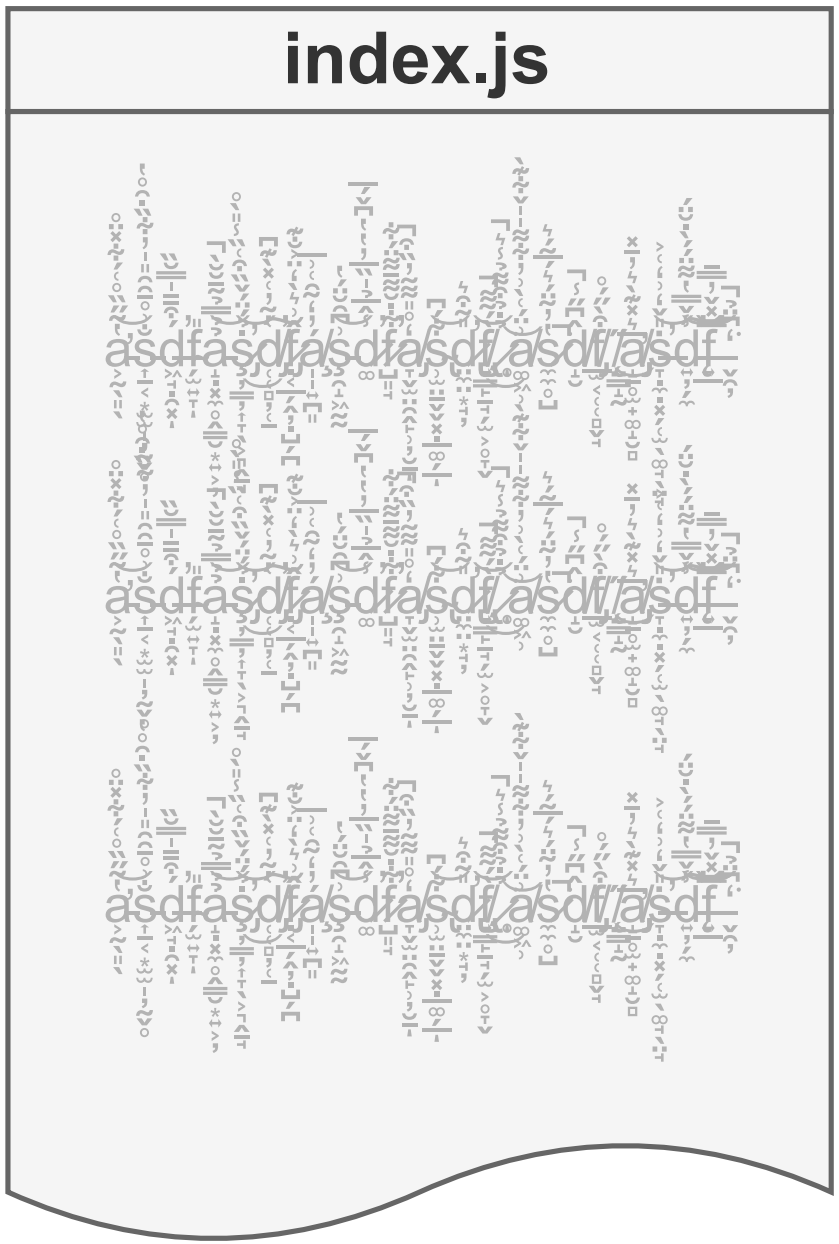


The index.js file can get messy really quickly

index.js

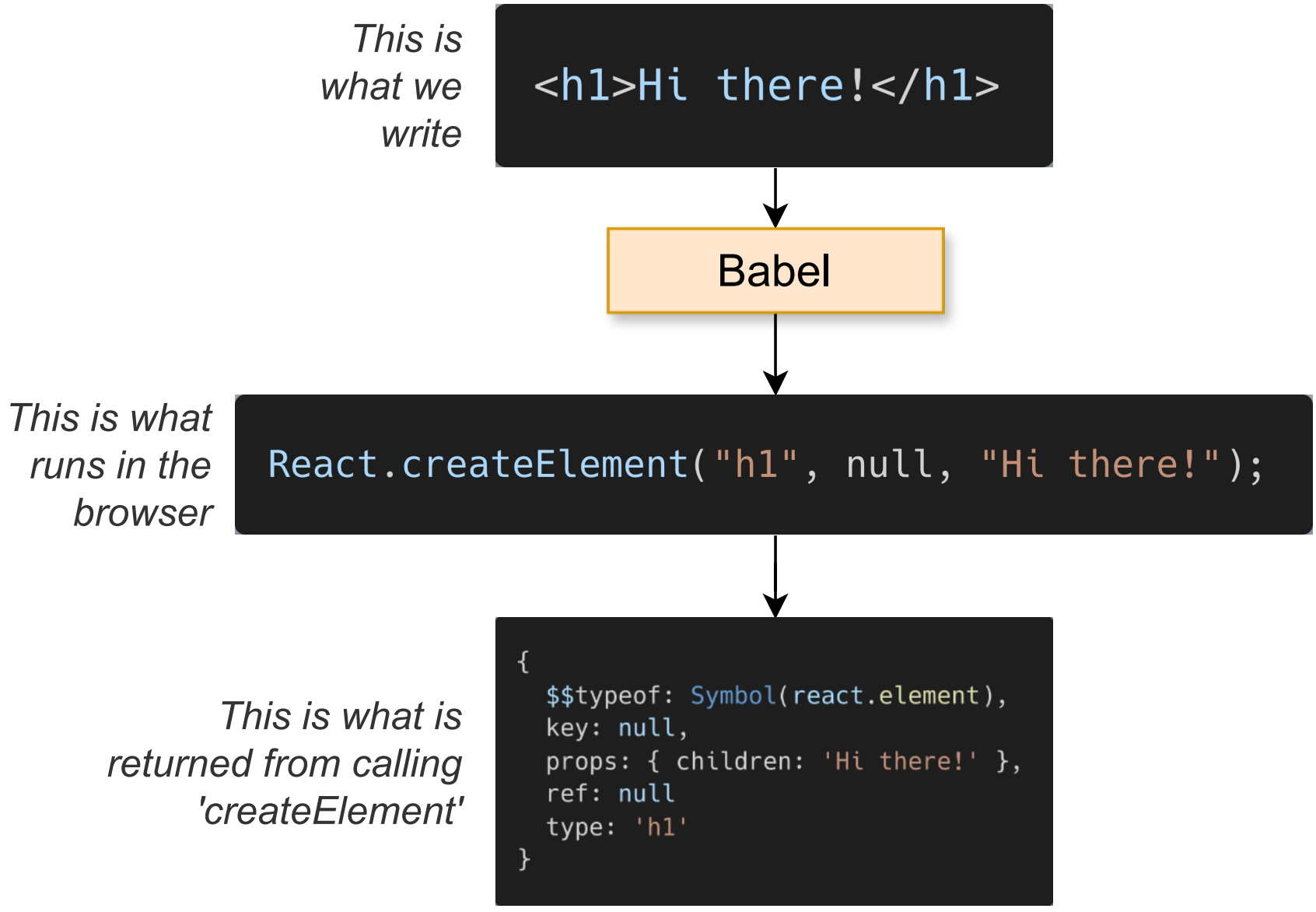
App.js

```
function App() {
  return <h1>Hi There!</h1>;
}
```

Create components in separate files. Easier to organize

Creating a Component

1	Create a new file. *(By convention)* File should start with a capital letter	JS App.js
2	Make your component. Should be a function that returns JSX.	`function App() { return <h1>Hi</h1> }`
3	Export the component at the bottom of the file	`export default App;`
4	Import the component into another file	`import App from './App';`
5	Use the component	`<App />`


This is what we write
<h1>Hi there!</h1>
Babel
This is what runs in the browser
React.createElement("h1", null, "Hi there!");
This is what is returned from calling 'createElement'
{
$$typeof: Symbol(react.element),
key: null,
props: { children: 'Hi there!' },
ref: null
type: 'h1'
}

babeljs.io/repl

Tool to show you what your JSX is turned into

```
<h1>Hi there!</h1>
```

Writing this doesn't make anything show up in the browser automatically

This creates an **instruction** for React, telling it to make an element

We have to **return** it from a component for React to use it

Good!
JSX on same line as return

```
function App() {
  return <h1>Hi!</h1>
}
```

Bad!
JSX on next line

```
function App() {
