

# **ReactJS.**

## Fundamentals

# Agenda

- **Background.**
- **JSX (JavaScript Extension Syntax).**
- **Developer tools.**
  - **Storybook.**
- **Component basics.**

# ReactJS.

- **A Javascript framework for building dynamic Web User Interfaces.**
  - **Single Page Apps technology.**
  - **Open-sourced in 2013**



- **Client-side framework.**
  - **More a library than a framework.**

# Before ReactJS.

- MVC pattern – **The convention for app design. Promoted by market leaders, e.g. AngularJS (1.x), EmberJS, BackboneJS.**
- **React is not MVC, just V.**
  - **It challenged established best practice (MVC).**
- Templating – **widespread use in the V layer;**
  - **React based on components.**

	<b>Templates</b>	<b>React components</b>
Separation of concerns	Technology (JS, HTML)	Responsibility
Semantic	New concepts and micro-languages	HTML and Javascript
Expressiveness	Underpowered	Full power of Javascript

# ReactJS

- **Philosophy:** *Build components, not templates.*
- **All about the User Interface (UI).**
  - **Not about business logic or the data model (Mvc)**
- **Component - A unit comprised:**
  - UI description (HTML) + UI behavior (JS)*
  - **Two aspects are tightly coupled and co-located.**
    - **Other frameworks decoupled them.**
  - **Benefits:**
    - 1. Improved Composition.**
    - 2. Greater Reusability.**
    - 3. Better Performance..**

# Creating the UI description

- `React.createElement()` – **create a HTML element.**
- `ReactDOM.render()` – **attach an element to the DOM.**
- `createElement()` **arguments:**
  1. **type (h1, div, span etc).**
  2. **properties (style, event handler etc).**
  3. **children.**
  - **We never use `createElement()` directly – far too cumbersome.**
- `ReactDOM.render()` **arguments:**
  1. **element to be displayed;.**
  2. **DOM node on which to mount the element.**
- **Ref. 01-UIDescription.html**
  - **Ref. 02-UIDescription.html - using nested method invocation**

# JSX.

- **JSX – JavaScript extension syntax.**
- **Declarative syntax for coding UI descriptions.**
- **Retains the full power of Javascript.**
- **Allows tight coupling between UI behavior and description.**
- **Must be transpiled (Babel) for browser execution compatibility.**
  - **Reference 03-JSX-error.html**
  - **Reference 04-JSX.html**

# REPL (Read-Evaluate-Print-Loop) transpiler.

The screenshot shows the Babel REPL interface in a Chrome browser. The address bar shows the URL: <https://babeljs.io/repl/#?babili=false&browsers=&build=&builtIns=false&spec=false&loose=false&co...>

**SETTINGS**

- ☒ Evaluate
- ☒ Line Wrap
- ☐ Minify
- ☒ Prettify
- ☐ File Size
- ☐ Time Travel

Source Type: Module

**PRESETS**

- ☒ es2015
- ☐ es2015-loose
- ☐ es2016
- ☐ es2017
- ☐ stage-0
- ☐ stage-1
- ☐ stage-2
- ☐ stage-3
- ☒ react

v6.26.0

**Input Code (Left):**

```
1 let rootElement =
2   <div className='myCSSstyle' >
3     <h1>Languages</h1>
4     <ul>
5       <li>Ruby</li>
6       <li>Javascript</li>
7     </ul>
8   </div> ;
9
10 ReactDOM.render(rootElement,
11  document.getElementById('mount-point') );
```

**Output Code (Right):**

```
1 'use strict';
2
3 var rootElement = React.createElement(
4   'div',
5   { className: 'myCSSstyle' },
6   React.createElement(
7     'h1',
8     null,
9     'Languages'
10  ),
11  React.createElement(
12    'ul',
13    null,
14    React.createElement(
15      'li',
16      null,
17      'Ruby'
18    ),
19    React.createElement(
20      'li',
21      null,
22      'Javascript'
23    )
24  )
25 );
```

