React Component Design

Download Demo Code

Goals

- Learn tips for deciding on components & state
- Practice designing a React app!
- Compare different patterns for writing components

Designing Components & State

Designing a React application is a challenging skill that takes lots of practice.

Here are some ideas to begin with.

Components

Generally, components should be small & do one thing

This often makes them more reusable

Example: component that displays a todo w/task could be used in lots of "lists".

"Dumb" Components

Often, small components are simple & don't have state:

```
function Todo(props) {
  return <div className="Todo">{ props.task }</div>;
}
```

This can be used like:

```
function ListOfTodos() {  // ... lots missing
  return (
    <div className="ListOfTodos">
      <Todo task={ todos[0] } />
      <Todo task={ todos[1] } />
      <Todo task={ todos[2] } />
    </div>
  );
}
```

Components like ***Todo*** are called "presentational" or "dumb" *[in a good way!]*

Don't Store Derived Info

If one thing can be calculated from another, don't store both:

```
function TaskList() {
  const [todos, setTodos] = useState(["wash car", "wash cat"]);
  const [numTodos, setNumTodos] = useState(2);

  return (
    <div>
      You have {numTodos} tasks ...
    </div>
  );
}
```

Yuck! Just calculate the number of todos as needed!

Example Design: Dice Game

Let's Design an App!

in App.js

```
<Dice />
```

Should show 6 dice

Value 1-20 generated when button clicked

Should Be Reusable, Flexible

in App.js

```
function App() {
  return (
    <div className="App">
      <Dice />
```

Should be able to control title, num dice to show, and max value

Design

- What components will we need?
- What props will they need?
- What state will we need?

Dice Component

- Props
  - ***title***: title of the game
  - ***numDice***: num of dice to display
  - ***maxVal***: max value of the die
- State
  - ***values***: array of `[val, val, val, ...]` for dice
- Events
  - ***onClick***: re-roll dice and regenerate values in state

Die Component

- Props
  - ***val***: value for this die
- State
  - none!
- Events
  - none!

Patterns for Writing Components

Destructuring Props

As with any other function, we can destructure arguments in our function components.

This is frequently used to destructure props.

what we've been doing:

```
function Dice(props) {
  // we can reference props via
  // props.title, props.numDice, props.
}
```

what we can do:

```
function Dice({ title, numDice, maxVal }) {
  // we can reference props via
  // title, numDice, maxVal
}
```

This often cleans up the code inside of our component.

Setting Default Props

When we destructure props in our component, we can also provide defaults!

This is a nice replacement for ***defaultProps***

what we've been doing:

```
function Dice(props) {
  // ... lots missing
}

Dice.defaultProps = {
  title: "Main Game",
  numDice: 6,
  maxVal: 20
};
```

what we can do:

```
function Dice({
  title = "Main Game",
  numDice = 6,
  maxVal = 20
}) {
  // ... lots missing
}
```

Arrow Functions

Components are just functions. So we can write them with arrow syntax if we choose.

If the component immediately renders, you can make use of an arrow function's implicit return.

what we've been doing

```
function Die(props) {
  return (
    <div className="Die">
```

what we can do:

```
const Die = ({ value }) => (
  <div className="Die">{value}</div>
);
```

Should I use arrow functions for my components?

Not necessarily.

But you'll see them used frequently when looking at documentation, code examples, etc.

Just be consistent!