

|  |    |
|--|----|
| WK4 - 4aJavaScript Objects   | 2  |
| WK4 - 5aLiteral Notation Versus Constructors                                     | 5  |
| WK4 - 6aConstructors and Prototypes  | 8  |
| WK4 - 7aConstructor and Prototype Methods  | 13 |
| WK4 - 9aObjects Within Objects   | 15 |
| WK4 - 10aAddress Book_ Objects Within Objects                                    | 18 |
| WK4 - 11aAddress Book_ Unique IDs  | 20 |
| WK4 - 12aAddress Book_ Finding and Deleting Contacts                             | 23 |
| WK4 - 17aAddress Book_ User Interface  | 25 |
| WK4 - 18aLooping Through Objects and Prototypal Inheritance                      | 27 |
| WK4 - 19aAddress Book_ Adding Interactivity                                      | 30 |
| WK4 - 20aAddress Book_ Event Bubbling, Event Delegation, and the<br>Event Object | 32 |
| WK4 - 21aAddress Book_ Delete Functionality and Polish                           | 35 |

Lesson

Weekend

Intermediate JavaScript (/intermediate-javascript)

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

/ JavaScript Objects

Text

Cheat sheet

## Terminology and Examples

- **Encapsulation:** In object-oriented programming languages, the concept of binding data and the functions that manipulate the data together in one package. In other words, the packing of data and functions into one component.
- **Object:** A structure for storing data that is composed of properties and methods. In other words, an object is a collection of properties.
- **Literal notation:** Creation of objects using their literal symbols; e.g. to create a string with literal notation, the quotes symbols " " are the literals that cause a new string to be generated; arrays use brackets [ ] and objects use curly braces { }.
- **Key-value pair:** The format of an object's properties including a key which names the property and a value which assigns the value to the key; in the example below, the property name has a key of name and a value of "Kitty Poppins":

```
let myCat = {  
  name: "Kitty Poppins"  
};
```

- **Method:** A object property whose value is a function. In the example below, the property `area` is a method.

```
let rectangle = {  
  length: 22,  
  width: 10,  
  area: function() {  
    return this.length * this.width;  
  }  
};
```

- **this :** A JavaScript keyword that references the object a method is called on. We use `this` inside of an object method to access an object's properties.
- **Dot Notation:** The syntax used with a period `.` to create, access, update and manipulate JavaScript properties.

```
> dog.name;  
"Sparky"  
> dog.speak();  
Woof!
```

- **Bracket Notation:** The syntax used with brackets `[ ]` to create, access, update and manipulate JavaScript properties.

```
>dog[ 'name' ];  
"Sparky"  
>dog[ 'speak' ]();  
Woof!
```

## Example

---

A JavaScript object with one property and one method:

```
let dog = {  
  name: "Sparky",  
  speak: function() {  
    console.log("Woof!");  
  }  
};
```

[Previous \(/intermediate-javascript/object-oriented-javascript/homework-and-class-structure\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/literal-notation-versus-constructors\)](#)

Lesson 4 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

[Lesson](#)[Weekend](#)

Intermediate JavaScript (/intermediate-javascript)

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

/ Literal Notation Versus Constructors

[Text](#)[Cheat sheet](#)

## Terminology

---

- **Literal Notation:** A manner of creating a JavaScript object by literally listing out all attributes. We can use literal notation for other data types as well, like strings and arrays.

## Summary

---

Using a constructor is a much more streamlined manner to make JavaScript objects, especially when you're creating multiple objects with the same properties. For example, the following two blocks of code create the exact same 3 dog objects, but one is much more brief:

### Using Constructors:

```
let dog1 = new Dog("Falcor", ["black"], 4);  
let dog2 = new Dog("Nola", ["white", "black"], 6);  
let dog3 = new Dog("Patsy", ["brown"], 7);
```

## Using Literal Notation:

```
let dog1 = {  
  name: "Falcor",  
  colors: ["black"],  
  age: 4,  
};  
  
let dog2 = {  
  name: "Nola",  
  colors: ["white", "black"],  
  age: 6,  
};  
  
let dog3 = {  
  name: "Patsy",  
  colors: ["brown"],  
  age: 7,  
};
```

