Introduction

In the previous chapters, you learned how to utilize conditional logic, loops, and the most common data structures. These form the groundwork and essentials for writing programs and building complex JavaScript applications. Still, building actual software is an inherently challenging task; focusing on only business logic is even more so. Therefore, as developers, we often rely on external software that lets us dedicate ourselves to the source code that's the most relevant to our product or business. This software does this by simplifying specific tasks and abstracting away complexity for us. Those pieces of external software are what we refer to as **frameworks** or **libraries**.

The following are some of the tasks that modern JavaScript frameworks can support us with:

- Performance rendering of complex or dynamic single-page applications (SPAs)
- Managing ongoing dataflow between the controllers and views of client-side applications
- Creating sophisticated animations
- Creating with fast and straightforward server APIs

Before we dive deeper into the whys and the wherefores of using external code, we need to clarify what the difference is between the terms "framework" and "library." This will be the topic of the following section.

Framework versus Library

Library describes an external collection of functions that perform a given task. These functions are made accessible to us as users of the library via APIs. One useful library is **`lodash`**, which can, for example, remove all duplicated values from an array:

```
const duplicatedArray = [1,2,1,2,3];
const uniqueArray = lodash.uniq(duplicatedArray)
// => [1,2,3]
```

Frameworks, on the other hand, are a particular form of library. They are reusable code frames that build the foundation of a JavaScript application. In contrast to libraries, which extend your code with functionality, a framework can stand alone and is enhanced with your source code to create an app as you like.

A popular framework is **Vue.js**, which we can use as follows:

library-vue.js

```
// example.html
<div id="example">
<input :value="text" @input="update"/>
<div v-html="myOwnText"></div>
</div>
//——————————————————————————————
// example.js
new Vue({
el: '#example',
data: {
text: 'My first framework'
},
 computed: {
myOwnText: function () {
return this.text
}
```

The full code is available at: https://packt.live/32MD4IN

As you can see, in general, there is more complexity to a framework than there is to a library. Nonetheless, both are equally important to software development.

Despite the technical differences between libraries and frameworks, we are going to use those terms interchangeably. Another synonym you'll encounter in the JavaScript world to describe external source code is "package." One of those packages you may encounter in JS resources is **Vanilla.js**. We'll have a look at it in the next section.

Vanilla.js

This specific framework follows the informal convention of including the JavaScript file extension with the name **nameOfFramework.js**. However, vanilla.js is not a framework; it's not even a library. People referring to **vanilla.js** are talking about plain JavaScript without any external code or tooling. The name is a running gag within the JavaScript community because some developers and non-developers think we need to use a framework for everything we build. We will discuss why this isn't the case later.

Popular JavaScript Frameworks

We have just looked at lodash.js, a library that helps developers handle data structures; (to be used, for example, making arrays unique) and Vue.js, a framework for building modular and dynamic user interfaces. These are just two examples of quite popular and widely used JS frameworks/libraries. In addition to those, there is a vast and ever-growing number of external packages you can choose from. Each one of them is useful for solving one specialized set of problems.

A few modern and often used alternatives that support creating browser applications are, for instance, React.js, Vue.js, and Angular.js. Other libraries that help you store and manage data in your app are MobX, VueX, and Redux.

Again, others can transform source code so that it supports older browser engines, for example, **Babel**, or handle and manipulate time for you, such as moment.js.

Then, there are frameworks such as Express.js or Hapi that let you create simple, easy-to-maintain, and performant REST APIs for Node.js.

Some packages make building **command-line interfaces** (**CLIs**) or desktop applications easy.

Most build and productivity tools for the JavaScript ecosystem are provided to the community as a library, too. Webpack, Parcel, and Gulp are a few of these tools.

