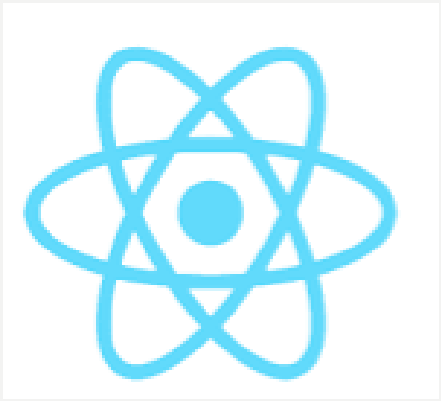


COMPOSANT PROPS

PREMIER APPROCHE COMPOSANT

- **Conceptuellement, les composants sont comme des fonctions JavaScript**
- le nom d'un composant commence pas une majuscule.
- Exemple composant sans état : fonction dont le résultat est le contenu du composant.

styles

JS index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import  First  from  './components/first.js';

ReactDOM.render(
    <First  />,
    document.getElementById('root')
);
```

```
//  /components/first.js
import  React  from  'react';

const  First = ()  =>
    <article>
        Loading  ...
    </article>

export  default  First;
```

COMPOSANT DANS UN COMPOSANT

- ***dans cet exemple : Garage.js***

1. On appelle ReactDOM.render() avec l'élément <h1> et <Car />.

2. React appelle le composant Car .

3. Notre composant Car retourne un élément <h2>car</h2>

```
import React from 'react';

class Car extends React.Component {
    render() {
        return <h2>car</h2>;
    }
}

class Garage extends React.Component {
    render() {
        return (
            <div>
                <h1>Garage</h1>
                <Car />
            </div>
        );
    }
}

export default Garage;
```

PROPS AVEC ES6

- **extends React.component**
- **props devient this.props**

dans cet exemple :

1.On appelle ReactDOM.render() avec l'élément <Hello name='classe es6' />.

2.React appelle le composant Hello avec comme props {name: 'classe es6'}.

3.Notre composant Hello retourne un élément <h1>Hello classe es6</h1> pour résultat.

4.React DOM met à jour efficacement le DOM pour correspondre à <h1>Hello classe es6</h1>.

```
import React from 'react'

class Hello extends React.Component {
    render() {
        return <h1>Hello {this.props.name}</h1>
    }
}

export default Hello
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import  Hello  from  './components/exemple1.js';

ReactDOM.render(
    <Hello name="classe es6" />,
 document.getElementById('root')
);
```

```
import React, { Component } from 'react';
import { render } from 'react-dom';


class FirstComponent extends Component {
  render() {
```

PROPS AVEC FONCTION FLÉCHÉE

Un composant peut recevoir des props .

Ce sont des propriétés passées par son parent afin de spécifier certaines valeurs que le composant ne peut pas connaître par lui-même;

```
//    /components/second.js

import  React  from  'react';

const  Personne  = (props)  =>
    <div  className="personne">Je suis  :
        <dl>
            <dt>nom</dt><dd>  {props.name}  </dd>
            <dt>age</dt><dd>  {props.age}  </dd>
        </dl>
    </div>

export default Personne;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import  Personne  from  './components/second.js';

ReactDOM.render(
    <Personne  name="Iano Randrianasoa"  age="50"  />,
    document.getElementById('root')
);
```

UTILISATION

- Les composants sont utilisés de la même manière que :

```
const ParentComponent = function (props) {
    const Text = "exemple";
    return (
        <FirstComponent content={Text} />
        <SecondComponent content={Text} />
        <ThirdComponent content={Text} />
    );
}
```

- les accessoires de réaction sont immuables une fois qu'ils ont été passés, ce qui signifie qu'ils ne peuvent pas être modifiés à partir d'un composant. Si le parent d'un composant modifie la valeur d'un accessoire, React gère le remplacement des anciens accessoires par le nouveau, le composant se répète en utilisant les nouvelles valeurs.

A RETENIR

- **Chaque composant React doit obligatoirement avoir la méthode render(). Il renvoie un seul élément React qui est la représentation du composant DOM natif. Si plusieurs éléments HTML doivent être rendus, ils doivent être regroupés dans une balise englobante telle que <form>, <group>, etc.**

- **Props est l'abréviation de Properties dans React. Ce sont des composants en lecture seule qui doivent être maintenus purs, c'est-à-dire immuables. Ils sont toujours transmis du composant parent aux composants enfants tout au long de l'application. Un composant enfant ne peut jamais renvoyer un Props au composant parent. Cela aide à maintenir le flux de données unidirectionnel**

PROPS DU COMPOSANT DANS UN COMPOSANT

- Exemple I

