Lesson Monday

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Address Book: User Interface

Text Cheat sheet

After following along with the weekend homework, we have a simple address book application that can create contacts and add, delete, and find them in an address book. Now let's start building the UI!

In the process, we'll experiment with adding dynamic elements to the DOM and learn about new concepts like **event bubbling** and **event delegation**. These are more advanced concepts that you won't be expected to apply to this section's independent project. However, you are encouraged to experiment with them, as they'll make you a better coder in the long run.

Address Book HTML

Let's get the basics of our UI up and running. We'll create an `index.html` file in the top-level of our directory with the following HTML boilerplate:

index.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <title></title>
</head>
<body>

</body>
</html>
```

Next let's fill in each section. First we'll add links to our `<head>` section:

index.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
  <link
    href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dis
t/css/bootstrap.min.css"
    rel="stylesheet"
    integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1R
WuH61DGLwZJEdK2Kadq2F9CUG65"
    crossorigin="anonymous">
  <script src="js/scripts.js"></script>
  <title>Address Book</title>
</head>
<body>

</body>
</html>
```

We've added the following, which should all be review:

- A link to the Bootstrap CDN in order to use Bootstrap styles in our project.

  - Remember that we don't have to specifically implement Bootstrap's classes or elements to benefit from Bootstrap — just including Bootstrap's CSS will improve our project's style.
  - Also, remember that you'll need to include a separate CDN in a `<script>` tag for Bootstrap's JS if you want to implement any Bootstrap elements that are interactive. We won't include the script in the Address Book project, but you are welcome to do so in the projects you build in this course section.
- A link to our local `scripts.js` file containing our JavaScript Logic.
- A `<title>` for our page.

We'll add a CSS style sheet later when we're ready to use it.

Now let's focus on the `<body>`. We'll keep it simple before later adding Bootstrap classes for styling and organization:

index.html

```
...

<body>
  <h1>Address Book</h1>
  <h2>Add a Contact:</h2>
  <form id="new-contact">
    <label for="new-first-name">First Name</label>
    <input type="text" id="new-first-name" name="new-first-
name">
    <label for="new-last-name">Last Name</label>
    <input type="text" id="new-last-name" name="new-last-na
me">
    <label for="new-phone-number">Phone Number</label>
    <input type="text" id="new-phone-number" name="new-phon
e-number">
    <button type="submit">Add</button>
  </form>

  <h2>Contacts:</h2>
  <div id="contacts">
  </div>

  <div id="contact-details" class="hidden">
    <p>First Name: <span id="first-name"></span></p>
    <p>Last Name: <span id="last-name"></span></p>
    <p>Phone Number: <span id="phone-number"></span></p>
  </div>
</body>

...
```

Here we have:

- A form to add new contacts, including fields for first names, last names, and phone numbers. Each field also has a corresponding `<label>`.
- A `<div>` element with an id of `contacts`, where we will add a list of `Contact`s.

- A `contact-details` div that will eventually display the details for a specific contact when selected from the list.

CSS

Next let's add CSS. We'll create a `css` subdirectory with a `styles.css` file inside containing a single rule:

css/styles.css

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

This ensures the `contact-details` div is hidden until we display a contact's details.

JavaScript

After following along with the past few lessons we should already have a `scripts.js` file in a `js` directory that looks like this:

js/scripts.js

```
// Business Logic for AddressBook ---------
function AddressBook() {
  this.contacts = {};
  this.currentId = 0;
}

AddressBook.prototype.addContact = function(contact) {
  contact.id = this.assignId();
  this.contacts[contact.id] = contact;
};

AddressBook.prototype.assignId = function() {
  this.currentId += 1;
  return this.currentId;
};

AddressBook.prototype.findContact = function(id) {
  if (this.contacts[id] !== undefined) {
    return this.contacts[id];
  }
  return false;
};

AddressBook.prototype.deleteContact = function(id) {
  if (this.contacts[id] === undefined) {
    return false;
  }
  delete this.contacts[id];
  return true;
};

// Business Logic for Contacts ---------
function Contact(firstName, lastName, phoneNumber) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.phoneNumber = phoneNumber;
}

Contact.prototype.fullName = function() {
  return this.firstName + " " + this.lastName;
};
```