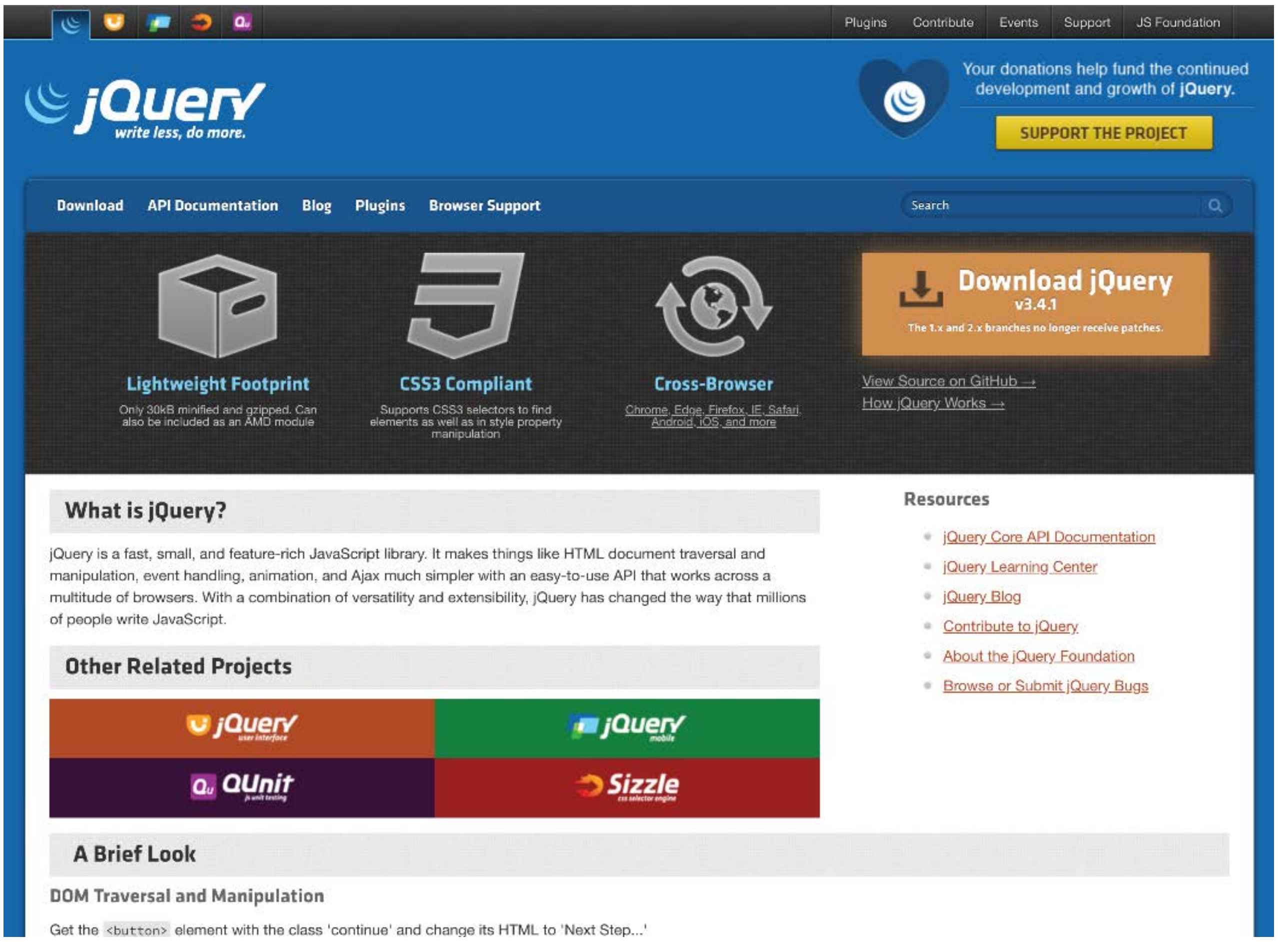
Not all of the available libraries are equally popular or useful. Their popularity depends on a few key facts:

- Whether they fix a problem that bothers many developers
- How well their API is defined and structured
- The quality of their documentation
- The level of performance optimization

Keep these in mind when crafting a package that you want to become well known.

Everlasting jQuery

One evergreen library that has been around for over a decade is jQuery. It touches almost every web app in one way or another and belongs in the toolkit of everybody who builds browser applications:



Your donations help fund the continued development and growth of **jQuery.**

The 1.x and 2.x branches no longer receive patches.

Lightweight Footprint

Only 30kB minified and gzipped. Can also be included as an AMD module

CSS3 Compliant

Supports CSS3 selectors to find elements as well as in style property manipulation

Cross-Browser

Chrome, Edge, Firefox, IE, Safari, Android, iOS, and more

What is jQuery?

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Other Related Projects

A Brief Look

DOM Traversal and Manipulation

Get the `<button>` element with the class 'continue' and change its HTML to 'Next Step...'

