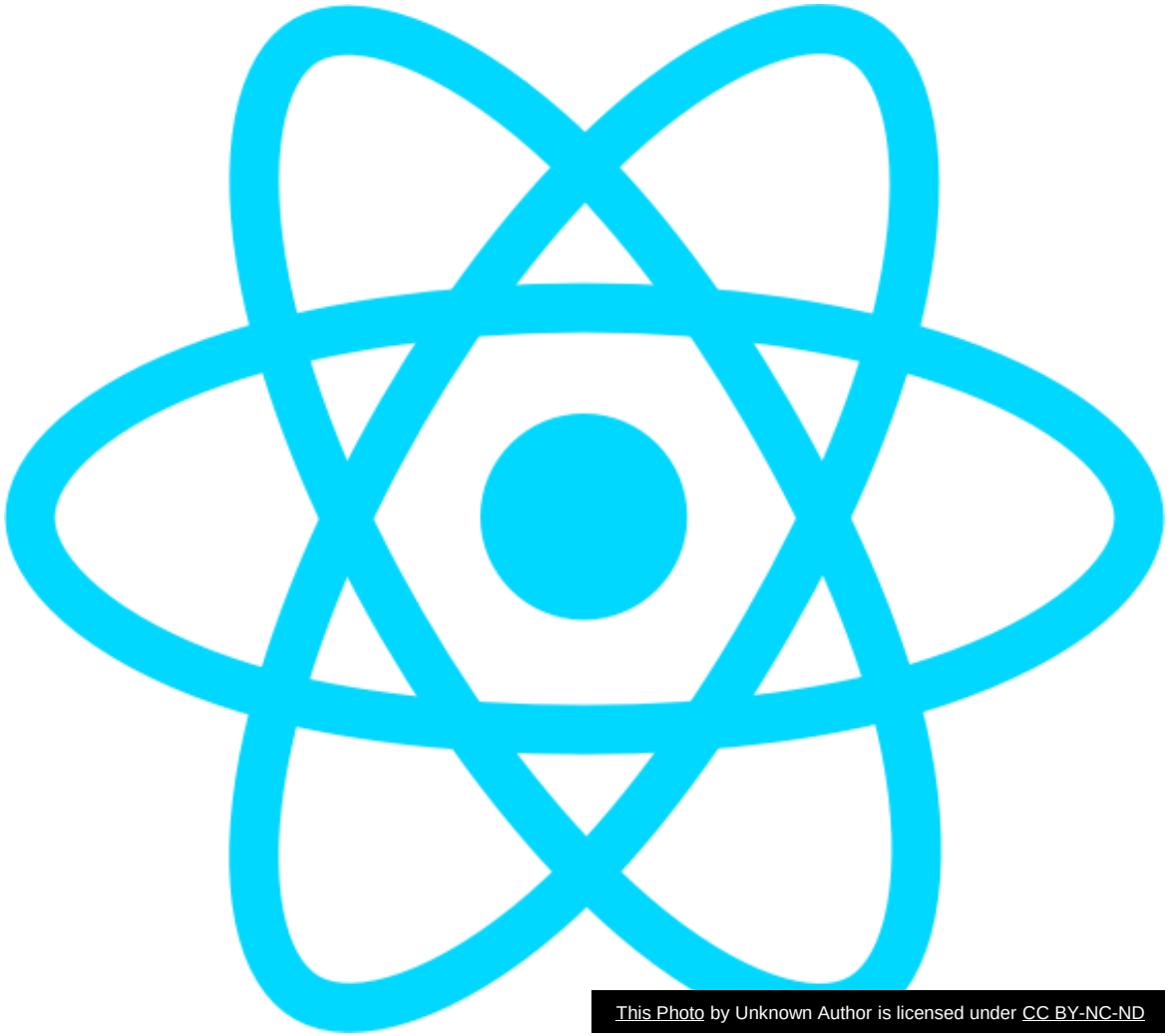
