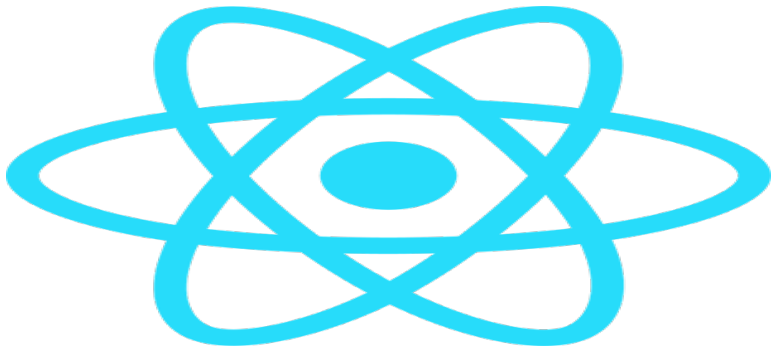
React : composants

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com



Plan

1. Création de composant
   - Classe
   - Fonction
2. Propriétés du composant : `props`
   - Attribut
   - Contenu
3. JSX
   - interpoler une variable primitive
   - interpoler un tableau
   - interpoler un objet
   - interpoler une expression
   - interpoler une fonction

Plan

React

Deux solutions pour la définition d'un composant **React**

- Fonctions
- Classes

React

Règles de nommage et bonnes pratiques

- Utiliser le **PascalCase** pour nommer les composants.
- Nommer le composant et le fichier le contenant de la même manière.
- Un composant $\Rightarrow$ un fichier.
- Choisir des noms de composants significatifs et descriptifs.
- Préférer des noms courts mais explicites, sans abréviations inutiles.
- Organiser les fichiers de composants dans des répertoires pour une structure claire et maintenable.
- Éviter les préfixes ou suffixes redondants dans les noms de composants.

React

Remarque

Les balises des composants doivent être refermées.

React

Composant de classe

- Une classe **ES6** qui étend `React.Component`
- Ayant une méthode `render` qui retourne du **JSX**

React

Dans `src/components`**, créons un fichier** `Hi.jsx`

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

class Hi extends Component {
    render() {
        return (
            <div className="Hi">
                <h2>Hi!</h2>
            </div>
        );
    }
}

export default Hi;
```

React

Dans `src/components`**, créons un fichier** `Hi.jsx`

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

class Hi extends Component {
    render() {
        return (
            <div className="Hi">
                <h2>Hi!</h2>
            </div>
        );
    }
}

export default Hi;
```


Explication

- La méthode `render` retourne un code **JSX** correspondant au rendu du composant.
- Les attributs utilisés dans des expressions **JSX** doivent être des propriétés **JavaScript** écrites en **camelCase**.
- Avant le `return` de la méthode `render`, on peut écrire un code **JavaScript** : `for`, `if`, `else`...

React

Question

Comment afficher ce composant ?

React

Question

Comment afficher ce composant ?

Réponse

En ajoutant ce nouveau composant dans `App.jsx` comme composant enfant.

React

Ajoutons ce nouveau composant dans `App.jsx` comme composant enfant

```
import logo from './logo.svg';
import './App.css';
import Hi from './components/Hi';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
        <Hi />
      </header>
    </div>
  );
}

export default App;
```

React

Composant de fonction

- Une fonction **JavaScript**
- Retournant un seul élément **JSX** ou

React

Le composant `Hello` déclaré comme fonction

```
import React from 'react';

function Hello() {
    return (
        <div className="Hello">
            <h1>Hello World!</h1>
        </div>
    );
}

export default Hello;
```

React

N'oublions pas de déclarer le composant dans `App.jsx`

```
import logo from './logo.svg';
import './App.css';
import Hi from './components/Hi';
import Hello from './components/Hello';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
        <Hi />
        <Hello />
      </header>
    </div>
  );
}

export default App;
```

React

Remarques

- L'utilisation des composants fonctionnels avec des hooks est la recommandation officielle pour le développement de nouveaux composants dans **React**.
- Les composants de classe restent supportés, mais les hooks offrent une manière plus moderne et efficace de gérer l'état et les effets dans les composants fonctionnels.

React

Remarques

- L'utilisation des composants fonctionnels avec des hooks est la recommandation officielle pour le développement de nouveaux composants dans **React**.
- Les composants de classe restent supportés, mais les hooks offrent une manière plus moderne et efficace de gérer l'état et les effets dans les composants fonctionnels.