**Reference**  
**04-JSX.html**

React is not defined



# JSX.

- **HTML-like markup.**
  - **It's actually XML code.**
- **Must be transformed (transpiled) into ES5.**
  - **The Babel tool suite.**
- **Some minor HTML tag attributes differences, e.g. className (class), htmlFor (for).**
- **Allows declarative description of the UI inlined in JavaScript.**
- **Combines the ease-of-use of templates with the power of JS.**

# Transpiling JSX.

- **What?**
  - **The Babel platform**
- **How?**
  1. **Manually, via REPL or command line.**
    - **When experimenting only.**
  2. **By the web server (using special tooling, i.e.Webpack).**
    - **Suitable for app development mode.**
  3. **As part of the build process for an app.**
    - **When deploying app for production.**

# React Components.

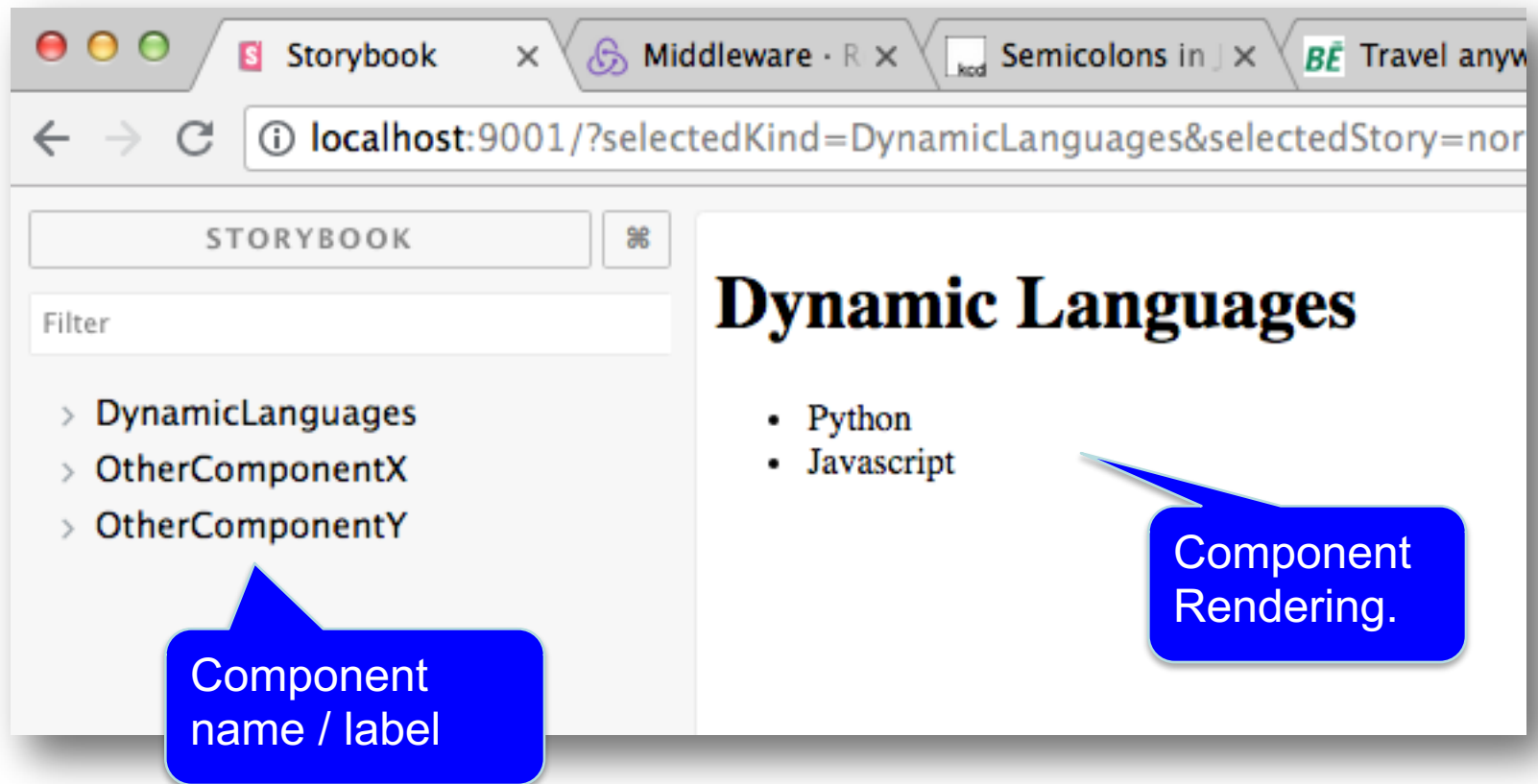
- **We develop COMPONENTS.**
  - **A JS class that extend** `React.Component`
  - The `render()` **method**:
    - **Mandatory.**
    - **Returns the component** UI description.
- **Can reference components as HTML tags.**
  - **E.g.** `ReactDOM.render(<ComponentName />, . . . . )`
- **Reference** `05-simpleComponent.html`

