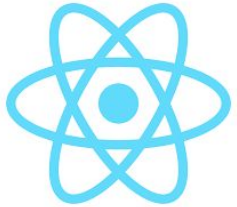
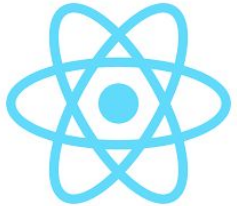


React-2

Nesli Erdoğmuş

JSX

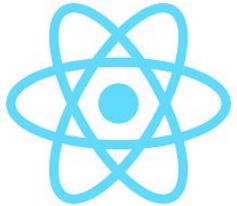


- Embedding Expressions in JSX:
  - You can put any valid JavaScript expression inside the curly braces in JSX.

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

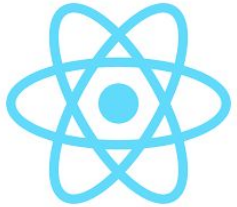
JSX

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

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

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

ReactDOM.render(
  element,
  document.getElementById('root')
);
```



We can embed the result of calling a JavaScript function, `formatName(user)`, into an `<h1>` element.

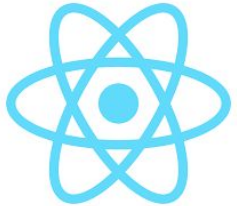
JSX



- JSX inside of if statements and for loops:
  - After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

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

JSX



- Specifying Attributes:
  - With string literals

```
const element = <div tabIndex="0"></div>;
```

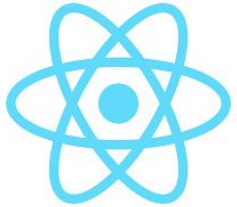
  - With JS expressions

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

- Specifying Children:

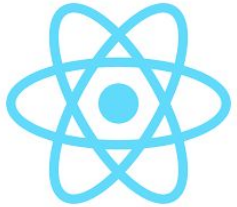
```
const element = (
    <div>
        <h1>Hello!</h1>
        <h2>Good to see you here.</h2>
    </div>
);
```

Elements



- Think of a component in React as a JavaScript Function that it’s return value is a UI.
  - Reusable
  - Can take in parameters
- Components are made of React “elements”.
  - Immutable
  - A React element is essentially a JavaScript object containing a type and props object.
  - React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.

Elements



```
const myElement = (
  <div foo="bar">
    <button>Test</button>
    <span>Hello</span>
  </div>
)

// Simplify myElement and print it to the console
console.log(JSON.parse(JSON.stringify(myElement)))
```

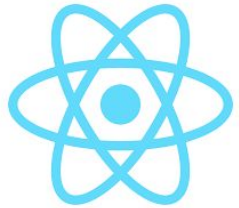
```
▸{type: "div", key: null, ref: null, props: Object, _owner: null}
        type: "div"
        key: null
        ref: null
      ▸props: Object
            foo: "bar"
          ▸children: Array(2)
              ▸0: Object
                  type: "button"
                  key: null
                  ref: null
                ▸props: Object
                      children: "Test"
                   owner: null
              ▸1: Object
                  type: "span"
                  key: null
                  ref: null
                ▸props: Object
                      children: "Hello"
                   _owner: null
    _owner: null
```

Components

```
<html>
  <body>
    <div id="root"></div>
  </body>
  <script src="https://unpkg.com/react@16.13.1/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16.13.1/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone@7.9.3/babel.js"></script>

  <script type="text/babel">

    function Message({children}){
      return <div className="message">{children}</div>
    }

    const element =  (
      <div className="container">
        <Message>Hello World</Message>
        <Message>Goodbye World</Message>
      </div>
    )

    ReactDOM.render(element, document.getElementById("root"))

  </script>
</html>
```

- a function: `Message`
- destructured input: `children`
  - regular HTML tag
  - Whatever content in between the <Message> and </Message> will be interpolated into the {children}.
- props

Components

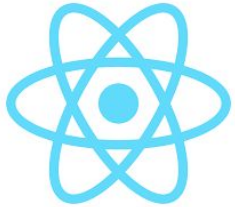