We intend for Hooks to cover all existing use cases for classes, but **we will keep supporting class components for the foreseeable future.** At Facebook, we have tens of thousands of components written as classes, and we have absolutely no plans to rewrite them. Instead, we are starting to use Hooks in the new code side by side with classes.

Source : `https://react.dev/reference/react/Component`

React

Remarque

- Jusqu'à **React 17**, il était nécessaire d'importer explicitement **React** (`import React from 'react';`) dans chaque fichier **JSX**, même si on ne l'utilisait pas directement.
- En effet, Le code **JSX** était transformé en appels de fonction comme `React.createElement()`.
- Cependant, depuis **React 17**, cette règle a changé avec la nouvelle transformation **JSX**. Désormais, il n'est plus nécessaire d'importer explicitement **React** dans les fichiers utilisant **JSX**.

Et si on voulait passer des paramètres au composant `Hi` ou `Hello` comme un attribut de balise

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import Hi from './Hi';
import Hello from './Hello';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
        </p>
          <a
            className="App-link"
            href="https://reactjs.org"
            target="_blank"
            rel="noopener noreferrer"
          >
            Learn React
        </a>
        </header>
        <Hi nom="Wick" />
        <Hello nom="Wick" />
      </div>

    );
  }
}
export default App;
```

React

`props`

- Valeurs passées à un composant **React**
- Utilisés comme attribut **HTML**
- Valeurs en lecture seule

React

Dans `Hi.jsx`, pour récupérer la valeur du paramètre `nom`, on utilise `props`

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

class Hi extends Component {
    render() {
        return (
            <div className="Hi">
                <h2>Hi {this.props.nom}</h2>
            </div>
        );
    }
}

export default Hi;
```

React

Et dans `Hello.jsx`

```
import React from 'react';

function Hello(props) {
    return (
        <div className="Hello">
            <h2>Hello {props.nom}!</h2>
        </div>
    );
}

export default Hello;
```

React

Ou

```
import React from 'react';

function Hello({ nom }) {
    return (
        <div className="Hello">
            <h2>Hello {nom}!</h2>
        </div>
    );
}

export default Hello;
```

Et si on voulait passer des paramètres au composant `Hi` ou `Hello` comme un contenu de balise

```
import logo from './logo.svg';
import './App.css';
import Hi from './components/Hi';
import Hello from './components/Hello';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
        <Hi nom="Wick">Marseille</Hi>
        <Hello nom="Wick">Paris</Hello>
      </header>
    </div>
  );
}

export default App;
```

React

`children`

- Attribut de `props`
- Permettant de récupérer le contenu d'une balise
- Valeurs en lecture seule

React

Dans `Hi.jsx`, pour récupérer le contenu, on utilise `props.children`

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

export default class Hi extends Component {
    render() {
        return (
            <div className="Hi">
                <h2>Hi {this.props.nom} from {this.props.children}</h2>
            </div>
        );
    }
}
```

React

Et dans `Hello.jsx`

```
import React from 'react';

export default function Hello(props) {
    return (
        <div className="Hello">
            <h2>Hello {props.nom} from {props.children}!</h2>
        </div>
    );
}
```

React

Ou

```
import React from 'react';

export default function Hello({ children, nom }) {
    return (
        <div className="Hello">
            <h2>Hello {nom} from {children}!</h2>
        </div>
    );
}
```

React

Question 1

Et si le parent passait des données à l'enfant dans un attribut nommé `children`, comment les récupérer ?

React

Question 1

Et si le parent passait des données à l'enfant dans un attribut nommé `children`, comment les récupérer ?

Réponse

- Utiliser `props.children` pour accéder au contenu passé entre les balises ouvrantes et fermantes du composant parent,
- Si le composant est utilisé avec à la fois un attribut `children` (Par exemple : `<Enfant children="text">content</Enfant>`) et un contenu entre balises, seul le contenu entre les balises (`content`) sera récupéré via `props.children`, et l'attribut `children` sera ignoré.

React

Question 2

Comment peut-on transmettre plusieurs données comme contenu de balise ?

React

Question 2

Comment peut-on transmettre plusieurs données comme contenu de balise ?

Deux techniques pour transmettre plusieurs données de manière séparée

- Utiliser un tableau d'éléments
- Utiliser des fragments nommés

React

Première méthode : transmettre les données dans une balise `div` (ou autre)