  return;
  <h1>Hi!</h1>;
}
```

Good!
Paren on same line as return

```
function App() {
  return (
    <h1>Hi!</h1>
  );
}
```

Bad!
Paren on next line

```
function App() {
  return
  (<h1>Hi!</h1>);
}
```

Standard HTML

Some tags can be self closing

<img /> *<link />*

<area /> *<hr />*

<col /> *<meta />*

JSX

Any tag can be self closing

* * **

*
* *
*

<button /> *<label />*

Use self-closing tags unless you want to nest an element

Very long elements should be multi-lined

```
<input id="number" type="number" min={5} max={10} value={7} placeholder={5} />
```

```
<input
  id="number"
  type="number"
  min={5}
  max={10}
  value={7}
  placeholder={5}
/>
```

Curly braces mean we are about to reference a JS variable or expression

```
function App() {
  const message = 'Hi there!';

  return <h1>{message}</h1>
}
```

Expressions are any snippets of code that result in a value.

Good

```
function App() {
  return <h1>{1 + 1}</h1>
}
```

Bad

```
function App() {
  return <h1>{const message = 'hi there'}</h1>
}
```

Good

```
function App() {
  return <h1>{'a,b,c'.split(',')}</h1>
}
```

Bad

```
function App() {
  const colors = {};

  return <h1>{colors.favorite = 'red'}</h1>
}
```

Some values might not display quite as you expect

```
function App() {
  const value = true;

  return <h1>{value}</h1>
}
```

What will be
in the h1?

