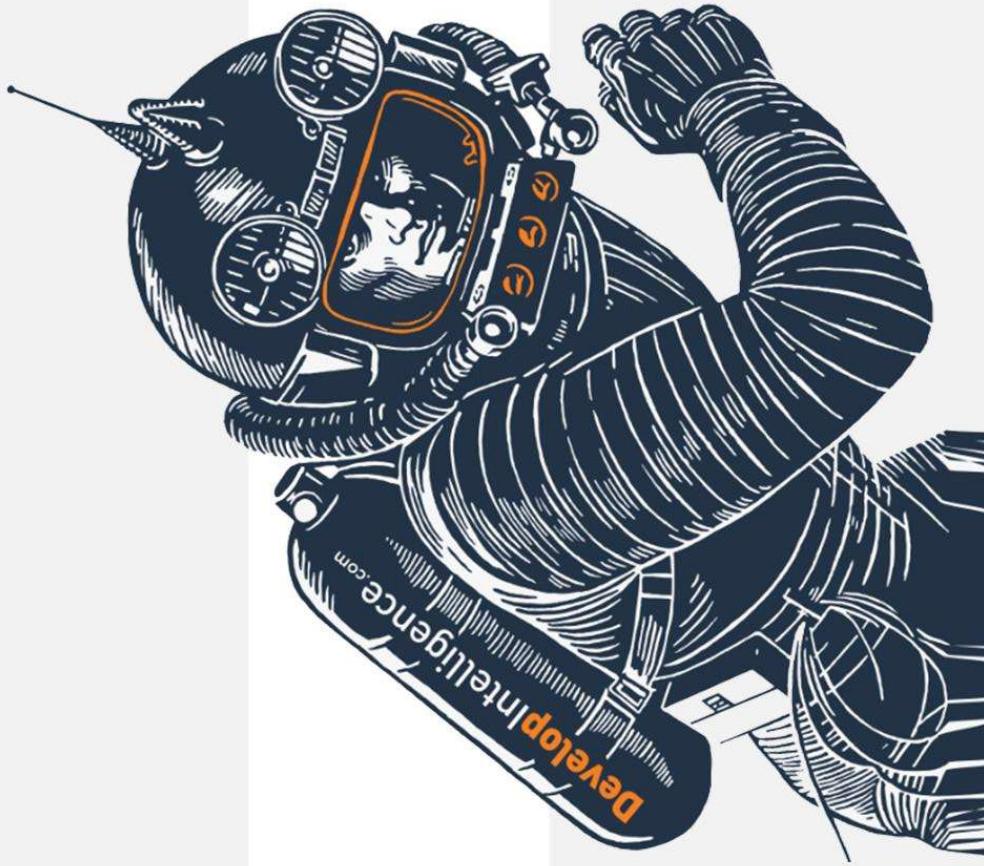


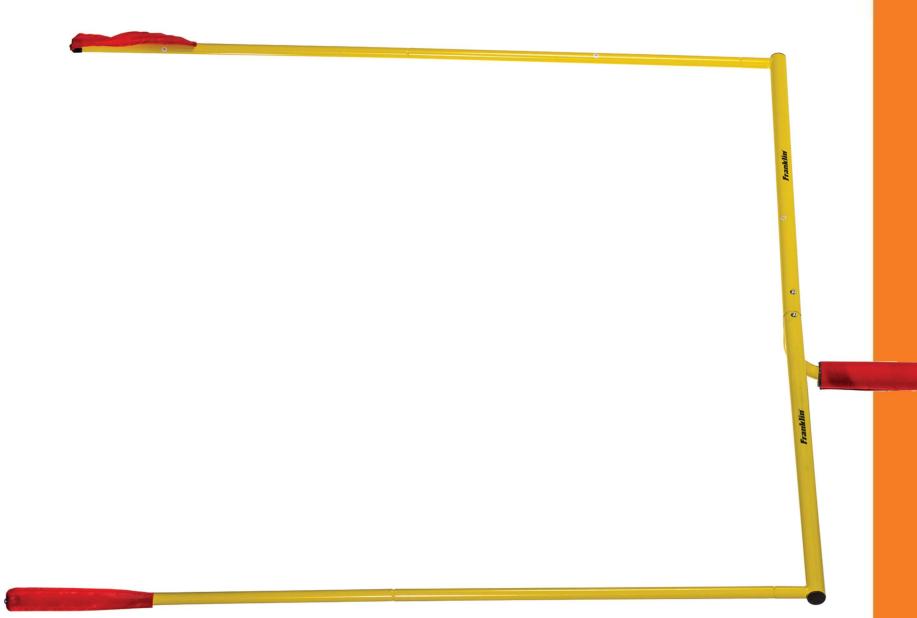
Components

Develop Intelligence



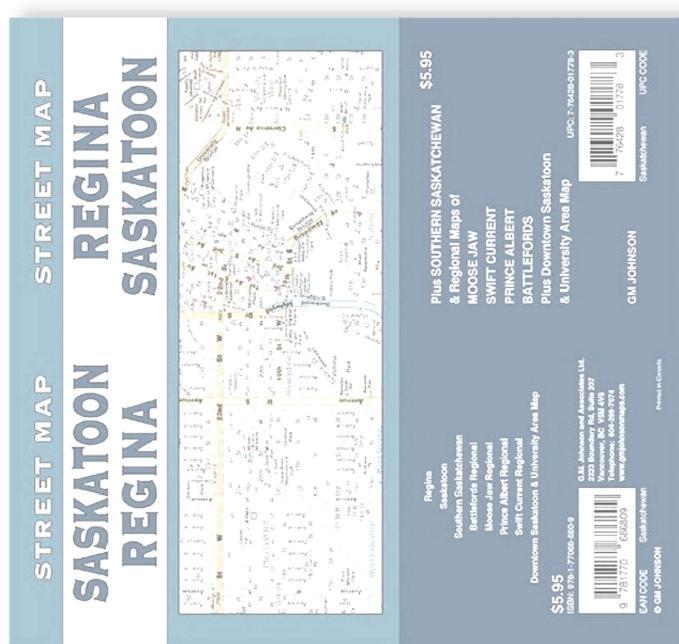
Goals

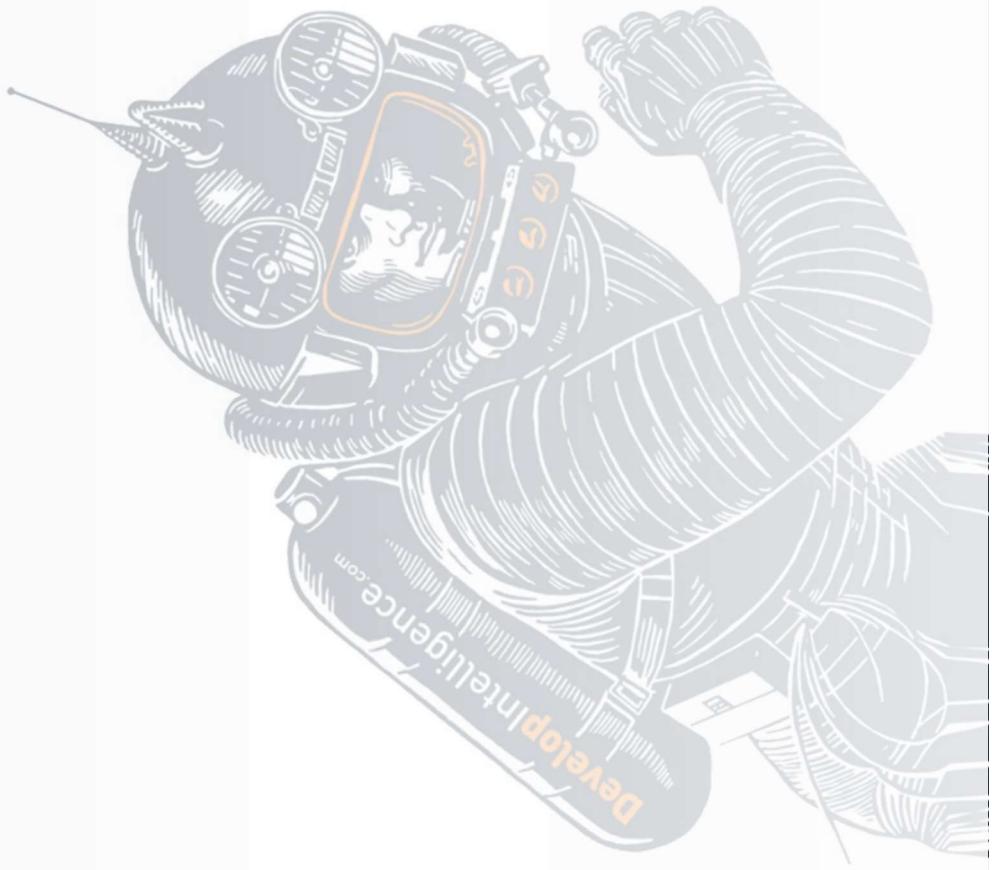
1. Compare JSX with standard HTML
2. List 3 advantages of functional components





1. JSX
2. Component Flavors
3. Properties
4. Events
5. Style

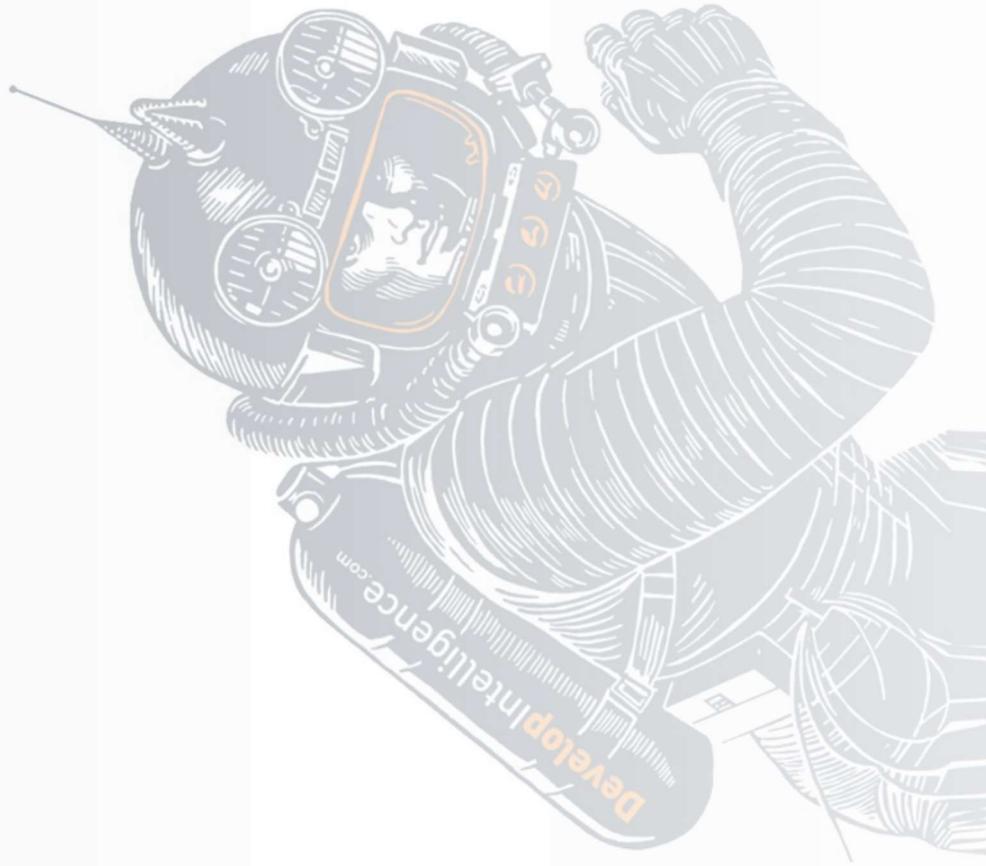




Develop
Intelligence



JSX



According to Hoyle

Spec says

JSX is an XML-like syntax extension to ECMAScript without any defined semantics. It's NOT intended to be implemented by engines or browsers. It's NOT a proposal to incorporate JSX into the ECMAScript spec itself. It's intended to be used by various preprocessors (transpilers) to transform these tokens into standard ECMAScript.