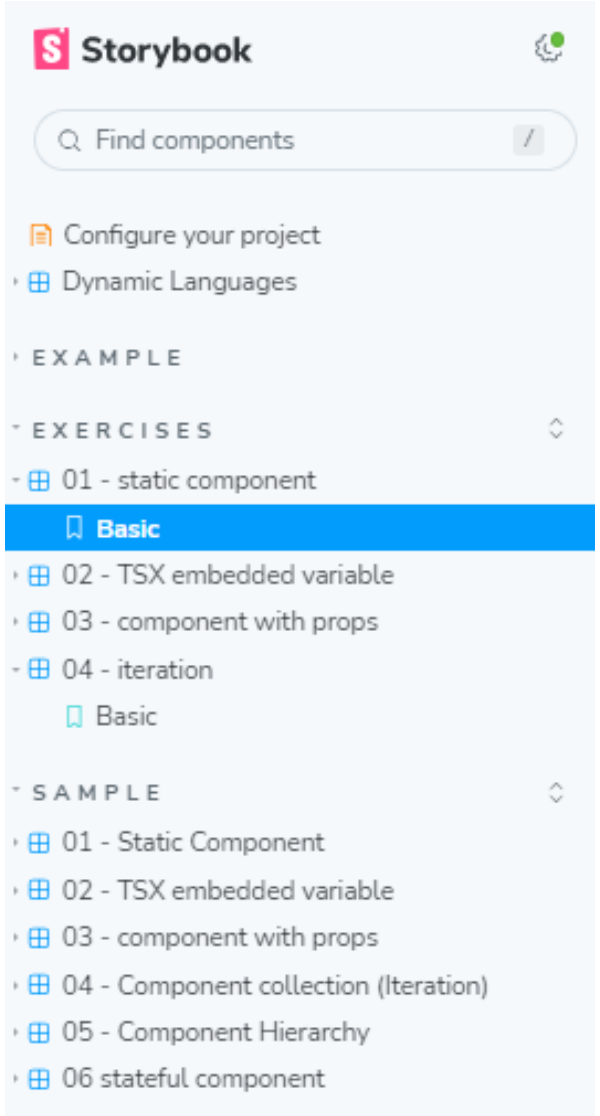
