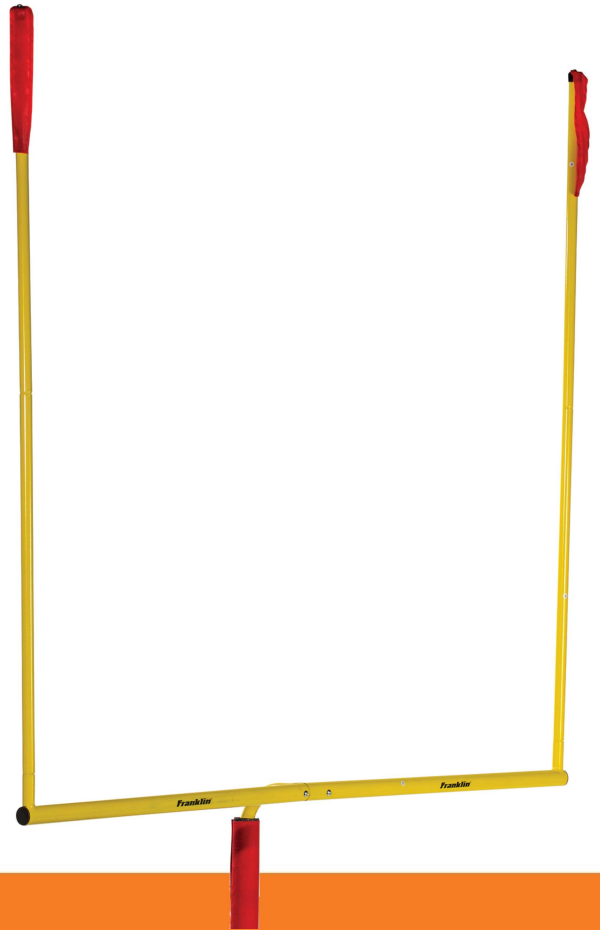
Components

Develop Intelligence

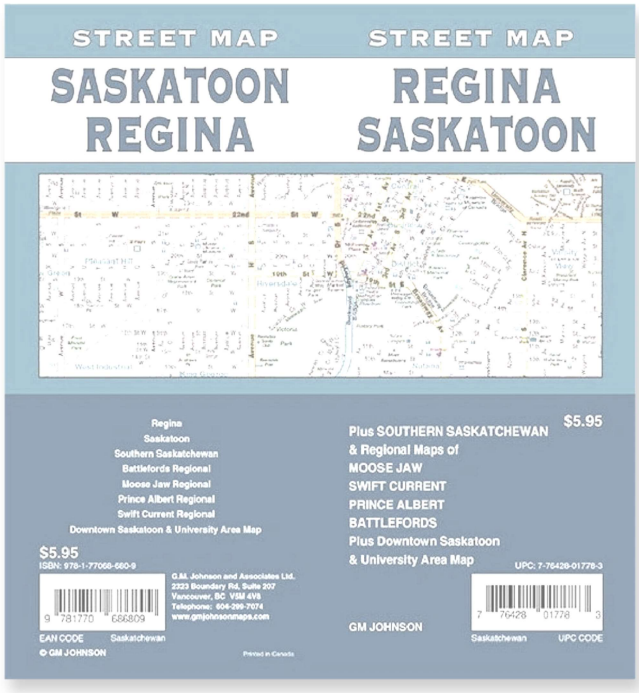
Goals

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



Roadmap

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



Develop
Intelligence

JSX

According to Hoyle

Develop Intelligence

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.

JSX

Develop Intelligence

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

Develop Intelligence

- 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

Develop Intelligence

- 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

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

Shorthand

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

On Parentheses

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

Not required

```
export const App = () =>
 <h1>Hello World.</h1>;

export const App =() =>{
  return <h1>Hello World.</h1>;
}
export const App = () => {
  return <h1>
          Helloo World.
         </h1>;
}
```

Required Parens

This doesn't work

```
export const App = () => {
  return
    <h1>
      Helloo World.
    </h1>;
}
```

Fixed

```
export const App = () => {
  return (
    <h1>
      Helloo World.
    </h1>;
  )
```

Templating *(sort of)*

Develop Intelligence

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

Example

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

Shorthand

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

```
export function App() {
  const day = new Date().getDay();
  return <>
          <h1>Hello World!</h1>
          {(day > 0 && day < 6)
            ? (<h2>Go to work!</h2>)
            : (<h2>Relax man!</h2>)
          }
         </>;
}
```

`null` for nothing

Develop Intelligence

- Returning null means the component renders nothing

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

Alternative

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

Short-circuit

Develop Intelligence

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

Lists

- Use Array.map as a functional loop

```
export function App() {
  const innerPlanets = ['Mercury', 'Venus','Earth','Mars'];
  return <>
           <h1>The inner planets are:</h1>
           <ul>
             {innerPlanets.map(p=><li>{p}</li>)}
           </ul>
         </>;
}
```

