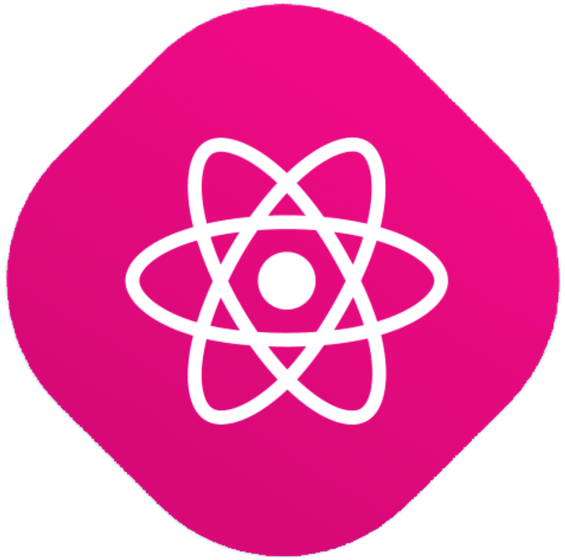
ЯВК

Junior phase



A Journey Through React

JSX - Components - States - Props

What is React

Welcome to the world of React JS! In today's web development landscape, React has become one of the most popular and widely used Javascript librairies. It offers developers a powerful set of tools for building dynamic and interactive user interfaces with ease.

But why is React so important? For starters, it allows developers to create reusable components that can be easily integrated into any project. This saves time and effort, making it easier to build complex applications.

What is React

Additionally, React simplifies the process of managing state and data flow, making it easier to build scalable and maintainable applications. So if you're looking to stay ahead in the world of web development, learning React is a must!

To further understand the full extent of React, we have to get familiar with a few fundamental concepts of it :

- **JSX (Javascript XML)**
- **Components**
- **States & Props**

Let’s Start

Our Journey Through React

Javascript XML (JSX for short)

JavaScript XML is a syntax extension of JavaScript that allows us to write HTML-like code in our JavaScript files. This is what JSX looks like in code :

```
const avatar = 'https://i.imgur.com/7vQD0fPs.jpg';
const description = 'Gregorio Y. Zara';
```

```
<img
  className="avatar"
  src={avatar}
  alt={description}
/>
```

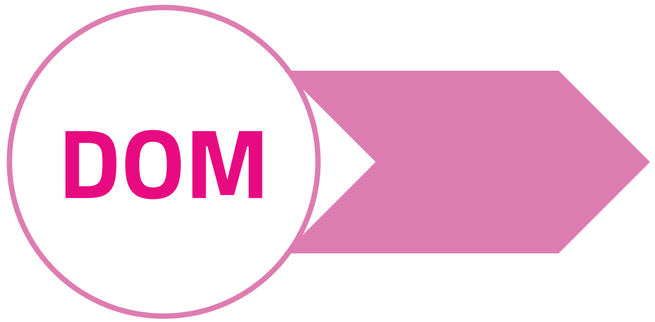
Important note: JSX is not a requirement for using React, but it does make it easier to work with React components.

Javascript XML (JSX for short)

When using JSX, we can use HTML-like tags to create React elements. These elements can then be rendered to the DOM just like any other HTML element. Here's what JSX looks like in action:

```
const greeting = "Hello world, "
const occupation = "web developer"
```

```
<p>{greeting}I'm a {occupation}</p>
```




```
Hello world, I'm a web developer
```

React Components

Now that we have a clear understanding of what JSX is in React, our next step is to delve into React components. React components are the building blocks of a React application, encapsulating reusable and self-contained sections of user interface logic and rendering.

There are two primary types of React components:

- **Functional components :** written as JavaScript functions that return JSX elements.
- **Class components :** ES6 classes that extend React.Component and include additional features like state management and lifecycle methods.

React Components

Functional vs Class

- Functional components are a fundamental part of React's component architecture.
- Functional components are primarily used for presenting UI elements and handling rendering logic.
- They don't have state or lifecycle methods by default, but with the introduction of React Hooks, functional components can now manage state and side effects.

- Class components were the traditional way of defining React components before the introduction of functional components and hooks.
- They offer a more structured approach to component development.
- Class components can define class properties (variables and functions) that can be accessed and reused throughout the component, promoting code organization and reusability.