# React Developer tools.

- **create-react-app (CRA) tool. Features:**
  - **Scaffolding/Generator.**
  - **Development web server: auto-transpilation on file change + live reloading.**
  - **Builder: build production standard version of app, i.e. minification, bundling.**
- **Storybook tool:**
  - **A development environment for UI components.**
  - **Runs outside of your app → develop component in isolation.**
  - **Leads to more reusable, testable components.**
  - **Quicker development – ignore app-specific dependencies.**



- **Tool interface.**

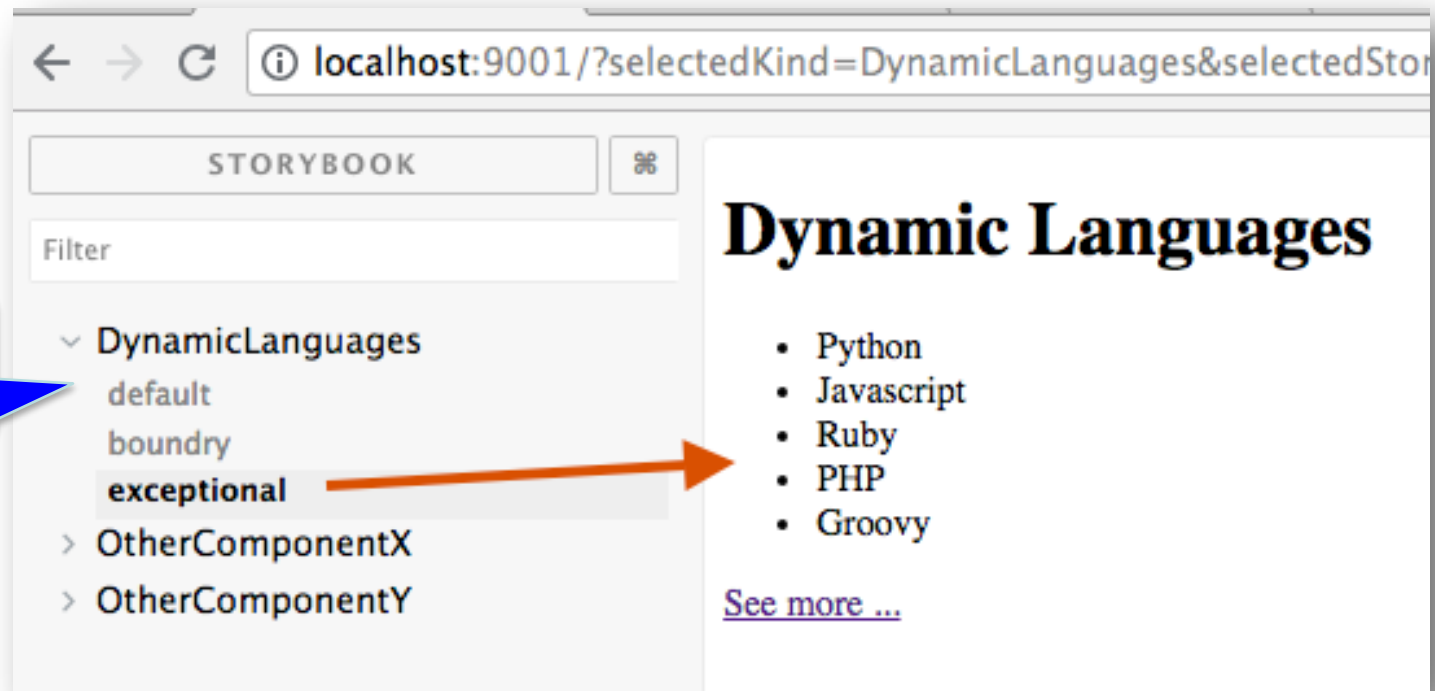




- **Component design considerations:**
  - **A component may have several STATES → State effects what it renders to the screen.**
- **EX.: DynamicLanguages component.**
  - **States might be:**
    - **Default – less than 5 languages → Render full list**
    - **Boundary – empty list → Render ‘No languages’ message**
    - **Exceptional – More than 5 languages → Render first 5 and a ‘See More...’ link to display next 5.**
- **Each state case termed a STORY.**

