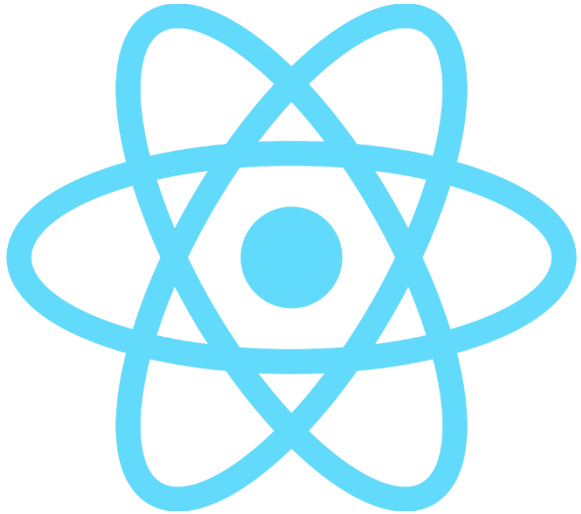
AFFICHAGE CONDITIONNEL

COURS : INTRODUCTION A REACT JS

Enseignant : M. Florent BALKOULGA

CONDITIONS



Affichage conditionnel

En React, vous pouvez concevoir des composants distincts qui encapsulent le comportement voulu. Vous pouvez alors n'afficher que certains d'entre eux, suivant l'état de votre application.

L'affichage conditionnel en React fonctionne de la même façon que les conditions en Javascript. On utilise l'instruction Javascript if ou l'opérateur ternaire pour créer des éléments représentant l'état courant, et on laisse React mettre à jour l'interface utilisateur (UI) pour qu'elle corresponde.