What do React Components look like?

React Components

Functional vs Class

```jsx
export default function Profile() {
  return (
    <div>
      <h1>Hedy Lamarr's Todos</h1>
      <img
        src="https://i.imgur.com/yXOvdOSs.jpg"
        alt="Hedy Lamarr"
        className="photo"
      />
    </div>
  );
}

```

```jsx
class Profile extends React.Component {
  render() {
    return (
      <div>
        <h1>Hedy Lamarr's Todos</h1>
        <img
          src="https://i.imgur.com/yXOvdOSs.jpg"
          alt="Hedy Lamarr"
          className="photo"
        />
      </div>
    );
  }
}
```

React Props

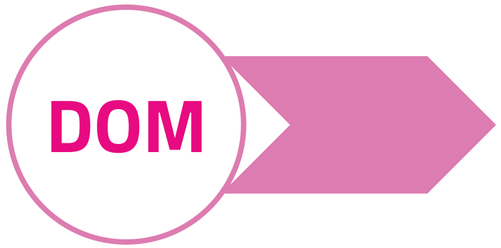
Props (short for "properties") are a way to pass data from a parent component to a child component in React. Props are read-only, meaning that the child component **cannot** modify them. They allow you to customize and configure child components based on the parent's data.

Let’s look at an example of a component that receives and uses props

React Props (Functional)

```jsx
const Greeting = (props) => {
  return <p>Hello, {props.name}!</p>;
}

export default App = () => {
  return (
    <div>
      <Greeting name="Alice" />
      <Greeting name="Bob" />
    </div>
  );
}
```



Hello, Alice!

Hello, Bob!

React Props (Class)

```
1  class Greeting extends React.Component {
2    constructor(props) {
3      super(props)
4    }
5    render() {
6      return <p>Hello, {this.props.name}!</p>;
7    }
8  }
9
10 class App extends React.Component {
11   render() {
12     return (
13       <div>
14         <Greeting name="Alice" />
15         <Greeting name="Bob" />
16       </div>
17     );
18   }
19 }
20
```

DOM

Hello, Alice!

Hello, Bob!

React States

State is used to manage and store data that can change over time and impact a component's rendering. **Unlike** props, which are passed from a parent component, state is managed internally within a component and can be updated.

Functional components can manage state using the useState **hook**

In class-based components, state is managed within the component using this.state and updated using this.setState

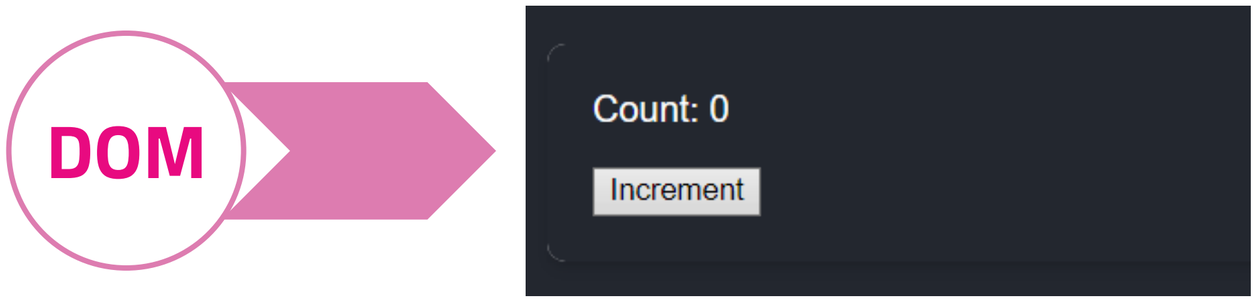
Let’s look at an example!