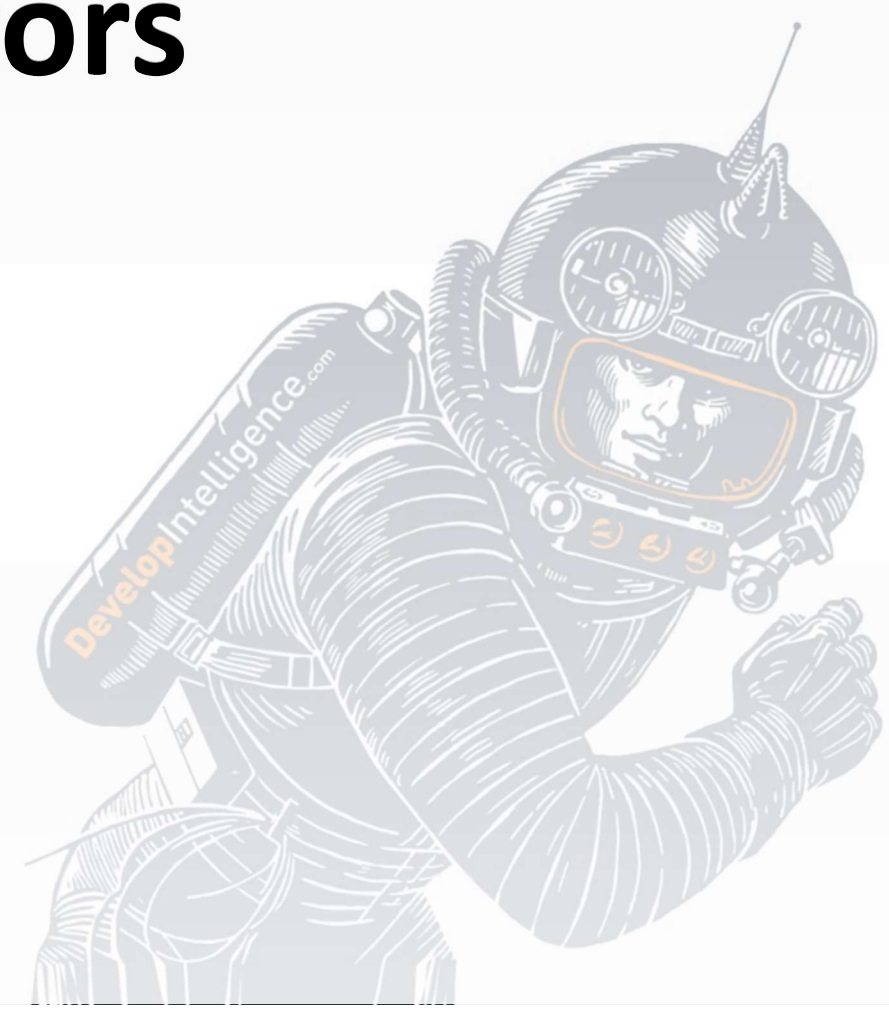
Lists: Better with key

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

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

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**
  - Reccomended
  - Simple rendering function
  - Keep state as hooks

Example: FizzBuzzComponent

Develop Intelligence

```
export class FizzBuzz extends Component {
  render() {
    return <div className="prose">
            <h1>FizzBuzz <i>[Complete]</i></h1>
            <ul>
              {getMessages().map(m => (
                <li>{m}</li>
              ))}
            </ul>
          </div>;
  }
}
```

Functional Equivalent

Develop Intelligence

```
export const FizzBuzz = () =>
  <div className="prose">
    <h1>FizzBuzz <i>[Complete]</i></h1>
    <ul>
      {getMessages().map(m => (
        <li>{m}</li>
      ))}
    </ul>
  </div>;
```