JavaScript Syntax Extension

- *Very similar to string-based html templates-- just without the quotes*
- Created by Facebook for React
- Used by other frameworks -- e.g. Vue.js
- Transformed into JavaScript by a compiler -- e.g. Babel



No Magic

- Shorthand syntax

```
1 const Greet = () => <h1>Hello, world!</h1>;
```

- Transformed (for React)

```
1 import { jsx as _jsx } from "react/jsx-runtime";
2 const Greet = () => /*#__PURE__*/_jsx("h1", {
3   children: "Hello, world!"
4 });
```

- Use babeljs.io/repl if you're ever curious



JSX Rules

- Stricter than HTML
 - Properly nested
 - 1 root element
 - All elements closed
- Subtleties: className, style
- No longer required
 - import React from 'react';

Root Element

- Every component must have 1 root element

Not this

```
1 const App = () => <h1>Hello World!</h1>
2 <h2>Part I: The Beginning</h2>
3 <p>It was a dark and stormy night....</p>;
```

This

```
1 const App = () => <div>
2   <h1>Hello World!</h1>
3   <h2>Part I: The Beginning</h2>
4   <p>It was a dark and stormy night....</p>
5 </div>;
```

Fragments

- Grouping construct, not rendered

Fragment

```
1 const App = () => <Fragment>
2             <h1>Hello World!</h1>
3             <h2>Part I: The Beginning</h2>
4             </Fragment>;
```

Shorthand

```
1 const App = () => <>
2             <h1>Hello World!</h1>
3             <h2>Part I: The Beginning</h2>
4             </>;
```

On Parentheses

- Only **required** in the presence of linebreaks and return
- Your formatter may add them for **all** multiline jsx

Not required

```
1 export const App = () =>
2   <h1>Hello World.</h1>;
3
4 export const App = () =>{
5   return <h1>Hello World.</h1>;
6 }
7 export const App = () => {
8   return <h1>
9     Helloo World.
10    </h1>;
11 }
```

Required Parentheses



This doesn't work

```
1 export const App = () => {
2   return
3   <h1>
4     Helloo World.
5   </h1>;
6 }
```

Fixed

```
1 export const App = () => {
2   return (
3   <h1>
4     Helloo World.
5   </h1>;
6 )}
```



Templating (*sort of*)

- Any valid *expression* gets evaluated inside curly brackets.

Example

```
1 export function App() {  
2   const time = new Date().toLocaleTimeString();  
3   return <div className='root'>  
4     <h2>It is {time}</h2>  
5   </div>;  
6 }
```

Shorthand

```
1 export const App = () => <div className='root'>  
2   <h2>It is {new Date().toLocaleTimeString()}</h2>  
3 </div>;
```

Conditionals

- If/then/else works fine in a component
- (But not inside JSX)

```
1 export function App() {  
2   const day = new Date().getDay();  
3   if (day > 0 && day < 6){  
4     return <h2>Go to work!</h2>  
5   }  
6   return <h2>Relax man!</h2>  
7 }
```

Ternary



- Use the ternary inside an expression

```
1  export function App() {  
2    const day = new Date().getDay();  
3    return <>  
4      <h1>Hello World!</h1>  
5      { (day > 0 && day < 6)  
6        ? (<h2>Go to work!</h2>)  
7        : (<h2>Relax man!</h2>)  
8      }  
9    </>;  
10 }
```

null for nothing

- Returning null means the component renders nothing

```
1 export const Greeter = ({isBirthday}:props) => {
2   if (!isBirthday){
3     return null;
4   }
5   return <h1>Happy birthday!</h1>
6 }
```

Alternative

```
1 export const Greeter = ({isBirthday}:props) =>
2   isBirthday ? <h1>Happy birthday!</h1> : null;
```



Short-circuit

```
1 export const Greeter = ({isBirthday}:props) =>
2   isBirthday && <h1>Happy birthday!;
```



- Use Array.map as a functional loop

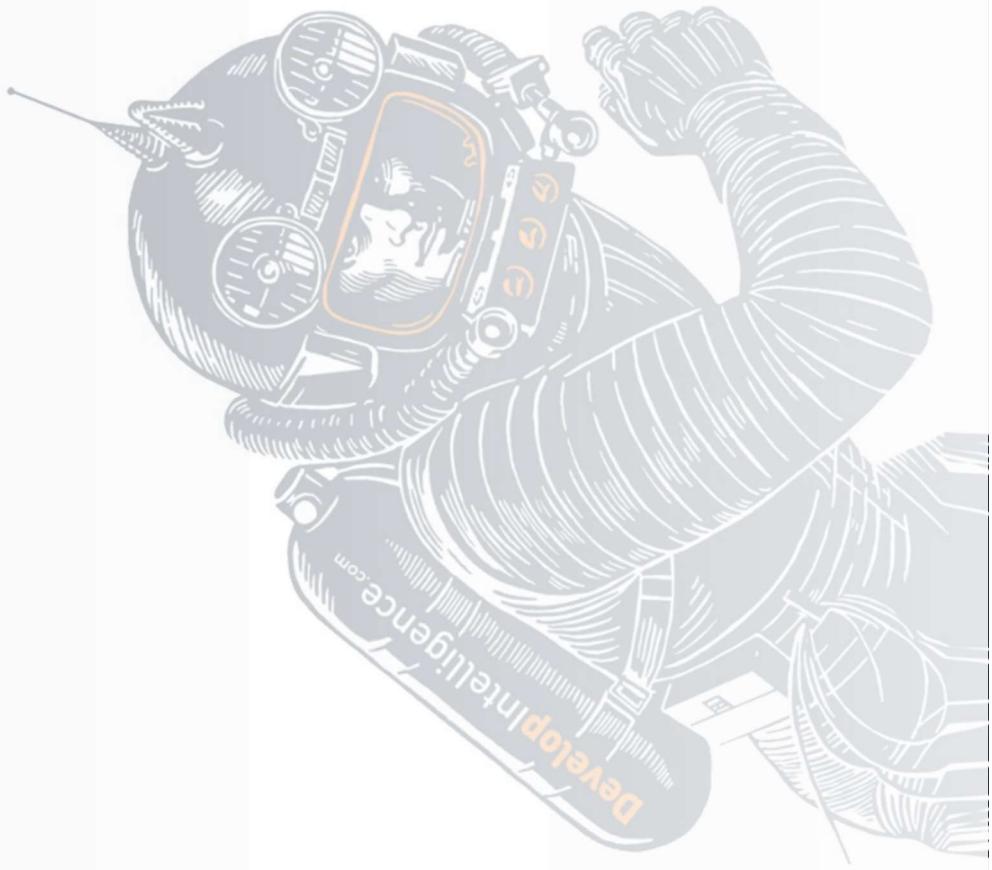
```
1 export function App() {  
2   const innerPlanets = ['Mercury', 'Venus', 'Earth', 'Mars'];  
3   return <>  
4   <h1>The inner planets are:</h1>  
5   <ul>  
6     {innerPlanets.map(p=><li>{p}</li>) }  
7   </ul>  
8   </>;  
9 }
```