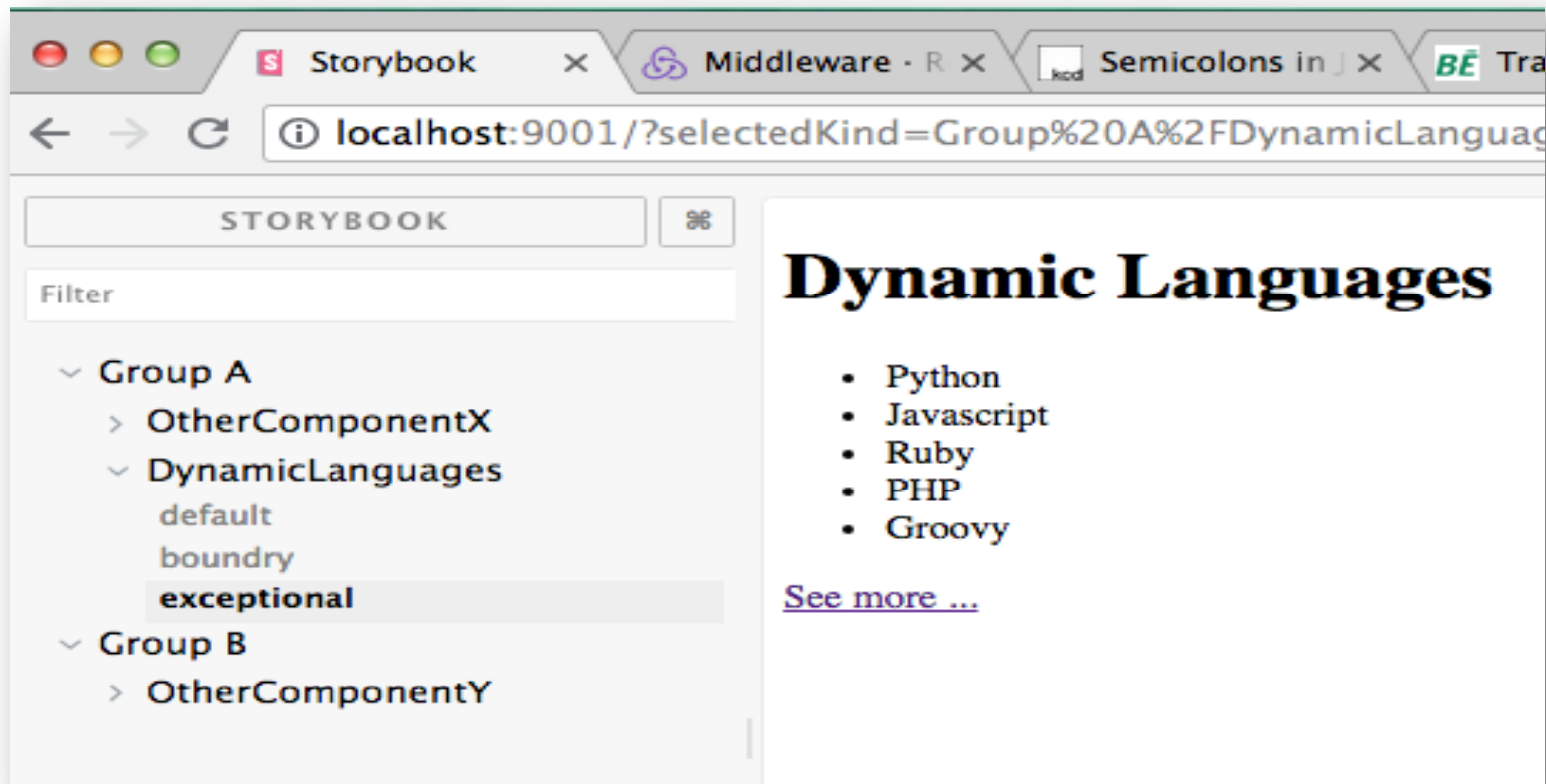


- Document **a component's stories** using nesting:





- For large component libraries, define component groups.
  - helps others understand the catalogue.





# Writing stories

- **Fluent-style API for writing stories.**
  - **Method chaining programming style.**

```
1 import React from 'react';
2 import { storiesOf } from '@storybook/react';
3 import DynamicLanguages from '../components/dynamicLanguages';
4
5 storiesOf('DynamicLanguages', module)
6   .add('default',
7     () => {
8       let languages = ['Python', 'Javascript', 'Ruby']
9       return <DynamicLanguages list={languages} />
10    }
11 )
12 .add('boundry',
13   () => . . . . .
14 )
15 .add('exceptional',
16   () => . . . . .
17 )
18
19 storiesOf('OtherComponentX', module)
20   .add('state 1',
21     () => . . . . .
22 )
23 . . . . .
```

Arrow  
function

3 stories for  
DynamicLanguages  
component

