# React.js

## Props & States

Leandro.frigerio@devugees.org

# Props vs. State

**Props**

*What does “props” mean?*
“props” is short for “properties”

*What makes props special?*
the property is passed to the component, similar to how an argument is passed to a function.

Example: https://codepen.io/anon/pen/aByERM?editors=1011

# Default Props

Now the “props as arguments” comparison is even clearer.

OK, so props “come from above.”

Often, but not always. A component can also have **default props**, so if a prop isn’t passed through it can still be set.

**Props cannot change**
During a component’s life cycle props should not change (consider them immutable).

Since props are passed in, and they cannot change, you can think of any React component that only uses props (and not state) as “pure,” that is, it will always render the same output given the same input. This makes them really easy to test - win!

# State

Like **props**, **state** holds information about the component.
However, the kind of information and how it is handled is different.

**By default, a component has no state.**

*So when would you use state?*

When a component needs to keep track of information between renderings the component itself can create, update, and use state.

We'll be working with a fairly simple component to see state working in action.

**Example:** https://codepen.io/lbain/pen/ENpzBZ

# State is created in Constructor()

State is created in the component's **constructor**. This is where state gets it's initial data. The inital data can be hard coded (as above), but it can also come from props! You can't change props but you can set states!!!

```
class Button extends React.Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
  }
```

## State is changeable:

state is updated with **setState**

```
  updateCount() {
    this.setState((prevState, props) => {
      return { count: prevState.count + 1 }
    });
  }
```

You can see we have access to **prevState** within the callback, this will contain the previous state, even if the state has already been updated somewhere else.
**setState** updates the state object and re-renders the component automagically.

# setState: WARNINGS

**setState warning one!**

It is tempting to write *this.state.count = this.state.count + 1.*

Do not do this!

React cannot listen to the state getting updated in this way, so your component will not re-render.

Always use setState.

**setState warning two!**

It might also be tempting to write something like this:

```
// DO NOT USE
this.setState({
  count: this.state.count + 1
});
```

Although this might look reasonable, doesn't throw errors, and you might find examples that use this syntax **online** ~~(stack & tutorials)~~, it is wrong.
This does not take into account the asychronous nature that setState can use and might cause errors with out of sync state data.

this is correct: this.setState(prevState => { return {counter: prevState.counter + 1}});

# Props in Initial State

https://reactjs.org/docs/react-without-es6.html (Initial State)

From docs:

*Using props to generate state in getInitialState often leads to duplication of "source of truth", i.e. where the real data is. This is because getInitialState is only invoked when the component is first created.*

The danger is that if the props on the component are changed without the component being 'refreshed', the new prop value will never be displayed because the constructor function (or getInitialState) will never update the current state of the component. The initialization of state from props only runs when the component is first created.

# Props+States but Anti-pattern

This a case of ‘anti-pattern’ because it's initializing state in the component with props.

**You won't do that!**

React Anti-Patterns: Props in Initial State

Why is passing the component initial state a prop an anti-pattern?

**Bad**

```
class SampleComponent extends Component {
  // constructor function (or getInitialState)
  constructor(props) {
    super(props);
    this.state = {
      flag: false,
      inputVal: props.inputValue
    };
  }

  render() {
    return <div>{this.state.inputVal && <AnotherComponent/>}</div>
  }
}
```

**Good**

```
class SampleComponent extends Component {
  // constructor function (or getInitialState)
  constructor(props) {
    super(props);
    this.state = {
      flag: false
    };
  }

  render() {
    return <div>{this.props.inputValue && <AnotherComponent/>}</div>
  }
}
```

# Props+States: componentWillReceiveProps()

**Updating states with componentWillReceiveProps() the right way**

*"Use this as an opportunity to react to a prop transition before render() is called by updating the state using this.setState(). The old props can be accessed via this.props. Calling this.setState() within this function will not trigger an additional render"*

https://facebook.github.io/react/docs/component-specs.html#updating-componentwillreceiveprops

**componentWillReceiveProps**() is invoked before a mounted component receives new props. If you need to update the state in response to prop changes, you may compare **this.props** and **nextProps** and perform state transitions using **this.setState()** in this method.

```
componentWillReceiveProps(newProps) {
    this.setState({items: newProps.items});
}
```

# Review

While props and state both hold information relating to the component, they are used differently and should be kept separate.

props contains information set by the parent component (although defaults can be set) and should not be changed.

state contains “private” information for the component to initialise, change, and use on it’s own.

**props** are a way of passing data from parent to child.
**state** is reserved only for interactivity, that is, data that changes over time.

# Resources

- Amazing guide on state vs. props
  https://github.com/uberVU/react-guide/blob/master/props-vs-state.md
- Stackoverflow question
  https://stackoverflow.com/questions/23481061/reactjs-state-vs-prop
- Docs on Components and Props
  https://reactjs.org/docs/components-and-props.html
- Docs on State and Lifecycle
  https://reactjs.org/docs/state-and-lifecycle.html
- Lucy Bain React Blog
  http://lucybain.com/blog/2016/react-state-vs-pros/
- Component will receive props
  https://reactjs.org/docs/react-component.html#componentwillreceiveprops