Lists: Better with `key`

- Uniquely identify a list item with `key`
- Best practice
- Potential performance improvement

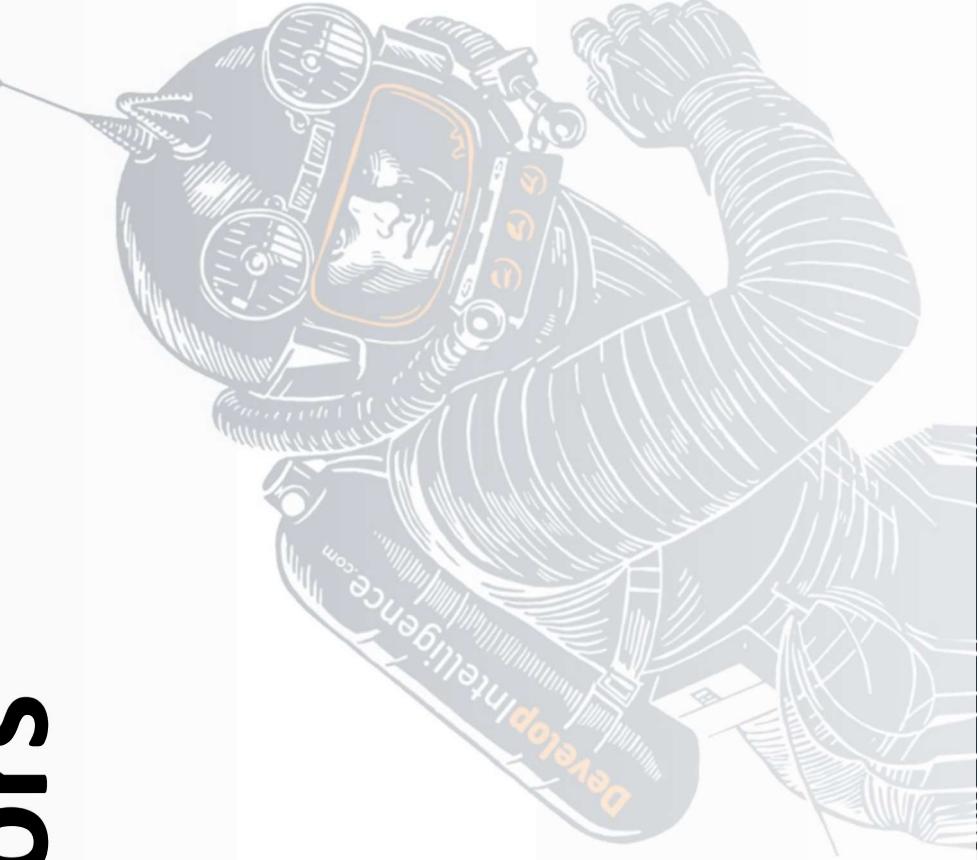
```
1 export function App() {  
2   const innerPlanets = ['Mercury', 'Venus', 'Earth', 'Mars'];  
3   return <>  
4     <h1>The inner planets are:</h1>  
5     <ul>  
6       {innerPlanets.map((p)=><li key={p}>{p}</li>)}  
7     </ul>  
8   </>;  
9 }
```



Develop
Intelligence



Component Flavors





Components come in 2 flavors

- **Class-based**

- Primary method for creating components before React 16.8

- Inherit from Component

- Implement a render method
- Keep state as member variables

- **Functional**

- Recommended
- Simple rendering function



Example: FizzBuzzComponent

```
1  export class FizzBuzz extends Component {
2    render() {
3      return <div className="prose">
4        <h1>FizzBuzz <i>[Complete]</i></h1>
5        <ul>
6          {getMessages().map(m => (
7            <li>{m}</li>
8          )))
9        </ul>
10       </div>;
11     }
12   }
```



Functional Equivalent

```
1 export const FizzBuzz = () =>
2   <div className="prose">
3     <h1>FizzBuzz <i>[Complete]</i></h1>
4     <ul>
5       {getMessages().map(m => (
6         <li>{m}</li>
7       ))}
8     </ul>
9   </div>;
```