```
class Car extends React.Component {
    render() {
        return <h2>{this.props.name}!</h2>;
    }
}
class Garage extends React.Component {
    render() {
        const carname = "Ford";
        return (
            <div>
                <h1>Garage</h1>
                <Car name={carname} />
            </div>
        );
    }
}
export default Garage;
```

```
class Car extends React.Component {
    render() {
      return <h2>{this.props.brand.model}!</h2>;
    }
}
class Garage extends React.Component {
    render() {
        const carinfo = {name: "Ford", model: "Mustang"};
        return (
            <div>
                <h1>Garage</h1>
                <Car brand={carinfo} />
            </div>
        );
    }
}
```

PROPS DU COMPOSANT DANS UN COMPOSANT

- Exemple 2

```
class Car extends React.Component {
    render() {
        return <h2>{this.props.name}!</h2>;
    }
}
class Garage extends React.Component {
    render() {
        const carname = "Ford";
        return (
            <div>
                <h1>Garage</h1>
                <Car name={carname} />
            </div>
        );
    }
}
export default Garage;
```

```
class Car extends React.Component {
    render() {
      return <h2>{this.props.brand.model}!</h2>;
    }
}
class Garage extends React.Component {
    render() {
        const carinfo = {name: "Ford", model: "Mustang"};
        return (
            <div>
                <h1>Garage</h1>
                <Car brand={carinfo} />
            </div>
        );
    }
}
```

EXTRACTION DES COMPOSANTS

- Exemple : décrivons un commentaire sur un réseau social en ligne

```jsx
function Commentaire(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
          src={props.author.avatarUrl}
          alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

```jsx
function Avatar(props) {
  return (
    <img className="Avatar"
      src={props.user.avatarUrl}
      alt={props.user.name}
    />
  );
}
```

```jsx
function Commentaire(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
```

EXTRACTION DES COMPOSANTS

- Ensuite, nous allons extraire un composant UserInfo

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} />
      <div className="UserInfo-name">
        {props.user.name}
      </div>
    </div>
  );
}
```

```
function Commentaire(props) {
  return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

MANIERES DE CREATION

- Il existe trois manières de créer un composant React

Composants fonctionnels ("sans état")

```
const FirstComponent = props => (
    <div>{props.content}</div>
);
```

React.createClass ()

```
const SecondComponent = React.createClass({
    render: function () {
        return (
            <div>{this.props.content}</div>
        );
    }
});
```

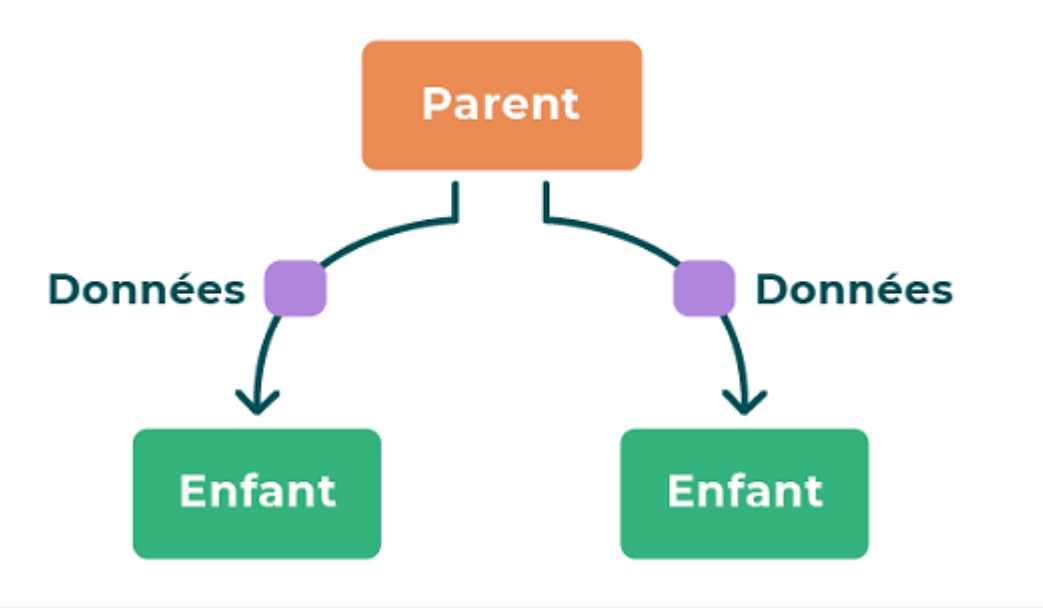
Classes ES

```
class ThirdComponent extends React.Component {
    render() {
        return (
            <div>{this.props.content}</div>
        );
    }
}
```