```
<Hello nom="Wick">
  <div>Paris</div>
  <div>IdF</div>
  <div>75000</div>
</Hello>
```

React

Première méthode : transmettre les données dans une balise `div` (ou autre)

```
<Hello nom="Wick">
  <div>Paris</div>
  <div>IdF</div>
  <div>75000</div>
</Hello>
```

Utiliser un tableau pour la récupération

```
const [ville, dep, cp] = children
```

React

Première méthode : transmettre les données dans une balise `div` (ou autre)

```
<Hello nom="Wick">
  <div>Paris</div>
  <div>IdF</div>
  <div>75000</div>
</Hello>
```

Utiliser un tableau pour la récupération

```
const [ville, dep, cp] = children
```

Et pour afficher les données

```
<ul>
    <li>{ville}</li>
    <li>{dep}</li>
    <li>{cp}</li>
</ul>
```

React

Deuxième méthode : transmettre les données dans des `Fragment`s nommés

```
<Hello nom="Wick">
  <Fragment key="ville">Paris</Fragment>
  <Fragment key="dep">IdF</Fragment>
  <Fragment key="cp">75000</Fragment>
</Hello>
```

React

Deuxième méthode : transmettre les données dans des `Fragment`s nommés

```
<Hello nom="Wick">
  <Fragment key="ville">Paris</Fragment>
  <Fragment key="dep">IdF</Fragment>
  <Fragment key="cp">75000</Fragment>
</Hello>
```


Pour la récupération

```
const ville = children.find(child => child.key === 'ville');
const cp = children.find(child => child.key === 'cp');
const dep = children.find(child => child.key === 'dep');
```

React

Deuxième méthode : transmettre les données dans des `Fragment`s nommés

```
<Hello nom="Wick">
  <Fragment key="ville">Paris</Fragment>
  <Fragment key="dep">IdF</Fragment>
  <Fragment key="cp">75000</Fragment>
</Hello>
```

Pour la récupération

```
const ville = children.find(child => child.key === 'ville');
const cp = children.find(child => child.key === 'cp');
const dep = children.find(child => child.key === 'dep');
```

Ou

```
const [ville, dep, cp] = children
```

React

Si les données sont toutes définies dans un objet

```
const obj = {
    ville: 'Paris',
    dep: 'IdF',
    cp: 75000
}
```

React

Si les données sont toutes définies dans un objet

```
const obj = {
    ville: 'Paris',
    dep: 'IdF',
    cp: 75000
}
```

On peut transmettre l'objet

```
<Hello nom="Wick">
  {obj}
</Hello>
```

React

Si les données sont toutes définies dans un objet

```
const obj = {
    ville: 'Paris',
    dep: 'IdF',
    cp: 75000
}
```

On peut transmettre l'objet

```
<Hello nom="Wick">
  {obj}
</Hello>
```

Et pour la récupération

```
const {ville, dep, cp} = children
```

React

`props` vs `children` : lequel est plus performant ?

- Les deux méthodes sont équivalentes, car les `children` sont finalement passés au composant comme une `prop` spéciale nommée `children`.
- Pour optimiser les performances, il est important d'utiliser des techniques comme la mémorisation des composants (`useMemo`, `useCallback`) pour éviter des rendus inutiles.

React

JSX

- Valeur de retour
  - d'un composant fonctionnel **React** ou
  - de la méthode `render` d'un composant de classe
- Permettant d'écrire des balises **HTML** dans le code **JavaScript**
- Rendant le code plus lisible et plus facile à comprendre
- Utilisant l'interpolation pour récupérer les variables/attributs et fonctions/méthodes du composant

React

Remarques

- Toutes les balises auto-fermantes doivent se terminer par `/>` en **JSX**.
- Certains attributs **HTML** doivent être renommés en **camelCase**, comme les évènements
  - `onclick` devient `onClick`
  - pareil pour `onChange`, `onSubmit`...
- Certains attributs attributs sont nommés autrement pour éviter toute ambiguïté avec les mots-clés **JavaScript**
  - `className` au lieu de `class` et
  - `htmlFor` au lieu de `for`.

React

Interpolation

- s'effectue avec les `{ ... }`
- permet d'afficher le contenu d'une variable
- la variable à afficher peut être
  - primitive,
  - tableau,
  - objet ou
  - expression.

React

Pour interpoler une variable primitive (ou une constante) dans une fonction