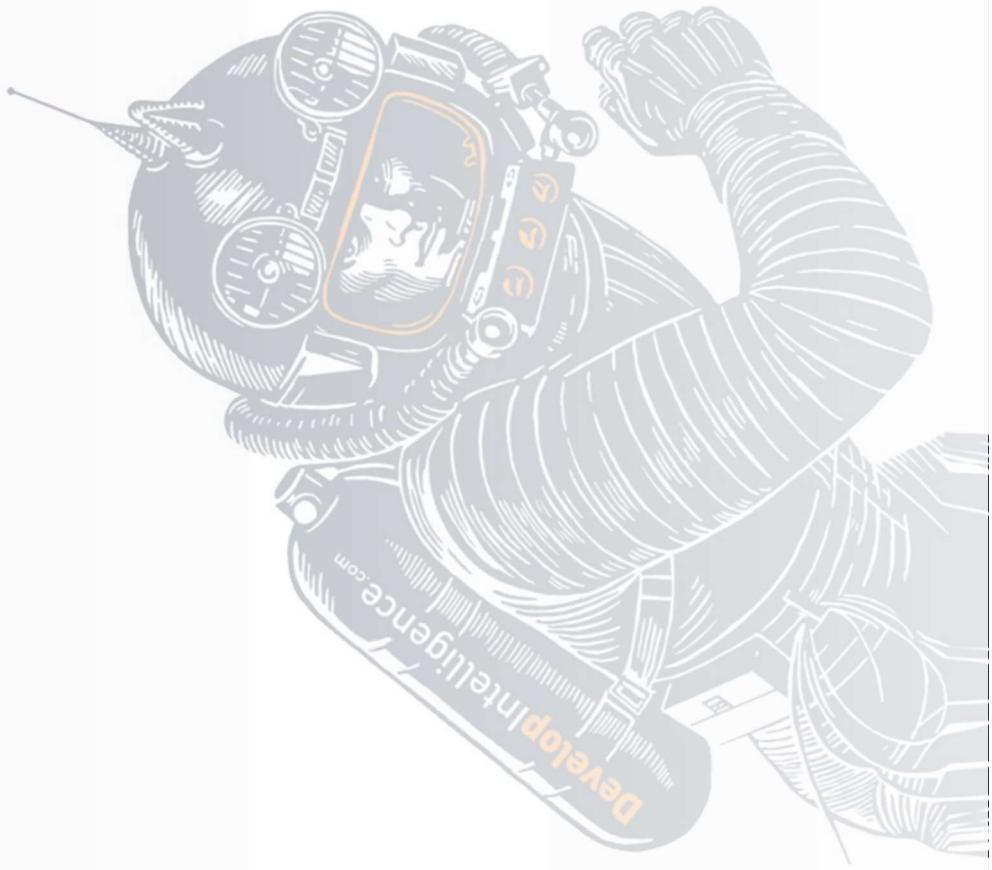


Functional advantages

- **Faster.** No instantiation overhead
- **Simpler.** Just a render function.
- **More composable.** No lifecycle methods.
- **Easier to learn.** Especially for programming noobs.
- **Popular** Everyone else uses them.

The big one

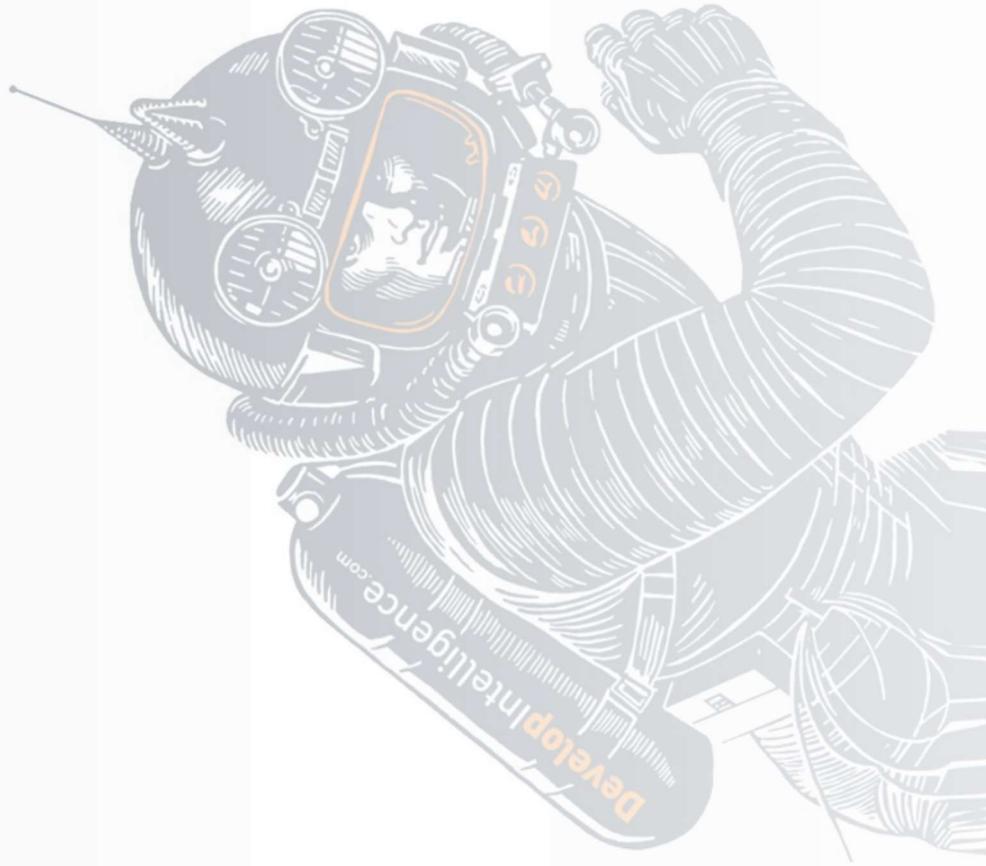
- *You can use hooks*



Develop
Intelligence



Properties





Overview

- Properties are passed by parameter
 - 1 parameter object
 - Keys correspond to JSX attributes

Example

```
1 const Greet = (props)=>
2   <h1>Hello {props.givenName} {props.surname}</h1>
3
4 export const App = () =>
5   <Greet givenName='Joe' surname='Bloggs' />
```



Aside: Destructuring

- ECMAScript ≥ 6 syntax
- Neat syntax for taking elements out of data structures
- Often used in idiomatic React

```
1 // Array destructure
2 const [first, second, ...rest] = [10, 20, 30, 40, 50];
3
4 // Object destructure
5 const person = {givenName: 'joe', surname: 'bloggs'};
6 const {givenName, surname} = person;
```



Destructured Props

- Arguably more readable
- Let you ignore anything else in props

```
1 const Greet = ({givenName, surname})=>
2   <h1>Hello {givenName} {surname}!</h1>
3
4 export const App = () =>
5   <Greet givenName='Joe' surname='Bloggs' />
```



Typed Props Pattern

- With typescript:
 - Clearer intent
 - Help from compiler & editor

```
1 interface Props {  
2   intro:Model.Intro;  
3   skin:Skin;  
4 }  
5  
6 export const Intro = ({ intro,skin }:Props) =>  
7   <Deck theme={skin.theme}>  
8     <TopicTitle skin={skin}># Introductions</TopicTitle>  
9     /*Etc...*/  
10    </Deck>;
```



Alternative

- Inline type definitions are good for simple props
- Get unwieldy

```
1 const Greet = ({givenName, surname}:{givenName:string, surname:string})=>
2 <h1>Hello {givenName} {surname}</h1>
3
4 export const App = () =>
5 <Greet givenName='Joe' surname='Bloggs' />
```



Props.children

- Props have a special (optional) property called children
- Used for nesting components
- Can be anything

```
1 const SectionTitle = (props)=>
2   <h1>{props.children}</h1>
3
4 const App = () =>
5   <>
6   <SectionTitle>Overview</SectionTitle
```