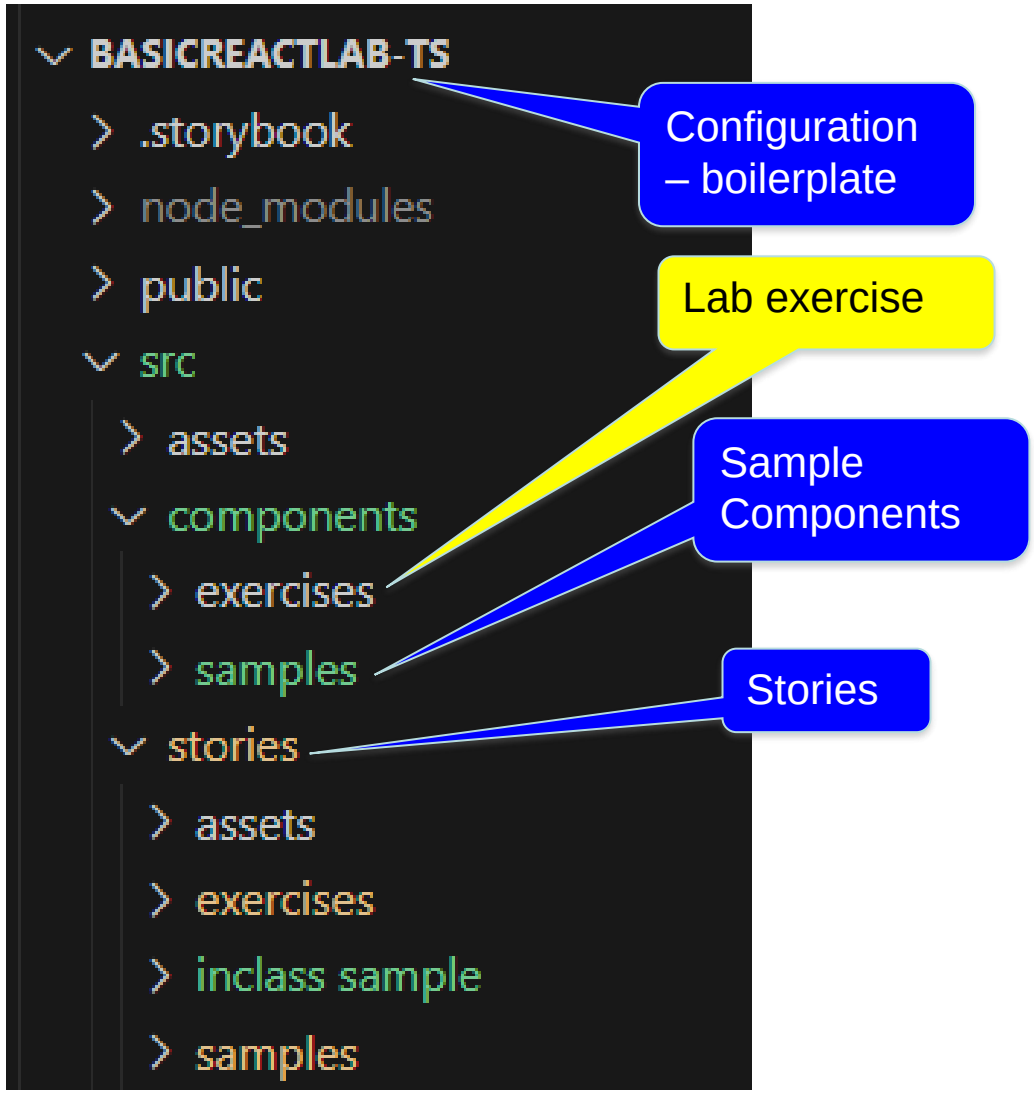
… back to components . . .



