REACT

composable components

HELLO WORLD

```html
<div id="root"></div>
```

```jsx
ReactDOM.render(
  <h1>Hello, world!</h1>,
  document.getElementById('root')
);
```

CodePen

https://reactjs.org/docs/hello-world.html

JSX

Syntax extension to JavaScript that produces produces React **"elements"**

```
const element = <h1>Hello, world!</h1>;
```

Comes with the **full power** of JavaScript!

```
const name = 'Deniz Arsan';
const element = <h1>Hello, {name}</h1>;
```

https://reactjs.org/docs/introducing-jsx.html

JSX

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Deniz',
  lastName: 'Arsan'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

You can put any valid JavaScript **expression** inside the curly braces in JSX

https://reactjs.org/docs/introducing-jsx.html

JSX

```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

JSX is an **expression** too

```
const element = <img src={user.avatarUrl} />;
```

Can be used to specify **attributes**

https://reactjs.org/docs/introducing-jsx.html

JSX

```
const element = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);
```

1 **Babel** compiles JSX down to **React.createElement()** calls

2 **React.createElement()** creates **React elements**

```
const element = React.createElement(
  'h1',
  { className: 'greeting' },
  'Hello, world!'
);
```

```
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

https://reactjs.org/docs/introducing-jsx.html

RENDERING ELEMENTS

```
let counter = 0;

function Timer() {
  return (
    <div>
      <h1>You loaded this page {counter} seconds ago.</h1>
    </div>
  );
}

function tick() {
  ReactDOM.render(<Timer />, document.getElementById('root'));
  counter = counter + 1;
}

setInterval(tick, 1000);
```

```
<div id="root"></div>
```

CodePen

https://reactjs.org/docs/rendering-elements.html

COMPONENTS

There are two ways to **declare** a component:

```
function Greeter(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

with **JS functions**

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

with **ES6 Classes**

https://reactjs.org/docs/components-and-props.html

COMPONENTS

Here's how to **render** a component:

```
function Greeter(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Greeter name="Deniz" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

1. Call **`ReactDOM.render()`** with **`<Greeter name="Deniz />`**
2. React calls **`Greeter`** with **`{ name: "Deniz" }`**
3. Greeter returns **`<h1>Hello, Deniz </h1>`**
4. ReactDOM updates the DOM

https://reactjs.org/docs/components-and-props.html

COMPONENTS

We can use components inside other components too.

This is called **composing**:

```
function Greeter(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Greeter name="Deniz" />
      <Greeter name="Ali" />
      <Greeter name="Carl" />
      <Greeter name="Sophia" />
    </div>
  );
}
```

```
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

https://reactjs.org/docs/components-and-props.html

COMPONENTS

IMPORTANT RULE

All React components must act like **pure functions** with respect to their props.

pure

```
function sum(a, b) {
  return a + b;
}
```

VS

impure

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

https://reactjs.org/docs/components-and-props.html

STATE

```
let counter = 0;

function Timer() {
  return (
    <div>
      <h1>You loaded this page {counter} seconds ago.</h1>
    </div>
  );
}

function tick() {
  ReactDOM.render(<Timer />, document.getElementById('root'));
  counter = counter + 1;
}

setInterval(tick, 1000);
```

We want to make this Timer **reusable** and **encapsulated**.

It needs **set up its own timer** and **update itself**.

https://reactjs.org/docs/state-and-lifecycle.html

STATE

1 Create a **stateful** component that keeps track of its state

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0
    };
  }

  render() {
    return (
      <h1>
        You loaded this page {this.state.counter} seconds ago.
      </h1>
    );
  }
}
```

https://reactjs.org/docs/state-and-lifecycle.html

STATE

2 Use **lifecycle methods** to change the state

```
componentDidMount() {
  this.timerID = setInterval(
    () => this.tick(),
    1000
  );
}

componentWillUnmount() {
  clearInterval(this.timerID);
}
```

```
tick() {
  this.setState((state, props) => ({
    counter: state.counter + 1
  }));
}
```

https://reactjs.org/docs/state-and-lifecycle.html

LIFECYCLE METHODS

`componentDidMount()`

Invoked immediately after a component is inserted into the tree.

Initialization that requires DOM nodes should go here

`componentDidUpdate()`

Invoked immediately after updating occurs.

This method is not called for the initial render.