Better: `PropsWithChildren<T>`

- No implicit any

```
1 const SectionTitle = (props: PropsWithChildren) =>
2   <h1>{props.children}</h1>;
3
4 export const App = () => (
5   <>
6     <SectionTitle>Overview</SectionTitle
```



Mix'n'Match

```
1 import { PropsWithChildren } from 'react'
2
3 interface Props{
4   fontSize:string;
5 }
6
7 const SectionTitle = (props: PropsWithChildren<Props>) =>
8   <h1 style={{'fontSize':props.fontSize}}>
9     {props.children}
10    </h1>
11
12 const App = () =>
13   <>
14     <SectionTitle fontSize='400%'>Overview</SectionTitle>
15     <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
16   </>
```

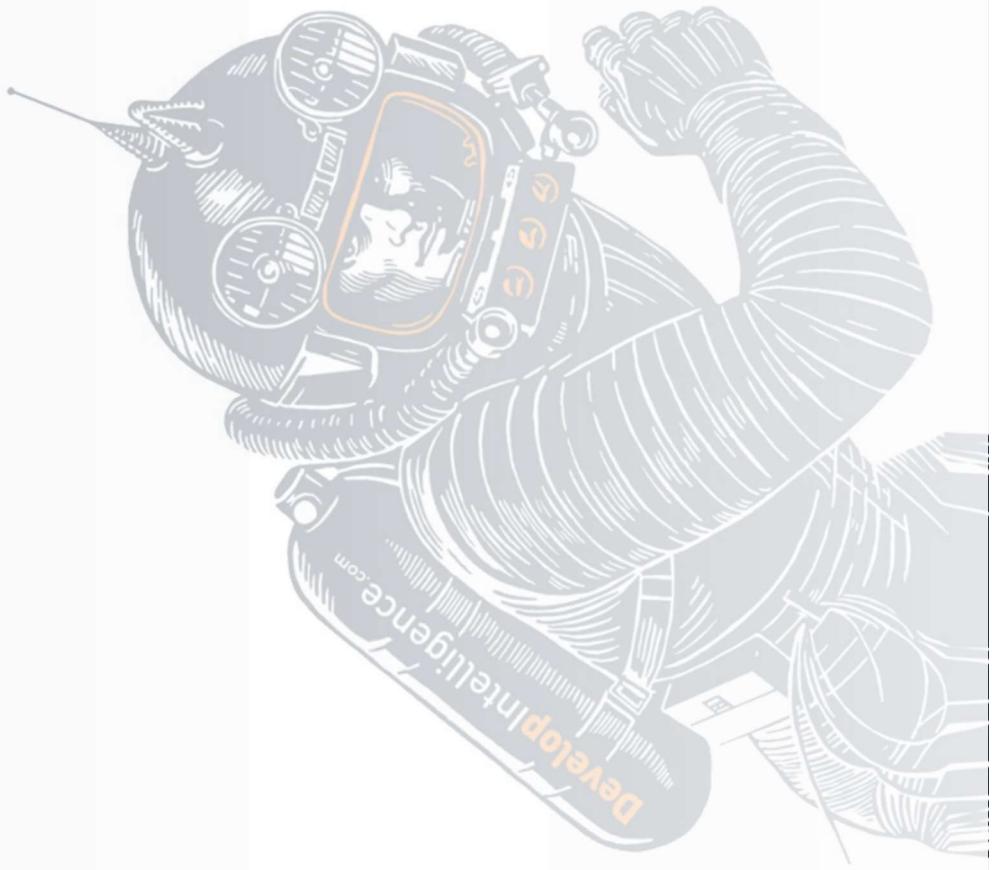


Component Rules

- Keep it pure
 - No side effects
 - Referentially transparent

Bad idea

```
1 let guest = 0;
2
3 const Cup = ()=>
4   <h2>Tea cup for guest #{guest++}</h2>;
5
6 export default TeaSet = () =>
7   <><Cup /><Cup /><Cup /></>;
```

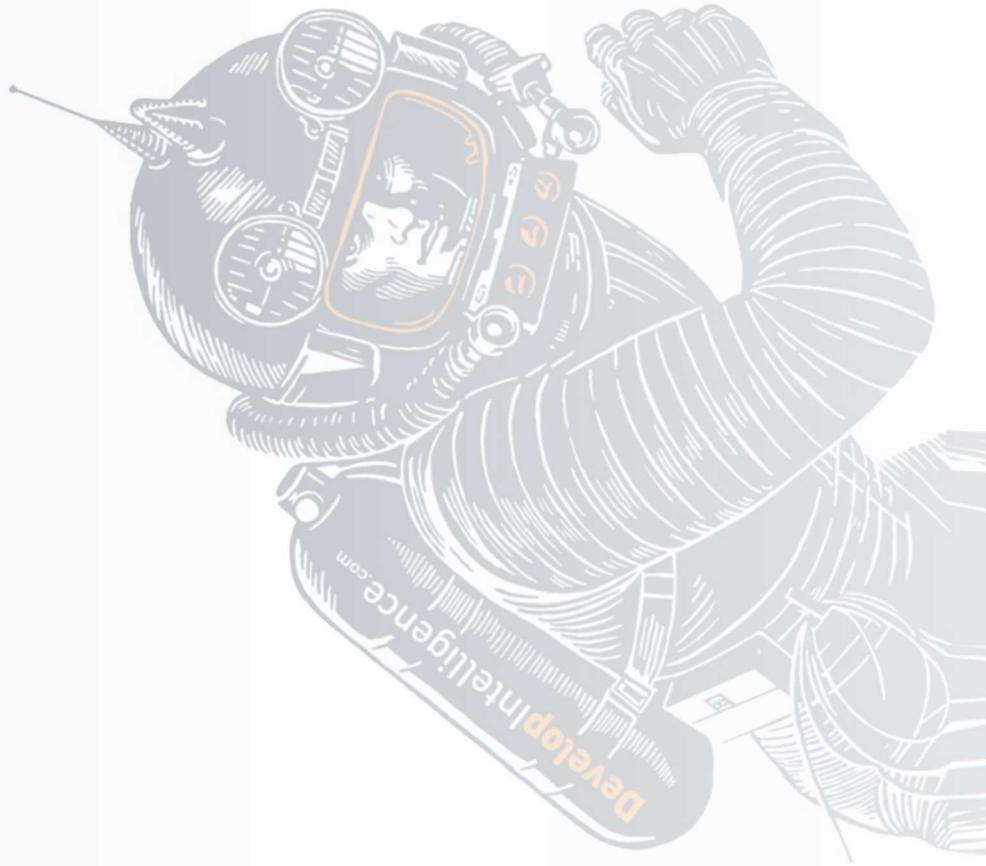


Develop
Intelligence



Events

Develop Intelligence





Overview

React lets you add event handlers to your JSX. Event handlers are your own functions that will be triggered in response to interactions like clicking, hovering, focusing form inputs, and so on.

