learn enough

JAVASCRIPT TO BE DANGEROUS

MICHAEL HARTL

Praise for Learn Enough Tutorials

"I have nothing but fantastic things to say about @LearnEnough courses. I am just about finished with the #javascript course. I must say, the videos are mandatory because @mhartl will play the novice and share in the joy of having something you wrote actually work!"

—Claudia Vizena

"I must say, this Learn Enough series is a masterpiece of education. Thank you for this incredible work!"

—Michael King

"I want to thank you for the amazing job you have done with the tutorials. They are likely the best tutorials I have ever read."

—Pedro Iatzky

```
1
> a.push(4);
2
> a.push("hello, world!");
3
> a;
[ 3, 4, 'hello, world!' ]
> a.pop();
'hello, world!'
```

Unlike strings and arrays, dates have no literal constructor, so we *have* to use **`new`** in this case:

```
> let now = new Date();
> now;
2022-03-16T 19:22:13.673Z
> let moonLanding = new Date("July 20, 1969 20:18");
> now - moonLanding;
1661616253673
```

The result here is the number of milliseconds since the day and time of the first Moon landing (Figure 4.2).[4] (Your results, of course, will vary, because time marches on, and your value for **`new Date()`** will differ.)

As with other JavaScript objects, **`Date`** objects respond to a variety of methods:

```
> now.getYear();        // Gives a weird answer
122
> now.getFullYear();    // This is what we want instead.
2022
> now.getMonth();
2
> now.getDay();
3
```

The first line here shows that sometimes the results of JavaScript methods are confusing, so it's important to be wary and double-check the values by hand from time to time.

Things like the month and day are returned as indices, and like everything in JavaScript they are zero-offset. For example, month **`0`** is January, month **`1`** is February, month **`2`** is March, etc.

4. Image courtesy of Castleski/Shutterstock.

Figure 4.2: Buzz Aldrin and Neil Armstrong somehow got to the Moon (and back!) without JavaScript.

Even though the official international standard is that Monday is the first day, JavaScript follows the American convention of using Sunday instead. We can get the name of the day by making an array of strings for the days of the week, and then using **`getDay()`** as an index in the array with the square bracket notation (Section 3.1):

```
> let daysOfTheWeek = ["Sunday", "Monday", "Tuesday", "Wednesday",
                       "Thursday", "Friday", "Saturday"];
> daysOfTheWeek[now.getDay()];
'Wednesday'
```

Your results will vary, of course, unless you happen to be reading this on a Wednesday.

As a final exercise, let's update our web page with an alert including the day of the week. The code appears in Listing 4.1, with the result as shown in Figure 4.3.

Listing 4.1: Adding a greeting customized to the day of the week.
index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Learn Enough JavaScript</title>
    <meta charset="utf-8">
    <script>
      const daysOfTheWeek = ["Sunday", "Monday", "Tuesday", "Wednesday",
                             "Thursday", "Friday", "Saturday"];
      let now = new Date();
      let dayName = daysOfTheWeek[now.getDay()];
      alert(`Hello, world! Happy ${dayName}.`);
    </script>
  </head>
  <body>