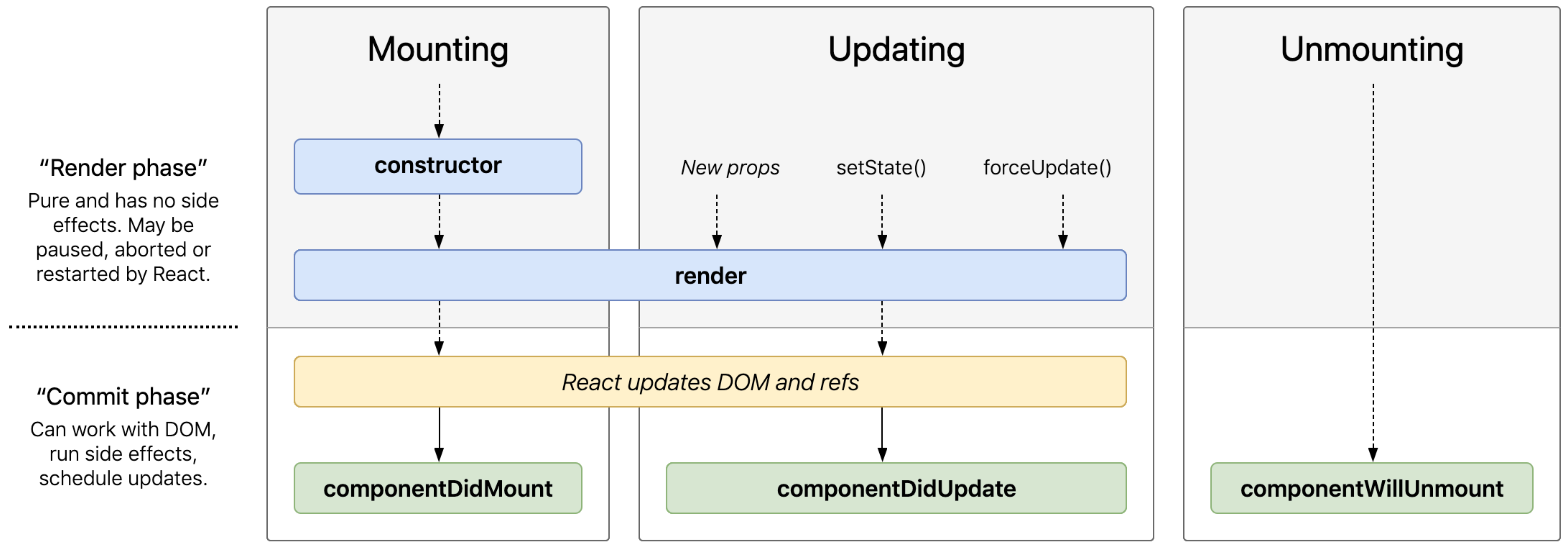
`componentWillUnmount()`

Invoked immediately before a component is unmounted and destroyed.

Any cleanup should go here

https://reactjs.org/docs/react-component.html

COMPONENT LIFECYCLE



setState()

Used to **update** the state

1 Do not update the state directly

```
this.state.name = 'Deniz';          // Wrong
this.setState({ name: 'Deniz' }); // Correct
```

2 Updates may be asynchronous

```
this.setState({ counter: this.state.counter + 1 });                  // Wrong
this.setState((state, props) => ({ counter: state.counter + 1})); // Correct
```

3 Updates are merged

```
this.state = { name: '', title: '' };
this.setState({ name: 'Deniz' });
```

https://reactjs.org/docs/state-and-lifecycle.html

Timer Component with State

CodePen

EVENTS

1 Create handler in the component

```
handleClick() {
  this.setState(state => ({ isActive: !state.isActive }));
}
```

2 Assign handler to event in the element

```
<button onClick={this.handleClick}>
```

3 Make **this** refer to the component

```
this.handleClick = this.handleClick.bind(this);
```

CodePen

https://reactjs.org/docs/handling-events.html

EVENTS

Resolving **this**

```
this.handleClick = this.handleClick.bind(this);
```

Bind it in the **constructor**

OR

```
<button onClick={(e) => this.handleClick(e)}>
```

Use **arrow functions** when assigning

OR

```
handleClick = () => {...}
```

Use **class fields syntax** when declaring

https://reactjs.org/docs/handling-events.html

EVENTS

Passing Arguments

arrow functions

```
<button onClick={(e) => this.handleClick(id, e)}>Go</button>
```

bind in element

```
<button onClick={this.handleClick.bind(this, id)}>Go</button>
```

https://reactjs.org/docs/handling-events.html

RENDERING LISTS

.map in jsx

```
function List(props) {
  const { numbers } = props;
  return (
    <ul>
      {numbers.map((number) =>
        <Item
         key={number.toString()}
         value={number}
        />
      )}
    </ul>
  );
}
```

```
function Item(props) {
  return <li>{props.value}</li>;
}

const numbers = [1, 2, 3, 4, 5];

ReactDOM.render(
  <List numbers={numbers} />,
  document.getElementById('root')
);
```

unique keys

CodePen

https://reactjs.org/docs/lists-and-keys.html

CONTROLLED COMPONENTS

`<input>`, **`<textarea>`**, and **`<select>`** maintain their own state

React components keep their own state and update it with **`setState()`**

Controlled components allow us to have a single source of truth - React controls the value of a form element

```
handleChange(event) {
  this.setState({ value: event.target.value });
}

<input
  type="text"
  value={this.state.value}
  onChange={this.handleChange} />
```

CodePen

https://reactjs.org/docs/forms.html

demo

https://gitlab.com/uiuc-web-programming/react-demo

RESOURCES

Step-by-step guide

https://reactjs.org/docs/hello-world.html

Learn-by-doing Guide

https://reactjs.org/tutorial/tutorial.html

NEXT CLASS:
REACT STATE/ROUTE MANAGEMENT

https://uiuc-web-programming.gitlab.io/fa21