Figure 4.1: jQuery documentation

jQuery was not the first but was definitely among the earliest JavaScript libraries ever to be used by developers all over the world to make their jobs easier. Since it was first released, a lot of maintainers and engineers have contributed to making jQuery what it is today – namely, a robust and essential part of the modern internet that offers lots of different functionalities.

jQuery provides, but is not limited to providing, the following features:

- DOM manipulations
- Event handling
- Animated effects and transitions

We will see how to do these things when we look at jQuery in more detail later in this chapter.

Where to Find and How to Use External Code

There are a few different approaches when it comes to including libraries in your program. Depending on those approaches, we get packages from different places.

One is to copy the library's source code and to handle it as we wish. This approach is the most secure in the sense that we have all the control of the software and can customize it to fit our needs. However, by doing so, we give up compatibility and automated updates and patches. Most open-source projects host their code on GitHub or any other version control platform. Therefore, it's rather easy to access and fork the package's code. As soon as we download the source code, we can do whatever we want to get it working with our software. Possible solutions could be hosting it on our **cloud distribution network** (**CDN**) and accessing it from there or bundling it with our source code.

Another approach is downloading the package from a CDN from the client at runtime. The most popular CDN to exclusively host JavaScript libraries is cdnjs.com. It hosts thousands of libraries you can include in your markup without you having to worry about where to store it or how to upgrade it:

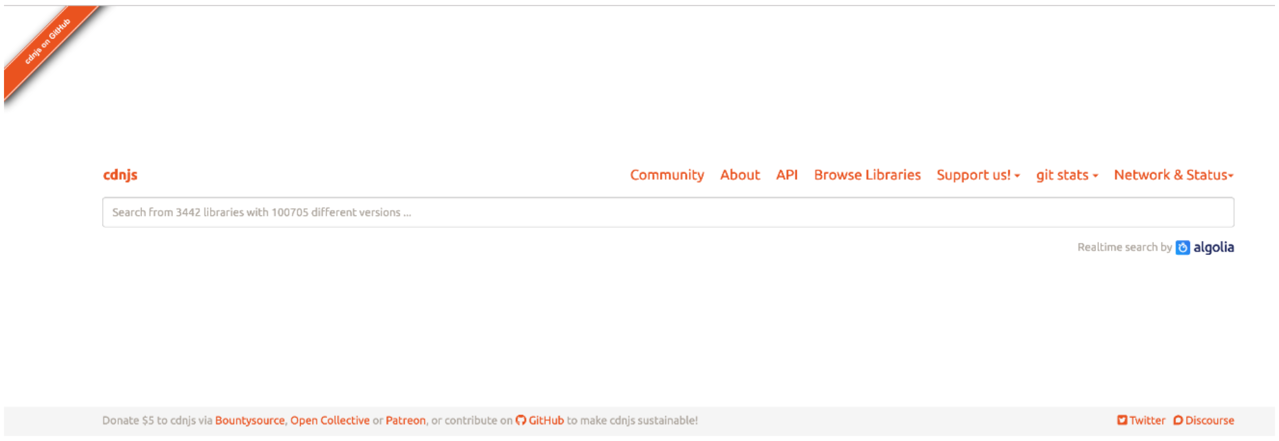

cdnjs on GitHub
cdnjs
Community
About
API
Browse Libraries
Support us!
git stats
Network & Status
Search from 3442 libraries with 100705 different versions ...
Realtime search by algolia
Donate $5 to cdnjs via Bountysource, Open Collective or Patreon, or contribute on GitHub to make cdnjs sustainable!
Twitter
Discourse

Figure 4.2: Downloading a package from cdnjs.com

The following is an example of how you'd include Vue.js with your markup:

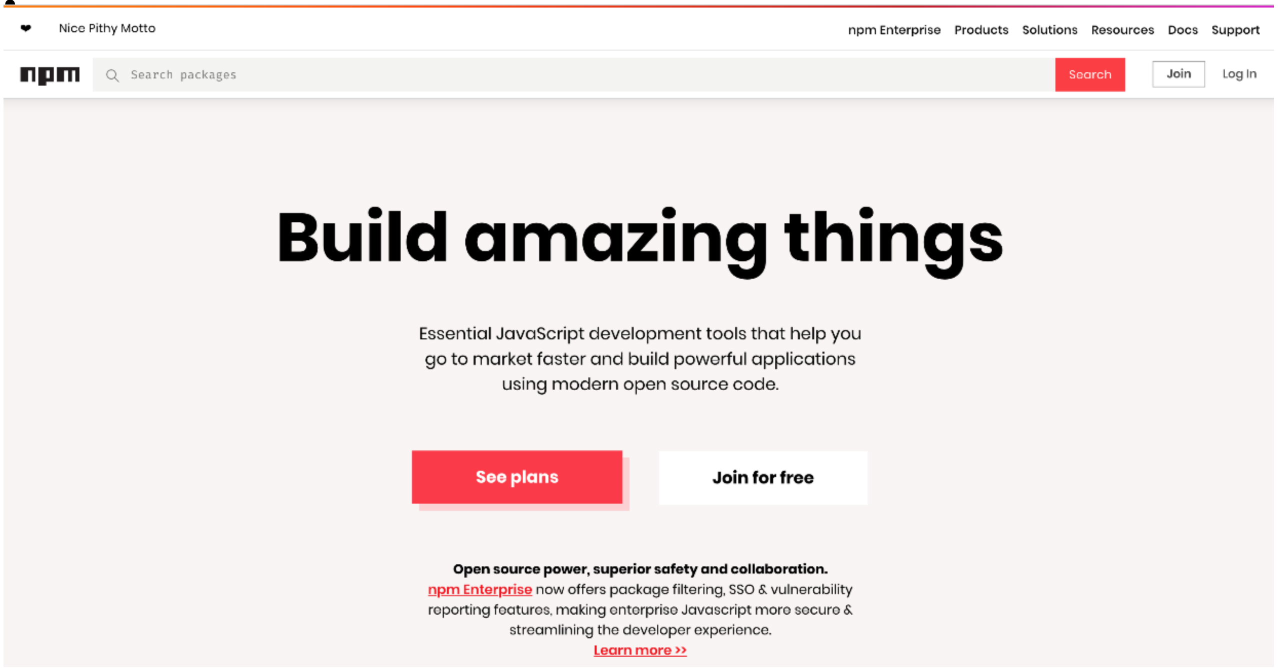
```
// myApplicationMarkup.html
<html>
<script src="https://cdn.com/vue.js"></script>
<script type="text/javascript">
console.log("Vue was loaded: ", !!Vue)
```

```
// => Vue was loaded: true
</script>
</html>
```

> **Note**
>
> If you include packages by loading them from the browser during runtime, you have to be aware of the order of the script tags. They're loaded from top to bottom. Therefore, if you switched the two script tags in the preceding example, console.log would print that there is no `Vue.js` loaded, even though, eventually, it will be.

The previous approach gained lots of popularity and is now by far the most common due to the development of the JavaScript ecosystem in recent years. It involves the **Node.js Package Manager (npm)**. npm is a tool that, as its name suggests, takes care of JavaScript packages within the Node.js ecosystem. npm itself consists of three parts:

- The website npmjs.com, for hosting all the documentation and package searches
- The CLI that gets installed with `Node.js`
- The registry, which is where all of the modules are stored and made installable:



Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

Open source power, superior safety and collaboration.
npm Enterprise now offers package filtering, SSO & vulnerability reporting features, making enterprise Javascript more secure & streamlining the developer experience.

Figure 4.3: NPM website

Using npm requires a **`Node.js`** version to be installed on your machine and any tool to bundle all your JavaScript together to make it executable in the browser.

Then, all you have to do is install any module you can find on npm:

```
// in your terminal
$ npm install <package>
```

This command then stores the package in a particular folder, called **node_modules**. This folder contains all the source code of the libraries you installed, and from there, the bundler will join it into your application during build time.

All of the aforementioned methods to include libraries and frameworks with your source code are valid and have their preferred use cases. However, it's likely that you are going to use the latter the most as new projects are set up within the **`Node.js`** ecosystem, which is where modules and npm come from, naturally. Nonetheless, knowing how to use external resources without npm can come in handy when you want something much more comfortable and quicker than an entire project setup. Therefore, let's perform an exercise in which we will load a third-party library into our code.