Below this business logic we'll add a new section for user interface logic, denoted by another comment:

js/scripts.js

```
...

// User Interface Logic ---------
let addressBook = new AddressBook();

function handleFormSubmission(event) {
  event.preventDefault();
  const inputtedFirstName = document.querySelector("input#new-first-name").value;
  const inputtedLastName = document.querySelector("input#new-last-name").value;
  const inputtedPhoneNumber = document.querySelector("input#new-phone-number").value;
  let newContact = new Contact(inputtedFirstName, inputtedLastName, inputtedPhoneNumber);
  addressBook.addContact(newContact);
  console.log(addressBook.contacts);
}

window.addEventListener("load", function (){
  document.querySelector("form#new-contact").addEventListener("submit", handleFormSubmission);
});
```

Let's walk through this new code:

- We create a new `AddressBook` object named `addressBook`. This is a global variable because it's declared at the 'top level' of our file. A quick reminder: while we generally want to avoid global variables, we've made an exception here to mimic a database.

- At the bottom of our scripts, we create an event listener for the `window` 's 'load' event. The callback function we pass into `window.addEventListener()` will be called when our webpage has loaded all resources and is ready to go.
- Within the `window` 's 'load' event listener we create an event listener for the form submission event. When the form is submitted the `handleFormSubmission()` function will be called. Take note that the naming convention to prefix event handler functions with `handle` is very common, but it's not required to follow. The baseline here is to always name your functions descriptively, so that anyone can immediately tell what the function's purpose is in the code.
- We then define the function `handleFormSubmission()` above where we call it in the `window` 'load' event listener. In this function we:
  - Call `event.preventDefault()` to prevent the refresh of the page.
  - Gather user-provided input from the form fields for first name, last name, and phone number, and assign them to the variables `inputtedFirstName` , `inputtedLastName` , and `inputtedPhoneNumber` .
  - Create a new `Contact` object, passing in this gathered information as arguments, and saving the new `Contact` object in the variable `newContact` .
  - Add the `newContact` to our `AddressBook` using the `AddressBook.prototype.addContact()` method.
  - Log the list of `Contacts` in our `AddressBook` to the console, to double-check that the new contact has been added.

In the next lesson, we'll add logic for displaying contacts to the webpage.

The entire updated `scripts.js` file looks like this:

js/scripts.js

```
// Business Logic for AddressBook ---------
function AddressBook() {
  this.contacts = {};
  this.currentId = 0;
}

AddressBook.prototype.addContact = function(contact) {
  contact.id = this.assignId();
  this.contacts[contact.id] = contact;
};

AddressBook.prototype.assignId = function() {
  this.currentId += 1;
  return this.currentId;
};

AddressBook.prototype.findContact = function(id) {
  if (this.contacts[id] !== undefined) {
    return this.contacts[id];
  }
  return false;
};

AddressBook.prototype.deleteContact = function(id) {
  if (this.contacts[id] === undefined) {
    return false;
  }
  delete this.contacts[id];
  return true;
};

// Business Logic for Contacts ---------
function Contact(firstName, lastName, phoneNumber) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.phoneNumber = phoneNumber;
}

Contact.prototype.fullName = function() {
  return this.firstName + " " + this.lastName;
};
```

```
// User Interface Logic ---------
let addressBook = new AddressBook();

function handleFormSubmission(event) {
  event.preventDefault();
  const inputtedFirstName = document.querySelector("input#n
ew-first-name").value;
  const inputtedLastName = document.querySelector("input#ne
w-last-name").value;
  const inputtedPhoneNumber = document.querySelector("input
#new-phone-number").value;
  let newContact = new Contact(inputtedFirstName, inputtedL
astName, inputtedPhoneNumber);
  addressBook.addContact(newContact);
  console.log(addressBook.contacts);
}

window.addEventListener("load", function (){
  document.querySelector("form#new-contact").addEventListen
er("submit", handleFormSubmission);
});
```

We can now launch our `index.html` page in the browser, fill out our form several times, and see a growing list of `Contact` s logged in the console! The next step will be to actually display our contacts in the DOM. In order to do that, though, we first need to learn how to loop through object properties.

📂Example GitHub Repo for the Address Book (https://github.com/epicodus-lessons/oop-address-book-v2/tree/5_address_book_user_interface)

Previous (/intermediate-javascript/object-oriented-javascript/address-book-places-you-ve-been-to-do)
Next (/intermediate-javascript/object-oriented-javascript/looping-through-objects-and-prototypal-inheritance)

Lesson 17 of 33
Last updated March 23, 2023

disable dark mode

Epicodus (http://www.epicodus.com)