

Lesson

Monday

Intermediate JavaScript (/intermediate-javascript)

/ Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript)

/ Address Book: Adding Interactivity

Text

Cheat sheet

Now that we know how to loop through an object's keys (properties), let's update our user interface code so we can display all of the `Contact`s on our site.

## Displaying Dynamic Contact Data

We'll start by writing an empty `listContacts()` function below our global variable but above our `handleFormSubmission()` function:

```
js/scripts.js
```

```
...

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

function listContacts(addressBookToDisplay) {

}

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);
}

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

As the function name states, this method will handle listing our contacts on the webpage. It takes an `AddressBook` object as an argument. Also notice that we're creating a new function to handle displaying contacts in the DOM instead of adding onto our existing functions. This is a best practice. By separating our code into different functions distinct purposes, our code is cleaner, easier to read, and easier to debug and refactor.

Now let's actually write the code that goes in this method. We'll apply a new best practice along the way:

## js/scripts.js

```
...

function listContacts(addressBookToDisplay) {
  let contactsDiv = document.querySelector("div#contacts");
  contactsDiv.innerHTML = null;
  const ul = document.createElement("ul");
  Object.keys(addressBookToDisplay.contacts).forEach(function(key) {
    const contact = addressBookToDisplay.findContact(key);
    const li = document.createElement("li");
    li.append(contact.fullName());
    li.setAttribute("id", contact.id);
    ul.append(li);
  });
  contactsDiv.append(ul);
}

...
```

- First we save the div that will contain our contacts in a variable called `contactsDiv`.
- Next, we clear the `innerHTML` of the `contactsDiv`. This clears our list of contacts *before* we populate it. We need to do this because our code is set up to loop through all contacts and print each one, not just newly added contacts. So, this ensures that the user can submit the form to create a new contact over and over and no duplicate contacts will be listed.
- Next, we create an empty unordered list element. During each iteration of the loop, we'll add new list items to this unordered list.
- Then, we use `Object.keys()` to get all the keys from `addressBookToDisplay.contacts` so we can iterate through them. Remember, `Object.keys()` returns an array, so we can call any array method on the returned value.

- Then, we use `Array.prototype.forEach()` to loop through the object keys.
- In our loop, we do a few things to populate our unordered list with list items for each contact in our address book:
  - We grab a `contact` object by using our `AddressBook.prototype.findContact()` method.
  - We create a new list item element for the contact. The text of the list item is set to the contact's full name, and we add an `id` attribute that is equal to the contact's `id` property.
  - Then we add the newly created list item to the unordered list.
- After the loop is finished and our unordered list is populated, we update the DOM by appending the unordered list to the `contactsDiv` element.

Take note that it is very important that each contact list item is being created with an `id` attribute matching the `Contact`'s `id` property. We can later retrieve the value of the `id` attribute to use with our `AddressBook.prototype.findContact()` method to locate an entire `Contact` object. If we didn't attach this information to the list element, our application would have no way to easily get the ID.

## A Best Practice: Limiting How Many Times We Update the DOM

One thing to note in the design of the `listContacts()` function is that we purposefully wrote our code to update the DOM just once. Querying the DOM takes time, whether we're simply accessing elements from it, or adding and removing elements. It's a best practice to design your code to access the DOM only when it's needed.

To understand this, let's look at a different design for the `listContacts()` function that is not as efficient as the one we currently have. In this alternate design, we don't use a `div` in our

HTML to locate our contacts. Instead, we directly use a UL tag like this:

```
...
    <h2>Contacts:</h2>
    <ul id="contacts">
    </ul>
...
```

With this, we can update our `listContacts()` function to look like the following. Notably this alternative version of `listContacts()` takes fewer lines of code to achieve the same functionality:

```
function listContacts(addressBookToDisplay) {
    let contactsList = document.querySelector("ul#contacts");
    Object.keys(addressBookToDisplay.contacts).forEach(function(key) {
        const contact = addressBookToDisplay.findContact(key);
        const li = document.createElement("li");
        li.append(contact.fullName());
        li.setAttribute("id", contact.id);
        contactsList.append(li);
    });
}
```

So what's different? Well, in each iteration of the loop we still create a list item that's set to the name of the contact. We also continue to add the contact to the unordered list. But now that unordered list is located in the DOM and not an element that we've newly created in our scripts. This means that every iteration of the loop we are querying the DOM to add a new list item to it.

Now let's say that our address book has one hundred contacts. This means that `listContacts()` will query the DOM 100 times. That can really slow down the webpage, especially if there are many users on

our address book application with lots of address books and contacts.

Now contrast that to the current setup of `listContacts()` :

### js/scripts.js

```
function listContacts(addressBookToDisplay) {
  let contactsDiv = document.querySelector("div#contacts");
  contactsDiv.innerHTML = null;
  const ul = document.createElement("ul");
  Object.keys(addressBookToDisplay.contacts).forEach(function(key) {
    const contact = addressBookToDisplay.findContact(key);
    const li = document.createElement("li");
    li.append(contact.fullName());
    li.setAttribute("id", contact.id);
    ul.append(li);
  });
  contactsDiv.append(ul);
}
```

There's more lines of code, but this code is way more efficient because we only update the DOM with our list of contacts once, after it is completely populated and ready to go. So whether we have one contact or 100 contacts, we still only update the DOM a single time with the populated list of contacts.

Limiting how many times you query the DOM is a best practice that you should work towards. That said, don't worry about getting it right 100% of the time, just start considering this when you are designing your UI logic.

## Calling the `listContacts()` Function

Let's call this new `listContacts` function whenever we add a new Contact . All we have to do is remove our `console.log()` and replace it with this new function.

**js/scripts.js**

