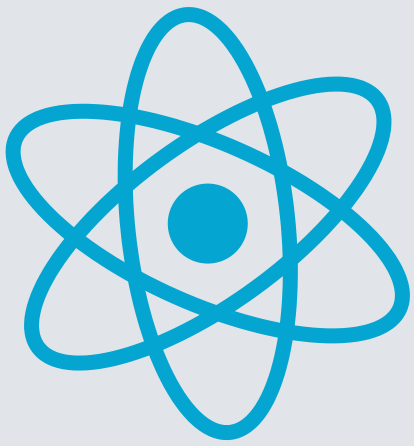
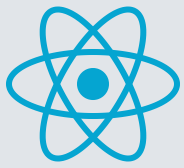


cours **React - J4**

IG2I 2021

une idée derrière la tech



React

1. Rappels
2. Syntaxes ES6+
3. Premiers pas avec React
4. **Les composants**
5. Formulaires et AJAX
6. Hooks & React Router

4. LES COMPOSANTS

- Imbrication
- Les "props"
- Function components
- DOM : événements & refs

IMBRICATION

MonSousComposant.js

```
import React from "react";

export default class MonSousComposant extends React.Component {
    render() {
        return <span>Je suis un sous composant !</span>;
    }
}
```

MonComposantPrincipal.js

```
import React from "react";
import MonSousComposant from "./MonSousComposant";

export default class MonComposantPrincipal extends React.Component {
    render() {
        return (
            <div>
                <strong>Je suis le composant principal !</strong>
                <MonSousComposant />
            </div>
        );
    }
}
```

Notes :

La force des composants React est qu'ils peuvent s'imbriquer à l'infini. Lorsque l'on crée un composant il devient possible de l'utiliser comme une nouvelle balise JSX dans un autre composant.

Cela présente plusieurs avantages comme la possibilité de créer des composants réutilisables mais également de scinder une interface graphique complexe en plusieurs "petits" composants.

4. LES COMPOSANTS

- Imbrication
- **Les "props"**
- Function components
- DOM : événements & refs

LES "PROPS"

- Paramètres transmis d'un composant parent à son enfant
- Equivalent des attributs HTML
- Disponible dans le composant enfant via "this.props"
- Peuvent être utilisées dans le render

Notes :

Les props sont des paramètres qui sont transmis d'un composant parent à son enfant au travers des "attributs" XML.

Ces paramètres passés par le parent sont alors disponibles dans le composant enfant via l'attribut "props".

Tout comme le "state", les props peuvent être utilisées dans la méthode render, et tout comme le "state", la modification d'une props entraîne un nouveau `render()`.

LES "PROPS"

HomePage.js

```
import Title from './Title';

export default class HomePage extends React.Component {
    render() {
        return (
            <div>
                <Title text="Car Wash" color="#ff0000" />
                Perfect place to launder my money ! 💰💰💰
            </div>
        );
    }
}
```

Title.js

```
export default class Title extends React.Component {
    render() {
        return (
            <h1 style={{ color: this.props.color }}>
                {this.props.text}
            </h1>
        );
    }
}
```

LES "PROPS" : CHILDREN

HomePage.js

```
export default class HomePage extends React.Component {
    render() {
        return (
            <div>
                <Title color="#ff0000">Car Wash</Title>
                Perfect place to launder my money ! 💰💰💰
            </div>
        );
    }
}
```

Title.js

```
export default class Title extends React.Component {
    // On récupère le contenu du titre via la prop children
    render() {
        return (
            <h1 style={{ color: this.props.color }}>
                {this.props.children}
            </h1>
        );
    }
}
```

Notes :

"children" est une props particulière dans la mesure ou elle ne se transmet pas via les attributs mais en tant qu'enfant du composant.

