Functional Components

Now that you know about state, let's talk about state in functional components. It used to be that you couldn't update state in a functional component but that has since changed with the addition of Hooks to React. Hooks, which we'll talk even more about next week, allow you to do things in functional components that used to be exclusive to classes. You also can abstract out code as needed and clean up your code immensely. Let's take a look at some code WITHOUT the proper design pattern.

Hooks reference: https://www.w3schools.com/react/react_hooks.asp

Link to the useState hook documentation: https://www.w3schools.com/react/react_usestate.asp

```
import React from "react";

const Todo = () => {
  let todos = [
    { id: 1, title: "Hello!", completed: true },
    { id: 2, title: "Hey!", completed: false },
    { id: 3, title: "Bye!", completed: false },
    { id: 4, title: "Huh?!", completed: true },
  ];
  // After 5 seconds it adds to the todos Array
  setTimeout(addTodo, 5000);

  const addTodo = () => {
    // This is technically updating state but not in a way that React can
understand so it doesn't trigger a re-render.
    todos.push({ id: 5, title: "Added!", completed: true });
    // Will show the full array with all 5 even though it won't re-render
    console.log(todos);
  };
  return (
    <div>
      {todos.map((t) => (
        <div key={t.id}>
          {t.title} {t.completed}
        </div>
      ))}
    </div>
  );
};
export default Todo;
```

As you can see above, this is using standard Array functions (push in this case). It still won't re-render on changes though. This is where the `useState` hook comes into play. The `useState` hook is a function that takes an argument (the initial value) and returns an array of items. The first item is the value of state while the second is a function to update state. So how would you initialize and update state then? It might look something like:

```
// useState also has to be imported
import React, { useState } from "react";

const Todo = () => {
  // Creating a starting variable just to clean up the code. This could be used
inline as an argument directly in `useState` but is a bit confusing to read that
way
  let startingTodos = [
    { id: 1, title: "Hello!", completed: true },
    { id: 2, title: "Hey!", completed: false },
    { id: 3, title: "Bye!", completed: false },
    { id: 4, title: "Huh?!", completed: true },
  ];

  // startingTodos is the array of todo objects that we want to initialize state
onto
  // todos is the current state value (initially set to 'startingTodos')
  // setTodos is the name of the function that we will call

  const [todos, setTodos] = useState(startingTodos);

  // After 5 seconds it adds to the todos Array

  const addTodo = () => {
    let newTodo = { id: 5, title: "Added!", completed: true };
    // By using the setTodos function we declared above, we pass it a new array
consisting of what we had and what we want to add and it updates accordingly
    setTodos((todos) => [...todos, newTodo]); // You might also see this as
setTodos((curr) => [...curr, newTodo]); where curr is just the current value of
state
    // Will show the full array and it will have re-rendered
    console.log(todos);
  };
  setTimeout(addTodo, 5000);

  return (
    <div>
      {todos.map((t) => (
        <div key={t.id}>
          {t.title} {t.completed}
        </div>
      ))}
    </div>
  );
};
export default Todo;
```

One thing to note about the `useState` above is as mentioned, it is used for one piece of state. The variable names in the `[]` can be anything you want but usually follows a pattern of:
`const [stateName, setStateName] = useState(initalValue)`

If you wanted to keep track of todos in state, a username in state, and a theme in state you would do so as:

```
// ....
const [todos, setTodos] = useState(initialTodoValue);
const [username, setUsername] = useState(initialUserName);
const [theme, setTheme] = useState(initialTheme);
// ....
```

If you have a bunch of pieces of state, you might even strip out that functionality into it's own custom hook which we'll do next week.

One important thing to note is that the pattern used to update state changes depending on why / how you need to update state. If you need to use the current value of state in order to update to a new version, then the `setTodos((prevTodos) => [...prevTodos, newTodo]);` pattern with `prevTodos` is needed. If you simply need to overwrite the current value, you can just put `setTodos([newTodo])` or whatever you want the new value to be.