  </body>
</html>
```

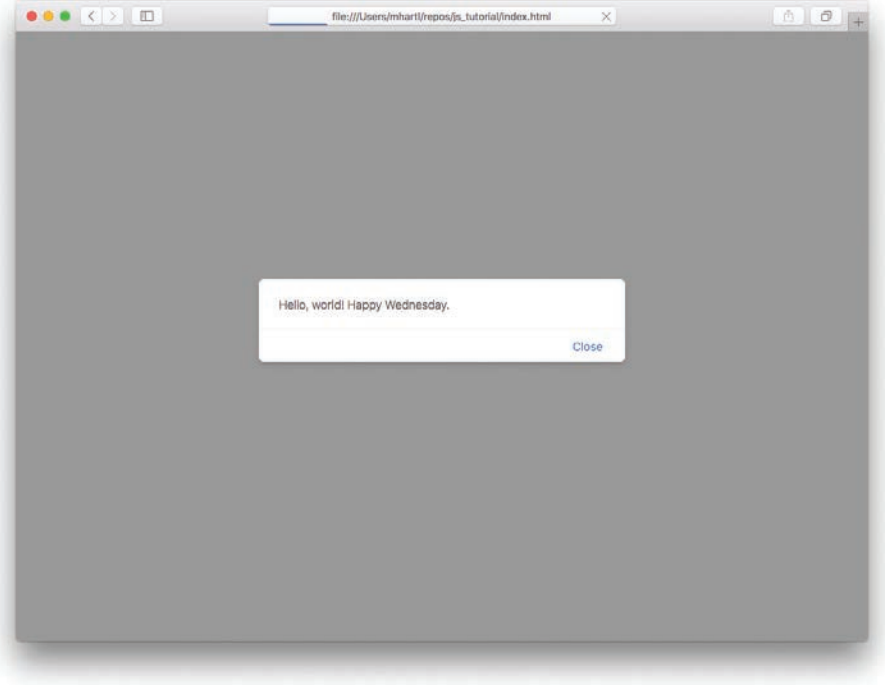

Figure 4.3: A greeting customized just for today.

Note that Listing 4.1 uses `const` instead of `let` when defining `daysOfTheWeek`:

```
const daysOfTheWeek = ["Sunday", "Monday", "Tuesday", "Wednesday",
                       "Thursday", "Friday", "Saturday"];
```

Here `const`, which (as you can probably guess) is short for "constant", gives us a way to indicate that the value of the variable won't change.[5] Some people even go so far as to use `const` in preference to `let` whenever possible. My preference is to use `let` as a default, and to use `const` as a signal that it's especially important for the value not to change.

4.2.1 Exercises

1. Create a new `Date` object by passing it a string for your birthday (including year). JavaScript supports a number of different formats, so it will probably work with whichever date format you prefer. Pretty cool, no?
2. How many seconds after the Moon landing were you born? (Or maybe you were even born *before* the Moon landing—in which case, lucky you! I hope you got to watch it on TV.)

4.3 Regular Expressions

JavaScript has full support for *regular expressions*, often called *regexes* or *regexps* for short, which are a powerful mini-language for matching patterns in text. A full mastery of regular expressions is beyond the scope of this book (and perhaps beyond the scope of human ability), but the good news is that there are many resources available for learning about them incrementally. (Some such resources are mentioned in "Grepping" (https://www.learnenough.com/command-line-tutorial/inspecting_files) in *Learn Enough Command Line to Be Dangerous* (https://www.learnenough.com/command-line) and "Global find and replace" (https://www.learnenough.com/text-editor-tutorial/advanced_text_editing#sec-global_find_and_replace) in *Learn Enough Text Editor to Be Dangerous* (https://www.learnenough.com/text-editor).) The most

5. Technically, `const` creates an *immutable binding*—i.e., the name can't change, but the value can. Mutating the contents of a variable created using `const` is a bad practice, though, and should be avoided to prevent confusion.

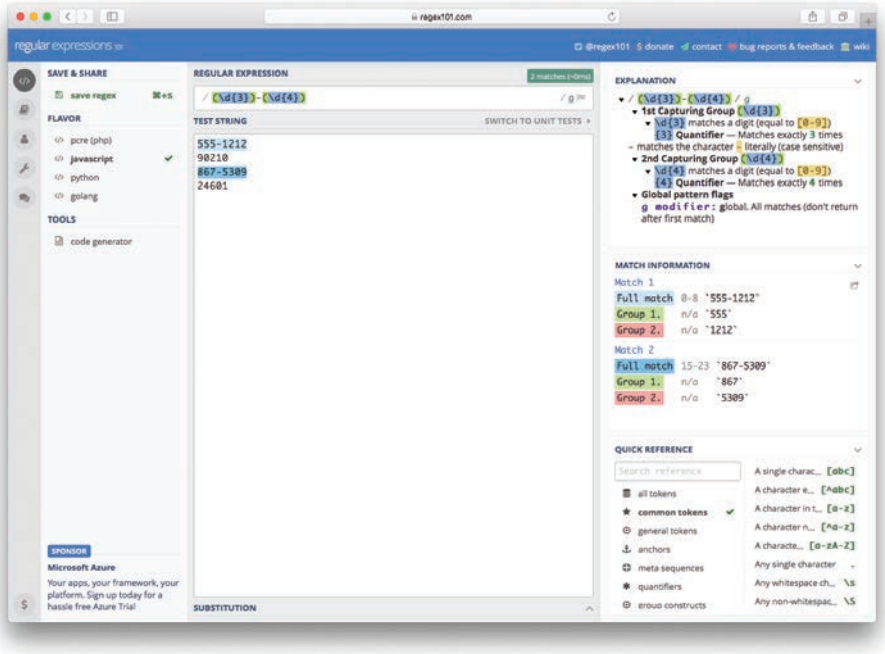

Figure 4.4: An online regex builder.

important thing to know about is the general idea of regular expressions; you can fill in the details as you go along.

Regexes are notoriously terse and error-prone; as programmer Jamie Zawinski famously said:

> Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

Luckily, this situation is greatly ameliorated by web applications like regex101 (https://regex101.com/), which let us build up regexes interactively (Figure 4.4). Moreover, such resources typically include a quick reference to assist us in finding the code for matching particular patterns (Figure 4.5).

If you look carefully at Figure 4.4, you might be able to see the checkmark in the menu on the left indicating that "javascript" has been selected for the regex input format. This arranges to use the exact regular expression conventions we need

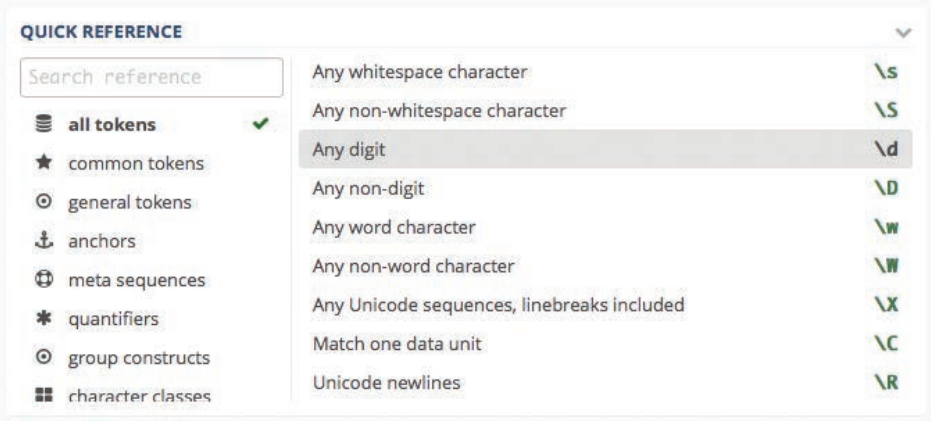

Figure 4.5: A close-up of the regex reference.

in this tutorial. In practice, languages differ little in their implementation of regular expressions, but it's wise to use the correct language-specific settings, and always to double-check when moving a regex to a different language.

Let's take a look at some simple regex matches in JavaScript. Our examples will draw on both the regex methods and string methods specialized for regexes. (The latter are often more convenient in practice.)

4.3.1 Regex Methods

A basic regex consists of a sequence of characters that matches a particular pattern. We can create a new regex using the `new` function (Section 4.2) on the `RegExp` object. For example, here's a regex that matches standard American ZIP codes (Figure 4.6),[6] consisting of five digits in a row:

```
> let zipCode = new RegExp("\\d{5}");
```

Here `\d` represents any digit (0–9), and the first backslash is needed to *escape* the second backslash to get a literal backslash in the string. (We'll see how to avoid

6. Image courtesy of 4kclips/123RF.