```
export default function Hello({ children, nom }) {

    const message = "Hello"

    return (
        <div className="Hello">
            <h2>{message} {nom} from {children}!</h2>
        </div>
    );
}
```

React

Pour afficher un tableau

```
export default function Hello({ children, nom }) {

    const message = "Hello"
    const numbers = [2, 3, 8, 5]

    return (
        <div className="Hello">
            <h2>{message} {nom} from {children}!</h2>
            <ul>
                <li>{numbers[0]}</li>
                <li>{numbers['1']}</li>
                <li>{numbers["2"]}</li>
            </ul>
        </div>
    );
}
```

React

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

React

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Solution

utiliser les structures itératives (section suivante)

React

Pour interpoler un objet

```
export default function Hello({ children, nom }) {

    const message = "Hello"
    const numbers = [2, 3, 8, 5]
    const personne = { id: 100, nom: "Doe", prenom: "John" }

    return (
        <div className="Hello">
            <h2>{message} {nom} from {children}!</h2>
            <ul>
                <li>{numbers[0]}</li>
                <li>{numbers['1']}</li>
                <li>{numbers["2"]}</li>
            </ul>
            <ul>
                <li>Nom : {personne.nom}</li>
                <li>Prénom : {personne['prenom']}</li>
            </ul>
        </div>
    );
}
```

React

JSX ne permet pas l'affichage direct d'un objet

```
<p>{personne}</p>
```

React

On peut aussi utiliser une expression (ternaire par exemple) dans une interpolation

```
<p>{numbers[0] % 2 === 0 ? "pair" : 'impair'}</p>
{/* affiche pair */}
```

React

On peut aussi utiliser une expression (ternaire par exemple) dans une interpolation

```
<p>{numbers[0] % 2 === 0 ? "pair" : 'impair'}</p>
{/* affiche pair */}
```

Ou une fonction/méthode JavaScript prédéfinie

```
<p>{numbers.length}</p>
{/* affiche 4 */}

<p>{message.length}</p>
{/* affiche 5 */}

<p>{Math.sqrt(4)}</p>
{/* affiche 2 */}
```

React

On peut interpoler une expression mais pas une instruction : ceci génère une erreur

```
{ var x = 2 }
```

React

Pour interpoler une fonction

```
function direBonjour() {
    return 'Bonjour React';
}

export default function Hello({ children, nom }) {

    const message = "Hello"
    const numbers = [2, 3, 8, 5]
    const personne = { id: 100, nom: "Doe", prenom: "John" }

    return (
        <div className="Hello">
            <h2>{message} {nom} from {children}!</h2>
            {/* contenu précédent */}
            <p>{ direBonjour() }</p>
        </div>
    );
}
```

React

La fonction peut être définie dans le composant

```
export default function Hello({ children, nom }) {

    const message = "Hello"
    const numbers = [2, 3, 8, 5]
    const personne = { id: 100, nom: "Doe", prenom: "John" }

    function direBonjour() {
        return 'Bonjour React';
    }

    return (
        <div className="Hello">
            <h2>{message} {nom} from {children}!</h2>
            {/* contenu précédent */}
            <p>{ direBonjour() }</p>
        </div>
    );
}
```

React

Dans ce cas, la fonction aura accès aux variables du composant

```
export default function Hello({ children, nom }) {
    const message = "Hello"
    const numbers = [2, 3, 8, 5]
    const personne = { id: 100, nom: "Doe", prenom: "John" }
    const framework = 'React'

    function direBonjour() {
        return `Bonjour ${framework}`;
    }

    return (
        <div className="Hello">
            <h2>{message} {nom} from {children}!</h2>
            {/* contenu précédent */}
            <p>{ direBonjour() }</p>
        </div>
    );
}
```

React

Question

Où faudrait-il définir de définir une fonction ? à l'intérieur ou à l'extérieur d'un composant fonctionnel **React** ?

React

Réponse

- **Si la fonction dépend du composant (props, état, ou hooks)**, elle devrait être à l'intérieur.
  - Cela permet d'utiliser directement les **props** et de les manipuler.
  - Si la fonction est appelée fréquemment, notamment dans le `render`, , il peut être nécessaire d'utiliser `useCallback` pour la mémoriser et éviter les recalculs inutiles.
- **Si la fonction est indépendante**, il vaut mieux la placer à l'extérieur, surtout si elle est utilisée par plusieurs composants ou si elle n'a pas besoin d'être recréée à chaque rendu.
  - Cela permet d'éviter sa recréation à chaque rendu.
  - La fonction pourrait aussi être utilisée par plusieurs composants.

React

Définissons un attribut `lien` dans la classe