# Grouping stories.

- **Use directory pathname pattern** – *dir / subdir / subsubdir*,  
storiesof('Group A/Component 1')  
    .add('...'), () => {.....}  
    .add('...'), () => {.....}  
storiesof('Group A/Component 2')  
    .add('...'), () => {.....}  
    .add('...'), () => {.....}  
storiesof('Group B/Component X')  
    .add('...'), () => {.....}  
    .add('...'), () => {.....}  
    .add('...'), () => {.....}

# Grouping storoes

- **Lots of flexibility with grouping approach. Ex:**

```
storiesof('Group A/Component 1/')
```

```
  .add('story1', () => {.....})
```

```
storiesof('Group A/Component 1/')
```

```
  .add('story2', () => {.....})
```

```
storiesof('Group A/Component 1/')
```

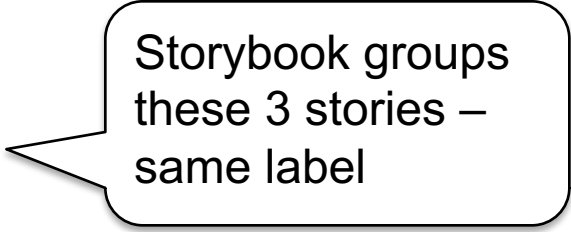
```
  .add('story3', () => {.....})
```

```
storiesof('Group A/Component 2/')
```

```
  .add('story1', () => {.....})
```

```
storiesof('Group A/Component 2/')
```

```
  .add('story1', () => {.....})
```

