Les bases

JSX, composants, props et state

- JSX
- Composants
- Props
- State

JSX

Qu'est-ce que JSX ?

JSX (JavaScript XML) est une extension syntaxique de JavaScript qui permet d'écrire du HTML directement dans du code JS. Il est compilé par Babel en appels `React.createElement()`.

Syntaxe de base

```
// JSX
const element = <h1 className="title">Bonjour React</h1>;

// Compilé en
const element = React.createElement('h1', { className: 'title' }, 'Bonjour React');
```

Règles importantes

- Un composant JSX doit retourner **un seul élément racine** (ou un `<Fragment>` / `<></>`)
- Utiliser `className` au lieu de `class`
- Les attributs sont en camelCase : `onClick`, `onChange`, `htmlFor`
- Les balises auto-fermantes doivent l'être : `<img />`, `<input />`
- Les expressions JS s'écrivent entre accolades : `{variable}`

Expressions dans JSX

```
const nom = "Hugo";
const age = 25;

return (
  <>
    <p>Bonjour {nom} !</p>
    <p>Tu as {age >= 18 ? "plus" : "moins"} de 18 ans.</p>
    <ul>
      {["pomme", "poire", "banane"].map((fruit, i) => (
        <li key={i}>{fruit}</li>
      ))}
```

```
    </ul>
  </>
);
```

Rendu conditionnel

```
// Opérateur ternaire
{isLoggedIn ? <Dashboard /> : <Login />}

// Court-circuit &&
{hasError && <ErrorMessage />}

// Variable
let content;
if (isLoggedIn) content = <Dashboard />;
else content = <Login />;
return content;
```

Composants

Les composants React

Un composant est une fonction (ou classe) qui retourne du JSX. Il est réutilisable, indépendant et encapsule sa logique.

Composant fonctionnel (recommandé)

```
// Déclaration avec function
function Bouton({ label, onClick }) {
  return <button onClick={onClick}>{label}</button>;
}

// Déclaration avec arrow function
const Bouton = ({ label, onClick }) => (
  <button onClick={onClick}>{label}</button>
);
```

Composant de classe (héritage, moins utilisé)

```
import { Component } from 'react';

class Compteur extends Component {
  state = { count: 0 };

  render() {
    return (
      <div>
        <p>{this.state.count}</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          +
        </button>
      </div>
    );
```

```
  }
}
```

Règles de nommage

- Un composant doit commencer par une **majuscule** : `MonComposant`
- Un nom en minuscule est interprété comme une balise HTML native

Composition

```
function Card({ title, children }) {
  return (
    <div className="card">
      <h2>{title}</h2>
      <div className="card-body">{children}</div>
    </div>
  );
}

// Utilisation
<Card title="Mon titre">
  <p>Contenu quelconque</p>
</Card>
```

Props

Les Props

Les props (propriétés) sont les paramètres passés à un composant depuis son parent. Elles sont **en lecture seule** — un composant ne doit jamais modifier ses propres props.

Passage de props

```
// Parent
<Profil nom="Hugo" age={25} admin={true} />

// Enfant
function Profil({ nom, age, admin }) {
  return (
    <div>
      <p>{nom}, {age} ans {admin && "□"}</p>
    </div>
  );
}
```

Valeurs par défaut

```
function Bouton({ label = "Cliquer", variant = "primary" }) {
  return <button className={variant}>{label}</button>;
}
```

Spread de props

```
const props = { nom: "Hugo", age: 25 };
<Profil {...props} />
```

La prop children

```
function Panel({ title, children }) {
  return (
    <section>
      <h3>{title}</h3>
      {children}
    </section>
  );
}

<Panel title="Info">
  <p>Texte dans le panel</p>
</Panel>
```

PropTypes (validation)

```
import PropTypes from 'prop-types';

Profil.propTypes = {
  nom: PropTypes.string.isRequired,
  age: PropTypes.number,
  admin: PropTypes.bool,
};

Profil.defaultProps = {
  age: 0,
  admin: false,
};
```

State

Le State (État)

Le state est la mémoire locale d'un composant. Contrairement aux props, il peut changer dans le temps. Chaque mise à jour du state déclenche un re-render du composant.

useState — déclaration

```
import { useState } from 'react';

function Compteur() {
  const [count, setCount] = useState(0); // valeur initiale = 0

  return (
    <div>
      <p>Compteur : {count}</p>
      <button onClick={() => setCount(count + 1)}>+</button>
      <button onClick={() => setCount(count - 1)}>-</button>
      <button onClick={() => setCount(0)}>Reset</button>
    </div>
  );
}
```

Mise à jour fonctionnelle

Quand le nouvel état dépend de l'ancien, toujours utiliser la forme fonctionnelle :

```
// ❌ Peut être stale dans certains cas
setCount(count + 1);

// ✅ Toujours sûr
setCount(prev => prev + 1);
```

State avec objets

```
const [user, setUser] = useState({ nom: "Hugo", age: 25 });

// □ Toujours créer un nouvel objet (immuabilité)
setUser(prev => ({ ...prev, age: 26 }));
```

Plusieurs états

```
const [nom, setNom] = useState("");
const [email, setEmail] = useState("");
const [loading, setLoading] = useState(false);
```

State vs Props

State	Props
Interne au composant	Vient du parent
Modifiable	En lecture seule
Déclenche un re-render	Déclenche un re-render