Demo Samples

(See lab exercise)





TSX - embedded variables.

- **Use { } to reference variable embedded in TSX.**
  - **Curly braces can contain any valid TS expression.**
- **Reference** src/components/samples/02_embeddedVariables.tsx

```tsx
import React from "react";

const Demo: React.FC = () => {
    const languages: string[] = ["Go", "Julia", "Kotlin"];
    const header: string = "Modern";

    return (
        <div>
            <h1>{`${header} Languages`}</h1>
            <ul>
                {languages.map(language => (
                    <li key={language}>{language}</li>
                ))}
            </ul>
        </div>
    );
};

export default Demo;
```

Reusability.

- We achieve reusability through **parameterisation.**
- **props –** Component properties / attribute / parameters.
  1. Passing props to a component:

```
<CompName  prop1Name={value}  prop2Name={value} . . . . />
```

  2. Access inside component via props object:

```
const ComponentName React.FC<PropInterface> = (props) => {
 const p1 = props.prop1Name
  ........
```

  3. Props are Immutable.
  4. Part of a component's design.
- Reference **src/components/samples/03_props.tsx (**and related story).

Aside.

- **We can assign a** single **TSX element to a variable.**

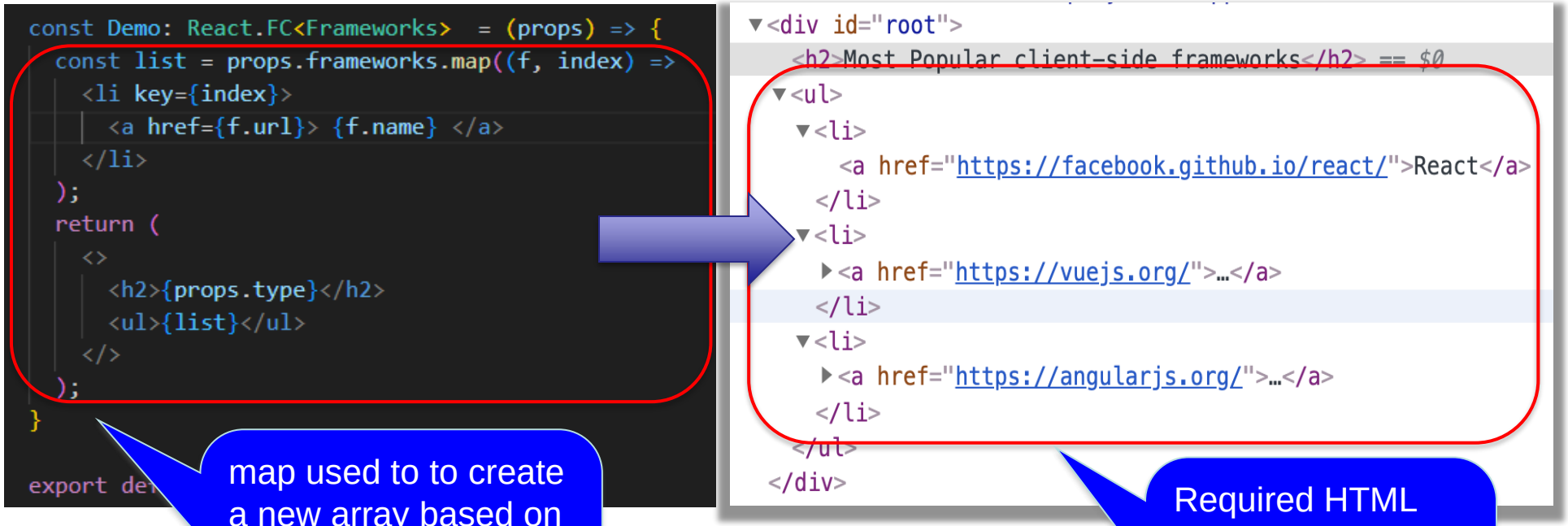
```
const demo = <div>
                <h1>Something</h1>
                <h2>Something else</h2>
             </div> ;
```

- **Why?**

```
const demo = React.createElement(
  "div",
  null,
  React.createElement("h1", null, "Something"),
  React.createElement("p", null, "Some text ...")
);
```

Component collection - Iteration

- **Use case:** Generate an array of (similar) component from a data array.
- **Reference:** src/components/samples/04_iteration.tsx



Component return value.

- **Examples:**;
  1. return <MyComponent prop1={…..} prop2={……} /> ;
  2. return (

```
<div>
  <h1>{this.props.type}</h1>
  <MyComponent prop1={…..} prop2={……} />
  <p>
      . . . . . .
  </p>
</div>
```

) ;

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

Component return value.

- **Must return only ONE element.**
- **Error Examples:**
  - return (

```
<h1>{this.props.type}</h1>
<MyComponent prop1={…..} prop2={……} />
<p>
    . . . . . .
</p>
```

) ;
  - **Error** – 'Adjacent JSX elements must be wrapped in an enclosing tag'
  - **Solution: Wrap elements in a tag.**

Component return value.

- **Old solution:**

```
return (
    <div>
        <h1> ……</h1>
        <MyComponent …… />
        <p> …….. </p>
     </div>
    ) ;
```

