Available PropTypes

Section 7, Lecture 100

Source: https://reactjs.org/docs/typechecking-with-proptypes.html

```
import PropTypes from 'prop-types';

MyComponent.propTypes = {
  // You can declare that a prop is a specific JS primitive. By default, these
  // are all optional.
  optionalArray: PropTypes.array,
  optionalBool: PropTypes.bool,
  optionalFunc: PropTypes.func,
  optionalNumber: PropTypes.number,
  optionalObject: PropTypes.object,
  optionalString: PropTypes.string,
  optionalSymbol: PropTypes.symbol,

  // Anything that can be rendered: numbers, strings, elements or an array
  // (or fragment) containing these types.
  optionalNode: PropTypes.node,

  // A React element.
  optionalElement: PropTypes.element,

  // You can also declare that a prop is an instance of a class. This uses
  // JS's instanceof operator.
  optionalMessage: PropTypes.instanceOf(Message),

  // You can ensure that your prop is limited to specific values by treating
  // it as an enum.
  optionalEnum: PropTypes.oneOf(['News', 'Photos']),
```

```
.
.    // An object that could be one of many types
.    optionalUnion: PropTypes.oneOfType([
.      PropTypes.string,
.      PropTypes.number,
.      PropTypes.instanceOf(Message)
.    ]),
.
.    // An array of a certain type
.    optionalArrayOf: PropTypes.arrayOf(PropTypes.number),
.
.    // An object with property values of a certain type
.    optionalObjectOf: PropTypes.objectOf(PropTypes.number),
.
.    // An object taking on a particular shape
.    optionalObjectWithShape: PropTypes.shape({
.      color: PropTypes.string,
.      fontSize: PropTypes.number
.    }),
.
.    // You can chain any of the above with `isRequired` to
   make sure a warning
.    // is shown if the prop isn't provided.
.    requiredFunc: PropTypes.func.isRequired,
.
.    // A value of any data type
.    requiredAny: PropTypes.any.isRequired,
.
.    // You can also specify a custom validator. It should
   return an Error
.    // object if the validation fails. Don't `console.warn`
   or throw, as this
.    // won't work inside `oneOfType`.
.    customProp: function(props, propName, componentName) {
.      if (!/matchme/.test(props[propName])) {
.        return new Error(
.          'Invalid prop `' + propName + '` supplied to' +
.          ' `' + componentName + '`. Validation failed.'
.        );
```

```
      }
    },

    // You can also supply a custom validator to `arrayOf` and `objectOf`.
    // It should return an Error object if the validation fails. The validator
    // will be called for each key in the array or object. The first two
    // arguments of the validator are the array or object itself, and the
    // current item's key.
    customArrayProp: PropTypes.arrayOf(function(propValue, key, componentName, location, propFullName) {
      if (!/matchme/.test(propValue[key])) {
        return new Error(
          'Invalid prop `' + propFullName + '` supplied to' +
          ' `' + componentName + '`. Validation failed.'
        );
      }
    })
  };
```

Requiring Single Child

With `PropTypes.element` you can specify that only a single child can be passed to a component as children.

```
  import PropTypes from 'prop-types';

  class MyComponent extends React.Component {
    render() {
      // This must be exactly one element or it will warn.
      const children = this.props.children;
      return (
        <div>
          {children}
        </div>
```

```
      );
    }
  }

  MyComponent.propTypes = {
    children: PropTypes.element.isRequired
  };
```

Default Prop Values

You can define default values for your `props` by assigning to the special `defaultProps` property:

```
  class Greeting extends React.Component {
    render() {
      return (
        <h1>Hello, {this.props.name}</h1>
      );
    }
  }

  // Specifies the default values for props:
  Greeting.defaultProps = {
    name: 'Stranger'
  };

  // Renders "Hello, Stranger":
  ReactDOM.render(
    <Greeting />,
    document.getElementById('example')
  );
```

The `defaultProps` will be used to ensure that `this.props.name` will have a value if it was not specified by the parent component.
The `propTypes` typechecking happens after `defaultProps` are resolved, so typechecking will also apply to the `defaultProps`.