

# RULES OF JSX

# RULES OF JSX

## GENERAL JSX RULES

# RULES OF JSX

## GENERAL JSX RULES

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## GENERAL JSX RULES

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

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## 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

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## 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`)

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## 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!");
```

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## 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!");
```

- 1 We can place **other pieces of JSX** inside `{}`

## DIFFERENCES BETWEEN JSX AND HTML



# RULES OF JSX

## 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!");
```


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

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## 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!");
```


- 1 We can place **other pieces of JSX** inside `{}`
  - 2 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 `<>`)

## DIFFERENCES BETWEEN JSX AND HTML

# RULES OF JSX

## 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!");
```

- 1 We can place **other pieces of JSX** inside `{}`
  - 2 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 `<>`)

## DIFFERENCES BETWEEN JSX AND HTML

- 👉 `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)