QUIZ

Can a web browser read JSX directly?

No, a web browser must be compiled before it can read JSX.

No, JSX must be compiled before it can be read by a web browser.

👏 Correct! JSX is compiled to code read-able by a browser.

Only Internet Explorer.

Yes.

What's wrong with this code?

```
let skateboardDog = (
  <img src="alfie.jpg" />
  <h1>Hiya kids!  I'm a dog on a skateboard.</h1>
);
```

Improper use of 'let'.

JSX expressions need an outermost element.

👏 Correct! This expression is wrong because it has two outermost elements.

What is the correct way to attach the function `yo` to a click event?

```
let yo = () => {
  alert('Yo');
};
```

```
<button onClick={yo}></button>
```

👏 That's right! The function `yo`, wrapped in curly braces because it is JavaScript, is set as the value of the `onClick` event handler.

What should you pass to `ReactDOM.render()` for its first argument?

`ReactDOM.render()` itself.

A JSX expression that you want to render.

👏 Correct! `ReactDOM.render()` takes a JSX expression for its first argument.

A JavaScript expression that you want to compile.

A selector that matches an HTML element.

What's a difference between a DOM object and a virtual DOM object?

A virtual DOM object will be updated if ANY JSX element renders.

All presented answers are correct.

Correct! The entire virtual DOM, without directly affecting any HTML, can be updated much more quickly than the regular DOM when a JSX element is rendered.

A virtual DOM object can't directly affect HTML.

A virtual DOM object can update much faster than a regular DOM object.

What should you pass to `ReactDOM.render()` for its second argument?

`ReactDOM.render()` itself.

A JSX expression that you want to render.

A JSX expression that you want to compile.

A selector that matches an HTML element.

That's right, the second argument should be a selector that matches an HTML element!

What problem does the virtual DOM attempt to solve?

Updating DOM objects doesn't happen when it is supposed to.

The regular DOM cannot recognize when an object has been updated.

Updating the DOM is not time efficient.

👏 Correct! Manipulating the virtual DOM is much faster, because it finds out exactly which objects have changed, and only those objects get updated on the DOM.

Updating the DOM will result in only partially rendered objects.

Which will render `100` to the screen?

```
ReactDOM.render(
  {<h1>10 * 10</h1>},
  document.getElementById('app')
);
```

```
ReactDOM.render(
  <h1>10 * 10</h1>,
  document.getElementById('app')
);
```

```
ReactDOM.render(
  <h1>{10 * 10}</h1>,
  document.getElementById('app')
);
```

👏 That's right! The expression wrapped in curly braces, `{10 * 10}`, will render as `100` in the browser.

Place the following steps in the right order:

```
a. Changes on the real DOM cause the screen to change.
b. A JSX element renders.
c. The virtual DOM is compared to what it looked like before it updated to figure out which objects have changed.
d. The entire virtual DOM gets updated.
e. The changed objects, and the changed objects only, get updated on the real DOM.
```

b, d, c, e, a

You got it! This sequence of events is how React leverages the virtual DOM to quickly and efficiently update only the parts of the regular DOM that changed.