React States (Functional)

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```



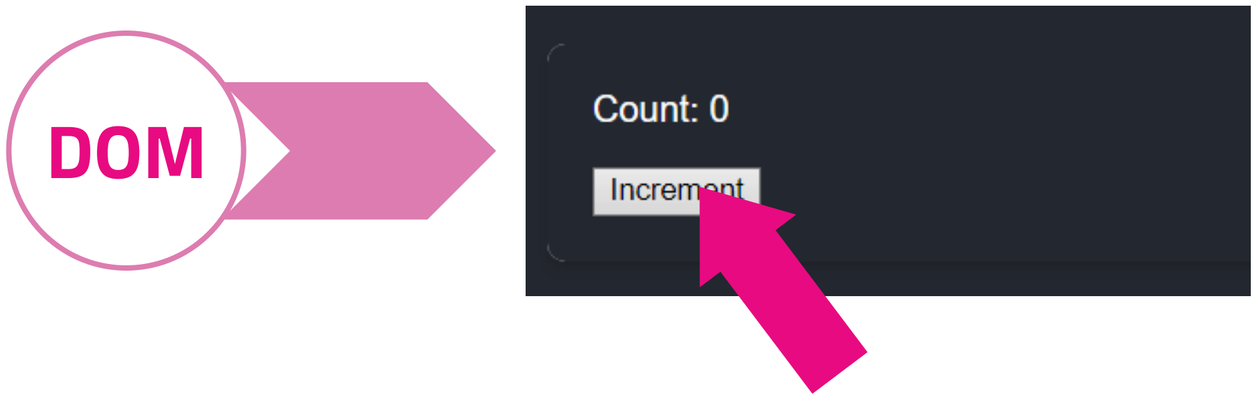
Junior phase

React States (Functional)

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```



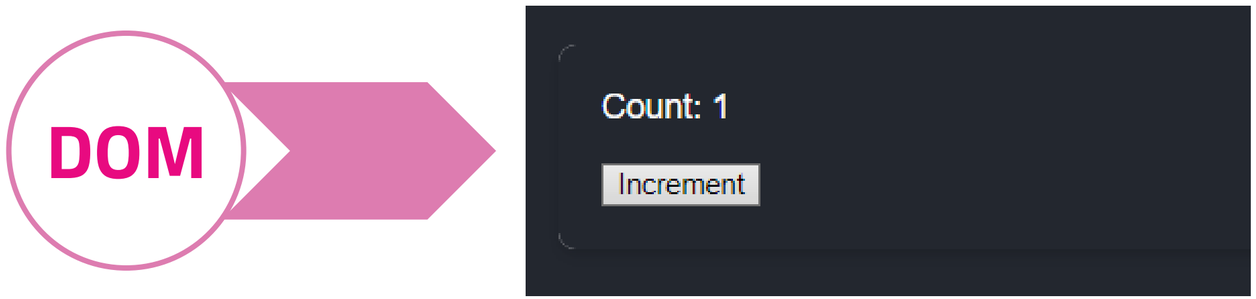
User Click

React States (Functional)

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
}
```



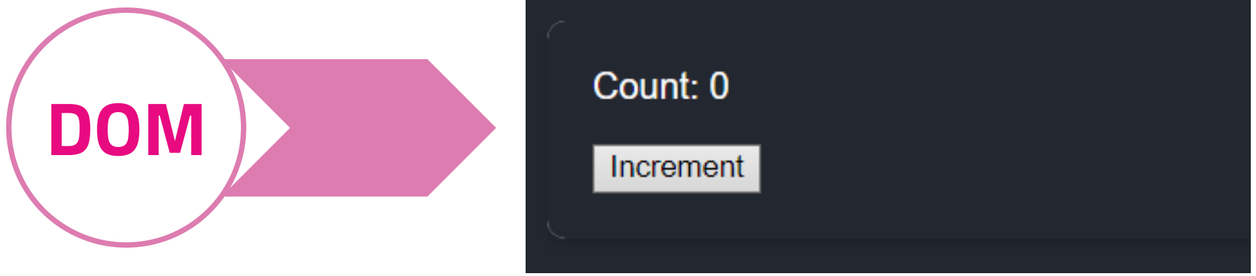
React States (Class)

```jsx
import React from 'react';

