**GENERAL JSX RULES** 

**GENERAL JSX RULES** 

### **GENERAL JSX RULES**

JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)

#### **GENERAL JSX RULES**

- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator

#### **GENERAL JSX RULES**

- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator
- Statements are not allowed (if/else, for, switch)

#### **GENERAL JSX RULES**

- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator
- Statements are not allowed (if/else, for, switch)
- JSX produces a JavaScript expression

```
const el = <h1>Hello React!</h1>;
const el = React.createElement("h1", null, "Hello React!");
```

#### **GENERAL JSX RULES**

- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator
- Statements are not allowed (if/else, for, switch)
- JSX produces a JavaScript expression

```
const el = <h1>Hello React!</h1>;
const el = React.createElement("h1", null, "Hello React!");
```

We can place other pieces of JSX inside {}

#### **GENERAL JSX RULES**

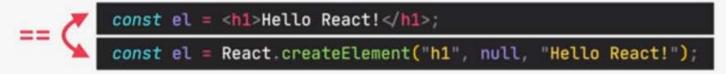
- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator
- Statements are not allowed (if/else, for, switch)
- JSX produces a JavaScript expression



- We can place other pieces of JSX inside {}
- We can write JSX anywhere inside a component (in if/else, assign to variables, pass it into functions)

#### **GENERAL JSX RULES**

- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator
- Statements are not allowed (if/else, for, switch)
- JSX produces a JavaScript expression



- We can place other pieces of JSX inside {}
- We can write JSX anywhere inside a component (in if/else, assign to variables, pass it into functions)
- A piece of JSX can only have one root element. If you need more, use <React.Fragment> (or the short <>)

#### **GENERAL JSX RULES**

- JSX works essentially like HTML, but we can enter "JavaScript mode" by using {} (for text or attributes)
- We can place JavaScript expressions inside {}.
  Examples: reference variables, create arrays or objects,
  [].map(), ternary operator
- Statements are not allowed (if/else, for, switch)
- JSX produces a JavaScript expression



- We can place other pieces of JSX inside {}
- We can write JSX anywhere inside a component (in if/else, assign to variables, pass it into functions)
- A piece of JSX can only have one root element. If you need more, use <React.Fragment> (or the short <>)

- className instead of HTML's class
- htmlFor instead of HTML's for
- Every tag needs to be closed. Examples: <img /> or <br />
- All event handlers and other properties need to be camelCased. Examples: onClick or onMouseOver
- Exception: aria-\* and data-\* are written with dashes like in HTML
- CSS inline styles are written like this: {{<style>}} (to reference a variable, and then an object)
- CSS property names are also camelCased
- Comments need to be in {} (because they are JS)