[Previous \(/intermediate-javascript/object-oriented-javascript/javascript-objects\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/constructors-and-prototypes\)](#)

Lesson 5 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

Lesson

Weekend

Intermediate JavaScript (/intermediate-javascript)

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

/ Constructors and Prototypes

Text

Cheat sheet

## Terminology

- **Constructor:** A blueprint for creating many of the same type objects — like strings and dogs. Constructors define the properties that every instance of an object will have.
  - Most built-in JavaScript objects have a constructor function that we can use to create objects of that type. An exception is the `Math` object, which does not have a constructor function. Some built-in objects can only be accessed via the constructor.
  - Generally speaking, if data can be created with literal notation, like a string, it's most common to use literal notation instead of the constructor function.
  - We can also create constructor functions for our own custom object types.
  - The first letter of the name of a constructor function is always capitalized, following UpperCamelCase, and it matches the name of the object type.



- **Array-like object:** Objects that have numbers as their property keys to represent an index location are called **array-like objects**. Similar to arrays, we can use bracket notation to access these objects, passing in an index. While the property keys look like numbers, they are actually strings. Here is an example:

```
> let testGreeting2 = new String("Hello!");
undefined
> testGreeting2;
String { Hello! }
  0: "H",
  1: "e",
  2: "l",
  3: "l",
  4: "o",
  5: "!",
  length: 6,
  [[Prototype]]: String,
  [[PrimitiveValue]]: "Hello!"
> testGreeting2[4];
"o"
```

- **Type:** The type (<https://developer.mozilla.org/en-US/docs/Glossary/Type>) of an object defines what data it contains, including all of the properties and methods that an object of that type has in it. In other words, the object type is the blueprint or template for creating an object, but not an actual object.
  - Note that you'll also see the terminology **class** if you are exploring constructor functions on MDN. A class is simply an object type, and in JavaScript we can create an object type by using class syntax. In a later course section, we'll learn how to use class syntax.
- **Instance:** Objects created with a constructor are instances of the object type defined by the constructor. A constructor can be used to create many instances of the same type.

- **Prototype:** Prototypes store properties to be shared by all objects of the same type. For example, `String.prototype` is shared by all `String` objects; `Dog.prototype` is shared by all `Dog` objects. Prototypes store properties that objects of that type inherit.
  - We choose to add methods to the prototype of an object type so that we can define the method once, but all object instances still have access to that method. This saves memory and is therefore more efficient. This is in contrast to adding a method to the constructor. In this case, when an instance of an object is created, so is a new instance of that method (and any other properties) which takes up more memory.
- **Prototype Chain:** Each object has a prototype property (called `__proto__`) that is a link to another object. This creates a **prototype chain** and is the mechanism by which one object inherits from multiple other object types. The end of the prototype chain is represented by the value `null`.

## Examples

---

Here is a constructor function for `Dog` that will initialize a new dog object with its attributes assigned to the values passed in to the constructor function.

```
function Dog(name, colors, age) {  
  this.name = name;  
  this.colors = colors;  
  this.age = age;  
}
```

Then to create a new dog we can do the following:

```
let myPuppy = new Dog("Ernie", ["brown","black"], 3);
```

We can access the name of the new dog:

```
> myPuppy.name;  
"Ernie"
```

The colors of the new dog:

```
> myPuppy.colors;  
["brown","black"]
```

And its age:

```
> myPuppy.age;  
3
```

We can add a method to the `Dog` prototype and all instances of the `Dog` object, current and future, will have access to this method:

```
> Dog.prototype.speak = function() {  
  console.log("Woof!");  
};
```

```
> myPuppy.speak();  
Woof!
```