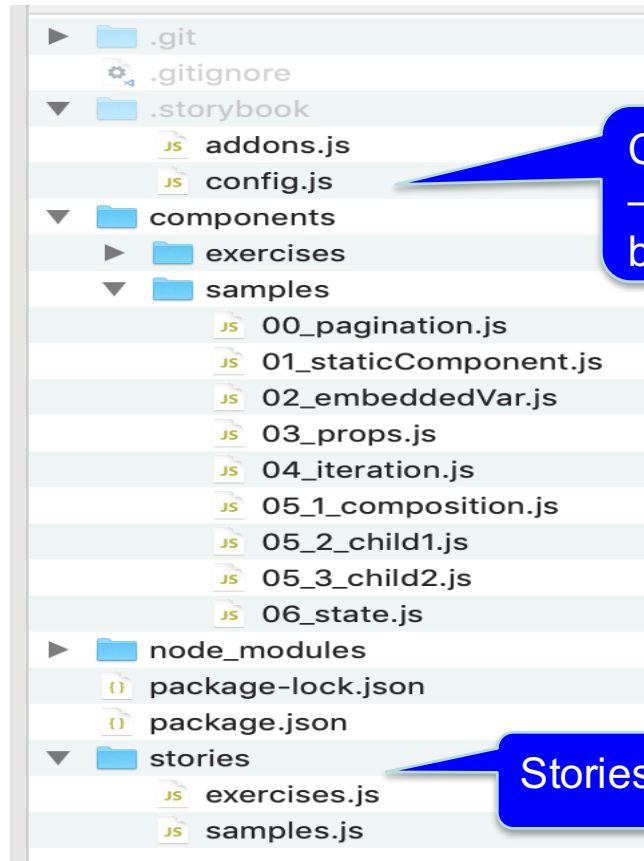
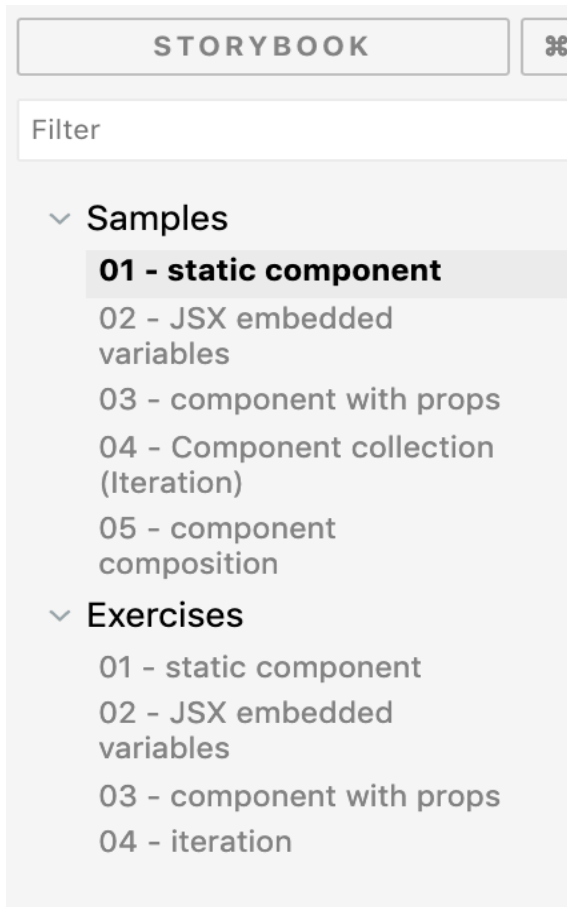


Storybook groups  
these 3 stories –  
same label

... back to components . . .