4. LES COMPOSANTS

- Imbrication
- Les "props"
- **Function components**
- DOM : événements & refs

FUNCTION COMPONENTS

Se passer de classe pour des composants simples

```
class Link extends React.Component {
    render() {
        return (
            <a href={this.props.url}>
                {this.props.label}
            </a>
        );
    }
}
```

```
function Link( props ) {
    return (
        <a href={props.url}>
            {props.label}
        </a>
    );
}
```

```
function Link({ url, label }) {
    return (
        <a href={url}>
            {label}
        </a>
    );
}
```

```
const Link = ({ url, label }) => (
    <a href={url}>
        {label}
    </a>
);
```

Notes :

Les **function components** permettent de définir des composants sous forme de fonction plutôt que de classe. Les props sont passées directement en paramètre de la fonction, et le JSX généré par le composant est celui retourné par la fonction (c'est plus ou moins l'équivalent de la méthode `render` d'un class component).

A la base, les **function components** étaient surtout utilisés pour des composants de présentation relativement simples. Lors de la sortie de cette fonctionnalité dans React 0.14, ils étaient nommés "Stateless function components" (cf. https://reactjs.org/blog/2015/10/07/react-v0.14.html#stateless-function-components)

Depuis React 16.8 (sorti en février 2019) les function components ont gagné de nouvelles capacités et notamment la possibilité de bénéficier d'un state local, grâce aux hooks (https://reactjs.org/docs/hooks-intro.html). Le mot "stateless" a donc disparu du nom.

Si vous souhaitez en savoir plus sur les hooks, les bonnes pratiques associées, et les erreurs à ne pas faire, rendez-vous dans la formation React avancé ;) !

4. LES COMPOSANTS

- Imbrication
- Les "props"
- Function components
- **DOM : événements & refs**

API DOM : LES ÉVÉNEMENTS


```
export default class MonComposant extends React.Component {
    state = { clicked: false };

    render() {
        return (
            <button onClick={event => this.handleButtonClick(event)}>
                {this.state.clicked ? 'Cliqué !' : 'Cliquez moi !'}
            </button>
        );
    }

    handleButtonClick( event ) {
        this.setState( { clicked: true } );
    }
}
```

```
export default class MonComposant extends React.Component {
    state = { clicked: false };

    constructor(props) {
        super(props);
        this.handleButtonClick = this.handleButtonClick.bind(this);
    }

    render() {
        return (
            <button onClick={this.handleButtonClick}>
                {this.state.clicked ? 'Cliqué !' : 'Cliquez moi !'}
            </button>
        );
    }

    handleButtonClick( event ) {
        this.setState( { clicked: true } );
    }
}
```

Notes :

Les écouteurs d'événement sur les éléments du DOM peuvent ajouté directement dans le JSX.

Pour cela il suffit d'ajouter l'attribut onNomEvenement (ex: onClick, onMouserOver, onSubmit, etc...) puis de lui attribuer une fonction à appeler lorsque cet événement survient.

NB: Lorsque l'on assigne une méthode à un événement il ne faut pas oublier de la binder dans le constructeur afin de s'assurer que la valeur du "this" ne soit pas perdue ou bien de faire une arrow function (mais dans ce cas la fonction est "recalculée" à chaque render...)

API DOM : REFS

L'attribut "ref" permet de "sélectionner" un élément

```
export default class MyForm extends React.Component {
    input = React.createRef();

    render() {
        return (
            <form>
                <input type="text" ref={this.input} >
                <button onClick={ () => this.input.current.focus() }>
                    focus sur le champ
                </button>
            </form>
        )
    }
}
```

```
export default class MyForm extends React.Component {
    input = null;

    render() {
        return (
            <form>
                <input type="text" ref={el => this.input = el} >
                <button onClick={ () => this.input.focus() }>
                    focus sur le champ
                </button>
            </form>
        )
    }
}
```

Notes :

Pour récupérer une référence vers un élément de la page HTML, il suffit, dans la méthode render, d'attribuer à l'élément que l'on souhaite récupérer une "ref". Cette technique permet d'accéder à l'API JavaScript d'éléments HTML spécifiques comme les canvas, la balise audio, video, etc...

On peut soit utiliser React.createRef() soit une fonction de callback.

NB: il est également possible d'assigner à l'attribut "ref" une chaîne de caractères, l'élément DOM étant alors accessible via `this.refs.maChaine` mais cette notation est dépréciée cf. https://github.com/facebook/react/issues/6250