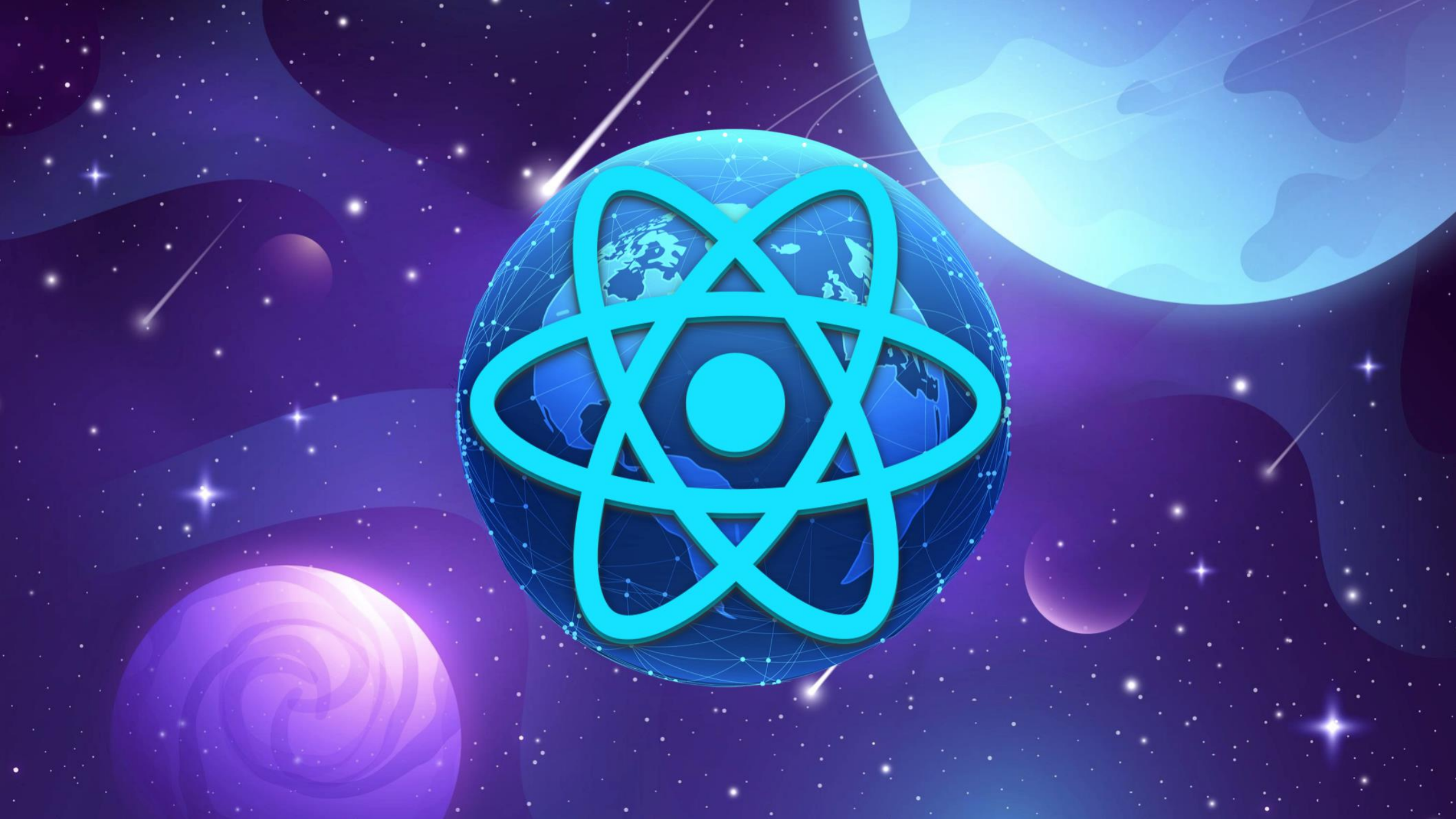
Affichage conditionnel

Considérons ces deux composants :

```
function UserGreeting(props) {
  return <h1>Bienvenue !</h1>;
}
function GuestGreeting(props) {
  return <h1>Veuillez vous inscrire.</h1>;
}
```

Nous allons créer un composant **Greeting** qui affiche l'un de ces deux composants, **selon qu'un utilisateur soit connecté ou non :**

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn)
    return <UserGreeting />;
  return <GuestGreeting />;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
// Essayez de changer ça vers isLoggedIn={true} :
root.render(<Greeting isLoggedIn={false} />);
```



Variables d'éléments

Vous pouvez stocker les éléments dans des variables. Ça vous aide à afficher de façon conditionnelle une partie du composant tandis que le reste ne change pas.

Prenons ces deux nouveaux composants, qui représentent les boutons de **Déconnexion** et de **Connexion** :

```
function LoginButton(props) {
  return (
    <button onClick={props.onClick}>
      Connexion
    </button>
  );
}

function LogoutButton(props) {
  return (
    <button onClick={props.onClick}>
      Déconnexion
    </button>
  );
}
```

Variables d'éléments

Dans l'exemple ci-dessous, nous allons créer un composant à état appelé **LoginControl**.

Il affichera soit **<LoginButton />**, soit <LogoutButton />, selon son état courant.
Il affichera aussi un **<Greeting />** de l'exemple précédent :

Essayer

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }
  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }
  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }
  render() {
    const isLoggedIn = this.state.isLoggedIn;
    let button;
    if (isLoggedIn)
      button = <LogoutButton onClick={this.handleLogoutClick} />;
    else
      button = <LoginButton onClick={this.handleLoginClick} />;

    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<LoginControl />);
```

Condition à la volée avec l'opérateur &&

Vous pouvez utiliser n'importe quelle expression dans du JSX en l'enveloppant dans des accolades. Ça vaut aussi pour l'opérateur logique Javascript &&. Il peut être pratique pour inclure conditionnellement un élément :

Essayer

Ça fonctionne parce qu'en JavaScript, true && expression est toujours évalué à expression, et false && expression est toujours évalué à false.

Du coup, si la condition vaut true, l'élément juste après && sera affiché. Si elle vaut false, React va l'ignorer et le sauter.

