React.js Cheat sheet

QUICK LEARNING

Components

```
import React from 'react’
 import ReactDOM from 'react-dom'
```

```
class Hello extends React.Component {
render ()
{ return <div className='message-box'> Hello {this.props.name}
 </div> }
 }
```

```
const el = document.body
const el = document.body ReactDOM.render(<Hello name='John' />, el)
```

Use the React.js jsfiddle to start hacking. (or the unofficial jsbin)

Import Multiple Exports

```
import React, {Component} from 'react’
 import ReactDOM from 'react-dom'
```

```
class Hello extends Component
 {
 …
}
```

Properties

```
<Video fullscreen={true} autoplay={false} />
```

```
render () {
 this.props.fullscreen const
{ fullscreen, autoplay } = this.props
...
}
```

Use this.Props to Access Properties Passed to the compentent

Children

```
<AlertBox> <h1>You have pending notifications</h1> </AlertBox>
```

```
class AlertBox extends Component {
render () {
 return <div className='alert-box'>
 {
this.props.children}
</div>
}
}
```

Children are passed as Child Property

States

```
constructor(props)
 {
super(props)
 this.state = { username: undefined
 }
}
this.setState({ username: 'rstacruz’ })

render ()
 { this.state.username const { username } = this.state ··· }
```

Use this.State to manage Dynamic Data With Babel you can use proposal-class-fields and get rid of constructor

```
class Hello extends Componen
t { state = { username: undefined }; ... }
```

Nesting

```
class Info extends Component {
render ()

{ const { avatar, username } = this.props return
 <div>

<UserAvatar src={avatar} />

 <UserProfile username={username} />
</div>
 }
}
```

As of React v16.2.0, fragments can be used to return multiple children without adding extra wrapping nodes to the DOM.

States

```
constructor(props)
 {
super(props)
 this.state = { username: undefined
 }
}
this.setState({ username: 'rstacruz’ })

render ()
 { this.state.username const { username } = this.state ··· }
```

Use this.State to manage Dynamic Data With Babel you can use proposal-class-fields and get rid of constructor

```
class Hello extends Componen
t { state = { username: undefined }; ... }
```

Nesting

```
import React,
{ Component,
 Fragment
 } from 'react’
 class Info extends Component {
render () {
const { avatar, username } = this.props
return
( <Fragment>
 <UserAvatar src={avatar} />
 <UserProfile username={username} />
</Fragment> )
}
}
```

Nest components to separate concerns.

Setting Default Props

```
Hello.defaultProps = { color: 'blue' }
```

See: defaultProps

Setting Default States

```
class Hello
 extends Component
 { constructor (props)
{ super(props) this.state = { visible: true }
}
}
```

See: defaultProps

Functional Components

```
function MyComponent
({ name })
{
 return
 <div className='message-box'> Hello {name}
 </div>
}
```

Functional components have no state. Also their props are passed as the First Parameter to the Function.

Pure Components

```
import React, {PureComponent} from 'react’
class MessageBox
extends PureComponent
{
…
}
```

Components API

```
this.forceUpdate()

 this.setState({ … })

this.setState(state => { … })

this.state this.props
```

State Hooks

```
import React, { useState } from 'react’;
 function Example()
{ // Declare a new state variable, which we'll call "count" const [count,
setCount] = useState(0);
 return
 (
 <div> <p>You clicked {count} times</p> <button onClick={() =>
setCount(count + 1)}> Click me </button> </div>
);
 }
```

Declare Multiple State variables

```
function ExampleWithManyStates()
{ // Declare multiple state variables!
const [age, setAge] = useState(42);
const [fruit, setFruit] = useState('banana’);
const [todos, setTodos] = useState([{ text: 'Learn Hooks’ }
]);
//
…
}
```