DONNEES DES PARENTS VERS ENFANTS

Lorsque nous créons un élément à l'aide de react, ces propriétés seront disponibles en tant qu'attribut de cet élément appelé .props. Nous pouvons ensuite utiliser des accessoires pour composer des enfants pour l'élément.
Tout ce que nous transmettons à l'élément sera ajouté en tant que nœud enfant. Il peut s'agir d'une simple chaîne.

Les composants parents partagent leurs données avec leurs enfants




```
class Greeting extends React.Component {
  static defaultProps = {
    name: 'bel inconnu'
  }

  render() {
    return (
      <div>Bonjour, {this.props.name}</div>
    )
  }
}
```

DONNEES DES PARENTS VERS ENFANTS

•*Une prop est toujours **passée par un composant parent à son enfant** : c'est le seul moyen normal de transmission.*

•*Une prop est considérée **en lecture seule** dans le composant qui la reçoit.*

EXEMPLE

```
<Banner>
    <img src={logo} alt='La maison jungle' />
    <h1 className='lmj-title'>La maison jungle</h1>
</Banner>
```

img et h1 sont les nœuds enfants dans le DOM de Banner
Et on peut **accéder à ces nœuds enfants de Banner dans ses paramètres**, un peu de la même manière qu'on récupérerait des props

```
function Banner({ children }) {
    return <div className='lmj-banner'>{children}</div>
}
```

Cette manière d'utiliser children est particulièrement **utile lorsqu'un composant ne connaît pas ses enfants à l'avance**,

SRC/COMPONENTS/CARESCALE.JS

CareScale est l'enfant, et ShoppingList est le parent

```
function CareScale({ scaleValue, careType }) {
    const range = [1, 2, 3]
    const scaleType = careType === 'light' ? '🌞' : '💧'

    return (
        <div>
            {range.map((rangeElem) =>
                scaleValue >= rangeElem ? (
                    <span key={rangeElem.toString()}>{scaleType}</span>
                ) : null
            )}
        </div>
    )
}
export default CareScale
```

SRC/COMPONENTS/SHOPPINGLIST.JS

```
import { plantList } from '../datas/plantList'
import Care from './Care'
import CareScale from './CareScale'
import '../styles/ShoppingList.css'

function ShoppingList() {
        <div>
            <ul>
                function ShoppingList() {
                    <li key={plant.id} className='lmj-plant-item'>
                        {plant.isBestSale && <span>🔥</span>}
                        {plant.name}
                        <Care careType='water' value={plant.water} />
                        <Care careType='light' value={plant.light} />
                        <CareScale careType='water' scaleValue={plant.water} />
                        <CareScale careType='light' scaleValue={plant.light} />
                    </li>
                }
            </ul>
        </div>
}
export default ShoppingList
```

SRC/COMPONENTS/CARE.JS

```
function Care({ value, careType }) {
    const scale = [1, 2, 3]
    const scaleType = careType === 'light' ? '☀️' : '💧'

    return (
        <div>
            {scale.map((elem) =>
                value >= elem ? <span key={elem.toString()}>{scaleType}</span> : null
            )}
        </div>
    )
}
export default Care
```

EXERCICE

```
// fichier plantList.js
import im from '../assets/monstera.jpg'

export const planteListes = [
    {
        id: 'aaa',
        nom: 'Dahlia',
        categorie: 'classique',
        lumiere: 2,
        eau: 3,
        photo: im
    },
    {
        id: 'aab',
        nom: 'Rose',
        categorie: 'extérieure',
        lumiere: 3,
        eau: 1,
        photo: im
    },
    {
        id: 'aac',
        nom: 'Vanille',
        categorie: 'classique',
        lumiere: 1,
        eau: 2,
        photo: im
    }
```

PLANTES

- [ ] classique
- [ ] extérieure
- [ ] plante grasse

Dahlia

Rose

Vanille

Inc

Tournesol

Prix

- niveau 1 : 8000 Ariary
- niveau 2 : 10000 Ariary
- niveau 3 : 15000 Ariary