```
const element = <Welcome name="Sara" />;
```

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Function components

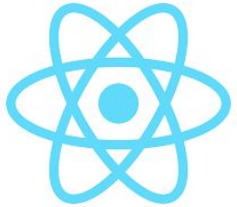
```
function Welcome(props) {
     return <h1>Hello, {props.name}</h1>;
}
```

Components



- Components accept arbitrary inputs and return React elements describing what should appear on the screen.
- When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object “props”.

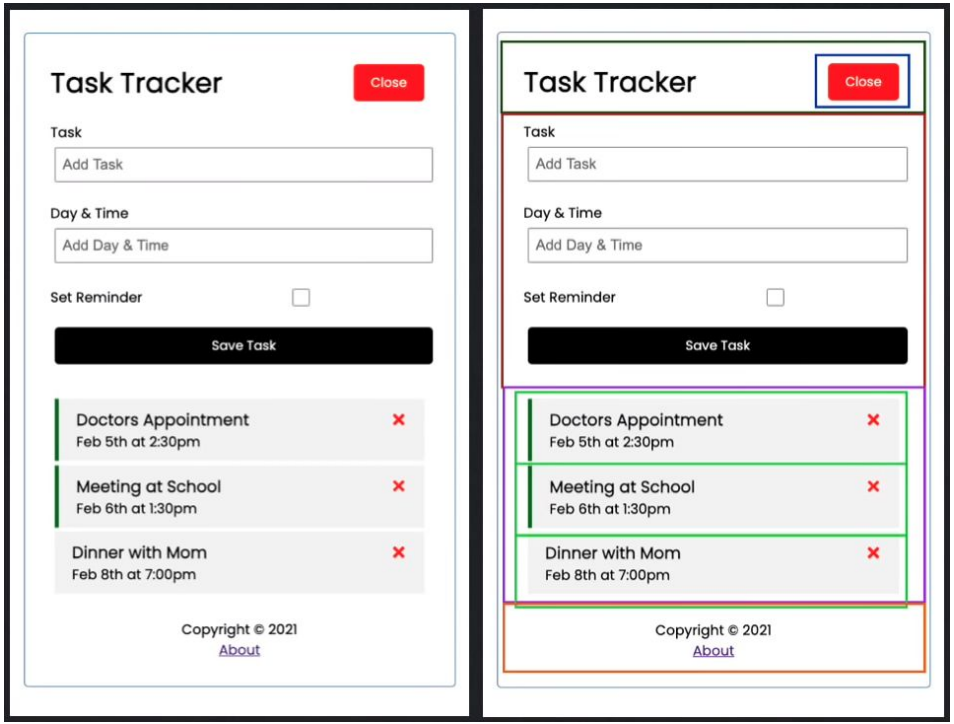
Components



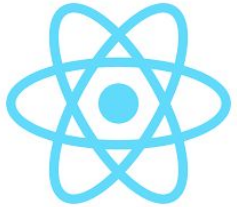
- Defining React Components and rendering them within each other makes Composition in React possible.
- If you want to render a React Component inside a Function Component, you define another component and render it as HTML element with JSX within the other component's body:

```
function App() {
  return <Headline />;
}

function Headline() {
  const greeting = 'Hello Function Component!';

  return <h1>{greeting}</h1>;
}
```

Components

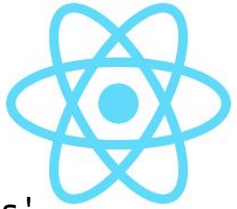

Task Tracker
Close
Task
Add Task
Day & Time
Add Day & Time
Set Reminder
Save Task
Doctors Appointment
Feb 5th at 2:30pm
Meeting at School
Feb 6th at 1:30pm
Dinner with Mom
Feb 8th at 7:00pm
Copyright © 2021
About
Task Tracker
Close
Task
Add Task
Day & Time
Add Day & Time
Set Reminder
Save Task
Doctors Appointment
Feb 5th at 2:30pm
Meeting at School
Feb 6th at 1:30pm
Dinner with Mom
Feb 8th at 7:00pm
Copyright © 2021
About

Components



- Typically, new React apps have a single App component at the very top.
- However, if you integrate React into an existing app, you might start bottom-up with a small component like Button and gradually work your way to the top of the view hierarchy.
- Don’t be afraid to split components into smaller components.
    - A good rule of thumb is that if a part of your UI is used several times (Button, Panel, Avatar), or is complex enough on its own (App, FeedStory, Comment), it is a good candidate to be extracted to a separate component.

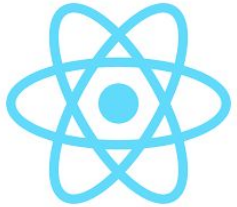
Components



- You can assign the special propTypes property using prop-types (if you don’t use JavaScript extensions like Flow or TypeScript for type-checking)
  - MyComponent.propTypes
- You can define default values for your props by assigning to the special defaultProps property
  - MyComponent.defaultProps

```
import PropTypes from 'prop-types'

function HelloWorldComponent({ name }) {
     return (
          <div>Hello, {name}</div>
     )
}

HelloWorldComponent.propTypes = {
     name: PropTypes.string
}

HelloWorldComponent.defaultProps = {
     name: 'stranger'
}

export default HelloWorldComponent
```

Components



- Props are Read-Only!
  - All React components must act like pure functions with respect to their props.
- Props are only used to pass data from one component to another component React, but only from parent to child components down the component tree.

- How to create dynamic UIs?