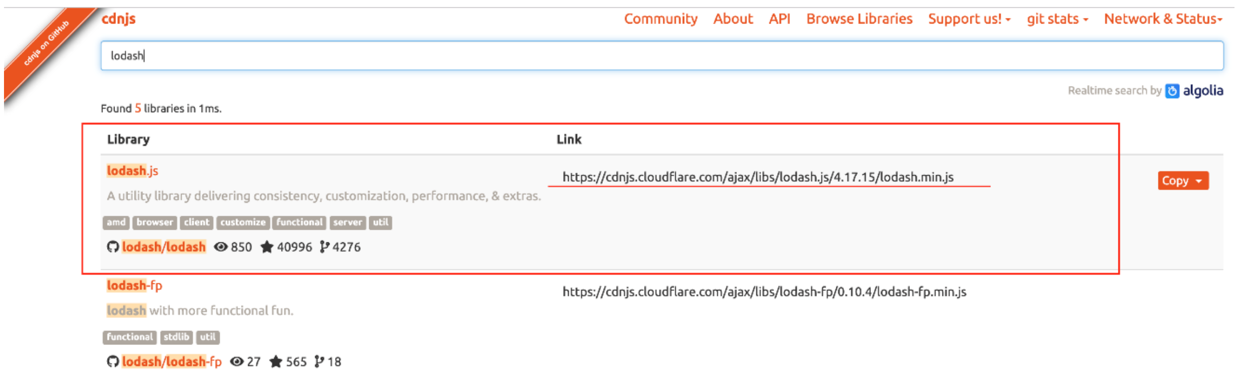
Exercise 4.01: Using a Third-Party Library in Your Code

As we've already discovered, using external software, namely libraries and frameworks, is an extremely useful skill as it can save a lot of resources and help you build highly functional apps. In this exercise, we are going to find and utilize a library ourselves. We'll use the **`lodash`** library to create an array of unique values. Let's get started:

1. Create a new HTML file:

```
<html>
<head></head>
</html>
```

2. Find the CDN URL for the latest **`lodash`** version. To do so, navigate to **`cdnjs.com`** and search for lodash, and then copy the URL highlighted in the figure:

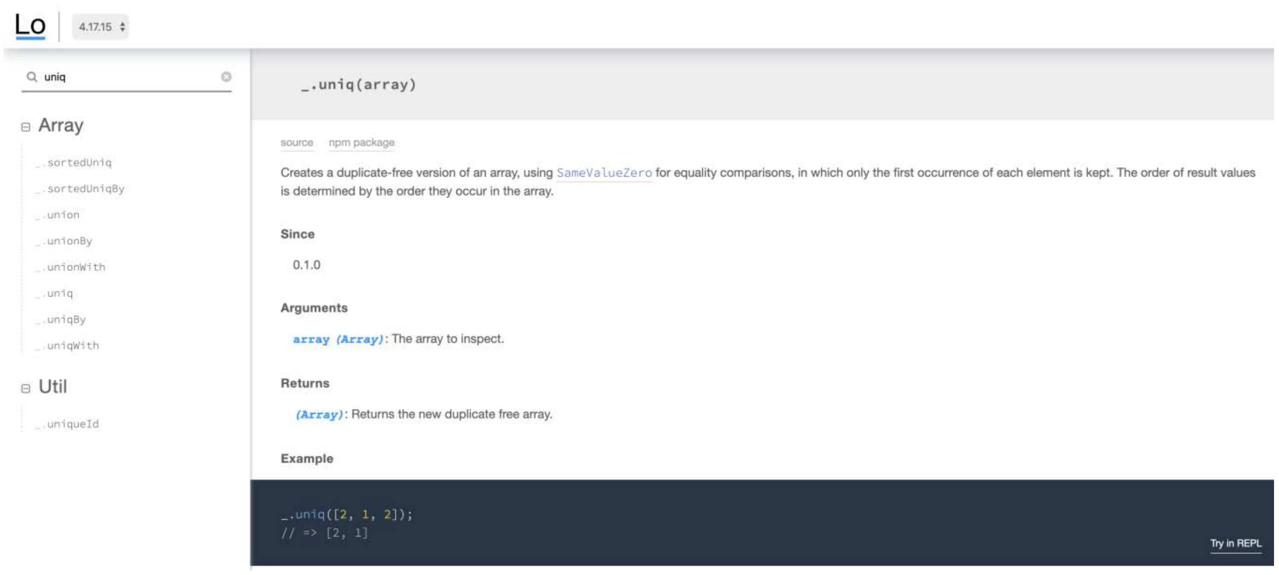


Found 5 libraries in 1ms.