```
const lien = "http://elmouelhia.free.fr/"
```

React

On peut utiliser Attribute binding sur l'attribut `href`

```
<p>
    Voici un lien vers ma page,
    <a href={lien}> Cliquez pour visiter</a>
</p>
```

Définissons un attribut `lienTarget` de type objet dans la classe

```
const lienTraget = {
    href: "http://elmouelhia.free.fr/",
    target: "_blank"
}
```

Définissons un attribut `lienTarget` de type objet dans la classe

```
const lienTraget = {
    href: "http://elmouelhia.free.fr/",
    target: "_blank"
}
```

Pour lier l'objet `lienTraget` à la balise `a`

```
<p>
    Voici un lien vers ma page,
    <a {...lienTraget}> Cliquez pour visiter</a>
</p>
```

Définissons un attribut `lienTarget` de type objet dans la classe

```
const lienTraget = {
    href: "http://elmouelhia.free.fr/",
    target: "_blank"
}
```

Pour lier l'objet `lienTraget` à la balise `a`

```
<p>
    Voici un lien vers ma page,
    <a {...lienTraget}> Cliquez pour visiter</a>
</p>
```


Constat

En cliquant, le lien s'ouvrira dans un nouvel onglet.

React

Définissons une variable `style`

```
const style = {
    backgroundColor: 'blue',
    color: 'tomato',
};
```

React

Définissons une variable `style`

```
const style = {
    backgroundColor: 'blue',
    color: 'tomato',
};
```


Pour associer ce style à une balise

```
<p style={style}>
    Texte avec un style défini en JS.
</p>
```

React

Pour définir le style en CSS

```
<p style={{ backgroundColor: 'white', color: 'teal' }}>
    Texte avec un style CSS.
</p>
```

React

Pour définir le style en CSS

```
<p style={{ backgroundColor: 'white', color: 'teal' }}>
    Texte avec un style CSS.
</p>
```

Explication

- La première paire d'accolades { } est utilisée pour indiquer une expression **JavaScript** à l'intérieur du **JSX**.
- La deuxième paire { } est l'objet **JavaScript** représentant les styles.

React

Ou en allant chercher les variables du composant

```
<div style={{ backgroundColor: bgCouleur, color: couleur }}>
        Texte avec un style défini en JS.
</div>
```

React

Définissons les classes suivantes dans `src/components/Hello/Hello.css`

```
.rouge {
    color: red;
}

.bleu {
    color: blue;
}

.gras {
    font-weight: bold;
}
```

React

Définissons les classes suivantes dans `src/components/Hello/Hello.css`

```
.rouge {
    color: red;
}

.bleu {
    color: blue;
}

.gras {
    font-weight: bold;
}
```

Importons `Hello.css` **dans** `Hello.jsx`

```
import './Hello.css'
```

React

Pour associer la classe `rouge` à la balise `<p>`

```
<p className='rouge'>
    {message}
</p>
```

React

Pour associer la classe `rouge` à la balise `<p>`

```
<p className='rouge'>
    {message}
</p>
```

On peut également utiliser une expression ternaire

```
<p className={message.length % 2 === 0 ? 'rouge' : 'bleu'}>
    {message}
</p>
```

React

Pour associer la classe `rouge` à la balise `<p>`

```
<p className='rouge'>
    {message}
</p>
```

On peut également utiliser une expression ternaire

```
<p className={message.length % 2 === 0 ? 'rouge' : 'bleu'}>
    {message}
</p>
```


On peut également associer plusieurs classes à une balise

```
<p className={`gras ${message.length % 2 === 0 ? 'rouge' : 'bleu'}`}>
    {message}
</p>
```

React

Problématique

Et si on avait plusieurs classes à associer à une balise selon une ou plusieurs conditions

React

Problématique

Et si on avait plusieurs classes à associer à une balise selon une ou plusieurs conditions

Solution

Utiliser le package `classNames`.

React

Problématique

Et si on avait plusieurs classes à associer à une balise selon une ou plusieurs conditions

Solution

Utiliser le package `classNames`.

Pour l'installer