Events drive changes in state!



Anonymous Event Handler

- Just like native browser events
 - *Except camel cased*

```
1 export const App = () => (
2   <button
3     onClick={() => alert('Hello world')}
4   >
5     Click me!
6   </button>
7 );
```



Named Equivalent

- Names are good: Express intent, easy to refactor
- Best practice: Use named event handlers for all but the simplest actions

```
1 export const App = () => {  
2   const handleClick = ()=>alert('Hello world!');  
3   return (  
4     <button  
5       onClick={handleClick}  
6     >  
7       Click me!  
8     </button>  
9   );  
10};
```



Handler Conventions

- Event handlers can be anything, but:
 - They're usually defined inside a component
 - Named 'handle' + *something*



Handlers as Props

- Child components can expose their own handlers

```
1 interface ButtonProps extends PropsWithChildren{  
2   onClick():void;  
3 }  
4  
5 const Button = ({onClick,children}:ButtonProps) =>  
6 <button onClick={onClick}>{children}</button>  
7  
8 export const App = () =>  
9 <Button onClick={()=>alert('Hello world! ')>}Click me</Button>
```



Event Information

- In React, events are 'synthetic'
- SyntheticEvent is a thin wrapper around the native browser event to abstract over browser differences
- Key members
 - target - For getting updated state
 - preventDefault - Useful for forms
 - stopPropagation

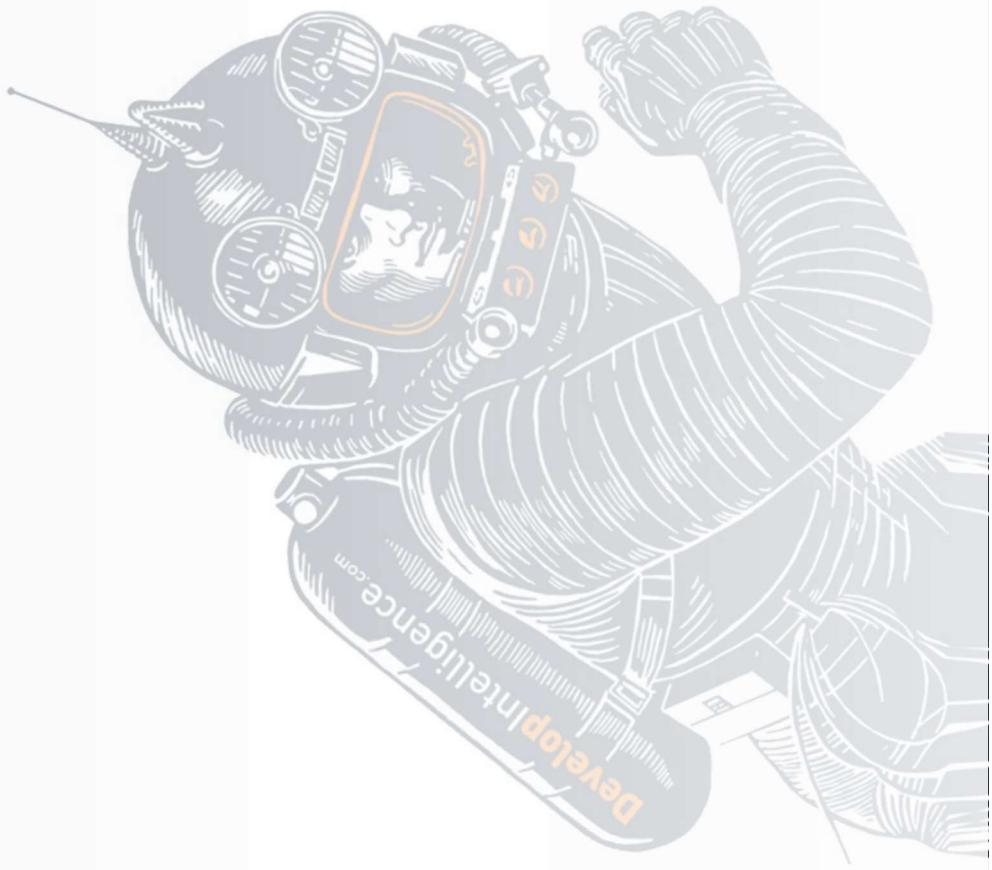
Example

```
1 interface Props {  
2   onChange(color: Color): void;  
3   color: Color;  
4 }  
5  
6 export const ColorPicker = (props: Props) => (  
7   <select  
8     value={props.color}  
9     onChange={event => props.onChange(event.target.value as Color)}  
10  >  
11    <option value={Color.Black}>Black</option>  
12    <option value={Color.Brown}>Brown</option>  
13    <option value={Color.Red}>Red</option>  
14    <option value={Color.Blue}>Blue</option>  
15  </select>  
16 );
```



Stopping Propagation

```
1 const Button = ({ onClick, children }) =>
2   <button onClick={e => {
3     e.stopPropagation();
4     onClick();
5   }}>
6   {children}
7 </button>;
```