We can access the prototype chain for the Dog object by accessing the `__proto__` property for each object in the chain. As we see in the image below, the `myPuppy` object inherits from the `Dog` object type, which itself inherits from the `Object` object type, which itself does not inherit from anything and ends the chain of inheritance.

```
> let myPuppy = new Dog("Ernie", ["brown", "black"], 3);
< undefined

> myPuppy.__proto__;
< ▶ {speak: f, humanYears: f, constructor: f}

> myPuppy.__proto__.constructor.name;
< 'Dog'

> myPuppy.__proto__.__proto__;
< ▶ {constructor: f, __defineGetter__: f, __defineSetter__: f, hasOwnProperty: f, __lookupGetter__: f, ...}

> myPuppy.__proto__.__proto__.constructor.name;
< 'Object'

> myPuppy.__proto__.__proto__.__proto__;
< null
```

[Previous \(/intermediate-javascript/object-oriented-javascript/literal-notation-versus-constructors\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/constructor-and-prototype-methods\)](#)

Lesson 6 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

Lesson

Weekend

# Intermediate JavaScript (/intermediate-javascript)

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

### / Constructor and Prototype Methods

Text

Cheat sheet

## Examples

Creating a Contact constructor:

```
function Contact(firstName, lastName, phoneNumber) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  this.phoneNumber = phoneNumber;  
}
```

Creating a prototype method for Contact :

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

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/1\\_address\\_book\\_constructor\\_and\\_prototype\\_methods](https://github.com/epicodus-lessons/oop-address-book-v2/tree/1_address_book_constructor_and_prototype_methods))

[Previous \(/intermediate-javascript/object-oriented-javascript/constructors-and-prototypes\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/accessing-code-from-different-branches\)](#)

Lesson 7 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

Lesson

Weekend

# Intermediate JavaScript (/intermediate-javascript)

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

### / Objects Within Objects

Text

Cheat sheet

## Terminology

- **Nested object:** A nested object is simply an object that is a property of another object. Or, an object that contains another object. Examples that we've seen with Web APIs are:

```
> window.document;  
#document  
> window.location;  
Location {ancestorOrigins: DOMStringList, href: 'https://www.learnhowtoprogram.com/introduction-to-.../getting-started-at-epicodus/learn-how-to-program', origin: 'https://www.learnhowtoprogram.com', protocol: 'https:', host: 'www.learnhowtoprogram.com', ...}  
> document.body;  
<body>...</body>
```

## Examples

---

To create relationships among objects, use properties whose values are arrays of other objects:

```
let pdx = { name: "Portland" };  
let sfo = { name: "San Francisco" };  
let sea = { name: "Seattle" };  
let usa = { name: "United States of America", cities: [pdx,  
sfo, sea] };
```

To add multiple objects within an object, we can use an empty array:

```
let uruguay = { name: "Uruguay", cities: [] };
```

Then we can add more objects to the `cities` array:

```
let mlz = { name: "Melo" };  
uruguay.cities.push(mlz);
```

[Previous \(/intermediate-javascript/object-oriented-javascript/accessing-code-from-different-branches\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/address-book-objects-within-objects\)](#)

Lesson 9 of 33

Last updated March 23, 2023

[disable dark mode](#)





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

[Lesson](#)[Weekend](#)

Intermediate JavaScript (/intermediate-javascript)

/ Object-Oriented JavaScript

(/intermediate-javascript/object-oriented-javascript)

/ Address Book: Objects Within Objects


[Text](#)[Cheat sheet](#)

## Examples

Add contacts to an `AddressBook` with its constructor and prototype methods:

```
js/scripts.js
```

```
// Business Logic for AddressBook -----  
function AddressBook() {  
    this.contacts = {};  
}  
  
AddressBook.prototype.addContact = function(contact) {  
    this.contacts[contact.firstName] = contact;  
};  
  
// 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;  
};
```

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/2\\_objects\\_within\\_objects](https://github.com/epicodus-lessons/oop-address-book-v2/tree/2_objects_within_objects))

[Previous \(/intermediate-javascript/object-oriented-javascript/objects-within-objects\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/address-book-unique-ids\)](#)

Lesson 10 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

[Lesson](#)[Weekend](#)