```
npm install classnames
```

React

Commençons par importer `classNames`

```
import classNames from 'classnames';
```

React

Commençons par importer `classNames`

```
import classNames from 'classnames';
```

Utilisons `classNames` pour lister les classes et leurs éventuelles conditions

```
const classes = classNames(
    'gras',
    { 'rouge': message.length % 2 === 0 },
    { 'bleu': message.length % 2 !== 0 }
);
```

React

Commençons par importer `classNames`

```
import classNames from 'classnames';
```

Utilisons `classNames` pour lister les classes et leurs éventuelles conditions

```
const classes = classNames(
    'gras',
    { 'rouge': message.length % 2 === 0 },
    { 'bleu': message.length % 2 !== 0 }
);
```

Associons ces classes à une balise HTML

```
<p className={classes}>
    {message}
</p>
```

React

Pour afficher les éléments du tableau `tab`, on utilise la fonction `map` pour transformer le tableau en liste HTML

```
<ul>
{
    numbers.map((number) =>
        <li>{number}</li>
    )
}
</ul>
```

React

Remarque

Le code a été compilé avec un warning affiché dans la console : `Each child in a list should have a unique "key" prop.`

React

Remarque

Le code a été compilé avec un warning affiché dans la console : `Each child in a list should have a unique "key" prop.`

Explication

React nous demande de spécifier l'identifiant : la clé de chaque élément (à traiter plus tard).

React

Et pour avoir l'indice de l'itération courante

```
<ul>
{
    numbers.map((number, index) =>
        <li>{index} : {number}</li>
    )
}
</ul>
```

React

Pour éviter le message d'erreur que React affiche, il faut spécifier ce qui pourrait être la clé

```
<ul>
{
    numbers.map((number, index) =>
        <li key={index}>{index} : {number}</li>
    )
}
</ul>
```

React

Considérons la liste suivante (à déclarer comme attribut de la classe)

```
const personnes = [
  { id: 100, nom: 'Wick', prenom: 'John' },
  { id: 101, nom: 'Abruzzi', prenom: 'John' },
  { id: 102, nom: 'Marley', prenom: 'Bob' },
  { id: 103, nom: 'Segal', prenom: 'Steven' }
]
```

React

Considérons la liste suivante (à déclarer comme attribut de la classe)

```
const personnes = [
  { id: 100, nom: 'Wick', prenom: 'John' },
  { id: 101, nom: 'Abruzzi', prenom: 'John' },
  { id: 102, nom: 'Marley', prenom: 'Bob' },
  { id: 103, nom: 'Segal', prenom: 'Steven' }
]
```


Exercice 1

Écrire un code **React** qui permet d'afficher dans une liste **HTML** les nom et prénom de chaque élément de la liste `personnes` (on n'affiche pas les clés).

React

Première solution

```
<ul>
{
    personnes.map((elt) =>
        <li key={elt.id}>{elt.prenom} {elt.nom}</li>
    )
}
</ul>
```

React

Deuxième solution (avec la déstructuration)

```
<ul>
{
    personnes.map(({ nom, prenom }, key) =>
        <li key={key}>{ prenom } { nom }</li>
    )
}
</ul>
```

React

Structure conditionnelle

- En **JSX**, on ne peut pas utiliser directement une instruction `if` comme dans un code **JavaScript** natif.
- Cependant, on peut intégrer des conditions en utilisant des expressions ternaires.
- On peut aussi utiliser des blocs `if` à l'extérieur du **JSX** pour organiser le rendu conditionnel avant de retourner le **JSX** final.

React

Exemple

```
export default function Hello({ children, nom }) {
    const message = "Hello"
    const numbers = [2, 3, 8, 5]
    const personne = { id: 100, nom: "Doe", prenom: "John" }
    const framework = 'React'
    const lien = "http://elmouelhia.free.fr/"
    const lienTraget = {
        href: "http://elmouelhia.free.fr/",
        target: "_blank"
    }
    const personnes = [
        { id: 100, nom: 'Wick', prenom: 'John' },
        { id: 101, nom: 'Abruzzi', prenom: 'John' },
        { id: 102, nom: 'Marley', prenom: 'Bob' },
        { id: 103, nom: 'Segal', prenom: 'Steven' }
    ]
    function direBonjour() {
        return `Bonjour ${framework}`;
    }
    if(numbers[0] % 2 == 0) {
        return <p>{numbers[0]} est pair</p>
    }
    // + le code précédent
}
```

React

Exercice : primeur-produit

- Créez deux composants : `Primeur` (parent) et `Produit` (enfant).
- Le composant `Primeur` définit un tableau `produits` (voir ci-dessous).
- Parcourez ce tableau afin de générer dynamiquement un composant `Produit` pour chaque élément.
- Chaque composant `Produit` reçoit en propriété l'objet correspondant et l'affiche.


