

⚠ Tuto → version étape par étape.

### Introduction à JSX :

JSX => extension syntaxique de Javascript.

ex: const element = <h1>Hello, World! </h1>;

↓  
pas un string, ni du HTML !

O

ReactDOM.render(??, ??);

?? → ce qu'on veut render (ex: JSX)

?? → où on veut render (ex: cibler avec un

.getElementById())

On peut utiliser des expressions JS dans JSX en utilisant des accolades {}. Cela peut être une variable, une fonction, une opération arithmétique, ...

ex1: const name = 'Alexandre Domeux';

O

const element = <h1>Bonjour, {name}</h1>

ReactDOM.render(

element,

document.getElementById('root')

);

ex2: function formatName(user){

return user.firstName + ' ' + user.lastName;

}

const user1 = {firstName: 'Alexandre', lastName: 'Domeux'};

const element = <h1>Bonjour, {formatName(user1)}</h1>

On peut utiliser du JSX partout dans du JS. Dans des if, des boucles for, l'affecter à des variables, l'accepter en tant qu'argument, le renvoyer depuis des fonctions,...

On peut, pour les attributs JSX, soit utiliser des guillemets pour définir des littéraux chaînes de caractères, soit utiliser des {} pour utiliser une expression JS. Pas les deux !

ex 1: const element = <div tabIndex="0">...</div>;

ex 2: const element = <img src={user.avatarUrl}></img>;

JSX représente des objets :

React utilise en réalité des "éléments React" pour construire sa DOM.

ex simplifié: const element = {  
 type: 'h1',  
 props: { className: 'greeting',  
 children: 'Hello, world!' },  
};

On obtiens cet élément de deux manières équivalentes

ex 1: const element = (  
 <h1 className='greeting'>  
 Hello, world!  
 </h1>  
>);

ex 2: const element = React.createElement(  
 'h1',  
 { className: 'greeting' },  
 'Hello, world!');  
  
avec  
React.createElement()

## Le rendu des éléments

ReactDOM.render(, );

↳ Déjà vu.

Les éléments React sont immuables. Une fois créés on ne peut plus modifier ses enfants ou ses attributs.

Avec nos connaissances actuelles, la seule façon de mettre à jour l'UI est de recréer un nouvel élément et le passer à ReactDOM.render().

⇒ Refaire un render de l'ensemble.



React met à jour le strict nécessaire !

## Composants et props

Composants => permettent de découper l'UI en éléments indépendants et réutilisables.

Ils acceptent des entrées quelconques (props) et renvoient des éléments React.



Définir un composant :

1<sup>e</sup> version, avec une fonction JavaScript :

```
function Welcome (props) {  
    return <h1> Bonjour, {props.name} </h1>;  
}
```

2<sup>e</sup> version, avec une classe ES6 :

```
class Welcome extends React.Component {  
    render () {  
        return <h1> Bonjour, {this.props.name} </h1>;  
    }  
}
```

## Produire le rendu d'un composant

On va définir un élément comme une balise avec le nom de la fonction ou de la classe composant.

ex: const element = <Welcome name="Sara"/>  
            ^  
            nom de la fct ou classe  
    ⇒ props = { name : 'Sara' }

L'élément prendra la forme renvoyé par la fonction ou classe préalablement définie.

⚠ Pour React, une balise commençant par une minuscule est une balise HTML (ex: <div>).  
Celles commençant par une majuscule sont des composants. → Materia UI utilise des balises avec maj. Pose des problèmes ?? -> Non!

## Composition de composants

Les composants peuvent faire référence à d'autres composants dans leurs sorties.

```
ex: function App () {  
    return (  
        <div>  
            <Welcome name="Sara" />  
            <Welcome name="Gahal" />  
            <Welcome name="Edite" />  
        </div>  
    );  
}
```

⇒ On fera un render de <App/>  
ReactDOM.render(<App/>, ...),

Il me faut alors avoir peur de scinder des composants en composants plus petits.

↳ Imbriquer des petits composants.

Il est conseillé de nommer les props du parent de ceux du composant plutôt que de celui du contexte dans lequel il est.

⚠ "Tout composant React doit agir comme une fonction pure nus-à-nus de ses props."

⇒ Une fonction ne peut pas modifier ses props.

Ils sont en lecture seule.

### Etat et cycle de vie

Créer un composant en tant que classe plutôt que fonction permet d'utiliser des fonctionnalités supplémentaires telles que l'état local et les méthodes de cycle de vie.

Etat local ⇒ similaire aux props, mais il est privé et complètement contrôlé par le composant.

Ajouter un état local à une classe :

ex: class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

render() {

return (

<div> <h1> Il est {this.state.date.toLocaleTimeString()}</h1> </div>

</div>

), {}

①

②

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

)

&lt;p

① On utilise un constructeur de classe qui initialise this.state.

