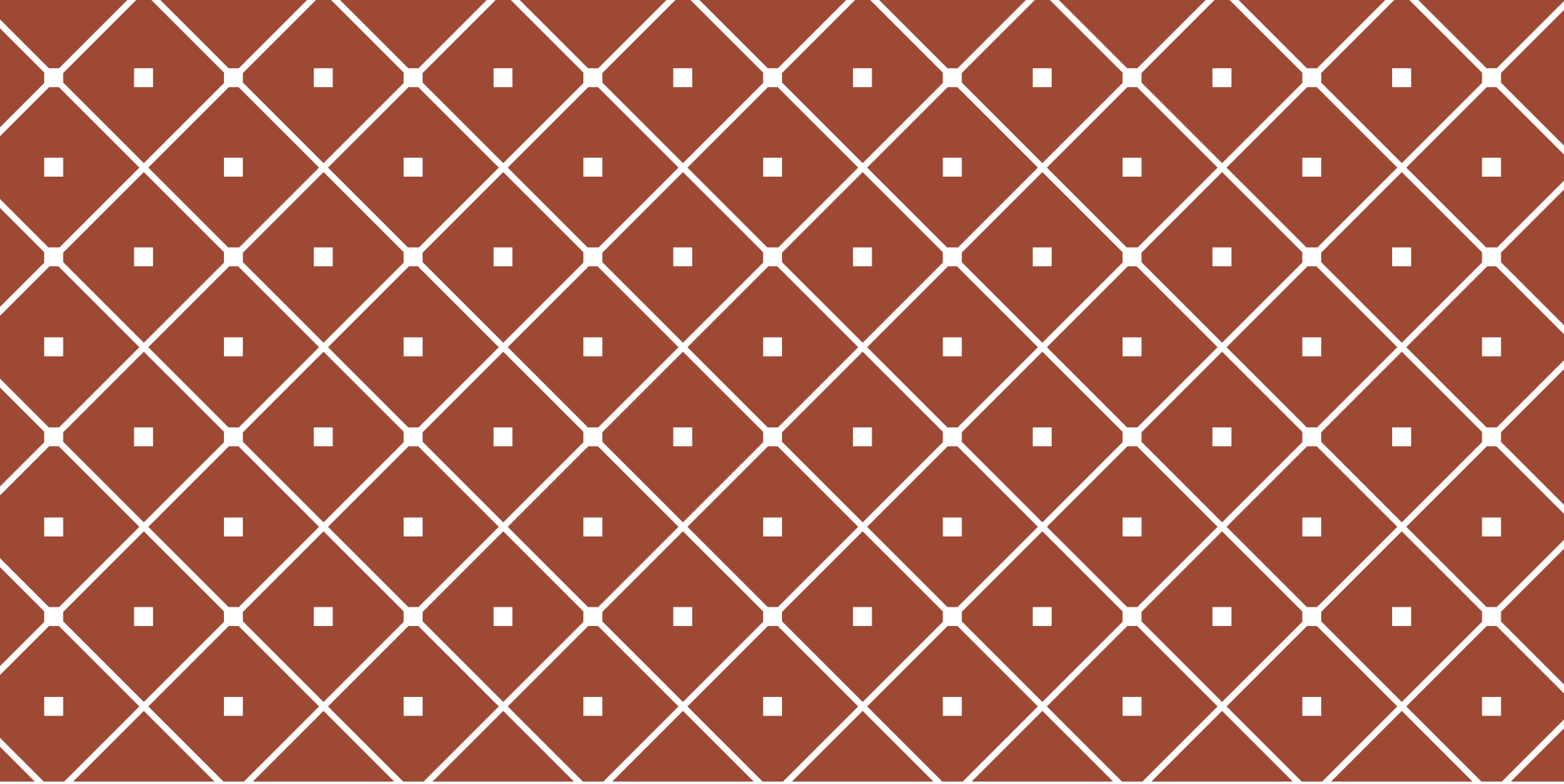


# LESSON 3 – INTRODUCTION OF JSX

# INTRODUCING JSX

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

JSX

- Syntax extension of JS.
- Describes what the UI should look like
- JSX produces React “elements”

# WHY JSX?

- React embraces the fact that **rendering logic is inherently coupled with other UI logic:** how events are handled, how the state changes over time, and how the data is prepared for display.
- React **doesn't require** using JSX, but most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

# WHY JSX?

```jsx
class Hello extends React.Component {
  render() {
    return <div>Hello {this.props.toWhat}</div>;
  }
}

ReactDOM.render(
  <Hello toWhat="World" />,
  document.getElementById('root')
);
```

**Code with JSX!**

```jsx
class Hello extends React.Component {
  render() {
    return React.createElement('div', null, `Hello ${this.props.toWhat}`);
  }
}

ReactDOM.render(
  React.createElement(Hello, {toWhat: 'World'}, null),
  document.getElementById('root')
);
```

**Code without JSX!**

# EMBEDDING EXPRESSION IN JSX

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

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

You can put any valid JavaScript expression inside the curly braces in JSX. For example, `2 + 2`, `user.firstName`, or `formatName(user)` are all valid JavaScript expressions.