# Samples

- **Samples to demonstrate Component features.**
  - **Basis for this week's lab..**



Configuration  
– mainly  
boilerplate

Stories

# JSX embedded variables.

- Dereference variable embedded in JSX code using `{ }` braces.
  - Braces can contain any valid JS expression.
- Reference `samples/02_embeddedVariables.js`

JS 02\_embeddedVar.js ×

```
1  import React , { Component } from 'react';
2  export default class DynamicLanguagesEmbeddedVars extends Component {
3      render() {
4          let languages = ['Go', 'Julia', 'Swift']
5          let header = 'Modern'
6          return (
7              <div className='myCSSstyle' >
8                  <h1>`${header} Languages`</h1>
9                  <ul>
10                     <li>{languages[0]}</li>
11                     <li>{languages[1]} </li>
12                     <li>{languages[2]} </li>
13                 </ul>
14             </div>
15         );
16     }
17 }
```

# Reusability.

- **Achieve reusability through** parameterization.
- **props – Component properties/attribute;**
  - **Passing props to a component:**  
`<CompName prop1Name={value} prop2Name={value} . . . . />`
  - **Access inside component via this.props object:**  
`let p1 = this.props.prop1Name`
  - **Immutable.**
- **Reference** `samples/03_props.js` **and related story.**

## Aside – Some JS issues

1. When an arrow function has only ONE statement, which is its return value, then you may omit:
  - Body curly braces; 'return' keyword; Semi-colon
2. The Array map method – returns a new array based on applying a function to each element of the source array.

```
1  let frameworks = [  
2    {name: 'React', url : 'https://facebook.github.io/react/'},  
3    {name: 'Vue', url : 'https://vuejs.org/'},  
4    {name: 'Angular', url : 'https://angularjs.org/'}  
5  ] ;  
6  const names = frameworks.map((f,index) => `${index+1}. ${f.name}` )  
7  console.log(names)  
8  // [ '1. React', '2. Vue', '3. Angular' ]  
9
```



## Aside – Some JS issues

3. We can assign a single **JSX** element to a variable.

```
9  
0  - const demo = <div>  
1      <h1>Something</h1>  
2      <h2>Something else</h2>  
3      </div> ;
```

# Component collection - Iteration

- **Obj.:** Generate a collection of component instances.
- **Reference** samples/04\_iteration.js

```
▼ <div>
  <h1>JS client-side Web</h1>
  ▼ <ul> = $0
    ▼ <li>
      <a href="https://facebook.github.io/react/">React</a>
    </li>
    ▼ <li>
      <a href="https://vuejs.org/">Vue</a>
    </li>
    ▼ <li>
      <a href="https://angularjs.org/">Angular</a>
    </li>
  </ul>
</div>
```

HTML produced by  
story  
(From Chrome  
Dev Tools)

# The render() return value.

- **Examples:**

- return <h1>Something</h1> ;
- return <MyComponent prop1={.....} prop2={.....} /> ;
- return (  
    <div>  
        <h1>{this.props.type}</h1>  
        <ul>  
            .....  
        </ul>  
    </div>  
);

- **Must enclose in ( ) when multiline.**

# The render() return value.

- **Must return only ONE element.**
- **Examples:**
  - return (  
    <h1>{this.props.type}</h1>  
    <ul>  
        .....  
    </ul>  
    );
  - **Error** – ‘Adjacent JSX elements must be wrapped in an enclosing tag’
  - **Solution: Wrap elements in a ‘container’ tag.**

# The render() return value.

- **Old solution:**

```
return (  
  <div>  
    <h1>{this.props.type}</h1>  
    <ul>  
      .....  
    </ul>  
  </div>  
) ;
```