Intermediate JavaScript (/intermediate-javascript)

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

/ Address Book: Unique IDs


[Text](#)[Cheat sheet](#)

## Example

Here's how `scripts.js` will look by the end of the lesson:

```
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;  
};  
  
// 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;  
};
```

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/3\\_unique\\_ids](https://github.com/epicodus-lessons/oop-address-book-v2/tree/3_unique_ids))

[Previous \(/intermediate-javascript/object-oriented-javascript/address-book-objects-within-objects\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/address-book-finding-and-deleting-contacts\)](#)

Lesson 11 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

[Lesson](#)[Weekend](#)

# Intermediate JavaScript (/intermediate-javascript)

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

### / Address Book: Finding and Deleting Contacts

[Text](#)[Cheat sheet](#)


## Examples

Find a Contact :

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

Delete a Contact :

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

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/4\\_finding\\_and\\_deleting\\_contacts](https://github.com/epicodus-lessons/oop-address-book-v2/tree/4_finding_and_deleting_contacts))

[Previous \(/intermediate-javascript/object-oriented-javascript/address-book-unique-ids\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/vs-code-bracket-colorization-and-guides\)](#)

Lesson 12 of 33

Last updated March 23, 2023

[disable dark mode](#)



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



Lesson

Monday

# Intermediate JavaScript (/intermediate-javascript)

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


### / Address Book: User Interface

Text

Cheat sheet

## Examples

See the branch below for the code added in this lesson, including HTML, CSS, and UI Logic in our `scripts.js`.

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/5\\_address\\_book\\_user\\_interface](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](#)



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

Lesson

Monday

Intermediate JavaScript (/intermediate-javascript)

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

/ Looping Through Objects and  
Prototypal Inheritance

Text

Cheat sheet

## Terminology

---

- **\_\_proto\_\_**: Objects have a **\_\_proto\_\_** property which allows them to access other properties and functionality via prototypal inheritance. In the DevTools this property may be labeled as `[[Prototype]]`, but we still access this property through the property name **\_\_proto\_\_**. See the example below!
- **Prototypal Inheritance**: Inheriting functionality via object prototypes.

## Examples

---

## Loop Through Properties with `Object.keys()`

```
let mathematician = {
  firstName: "Ada",
  lastName: "Lovelace",
  profession: "Mathematician",
  funFact: "Daughter of Lord Byron",
  countryOfBirth: "England",
  yearOfBirth: 1815,
  yearOfDeath: 1852
}
const adaKeys = Object.keys(mathematician);
let adaString = "";
adaKeys.forEach(function(key) {
  adaString = adaString.concat(key + ": " + mathematician[key] + "\n");
});
```

## Loop Through Properties with `for...in`

```
let mathematician = {
  firstName: "Ada",
  lastName: "Lovelace",
  profession: "Mathematician",
  funFact: "Daughter of Lord Byron",
  countryOfBirth: "England",
  yearOfBirth: 1815,
  yearOfDeath: 1852
}
for (const key in mathematician) {
  if (contact.hasOwnProperty(key)) {
    console.log(mathematician[key]);
  }
}
```

**Note:** Use `Object.prototype.hasOwnProperty()` if you only want the properties of the object itself to be iterated over.

## Accessing the Prototype of an Object

```
> function Contact(firstName, lastName, phoneNumber) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  this.phoneNumber = phoneNumber;  
}  
> Contact.prototype.fullName = function() {  
  return this.firstName + " " + this.lastName;  
};  
> let newContact = new Contact("Ada", "Lovelace", "111-111-1111");  
> newContact;  
Contact {firstName: 'Ada', lastName: 'Lovelace', phoneNumber: '111-111-1111'}  
> newContact.__proto__;  
{fullName: f, constructor: f}  
> newContact.__proto__.__proto__;  
{constructor: f, __defineGetter__: f, __defineSetter__: f, hasOwnProperty: f, __lookupGetter__: f, ...}
```

[Previous \(/intermediate-javascript/object-oriented-javascript/address-book-user-interface\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/address-book-adding-interactivity\)](#)

Lesson 18 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

Lesson

Monday

Intermediate JavaScript (/intermediate-javascript)

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

/ Address Book: Adding Interactivity

Text

Cheat sheet

## Best Practices

---


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.