# EMBEDDING EXPRESSION IN JSX

In the example below, we embed the result of calling a JavaScript function, `formatName(user)`, into an `<h1>` element.

```
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')
);
```

# JSX IS AN EXPRESSION

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

This means that you can use JSX inside of `if` statements and `for` loops, assign it to variables, accept it as arguments, and return it from functions:

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

# ATTRIBUTES IN JSX

You may use quotes to specify string literals as attributes:

```
const element = <a href="https://www.reactjs.org"> link </a>;
```

You may also use curly braces to embed a JavaScript expression in an attribute:

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

Warning: Don't put quotes around curly braces when embedding a JavaScript expression in an attribute. You should either use **quotes (for string values)** OR **curly braces (for expressions)**, but not both in the same attribute.

# CHILDREN WITH JSX

If a tag is empty, you may close it immediately with />, like XML:

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

JSX tags may contain children:

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

# JSX REPRESENTS OBJECTS

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

These two examples are identical:

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

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

# JSX REPRESENTS OBJECTS

`React.createElement()` performs a few checks to help you write bug-free code but essentially it creates an object like this:

```
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

# JSX REPRESENTS OBJECTS

These objects are called "**React elements**".

You can think of them as descriptions of what you want to see on the screen.

React reads these objects and uses them to construct the DOM and keep it up to date.

# WHAT IS “ELEMENT”?

**Elements** are the smallest building blocks of React apps.

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

**Note:**
One might confuse elements with a more widely known concept of “**components**”. Elements are what components are “made of”.

# RENDERING AN ELEMENT INTO THE DOM

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

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element, document.getElementById('root'));
```

# UPDATING THE RENDERED ELEMENTS

React elements are immutable. Once you create an element, you can't change its children or attributes. An element is like a single frame in a movie: it represents the UI at a certain point in time.

With our knowledge so far, the only way to update the UI is to create a new element, and pass it to `ReactDOM.render()`.

# UPDATING THE RENDERED ELEMENTS

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById('root'));
}

setInterval(tick, 1000);
```

# UPDATING THE RENDERED ELEMENTS

It calls `ReactDOM.render()` every second from a `setInterval()` callback.

**Note:**

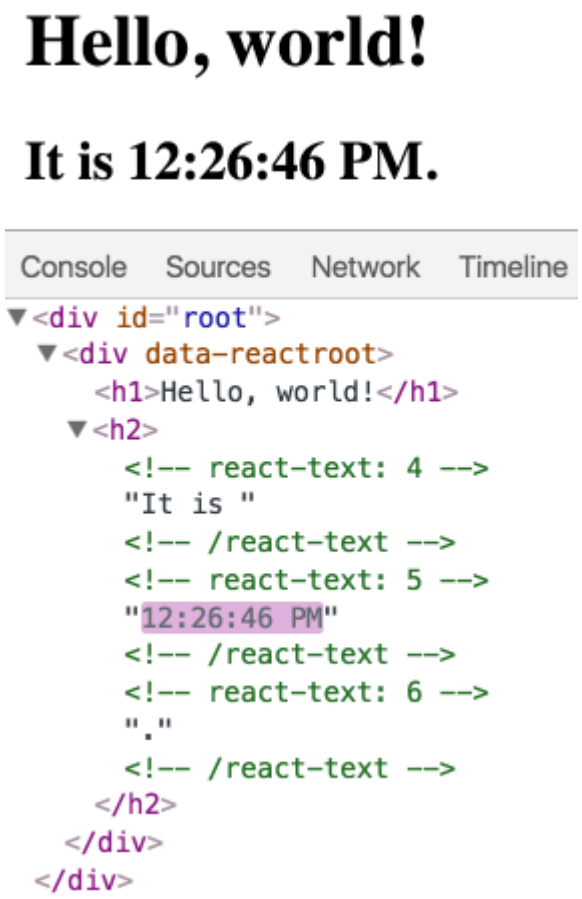
In practice, most React apps only call `ReactDOM.render()` once. In the next sections we will learn how such code gets encapsulated into stateful components.

We recommend that you don't skip topics because they build on each other.

# REACT ONLY UPDATES WHAT'S NECESSARY

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.

You can verify by inspecting the last example with the browser tools:



# REFERENCES

SimpliLearn. (n.d.). ReactJS Tutorial: A Step-by-Step Guide To Learn React. https://www.simplilearn.com/tutorials/reactjs-tutorial

ReactJS Documentation: https://reactjs.org/docs/getting-started.html