WISE GUIDES
CAREER CHANGER

Rev. 25Jan2019

Exercise 19

KIND OF A DUMB MOVE EFFECT

In this exercise, we're going to create an effect where clicking an item in column A moves it to column B. Then, clicking an item in column B moves it to column A.

As advertised, this is kind of a dumb move effect. There are much better ones, but we're still in preschool here. And while we may not know a lot, we'd still like to have some fun. And we'll definitely learn something.

Look at the HTML for this exercise. We have two near-identical `<div>`s: `#yours` and `#mine`. Every item in `#mine` has a `class` of `hidden` by default:

```
.hidden {
 display: none;
}
```

You've used the `display` property before when we set it to `inline-block`. When it's set to `none`, the browser ignores the element altogether. It doesn't take up any space on the page. It's as if it doesn't exist — only it does, as we'll see.

What we want to happen is this: when someone clicks an item in `#yours`, we want to have that disappear and the corresponding one in `#mine` to appear. We can do that by adding or removing the `hidden class` from any element. And we implement this with something else we've used before: the `onclick` event handler.

When someone clicks on a visible element (one lacking the `class` value of `hidden`), two things should happen: a `class` of `hidden` should be added to the clicked element (hiding it) and the existing `class` of `hidden` should be removed from the corresponding hidden element.

If we have an event handler, we must have a function that will be called, and we can see that we do.

This is the pattern:

```
onclick="toggle('this-id', 'hidden-id')"
```

Here's the first implementation of that pattern that appears on the page:

```
onclick="toggle('y-bottle', 'm-bottle')"
```

We know a little about functions. Based on what we know, we can say that the `toggle` function accepts two arguments. Let's look at the `toggle` function, found in `index.js`:

```
let toggle = (idToHide, idToShow) => {
 document.getElementById(idToHide).classList.add('hidden')
 document.getElementById(idToShow).classList.remove('hidden')
}
```

See the pattern?

`.classList.add(class-name)` adds the specified *`class-name`* to the found element.

`.classList.remove(class-name)` removes the specified *`class-name`* from the found element.

The effect of this JavaScript is that items appear to move back and forth between `#yours` and `#mine`. Now that you see how the trick is performed, you can see that no moving is involved: we're just hiding/unhiding different elements.

Study `index.css` to spot a couple of things:

1. I'm using the pseudo-class of `:hover` to change the `cursor` to a `pointer` when either `#yours` or `#mine` is moused over.

2. I can specify multiple selectors on the same line by using commas between selectors. You might want to do this in cases where the rules are identical for all selectors thus identified — just to save some typing.

- [ ] Now, your turn: add 3 more elements to both `#yours` and `#mine`, following the pattern established in the existing code. The add/remove functionality should work for the new items too. For this exercise, I haven't provided you with an answer file. I'm confident you can work this out on your own.