- **Adds unnecessary depth to DOM → affects performance.**

- **Alternative solution:**

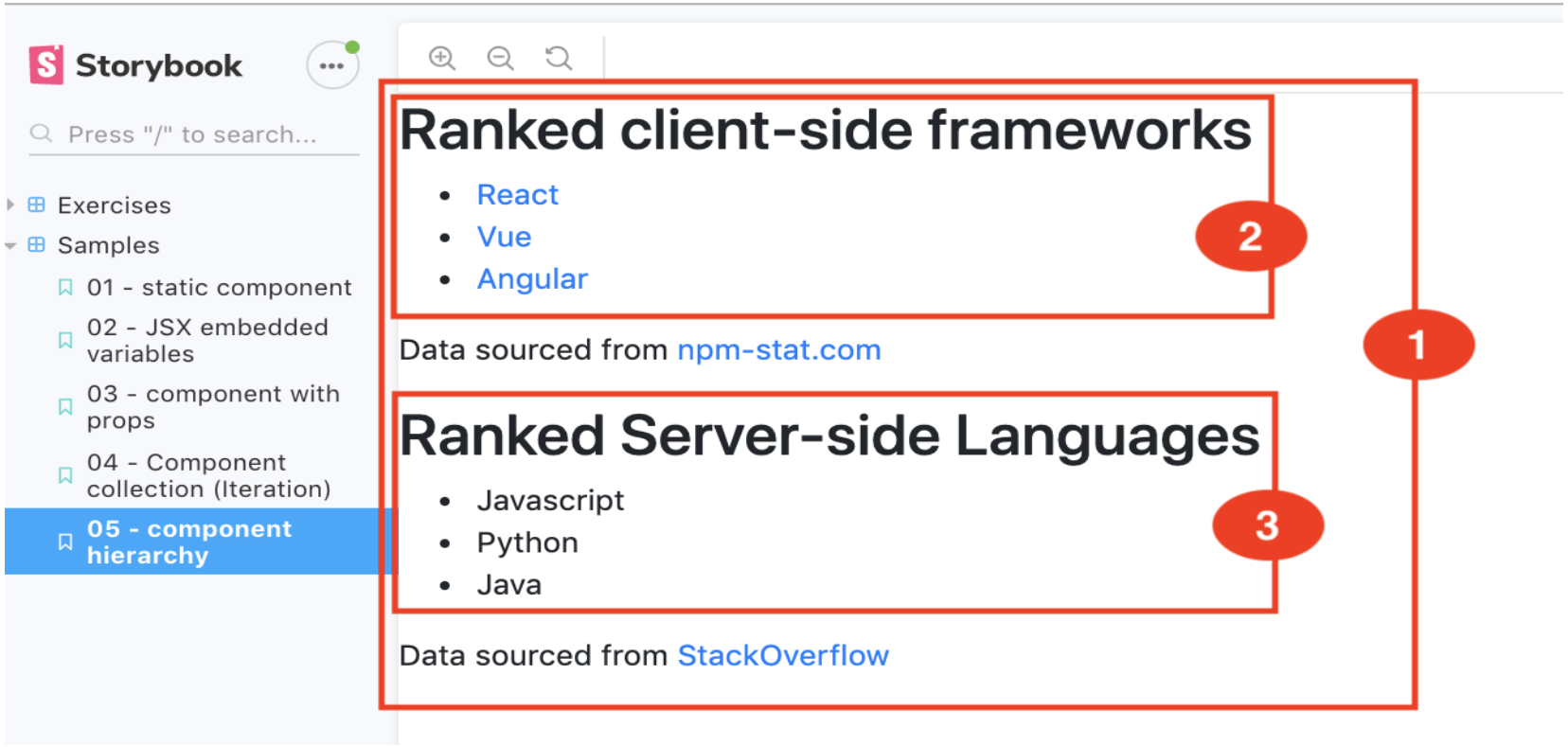
```
return (
    <>
        <h1> ……</h1>
        <MyComponent …… />
        <p> …….. </p>
     </>
    ) ;
```

- **<> </> – special React element, termed** Fragment**.**
  - **No DOM presence.**

Component ***Hierarchy***.

All React application are designed as a hierarchy of components.

- **Components have** children **– nesting.**
- **Ref**. src/components/samples/05_hierarchy.ts.



Summary.

- **TSX.**
  - **UI** description **and** behaviour **tightly coupled.**
  - **Can embed variables/expressions with braces.**
- **All about components.**
  - **A function that takes a props argument and returns a single TSX element .**
  - **Components can be nested.**
- **Storybook tool.**
  - **Develop components in isolation.**
  - **Story – the state (data values) of a component can affect its rendering (and behaviour).**