## Examples

---

Create a UI function to display contacts in an address book:

```
function listContacts(addressBookToDisplay) {  
  let contactsDiv = document.querySelector("div#contacts");  
  contactsDiv.innerText = 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);  
}
```

 **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.

Lesson

Monday

Intermediate JavaScript (/intermediate-javascript)

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

/ Address Book: Event Bubbling, Event Delegation, and the Event Object

Text

Cheat sheet

## Terminology

---

- **Event Bubbling:** The process of events bubbling upward when an event is triggered in the DOM.
- **Event Delegation:** The process of creating an event listener on a parent element that fires for all specified child elements.

## Examples

---

Here's an example of **event bubbling**:



```
<div id="top-level">
  <ul id="contacts">
    <li id=1>Contact 1</li>
    <li id=2>Contact 2</li>
    <li id=3>Contact 3</li>
  </ul>
</div>
```

If an `li` in the sample HTML above is clicked, it will first trigger any listeners on `li`, then listeners on `#contacts`, then listeners on `#top-level`. The event 'bubbles up' the elements to the outermost element, triggering any event listeners as it goes.


Here's an example of **event delegation**:

```
function displayContactDetails(event) {
  const contact = addressBook.findContact(event.target.id);
  document.querySelector(".first-name").innerText = contact.firstName;
  document.querySelector(".last-name").innerText = contact.lastName;
  document.querySelector(".phone-number").innerText = contact.phoneNumber;
  document.querySelector("div#contact-details").removeAttribute("class");
}

...

window.addEventListener("load", function () {
  ...
  document.querySelector("div#contacts").addEventListener(
    "click", displayContactDetails);
});
```

Thanks to event delegation (and event bubbling), we can attach a click event listener to the parent element "div#contacts" , and it will trigger when we click on a list item inside of the div. We can then access information about the list item element through `event.target` . The power of event delegation is being able to write code for and target data from elements that do not exist yet in the DOM.

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/7\\_event\\_delegation](https://github.com/epicodus-lessons/oop-address-book-v2/tree/7_event_delegation))

[Previous \(/intermediate-javascript/object-oriented-javascript/address-book-adding-interactivity\)](#)

[Next \(/intermediate-javascript/object-oriented-javascript/address-book-delete-functionality-and-polish\)](#)

Lesson 20 of 33

Last updated March 23, 2023

[disable dark mode](#)



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

**Lesson****Monday**

Intermediate JavaScript (/intermediate-javascript)

/ Object-Oriented JavaScript

(/intermediate-javascript/object-oriented-javascript)

/ Address Book: Delete Functionality and Polish

Text

Cheat sheet

## Examples

---

To create a UI function to delete a contact in the DOM:


```
...

function displayContactDetails(event) {
    ...
    document.querySelector("button.delete").setAttribute("id", contact.id);
    document.querySelector("div#contact-details").removeAttribute("class");
}

function handleDelete(event) {
    addressBook.deleteContact(event.target.id);
    document.querySelector("button.delete").removeAttribute("id");
    document.querySelector("div#contact-details").setAttribute("class", "hidden");
    listContacts(addressBook);
}

...

window.addEventListener("load", function () {
    ...
    document.querySelector("button.delete").addEventListener("click", handleDelete);
});
```

 **Example GitHub Repo for the Address Book**  
([https://github.com/epicodus-lessons/oop-address-book-v2/tree/8\\_adding\\_delete\\_functionality\\_and\\_polish](https://github.com/epicodus-lessons/oop-address-book-v2/tree/8_adding_delete_functionality_and_polish))

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

[Next \(/intermediate-javascript/object-oriented-javascript/imposter-syndrome\)](#)

Lesson 21 of 33

Last updated March 23, 2023

[disable dark mode](#)



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