```
let produits = [
    { nom: "banane", prix: 3, quantite: 10 },
    { nom: "fraise", prix: 10, quantite: 20 },
    { nom: "poivron", prix: 5, quantite: 10 }
]
```

React

Correction : composant `Primeur`

```
import Produit from './Produit'

export default function Primeur() {
    let produits = [
        { nom: "banane", prix: 3, quantite: 10 },
        { nom: "fraise", prix: 10, quantite: 20 },
        { nom: "poivron", prix: 5, quantite: 10 }
    ]
    return (
        <div>
            <h2>Primeur</h2>
            <ul>
                {
                    produits.map((elt, ind) =>
                        <Produit key={ind} elt={elt} />
                    )
                }
            </ul>
        </div>
    )
}
```

React

Correction : composant `Produit`

```
export default function Produit({ elt }) {

    return (
        <li>{elt.nom} {elt.quantite} {elt.prix}</li>
    )
}
```

React

Évènement (Event)

- Action appliquée par l'utilisateur ou simulée par le développeur sur un élément **HTML**.
- Évènement déclenché ⇒ Fonction, définie dans le composant, exécutée.

React

Définissons la fonctions `alertBonjour()` dans le composant.

```
function alertBonjour() {
    alert('Bonjour React');
}
```

React

Définissons la fonctions `alertBonjour()` dans le composant.

```
function alertBonjour() {
    alert('Bonjour React');
}
```

Pour exécuter la fonction `alertBonjour` lorsqu'on clique sur le bouton

```
<button onClick={alertBonjour}>
    Dire Bonjour
</button>
```

React

Définissons la fonctions `alertBonjour()` dans le composant.

```
function alertBonjour() {
    alert('Bonjour React');
}
```

Pour exécuter la fonction `alertBonjour` lorsqu'on clique sur le bouton

```
<button onClick={alertBonjour}>
    Dire Bonjour
</button>
```

Ajouter des parenthèses à la fonction déclenchera son exécution immédiate.

React

Remarque

On peut utiliser l'objet `event` pour récupérer des informations sur l'évènement **JavaScript**.

React

Considérons la zone de saisie suivante dans laquelle nous voudrions uniquement accepter des lettres en minuscule (`for` est remplacé par `htmlFor` en JSX)

```
<div>
    <label htmlFor="texte">
        Merci de bien saisir un texte :
    </label>
    <input type="text" id="texte" onInput={showValue} />
</div>
```

React

Considérons la zone de saisie suivante dans laquelle nous voudrions uniquement accepter des lettres en minuscule (`for` est remplacé par `htmlFor` en JSX)

```
<div>
    <label htmlFor="texte">
        Merci de bien saisir un texte :
    </label>
    <input type="text" id="texte" onInput={showValue} />
</div>
```

Pour connaître le caractère saisi, utilisons l'objet `event` dans `showValue()`

```
function showValue(event) {
    console.log(event.target.value);
    // affiche tous les caractères saisis

    console.log(event.nativeEvent.data);
    // affiche le dernier caractère saisi
}
```

React

Question

Et si la fonction envoyait des arguments (paramètres) ?.

React

Pour envoyer des paramètres, on peut utiliser une fonction fléchée

```
<div>
  <label htmlFor="texte">
    Merci de bien saisir un texte :
  </label>
  <input type="text" id="texte" onInput={() => showValue('value1')} />
</div>
```

React

Pour envoyer des paramètres, on peut utiliser une fonction fléchée

```
<div>
  <label htmlFor="texte">
    Merci de bien saisir un texte :
  </label>
  <input type="text" id="texte" onInput={() => showValue('value1')} />
</div>
```


Pour récupérer le paramètre

```
function showValue(param1) {
    console.log(param1);
}
```

React

Un cas avec un paramètre et `event`

```
<div>
    <label htmlFor="texte">
        Merci de bien saisir un texte :
    </label>
    <input type="text" id="texte" onInput={(event) => showValue('value1', event)} />
</div>
```

React

Un cas avec un paramètre et `event`