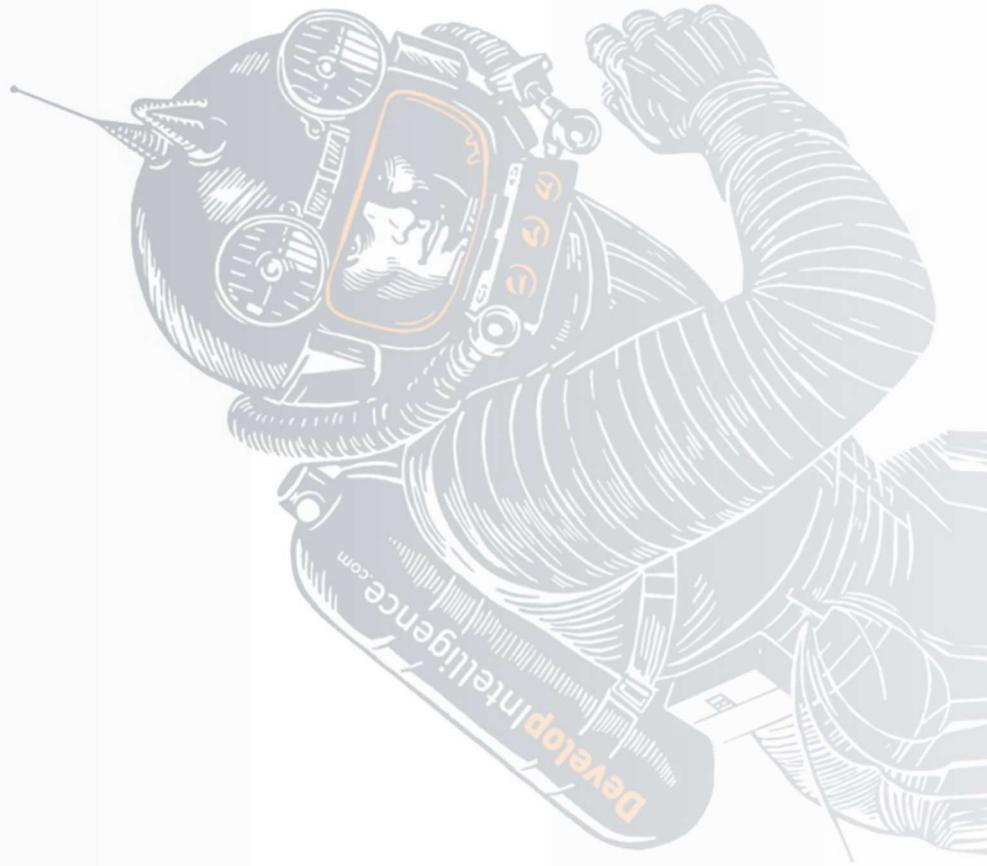


Develop
Intelligence



Style

Develop Intelligence



💡 Styling

- In ECMAScript, `class` is a keyword
- In JSX, use `className` instead

Not this

```
1 const App = () => <div class='root'>
2   <h1>Hello World!</h1>
3 </div>;
```

This

```
1 const App = () => <div className='root'>
2   <h1>Hello World!</h1>
3 </div>;
```

Style

- Style is an object, not a string
- For script friendliness, properties are camelCase

```
1 export const App = () => {
2   const greetingStyle = {
3     fontWeight: 'bold', //NOT font-weight
4     color: 'green'
5   };
6
7   return <h1 style={greetingStyle}>Hello World.</h1>;
8 }
```



Style Pattern

- Common pattern: Wrap HTML with styled
- Passing props with spread syntax

```
1 export const TextInput = (props)=>
2   <input
3     className="border-2 border-gray-300 p-2"
4     type="text"
5     {...props}
6   />
```



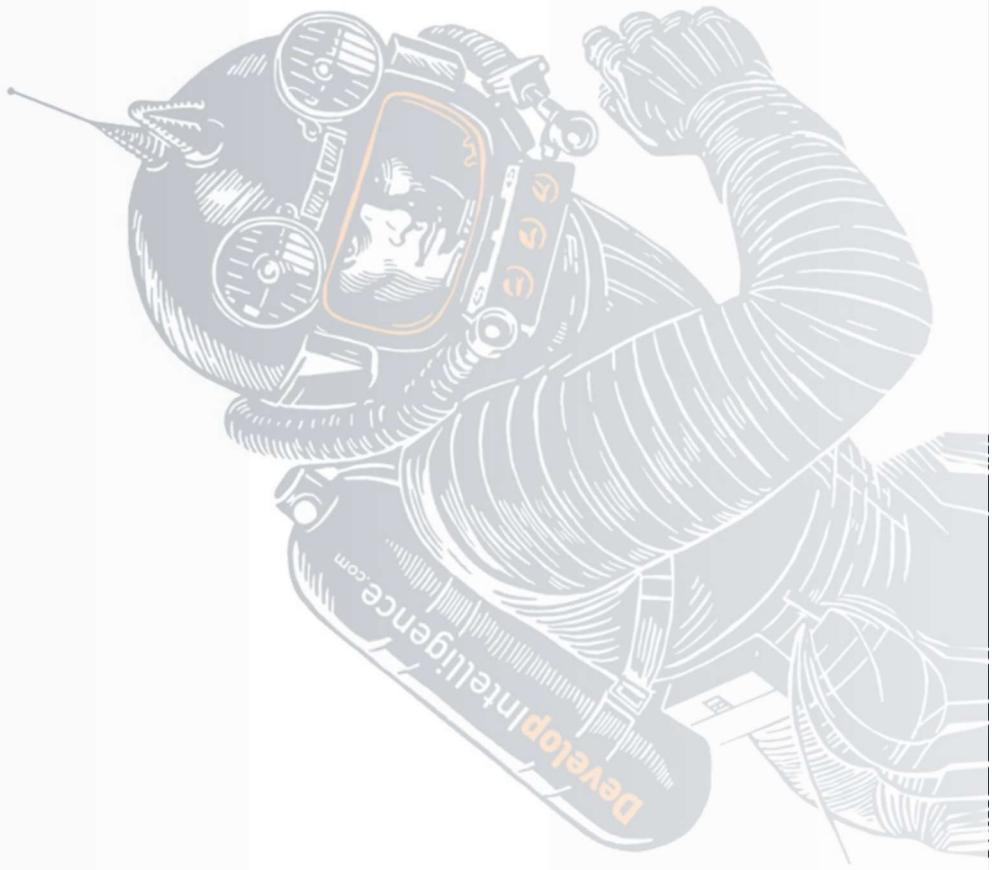
Lab:FizzBuzz

Instructions: `src/practice/fizz-buzz`

Summary:

- List numbers 1 to 100.
- For multiples of 3 print 'Fizz' instead
- For multiples of 5 print 'Buzz'
- Multiples of 3 and 5 print 'FizzBuzz'





Develop
Intelligence



Review



1. Compare JSX with standard HTML
2. List 3 advantages of functional components