Functional advantages

Develop Intelligence

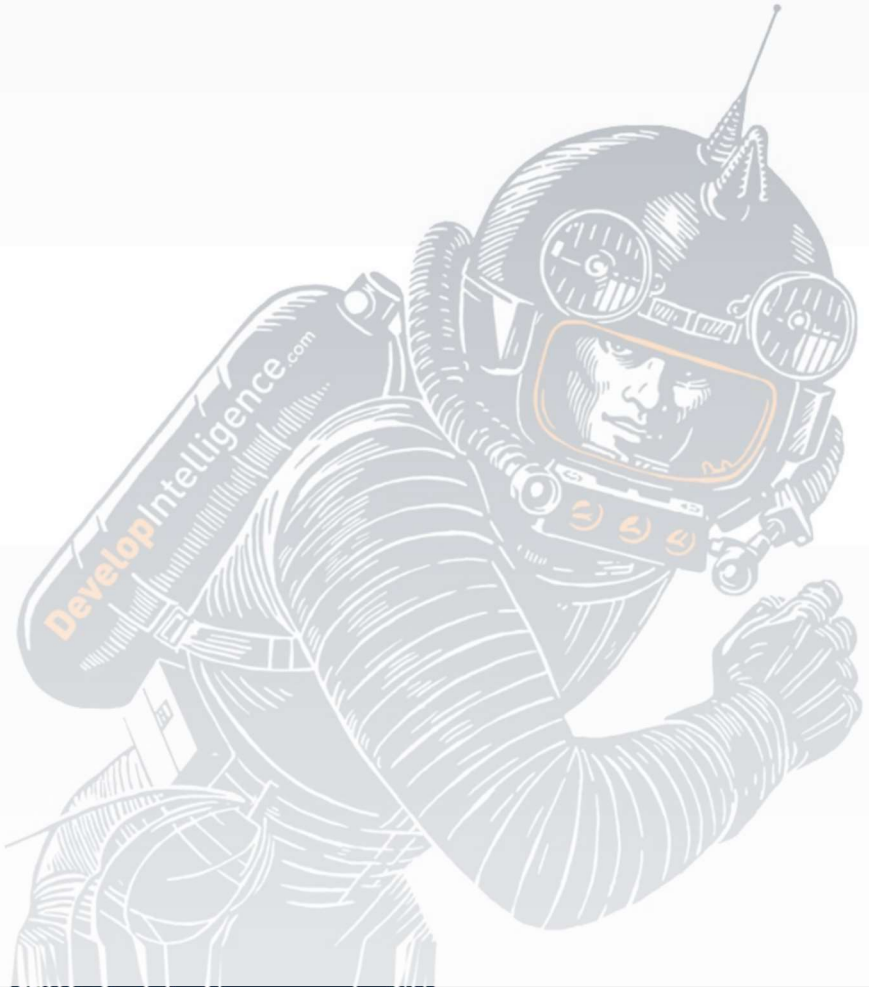
- **Faster.** No instantiation overhead
- **Simpler.** Just a render function.
- **More composable.** No lifecycle methods.
- **Easier to learn.** Especially for programming noobs.
- **Popular** Everyone else uses them.

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

Develop Intelligence

- 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

Develop Intelligence

- 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

Develop Intelligence

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

```
interface Props {
  intro:Model.Intro;
  skin:Skin;
}

export const Intro = ({ intro,skin }:Props) =>
  <Deck theme={skin.theme}>
    <TopicTitle skin={skin}># Introductions</TopicTitle>
    /*Etc...*/
  </Deck>;
```

Alternative

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

```
const Greet = ({givenName,surname}:{givenName:string,surname:string})=>
  <h1>Hello {givenName} {surname}!</h1>

export const App = () =>
  <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>
7      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
8    </>
```

Better: `PropsWithChildren<T>`

Develop Intelligence

- No implicit any

```
const SectionTitle = (props: PropsWithChildren) =>
  <h1>{props.children}</h1>;

export const App = () => (
  <>
     <SectionTitle>Overview</SectionTitle>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit...</p>
  </>
);
```

Mix'n'Match

Develop Intelligence

```
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

Develop Intelligence

- Keep it pure
  - No side effects
  - Referentially transparent

Bad idea

```
let guest = 0;

const Cup = ()=>
  <h2>Tea cup for guest #{guest++}</h2>;

export default TeaSet = () =>
   <><Cup /><Cup /><Cup /></>;
```

Develop
Intelligence

Events

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

Develop Intelligence

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

```
export const App = () => (
  <button
    onClick={() => alert('Hello world')}
  >
    Click me!
  </button>
);
```

Named Equivalent

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

```
export const App = () => {
  const handleClick = ()=>alert('Hello world!');
  return (
    <button
      onClick={handleClick}
    >
      Click me!
    </button>
  );
};
```

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

```
interface ButtonProps extends PropsWithChildren{
  onClick():void;
}

const Button = ({onClick,children}:ButtonProps) =>
  <button onClick={onClick}>{children}</button>

export const App = () =>
  <Button onClick={()=>alert('Hello world!')}>Click me</Button>
```

Event Information

Develop Intelligence

- In React, events are 'sythetic'
- `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


```tsx
interface Props {
  onColorChange(color: Color): void;
  color: Color;
}

export const ColorPicker = (props: Props) => (
  <select
    value={props.color}
    onChange={event => props.onColorChange(event.target.value as Color)}
  >
    <option value={Color.Black}>Black</option>
    <option value={Color.Brown}>Brown</option>
    <option value={Color.Red}>Red</option>
    <option value={Color.Blue}>Blue</option>
  </select>
);
```

Stopping Propagation

Develop Intelligence

```
const Button = ({ onClick, children }) =>
    <button onClick={e => {
      e.stopPropagation();
      onClick();
    }}>
      {children}
    </button>;
```

Develop
Intelligence

Style

Styling

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

Not this

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

This

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

Style

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

```
export const App = () => {
  const greetingStyle = {
    fontWeight:'bold', //NOT font-weight
    color:'green'
  };

  return <h1 style={greetingStyle}>Hello World.</h1>;
}
```

Style Pattern

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

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

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