```
...
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);
  listContacts(addressBook); // <--- This is the new line!
}

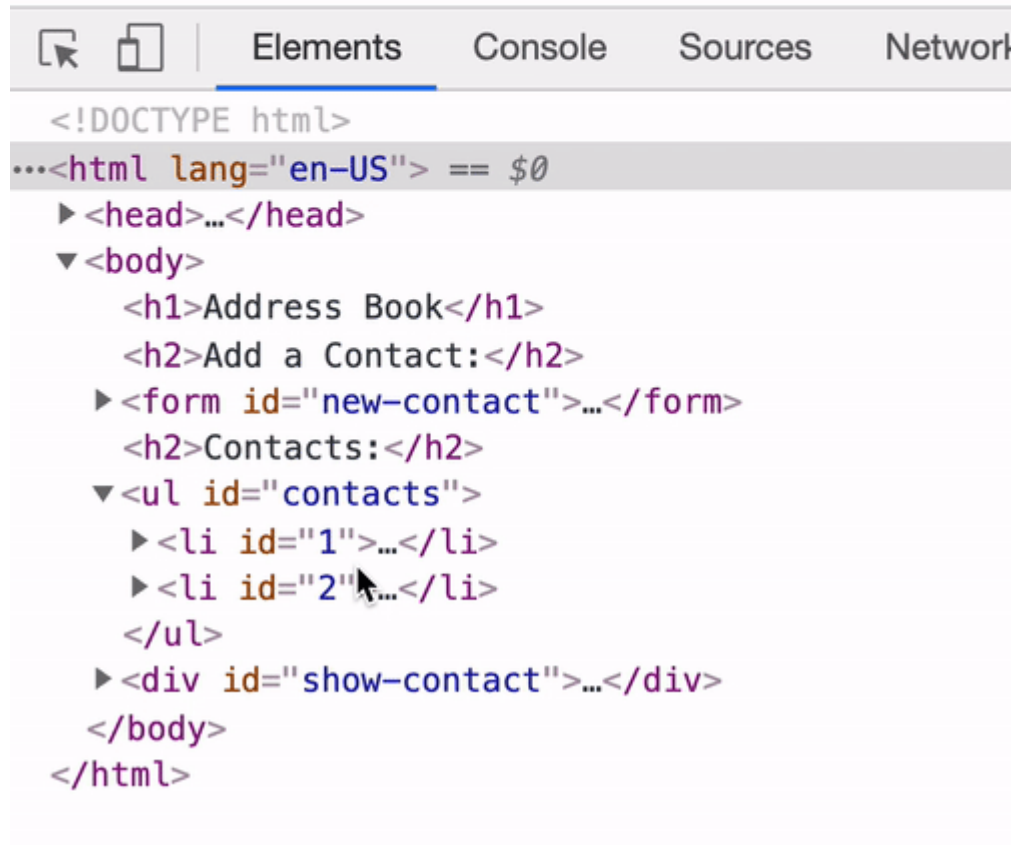
window.addEventListener("load", function () {
  document.querySelector("form#new-contact").addEventListener("submit", handleFormSubmission);
});
```

Now each time we add a new Contact, the page will update and add the contact's first and last names to the list of contacts. Try it out for yourself in the browser.

If we open Chrome Developer Tools and inspect elements on the page (right-click and then select *Inspect*) we'll see each Contact's `<li>` entry has a unique `id` corresponding to the Contact object's automatically-assigned `id` property. The GIF below demonstrates this:

# Contacts:

- contact1 contact1
- contact2 contact2



While this ID isn't doing anything yet, we'll soon be adding functionality so that we can click on a contact and get additional information based on its ID.

After following along, our 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 listContacts(addressBookToDisplay) {
  let contactsDiv = document.querySelector("div#contacts");
  contactsDiv.innerHTML = null;
  const ul = document.createElement("ul");
  Object.keys(addressBookToDisplay.contacts).forEach(function(key) {
    const contact = addressBookToDisplay.findContact(key);
    const li = document.createElement("li");
    li.append(contact.fullName());
    li.setAttribute("id", contact.id);
    ul.append(li);
  });
  contactsDiv.append(ul);
}

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);
  listContacts(addressBook);
}

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

## Best Practices Review

Let's recap some of the best practices we just used:


1. We created a **separate UI function** instead of adding the code to an existing function, like our `handleFormSubmission()` function. This allows us to focus on writing one function at a time, and helps keep code modular.
2. We create a list of all elements we want to append to the DOM, and add them **all at once** instead of one a time. This is faster and more efficient.

You aren't expected to master these best practices just yet, but you should consider them and practice integrating them in your code throughout the course. These kinds of details separate beginning coders from more experienced ones.

In addition to these best practices, we also applied our new knowledge of looping through object keys.

In the next lesson, we'll add UI functions that will allow us to display the detailed information of an individual contact onscreen.

---

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/6\\_adding\\_interactivity](https://github.com/epicodus-lessons/oop-address-book-v2/tree/6_adding_interactivity))

[Previous \(/intermediate-javascript/object-oriented-javascript/looping-through-objects-and-prototypal-inheritance\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/address-book-event-bubbling-event-delegation-and-the-event-object\)](#)

Lesson 19 of 33

Last updated March 23, 2023

[disable dark mode](#)



© 2023 Epicodus (<http://www.epicodus.com/>), Inc.