap offers good looking form styling that you can explore in your projects. Check out the Bootstrap version 5.2 documentation on forms here (https://getbootstrap.com/docs/5.2/forms/overview/). Take note that we'll explore and practice with other form types in upcoming lessons. These include select boxes, radio buttons, and check boxes, as well as using `<input>` types for dates and colors.

Summary

We learned a lot in this lesson. Here are some important takeaways:

- When creating forms, use a button element with a `type` attribute that's set to a value of `submit`. This button should always be inside of the `<form>` tags. With this HTML, we can use the `onsubmit` event handler property on the form to target the submission event.
- An `event` object is automatically created for every event handler when the event happens. We can only access the `event` object from within the event handler. The `event` object give us information and tools like:
  - What type of event it is (like 'click' or 'submission').
  - Where in the webpage the event originated from.
  - Stopping the default behavior for the event with the `event.preventDefault()` method.

- When we're working with a submission event on a form, we'll need to prevent the default behavior of refreshing the page. Use `event.preventDefault()` within the handler function and don't forget to add the parameter `event`.
- `HTMLInputElement`s have a `value` property that we can access to get what the user has typed into the input.

Visit the cheat sheet for the completed code for the Mad Libs project.

Previous (/introduction-to-programming/javascript-and-web-browsers/practice-event-handling-with-event-handler-properties)
Next (/introduction-to-programming/javascript-and-web-browsers/other-ways-to-organize-ui-logic)

Lesson 54 of 75
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)