Library	Link
lodash.js A utility library delivering consistency, customization, performance, & extras. amd browser client customize functional server util lodash/lodash 850 40996 4276	https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.15/lodash.min.js
lodash-fp lodash with more functional fun. functional stdlib util lodash/lodash-fp 27 565 18	https://cdnjs.cloudflare.com/ajax/libs/lodash-fp/0.10.4/lodash-fp.min.js

Figure 4.4: Search result of lodash at cdnjs.com

3. To look at the **lodash** documentation, navigate to lodash.com. There, you can use the search bar to find the "**uniq**" function:



```
_.uniq(array)
```

Creates a duplicate-free version of an array, using SameValueZero for equality comparisons, in which only the first occurrence of each element is kept. The order of result values is determined by the order they occur in the array.

Since

0.1.0

Arguments

array (Array): The array to inspect.

Returns

(Array): Returns the new duplicate free array.

Example

```
_.uniq([2, 1, 2]);
// => [2, 1]
```

Figure 4.5: lodash.com documentation for uniq function

4. Load the CDN URL in a script tag's **src** attribute. To do so, paste the URL you previously copied in *step 2*:

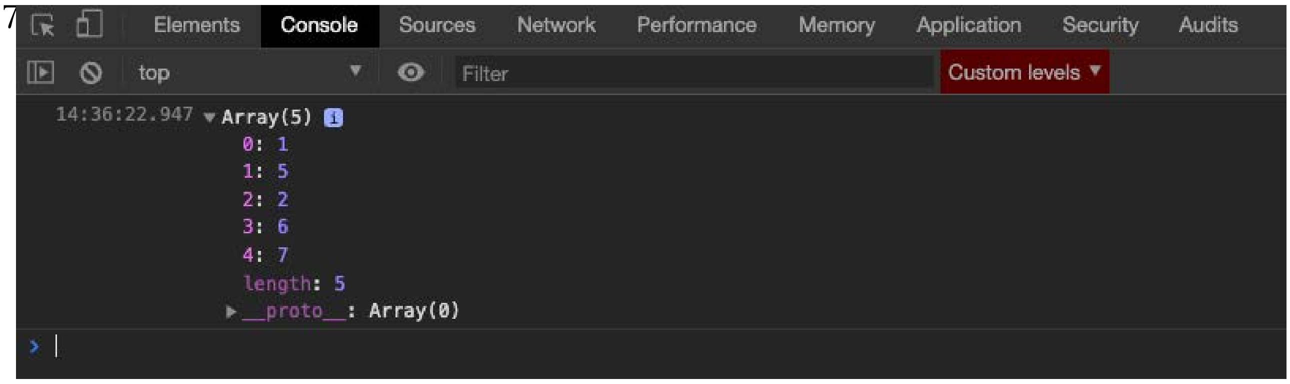
```
<html>
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.15/lodash.min.
    js"></script>
  </head>
</html>
```

5. Create another **script** tag and write JS code using *lodash* to make an array, **[1,5,5,2,6,7,2,1]**, that contains **unique** values:

```
<html>
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.15/lodash.min.
    js"></script><script type=„text/javascript">
      // create an array with duplicated values
      const exampleArray = [1,5,5,2,6,7,2,1];
      // use lodash.uniq to make the array contain unique values
      const uniqueArray = _.uniq(exampleArray);
      // print the unique array to the console
      console.log(uniqueArray);
```

```
          // => [1,5,2,6,7]
        </script>
      </head>
    </html>
```

6. Open your HTML, including the JavaScript, in a browser and verify that you created an array with unique values inside the browser's development tools console:



```
14:36:22.947 ▾Array(5)
    0: 1
    1: 5
    2: 2
    3: 6
    4: 7
    length: 5
  ▸__proto__: Array(0)
```

Figure 4.6: Unique array values in the browser's development tools console

In this exercise, we used the *lodash* library to create an array that contains sole unique values.

jQuery versus Vanilla.js

Earlier, in the *Everlasting jQuery* section of this chapter, we had a look at jQuery and how it has an exceptional standing in the JavaScript community. To demonstrate why libraries and frameworks, but mainly jQuery, became popular, we will compare it to **`Vanilla.js`** (plain JS).

Manipulating the DOM

If we wanted to fade out and then remove one element in plain JavaScript, we would write verbose and less comprehensive code:

```
// Vanilla.js
const styles = document.querySelector('#example').style;
styles.opacity = 1;(function fade() {
styles.opacity -= .1;

styles.opacity< 0
? styles.display = "none"
: setTimeout(fade, 40)
})();
```