Basics of JSX

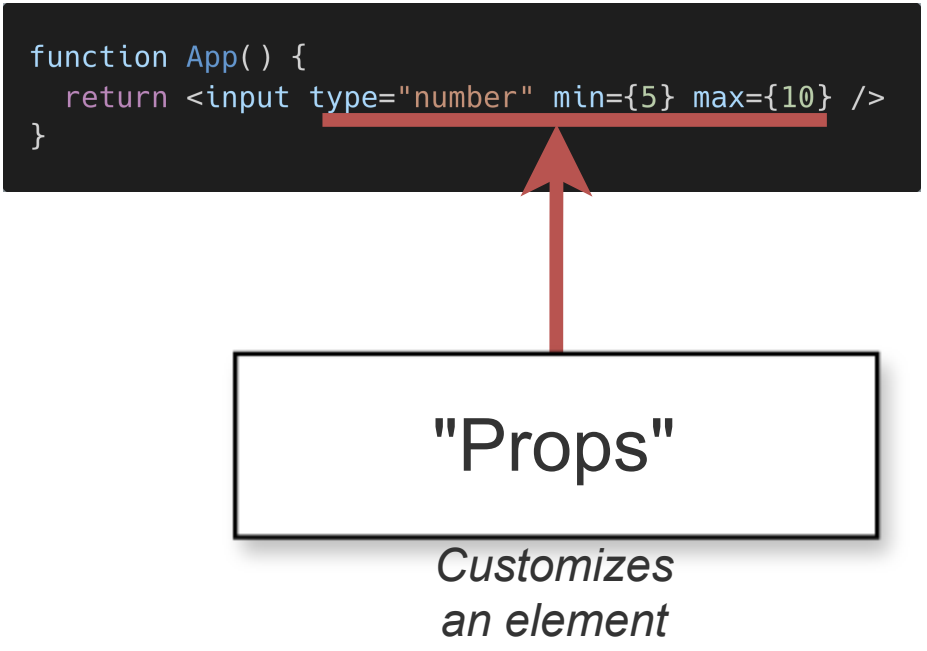
Returning JSX

Handling JSX tags

Referencing JS values in JSX

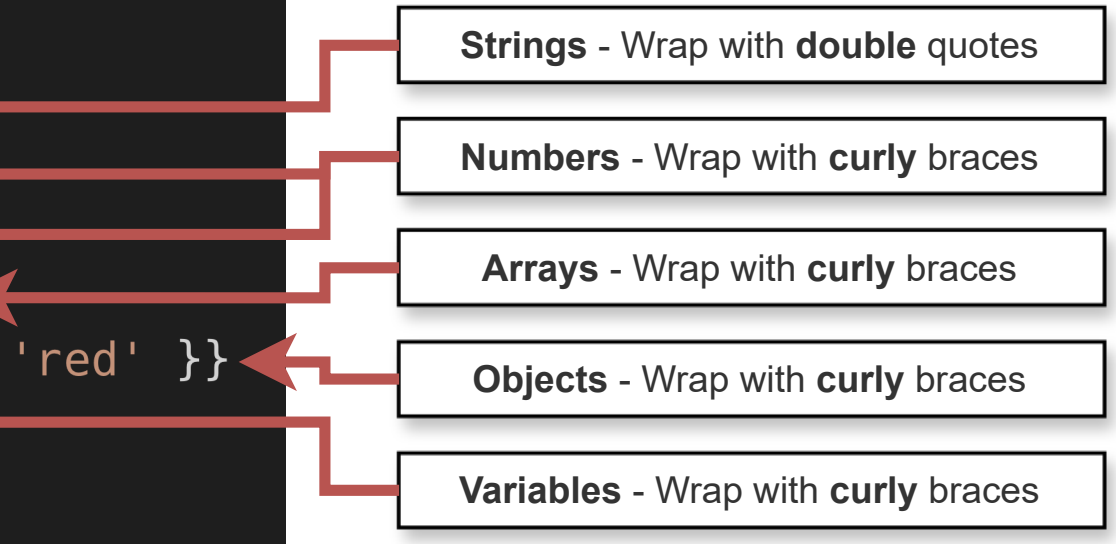
Element Props

Inline Styles


function App() {
return <input type="number" min={5} max={10} />
}
"Props"
Customizes
an element

```jsx
function App() {
  const message = 'Enter age';

  return (
    <input
      type="number"
      min={5}
      max={10}
      list={[1,2,3]}
      style={{ color: 'red' }}
      alt={message}
    />
  );
}
```



HTML

```
<a href="google.com" style="text-decoration: none; font-size: 20px;">
  Google
</a>
```

JSX Equivalent

```
<a href="google.com" style={{ textDecoration: 'none', fontSize: '20px' }}>
  Google
</a>
```

Inline styling is given as an object

Property names are camelCase (no dashes)

Pixel values are strings with 'px'