



Rev. 08Aug2019

Exercise 16

DATES IN JAVASCRIPT

I'm so tempted to make a pun about JavaScript "dates" and Tender, but I shall eschew the childish impulse... What I'm referring to is working with a date like July 4, 1776. Although JavaScript makes many things easy, apparently the working group for dealing with dates got confused with the ones on Tender. (I'm sorry — I tried. Really, I tried.)

So, let's learn a little about how we can use dates in JavaScript. We'll start with the HTML available for this exercise. There we have `` elements for the current date, and one for a custom formatted date.

- Let's start by creating a current date in `index.js`:

```
let currentDate = new Date()
```

- Using your choice of `document.querySelector('#current-date').innerHTML` or `document.getElementById('current-date').innerHTML`, set the value to `currentDate`. It works...but it's not very attractive.

- Let's fix that by using a function called `toDateString`:

```
let dateString = currentDate.toDateString()
```

- Then output that to the element with an `id` of `current-date`.

- Run the repl. That's good. But we can do even better. Let's say we want to have a date format that looks like this:

Sunday, June 23, 2019

For this, we're going to have to dig deeper into JavaScript.

In a previous exercise, we learned that elements that start on a separate line are called "block-level" elements. Elements that stay on the same line they're placed (like ``) are called "inline" elements.

There are a few useful functions we can use with a date:

This "numbering from 0" really throws people new to programming. Why start from 0? As we get deeper into the course, it will make more sense. For now, just think of it as an "offset" from 0. So the first item in a list has a zero offset. Don't worry: it confuses everyone at first. Once you understand it deeper, you'll be glad they did it this way!

`getFullYear` will return a 4-digit year according to the local time.

● `getMonth` returns the month of the year according to the local time. This one is going to take more work. It returns the month, not as text such as "June", but as a number. And to make it just a little harder, we don't start numbering from 1. We start numbering from 0. So January is not 1, but 0. December is not 12, but 11. And we don't really want a number; we want the textual representation. We'll come back to solve this shortly.

`getDate` returns the day of the month according to local time. Do the days of the month start numbering from 0? No, they start with 1, as you'd expect. Why the difference from `getMonth`? No one knows.

`getDay` returns the day of the week according to local time. This one is like `getMonth`, where it returns a number — and that number starts with 0.

JAVASCRIPT OBJECTS

To solve the problem with `getMonth` and `getDay`, I need to introduce a new idea. It's called a JavaScript object. Let me show you what it looks like:

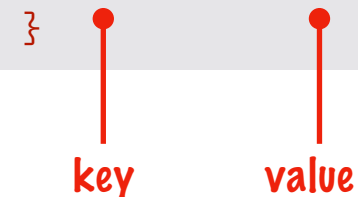
```
let days = {
  0: "Monday",
  1: "Tuesday",
  2: "Wednesday",
  3: "Thursday",
  4: "Friday",
  5: "Saturday",
  6: "Sunday"
}
```

We still have the idea of `let` followed by a variable name (`days` in this case) followed by the `=` symbol. But what is the thing to the right of the `=` symbol, the thing that starts with curly braces? That is a JavaScript object.

It certainly looks...intimidating, but it's really quite simple once you spot the pattern.

And what's the pattern? A JavaScript object holds one or more key:value pairs, separated by commas. Let's look at an object with only one key:value pair:

```
let abbreviations = {
  uncf: "United Nations Children's Fund"
}
```



To access the value, we use a different pattern: *object-name.key*. In the above example, `abbreviations.uncf` will return the text: **United Nations Children's Fund**.

- Try that out. After creating the `abbreviations` variable above, output `abbreviations.uncf` into the `<p id="scratch"></p>` element.

How does this concept of objects help us with our current problem of getting "Monday" — or whatever the day happens to be? Well, I said that `getDay` returns a value and that it is between 0 and 6. Knowing that, we can construct an object where the keys are 0 through 6 and the values are the names of the day corresponding to each number. And that brings us back to the original look we had into objects:

```
let days = {
  0: "Monday",
  1: "Tuesday",
  2: "Wednesday",
  3: "Thursday",
  4: "Friday",
  5: "Saturday",
  6: "Sunday"
}
```

Write that in your `index.js` file. Then output to the `scratch` element the value that belongs to `2` key. Only there's a small problem. Normally, the pattern I gave you is correct: *object-name.key*. But there's a small variant to this. If the key is a number and not text, the pattern changes to this: *object-name[key]*.

```
document.getElementById('scratch').innerHTML = days.2 // not this
document.getElementById('scratch').innerHTML = days[2] // but this
```

- Now try it.

Now, for one last piece to this puzzle: tying the results of `getDay` to the `days` object we created.

- We want a custom date format, beginning with the day of the week. I wrote this code on Sunday, June 23d, 2019. That was my `currentDate`. That means that `currentDate.getDay()` returns 6 (since Sunday is the sixth day, when we begin numbering from 0). Let's write that out in code:

```
let currentDate = new Date() // returns the current date in local time
let dayOfWeekByNum = currentDate.getDay() // returns the number 6 if the day is Sunday
```

- Now, let's find out what the formatted day is:

```
let dayOfWeekInText = days[dayOfWeekByNum] // returns Sunday
```

That's a headscratcher. But walk through the discussion slowly, step-by-step and, above all else, try things out.

- Getting the current month is very similar to getting the current day. Use `getMonth` to get a zero-based number for the current month. But you won't stop there. Create a JavaScript object for `months` that's extremely similar to the one you did for `days`.
- And finally, use concatenation (joining things together) to put together the date like this:

Sunday, June 23, 2019

As always, if you get stuck, consult my solution in [answer.js](#).

Finally, I want to encourage you if you did get stuck and frustrated. This is, by far, the most difficult exercise you've done. If you ended up looking at the answer, don't feel you've failed. You're learning. Sometimes, learning *is* frustrating. That doesn't mean you won't get it; it just means it takes more work on your part. That's literally all it means. No nonsense about not having the programmer gene. Or not being smart enough. There is no programming gene and, if you've made it this far, you're plenty smart.

Just. Don't. Give. Up.

The two forward slashes indicates a comment — something written for the programmer but ignored by the browser. We use comments a lot to remind us of things, explain some difficult-to-understand piece of code, etc.