useMemo and useCallback

As we saw, useMemo and useCallback help us to optimize and speed up a React application. Our app isn't in need of useMemo or useCallback. But just so you can get some practice with them and see their effects, here are some exercises.

Simulating a slow function

1. Edit Cart.tsx. Add this to the code at the top.

```
//Simulates a slow-running function that returns a value we need
const slowFunction = (val: number | undefined) => {
  console.time('slowFunction')
  for (let i = 0; i < 1e9; i++) { }
  console.timeEnd('slowFunction')
  return val;
}
const slowTip = slowFunction(tip);
```

2. Examine that code briefly. Note that we're incrementing a variable to a billion and then returning the same value supplied. It does nothing but pass the value through very, very slowly.

3. Let's use it. Change the tip input to use slowTip instead of tip:

```
<div>Tip: <input value={slowTip} onChange={e => setTip(+e.target.value)} type="number"
step="0.01" min="0.00" /></div>
```

4. Run and test. Open the console so you can see when and how slowly slowFunction runs.

5. Change the tip in the web app. If you type fast, the page should have trouble keeping up. (If it's still not slow enough, change the "1e9" to a "1e10" or 11 or 12 until it's obviously slow.)

This is to be expected. After all, we're simulating the effects of a large calculation based on the tip value.

But here's where it gets bad.

6. Change the For field. Or the Special Requests field. Or the CVV field. Changing even one character in any of them runs painfully slowly! It's slowing down our whole app! Our plain and fast fields are being hindered by the slow calculation because that calculation needs to happen every time the component is re-rendered.

Let's fix the problem!

7. Wrap the execution of slowFunction in a useMemo() so that it's result will be remembered and only re-run if tip changes.

```
const slowTip = useMemo(() => slowFunction(tip), [tip]);
```

8. Run and test again. Change the tip in the web app. What happens when you change it? __________
   ____
   It should continue to run real slow. And that's fine; remember, we're pretending that the calculation is needed to calculate the tip.

9. Change the CVV or the Deliver to... or the For or the Special Requests. Now what happens? ______
   ____ It is back to running fast again. Why? Because rather than re-calculating the slowTip on every re-render, it only recalculates it when the tip changes.

10. Go back in and remove all that testing code. We won't need it going forward and we want to leave our code clean.

UseCallback

UseCallback is typically run to obviate the need for rerendering a sub component. Our app doesn't have any place where that would be needed but we can still demonstrate how useCallback works and when it is most useful.

11. Edit Cart.tsx. Find the setFirstName function. Move it above the return statement and make it look like this:

```
const random = Math.random();
const setFirstName = function (firstName: string, ci: CartItem) {
  console.log(`set first name ${random}`)
  const newCartItem = { ...ci, firstName }
  changeCartItem(newCartItem);
};
```

12. Now run and test. Put something in your cart. Navigate to Cart/Check out and type your name in the "For" box. Look at the console. What's happening to the number? ______________________________

13. This proves that every time Cart is re-rendered, setFirstName is re-created from scratch. What a waste of resources! It's not doing anything different on each render. Let's useCallback to remember the function.

14. Wrap the function in useCallback()

```
const random = Math.random();
const setFirstName = useCallback(function (firstName: string, ci: CartItem) {
  console.log(`set first name ${random}`)
  const newCartItem = { ...ci, firstName }
  changeCartItem(newCartItem);
}, []);
```

15. Run and test again. This time what happens to the random number? _________________________

This proves that the first time the component is rendered, React memoizes the function definition along with its JavaScript closure and thus the number generated on the first render.

16. Undo all that testing, leaving our project clean again.