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Bonjour !</h1>
      {unreadMessages.length > 0 &&
        <h2>
          Vous avez {unreadMessages.length} message(s) non-lu(s).
        </h2>
      }
    </div>
  );
}

const messages = ['React', 'Re: React', 'Re:Re: React'];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Mailbox unreadMessages={messages} />);
```

Alternative à la volée avec opérateur ternaire

Une autre méthode pour l'affichage conditionnel à la volée d'éléments consiste à utiliser l'opérateur ternaire Javascript condition ? trueValue : falseValue.

Dans l'exemple ci-dessous, on l'utilise pour afficher conditionnellement un bloc de texte.

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      L'utilisateur <b>{isLoggedIn ? 'est actuellement' : 'n'est pas'}</b> connecté.
    </div>
  );
}
```

Alternative à la volée avec opérateur ternaire

On peut aussi l'utiliser pour des expressions plus longues, même si c'est moins clair :

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      {isLoggedIn
        ? <LogoutButton onClick={this.handleLogoutClick} />
        : <LoginButton onClick={this.handleLoginClick} />
      }
    </div>
  );
}
```

Tout comme en Javascript, c'est à vous de choisir un style approprié selon les préférences de lisibilité en vigueur pour vous et votre équipe. Souvenez-vous aussi que chaque fois que des conditions deviennent trop complexes, c'est peut-être le signe qu'il serait judicieux d'en extraire un composant.

Empêcher l'affichage d'un composant

Dans de rares cas, vous voudrez peut-être qu'un composant se masque alors même qu'il figure dans le rendu d'un autre composant. Pour ce faire, il suffit de renvoyer **null** au lieu de son affichage habituel.

Dans l'exemple ci-dessous, **<WarningBanner />** s'affichera en fonction de la valeur de la **prop warn**. Si la valeur est false, le composant ne s'affiche pas :

Empêcher l'affichage d'un composant

Essayer

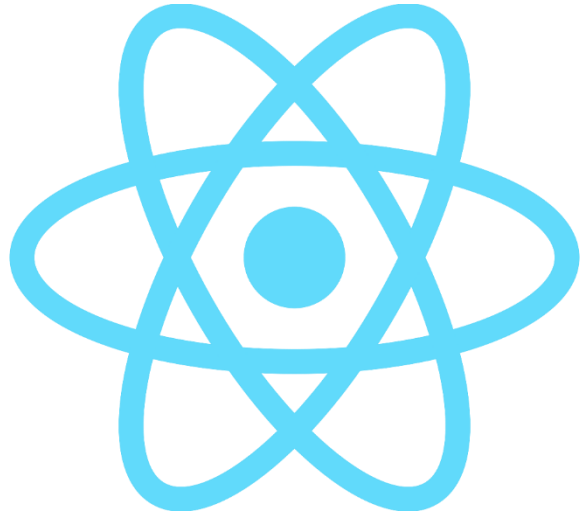
Renvoyer **null** depuis la méthode **render** d'un composant n'affecte pas l'appel aux méthodes du cycle de vie du composant. Par exemple, **componentDidUpdate** sera quand même appelée.

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }
  return (
    <div className="warning">
      Attention !
    </div>
  );
}
class Page extends React.Component {
  constructor(props) {
    super(props);
    this.state = {showWarning: true};
    this.handleToggleClick = this.handleToggleClick.bind(this);
  }
  handleToggleClick() {
    this.setState(state => ({
      showWarning: !state.showWarning
    }));
  }

  render() {
    return (
      <div>
        <WarningBanner warn={this.state.showWarning} />
        <button onClick={this.handleToggleClick}>
          {this.state.showWarning ? 'Masquer' : 'Afficher'}
        </button>
      </div>
    );
  }
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Page />);
```

RESSOURCES



Ressources

- ✓ https://kinsta.com/fr/blog/rendu-conditionnel-react/
- ✓ https://developpement-web-facile.com/affichage-conditionnel-react/
- ✓ https://www.savoirdanslavie.com/conditional-rendering-react-js/
- ✓ https://technoglitz.com/france/affichage-conditionnel-dans-react-cloudsavvy-it/
- ✓ https://blog.logrocket.com/react-conditional-rendering-9-methods/
- ✓ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Conditional_operator
- ✓ https://fr.javascript.info/ifelse
- ✓ https://www.florencebergaut.com/journal/

- ✓ https://www.youtube.com/watch?v=NWIAy4McC8g
- ✓ https://www.youtube.com/watch?v=zNznTnrWfAY
- ✓ https://www.youtube.com/watch?v=zOo69XPWmZc
- ✓ https://www.youtube.com/watch?v=_D6C3BWTGX4
- ✓ https://www.youtube.com/watch?v=H_dEaebG2u4

“ A LA PROCHAINE POUR LE COURS SUIVANT ”