Un constructeur définit des variables, etc. lorsque un objet est créé avec une classe.

this.state => état local, ensemble de variables générées par le composant.

super() -> permet d'appeler le constructeur de la classe parente.

② this.state à la forme d'un objet.

On utilise this.state.date à la place de this.props.date.

↳ On utilise l'état local, plus les props.

L'état local nous permet d'avoir des variables dynamiques. Le composant peut les modifier.

Ajouter des méthodes de cycle de vie à une classe :

componentDidMount()

↳ Exécuté après que la sortie du composant a été injecté dans la DOM.

componentWillUnmount()

↳ Exécuté après que la sortie du composant ait été retiré de la DOM.

Dans le cas de l'horloge on va commencer à mettre à jour la date quand celle-ci est montée ("mounted", ajouté dans la DOM) et on arrêtera lorsqu'elle sera démontée ("unmounted").

ex: Dans la classe Clock:

```
componentDidMount () {  
    this.timerID = setInterval (() => this.tick(), 1000);  
}  
*  
componentWillUnmount () {  
    clearInterval (this.timerID);  
}  
}  
tick () {  
    this.setState ({ date: new Date () });  
}
```

① Dès que l'horloge est mise dans la DOM on effectue la méthode tick de la classe toutes les secondes.

② Dès qu'on enlève l'horloge de la DOM on arrête le setInterval.

③ La méthode tick() met à jour la valeur date dans l'état local

▷ On peut définir des valeurs de this par nous même ✖) en plus de celles existantes de base (this.props, this.state,...).

this.setState ({ }),

↳ Mets à jour l'état local du composant

✖ → Objet contenant l'ensemble des nouvelles valeurs de l'état local.

▷ Ne pas modifier l'état directement.

this.state.comment = "Bonjour";

↳ ne marche pas.

Il faut utiliser this.setState(). Le seul endroit où on peut utiliser this.state est le constructeur

Les mises à jour de l'état peuvent être asynchrone.

On ne doit donc pas se baser sur les valeurs pour calculer le prochain état :

ex: `this.setState({`

`counter: this.state.counter + this.props.increment,`

`});`

$\Rightarrow$  Pas bon !

Solution: Utiliser une fonction comme argument

pour `useState()`.

ex: `this.setState ((state, props) => {`

`counter: state.counter + props.increment`

`});`

$\Rightarrow$  Pas de problème avec l'asynchrone.



A) Il est impossible pour un composant d'accéder à l'état local d'un autre composant

On peut cependant faire passer les valeurs d'un état local à un composant enfant en les considérant comme `props`.

ex: `<FormatteDate date={this.state.date} />`

$\Rightarrow$  On définit la valeur 'date' de l'état local comme la valeur 'date' des props du composant enfant `FormatteDate`.

## Gérer les événements:

La gestion des événements pour les éléments React est très similaire à celle des éléments du DOM.

### Quelques différences:

- On nomme les événements React en camelCase plutôt qu'en minuscule.

- En JSX on passe une fonction comme gestionnaire d'événements plutôt qu'une chaîne de caractère.

ex: `<button onClick = "activatelausers()"> ... </button>`  
↓ devient

```
<button onClick = {activatelausers}> ... </button>
```

- Pour empêcher l'ouverture de lien (= recharge de la page, comportement par défaut) on renvoyait false avec la fonction de l'événement.

Avec React on appelle preventDefault.

ex: `<a href = "#" onClick = "console.log('Cliqué'); return false">`

Clique ici!

`</a>`

↓

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('Cliqué');
  }
}
```

```
, return <a href = "#" onClick = {handleClick}> ... </a>
```

On peut définir ça dans le constructeur pour éviter de la réécrire chaque fois.

this.handleClick = this.handleClick.bind(this);

e => événement synthétique React suivant les spécifications W3C. e représente l'événement React.

Quand on utilise React on évite d'ajouter des écouteurs d'événements à la DOM par après (avec `addEventListener`).  
qu'ils soient créés

On fournit l'écouteur dès le début lors du render.

Quand on définit un composant avec une classe on fait en sorte que le gestionnaire d'événements soit une méthode de la classe.

Dans JS, pour utiliser "this" correctement dans une méthode de classe il faut lier cette classe.

on rajoute  
====

<button onClick={this.handleClick.bind(this)}> ... </button>

\*

Si on ne lie pas la méthode avec `.bind()`, this sera undefined à l'intérieur de celle-ci !

Il y a moyen de lier une méthode sans utiliser `.bind()`.  
↳ Avec fonctions fléchées.

\* Passer des arguments :

<button onClick={this.handleClick.bind(this, id)}> ... </button>

↳ Version avec `.bind()`.

<button onClick={(e) => this.handleClick(id, e)}> ... </button>

↳ Version avec fonction fléchée.