



Rev. 08Aug2019

Exercise 45

USING THIRD-PARTY CODE

The myth of the lonely coder leaves out the extent to which they rely on the work of others. From the committees that work to define HTML, CSS, and Javascript to the people that spend countless hours developing VS Code to the network of fellow programmers that even the "loneliest" of coders developers, our lives and careers are bound up with other people's work.

And while this dependency is somewhat removed, programmers *directly* rely to a great extent on other people's work in the form of third-party code. If you decide to pursue a programming career with my **CareerChanger** course, you'll learn about two extremely popular (and in-demand) front-end technologies called *React* (created and released for free) by Facebook, and *Redux*, which works hand-in-hand with React (and is also free).

React and Redux both fall into the category of software known as *open source*. In open-source software, the original source code is made freely available and may be modified and redistributed without additional permission. But React and Redux are the tip of the iceberg for open-source projects.

Suppose that you want a bit of JavaScript that will allow users to select a date (or range of dates). That's far more complicated than it might seem at first, but chances are you won't need to write a single line of code. Why not? Because you'll head over to **npmjs.com**, which is a massive library of open-source projects. In the search bar, type "**date picker**" — and you'll find 1,765 packages. Chances that one already exists that will meet your need? 100%.

In this exercise, we're going to use an npm package called *Toastify*. It provides for messages that rise up to the top of the user's screen and provide them with some information.

Full information on
CareerChanger available
at careerchanger.org

"npm" stands for "Node
Package Manger". Open-
source projects added to
the library are called
"packages".

Notifications "rise up" like
toast — hence, the name,
Toastify. Only, in this case,
they go down...

Normally, we'd use a piece of software known as **npm** to install Toastify, but the process of installing npm requires certain permissions be set. In the CareerChanger course, I'll walk you through that, but for now, we'll use an alternate way of working with Toastify.

- ❑ Create a sub-directory called **ex45** under your **exercises** directory.
- ❑ In the folder you just created for this exercise, create 3 sub-folders: **html**, **css**, and **javascript**. Inside the html folder, create **index.html**, using **doc** followed by pressing **Tab** to generate HTML boilerplate.

- ❑ Just above the closing `</body>` tag, insert this code:

```
<script type="text/javascript" src="https://cdn.jsdelivr.net/npm/toastify-js"></script>
```

- ❑ And just above the closing `</head>` tag, insert this code:

```
<link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/npm/toastify-js/src/toastify.min.css">
```

Now, we're ready to get our toast on!

CREATING INDEX.HTML

- ❑ In **index.html**, add this code just after the opening `<body>` tag:

```
<p class="blue pill" data-type="blue">Take the blue pill</p>
<p class="red pill" data-type="red">Take the red pill</p>
```

CREATING INDEX.CSS

- ❑ Create **css/index.css** and add a selector targeting elements with a class of **pill** with a selector that changes the **cursor** to **pointer**.

CREATING INDEX.JS

- ❑ Create **javascript/index.js**

This is the first time we've seen an attribute starting with "data-". Such attributes allow us to create our own, custom properties for HTML elements. When we write the JavaScript file, we'll see how we can use them.

ALGORITHMS AND COMMENTS

The term for creating steps to accomplish a task is "algorithm". The term derives from the name of an Arabic mathematician, "al-Kwarizmi", author of widely-translated works on algebra and arithmetic.

Until now, we haven't used comments very much. From this point on, we're going to use them much more extensively. People who don't understand software development assume that the primary work of coders is, well, writing code. It's not. The primary work of coders is determining a plan that, when implemented, will accomplish the task at hand.

— That might sound confusing. Let's think about a completely unrelated task in order to illustrate the point.

Let's say you have developed software that is purchased on a per-seat basis. That means that however many seats were purchased is the limit to the number of users that can log on simultaneously.

Here's where we're going to use comments. Instead of jumping to code, a wise coder lays out a plan for accomplishing this — and they then list the steps involved as a series of comments, like this:

```
// Create a variable that holds the number of seats purchased

// Create a variable that has the current number of people logged in

// When someone logs in, check to see if the current number of people logged in is
  below the number of seats purchased.

// If that's true, allow the person to log in.

// If it's false, disallow the login and give the user a message to buy more seats.
```

Once the plan is done, you have the steps needed to accomplish the task and you'll implement each comment separately:

```
// Create a variable that holds the number of seats purchased
let seatsPurchased = 5;

// Create a variable that has the current number of people logged in
let currentlyLoggedIn = 0;

// When someone logs in, check to see if the current number of people logged in is
  below the number of seats purchased.
let availableSeat = seatsPurchased - currentlyLoggedIn;

// If that's true, allow the person to log in.
if (availableSeat) { acceptLogin() }
```

BACK TO INDEX.JS

That was a long digression, but a very important one if you plan to take up software as a career. Here are the comments I wrote prior to writing any code:

```
// create pillListener function for event listener that will read the value
// of data-type

// get the pill type

// set a default message

// if the pill is red, provide the appropriate message

// call Toastify to get a function, passing it the message and how long it
// should remain

// Invoke the function returned by Toastify

// add pillListener as event listeners to all .pill elements
```

- ❑ I'm going to give you my implementation of each of these comments. Your job is to determine which code snippet to use for each comment. (They're mixed up right now.)

```
let toastMessage = Toastify({
  text: message,
  duration: 5000
});
```

```
let pillType = event.target.getAttribute('data-type');
```

```
toastMessage.showToast()
```

```
let pills = document.querySelectorAll('.pill')
let pillsArray = Array.prototype.slice.call(pills)

pillsArray.forEach( element => {
  element.addEventListener('click', pillListener)
})
```

```
let toastMessage = Toastify({  
  text: message,  
  duration: 5000  
})
```

```
let message = null;
```

```
} else {  
  message = "That was just a dream. Relax, go back to sleep..."  
}
```

```
if ( pillType === 'red') {  
  message = "You've chosen to take the red pill. Remember: all I can offer you is  
  the truth."
```

The rest of the exercises require this kind of planning. You may find it hard at first to think through an entire algorithm, using comments to document each step, but with practice, it becomes much easier — and the payoff is enormous.