- **Added unnecessary depth to HTML / DOM → effected performance.**

- **New solution:**

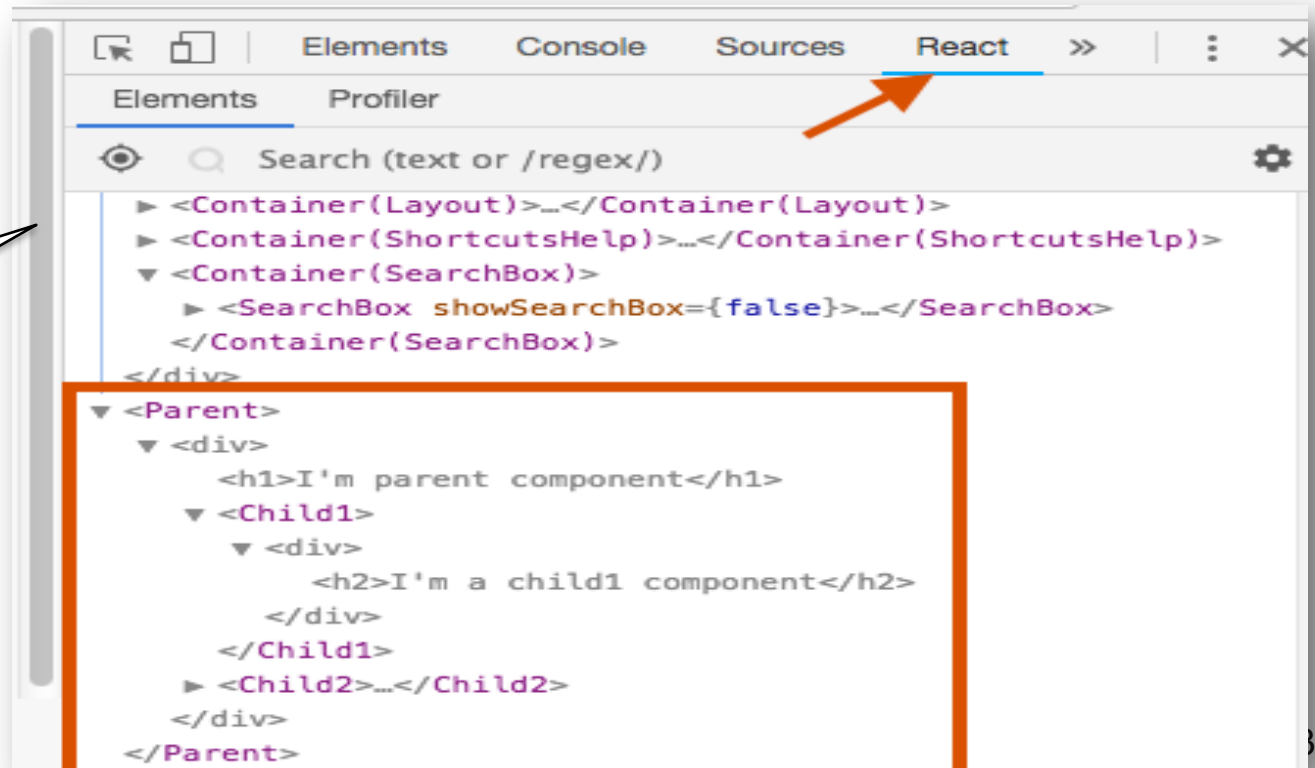
```
return (  
  <Fragment>  
    <h1>{this.props.type}</h1>  
    <ul>  
      .....  
    </ul>  
  </Fragment>  
) ;
```

- **Fragment – special React element.**
  - **No DOM presence.**

# Component **Composition**.

*A React application is designed as a hierarchy of components.*

- **Components have children – nesting.**
- **See 05\_1\_composition.js.**



From Chrome  
Dev Tools React  
extension

# Summary.

- **JSX.**
  - **Facilitates tight coupling between UI description and behaviour.**
- **All about components.**
  - **A class that extends `React.Component`.**
  - **The render method.**
  - **Parameterization via props.**
- **Storybook tool.**
  - **Develop components in isolation.**
  - **Story – the state (data values) of a component can effect its rendering (and behaviour).**