```
<div>
    <label htmlFor="texte">
        Merci de bien saisir un texte :
    </label>
    <input type="text" id="texte" onInput={(event) => showValue('value1', event)} />
</div>
```

Pour récupérer le paramètre et `event`

```
function showValue(param1, event) {
    console.log(param1);
    console.log(event.target.value);
}
```

React

Considérons le composant `Calculette.js` suivant (à créer)

```
export default function Calculette() {

    let valeur1 = 0
    let valeur2 = 0

    return (
        <div>
            <h2>Calculette</h2>
            Valeur 1 : <input type="number" name="valeur1" />
            Valeur 2 : <input type="number" name="valeur2" />
            <button>+</button>
        </div>
    )
}
```

React

Considérons le composant `Calculette.js` suivant (à créer)

```
export default function Calculette() {

    let valeur1 = 0
    let valeur2 = 0

    return (
        <div>
            <h2>Calculette</h2>
            Valeur 1 : <input type="number" name="valeur1" />
            Valeur 2 : <input type="number" name="valeur2" />
            <button>+</button>
        </div>
    )
}
```


Exercice 1

En cliquant sur le bouton `+`, nous voudrions que le résultat de la somme des deux inputs s'affiche dans une `alert`.

React

Correction : composant `Calculette`

```
export default function Calculette() {

    let valeur1 = 0
    let valeur2 = 0

    const sendValue = (e) => {
        if (e.target.name === 'valeur1') {
            valeur1 = e.target.value
        } else {
            valeur2 = e.target.value
        }
    }

    const somme = () => {
        alert(Number(valeur1) + Number(valeur2))
    }

    return (
        <div>
            <h2>Calculette</h2>
            Valeur 1 : <input type="number" name="valeur1" onInput={sendValue} />
            Valeur 2 : <input type="number" name="valeur2" onInput={sendValue} />
            <button onClick={somme}>+</button>
        </div>
    )
}
```

React

Exercice 2

- Ajouter 3 autres boutons : `-`, `*` et `/`.
- En cliquant sur un des 4 boutons, le résultat de l'opération arithmétique correspondante s'affiche dans une `alert`.
- Les 4 boutons appellent la même fonction.

React

Correction : composant `Calculette`

```
export default function Calculette() {

    let valeur1 = 0
    let valeur2 = 0

    const sendValue = (e) => {
        if (e.target.name === 'valeur1') {
            valeur1 = e.target.value
        } else {
            valeur2 = e.target.value
        }
    }

    const calculer = (op) => {
        alert(eval(`${valeur1} ${op} ${valeur2}`))
    }

    return (
        <div>
            <h2>Calculette</h2> {valeur1}
            Valeur 1 : <input type="number" name="valeur1" onInput={sendValue} />
            Valeur 2 : <input type="number" name="valeur2" onInput={sendValue} />
            <button onClick={() => calculer('+')}>+</button>
            <button onClick={() => calculer('-')}>-</button>
            <button onClick={() => calculer('*')}>*</button>
            <button onClick={() => calculer('/')}>/</button>
        </div>
    )
}
```

React.js

Bonnes pratiques

- Utilisez des parenthèses pour les retours multi-lignes dans le **JSX** afin de rendre la structure plus claire et d'éviter les erreurs de syntaxe.
- Le **JSX** doit toujours retourner un seul élément parent. Si vous avez besoin de retourner plusieurs éléments, utilisez un élément conteneur comme une `div` ou un `React.Fragment`.
- Les balises qui n'ont pas de contenu doivent être auto-fermées.
- Les noms des composants doivent suivre la convention **PamelCase** pour les distinguer des éléments **HTML** natifs.
- Les attributs doivent suivre la convention **camelCase** : `className` au lieu de `class` et `onClick` au lieu de `onclick`.
- Utilisez des méthodes de rendu conditionnel pour éviter d'encombrer le **JSX** avec des conditions trop complexes.

```
const maFonction = () => {
  if (loading) return <Loader />;
  if (error) return <Error />;
  return <Content />;
};

return <div>{maFonction()}</div>;
```

React.js

Bonnes pratiques

- Utilisez des accolades et des barres obliques `{/* comment */}` pour insérer des commentaires dans le **JSX**.
- Utilisez les méthodes de tableau (`map`, `filter`, `reduce`...) pour générer dynamiquement des listes d'éléments **JSX**.