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  incrementCount() {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.incrementCount()}>
          Increment
        </button>
      </div>
    );
  }
}
```



React States (Class)

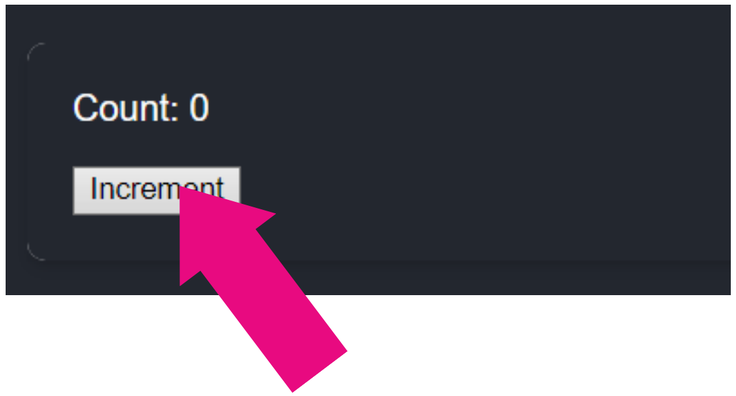
```jsx
import React from 'react';

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  incrementCount() {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.incrementCount()}>
          Increment
        </button>
      </div>
    );
  }
}
```

DOM



User Click

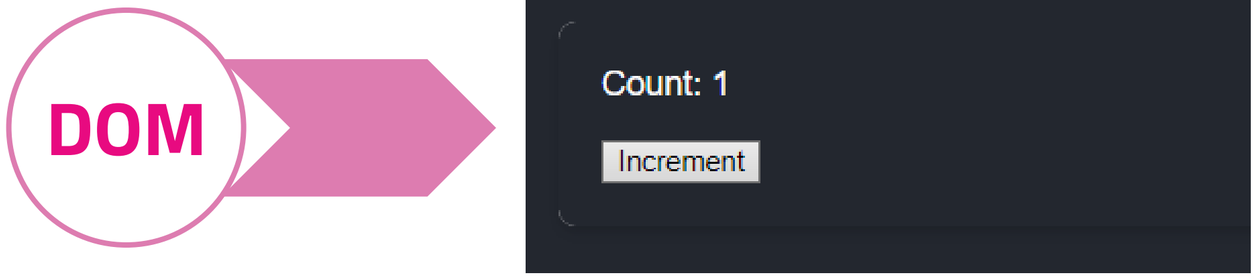
React States (Class)

```jsx
import React from 'react';

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }

  incrementCount() {
    this.setState({ count: this.state.count + 1 });
  }

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.incrementCount()}>
          Increment
        </button>
      </div>
    );
  }
}
```



States vs. Props

- Read-only and provide a way to configure and customize child components based on data from their parent.
- Defined in the parent component and are passed down to child components as attributes.
- Flow in a unidirectional manner, from parent to child, and cannot be modified by child components

- A way to manage and store data that can change over time within a component.
- Allows a component to keep track of its internal data and re-render when that data changes.
- Commonly used for managing component-specific data, such as form input values, counters, or UI-related information that may change during a component's lifecycle.

States vs. Props

Props

- Read-only and provide a way to configure and customize child components based on data from their parent.
- Defined in the parent component and are passed down to child components as attributes.
- Flow in a unidirectional manner, from parent to child, and cannot be modified by child components

States

- A way to manage and store data that can change over time within a component.
- Allows a component to keep track of its internal data and re-render when that data changes.
- Commonly used for managing component-specific data, such as form input values, counters, or UI-related information that may change during a component's lifecycle.

What Makes React Reactive

Introducing The V-DOM (Virtual DOM)

What Makes React Reactive

The Virtual DOM is an in-memory representation of the actual DOM. It's a lightweight copy of the real DOM, constructed and managed by React. Instead of interacting directly with the browser's DOM, React works with the Virtual DOM.

To visualize the virtual DOM in action, imagine a complex web application with multiple components. When data changes, React calculates the difference between the previous and current virtual DOM representations, making minimal updates to the real DOM. This process, known as **reconciliation**, ensures an efficient and responsive UI.

But, How does it work?

What Makes React Reactive

1. When you update a component's state or props, React generates a new Virtual DOM tree.
2. React then performs a process called "reconciliation" to identify the differences (or "diffs") between the new Virtual DOM and the previous one.
3. React determines the most efficient way to update the real DOM to match the changes in the Virtual DOM.
4. Finally, React updates the actual DOM with the minimum number of changes required to reflect the new state or props.

Recap