Accéder aux accessoires d'un composant

Chaque composant a quelque chose appelé `props`.

Un `props` de composant est un objet. Il contient des informations sur ce composant.

Vous avez déjà vu cela, mais vous ne vous en êtes peut-être pas rendu compte ! Jetons un coup d'œil à la balise de bouton HTML. Il existe plusieurs informations que nous pouvons transmettre à la balise bouton, telles que la valeur `type` du bouton.

```
<button type="submit" value="Submit"> Submit </button>
```

Dans cet exemple, nous avons transmis deux informations à la balise bouton, un `type` et une `value`. Selon l' attribut `type` que nous attribuons à l'élément `<button>`, celui-ci traitera le formulaire différemment. De la même manière, nous pouvons transmettre des informations à nos propres composants pour préciser leur comportement !

Les accessoires ont le même objectif pour les composants que les arguments pour les fonctions.

Pour accéder à l'objet d'un composant `props`, vous pouvez référencer l' objet `props` et la notation par points pour ses propriétés. Voici un exemple :

```
props.name
```

Cela récupérerait la propriété `name` de l'objet `props`.

Passer props à un composant

Pour profiter de `props`, nous devons *transmettre des informations* à un composant React.

Comment passe-t-on `props`? En donnant un *attribut* au composant :

```
name="Jamel" />
```

Disons que vous souhaitez transmettre à un composant le message `"We're great!"`. Voici comment procéder :

```
message="We're great!" />
```

Comme vous pouvez le constater, pour transmettre des informations à un composant, vous avez besoin d'un *nom* pour les informations que vous souhaitez transmettre.

Dans l'exemple ci-dessus, nous avons utilisé le nom `message`. Vous pouvez utiliser le nom de votre choix.

Si vous souhaitez transmettre des informations qui ne sont pas une chaîne, placez ces informations entre accolades. Voici comment transmettre un tableau :

```
myInfo={["Astronaut", "Narek", "43"]} />
```

Dans cet exemple suivant, nous transmettons plusieurs informations à `<Greeting />`. Les valeurs qui ne sont pas des chaînes sont placées entre accolades :

```
  name="The Queen Mary"city="Long Beach, California"age={56}haunted={true} />
```

Rendre les accessoires d'un composant

Les props nous permettent de personnaliser le composant en lui transmettant des informations.

Nous avons appris comment *transmettre* des informations à l'objet d'un composant `props`. Vous souhaiterez souvent qu'un composant *affiche* les informations que vous transmettez.

Pour vous assurer qu'un composant fonction peut utiliser l'objet `props`, définissez votre composant fonction avec `props` comme paramètre :

```
function Button(props) {
  return <button>{props.displayText}</button>;
}
```

Dans l'exemple, `props`est accepté comme paramètre et les valeurs de l'objet sont accessibles avec le modèle d'accesseurs de notation par points (`object.propertyName`).

Alternativement, puisqu'il `props`s'agit d'un objet, vous pouvez également utiliser une syntaxe de déstructuration comme celle-ci :

```
function Button({displayText}) {
  return <button>{displayText}</button>;
}
```

exemple :

Explorez le code ci-dessous.

```
import React from 'react';

function Product() {
  return (
    <div>
      <h1></h1>
      <h2></h2>
      <h3></h3>
    </div>
  );
}

export default Product;
```

Product.js contient les grandes lignes d'un composant chargé d'afficher les produits d'un site e-commerce.

App.js `App` contient le composant de niveau supérieur , qui appelle le composant `Product` avec trois informations : `name`, `price` et `rating`.

```
import React from 'react';
import Product from './Product';

function App() {
  return (
<div>
        <Product name="Apple Watch" price = {399} rating = "4.5/5.0" />;
</div>
)
}

export default App;
```

Cependant, le composant `Product` n'accepte pas les accessoires.

Dans **Product.js** , modifiez le composant `Product` pour qu'il accepte les `props` dans la définition de la fonction.

```
function Product(props) {
  //instruction
}
```

Maintenant, à partir de `props` nous pouvons injecter les valeurs appropriés données par l'attribut.

```
import React from 'react';

function Product(props) {
  return (
    <div>
      <h1>{props.name}</h1>
      <h2>{props.price}</h2>
      <h3>{props.rating}</h3>
    </div>
  );
}

export default Product;
```

Passer les accessoires du composant au composant

Vous avez appris à passer `prop` à un composant :

```
<Greeting firstName="Esmerelda" />
```

Vous avez également appris à accéder et afficher un `prop` transmis:

```
return <h1>{props.firstName}</h1>;
```

L'utilisation la plus courante de `props` consiste à transmettre des informations à un composant à partir *d'un autre composant* .

Les accessoires dans React se déplacent dans un sens unique, de haut en bas, du parent vers l'enfant.

Explorons un peu plus loin la relation parent-enfant de `props`.

```
function App() {
    return <Product name="Apple Watch" price = {399} rating = "4.5/5.0" />;
```

```
}
```

Dans cet exemple, `App` est le parent et `Product` est l'enfant. `App` passe trois accessoires à `Product`(`name`, `price`, et `rating`), qui peuvent ensuite être lus à l'intérieur du composant enfant.

Les accessoires transmis sont immuables, ce qui signifie qu'ils ne peuvent pas être modifiés. Si un composant souhaite de nouvelles valeurs pour ses accessoires, il doit s'appuyer sur le composant parent pour lui en transmettre de nouvelles.

Rendre une interface utilisateur différente en fonction des accessoires

Vous pouvez faire plus avec les accessoires que simplement les afficher. Vous pouvez également utiliser des accessoires pour prendre des décisions.

```
function LoginMsg(props) {
  if (props.password === 'a-tough-password') {
    return <h2>Sign In Successful.</h2>
  } else {
    return <h2>Sign In Failed..</h2>
  }
}
```

Dans cet exemple, nous utilisons les accessoires transmis pour prendre une décision plutôt que d'afficher la valeur à l'écran.

Si le `password` reçu est égal à `'a-tough-password'`, le message résultant `<h2> </h2>`sera différent !

Dans les deux cas , le montant transmis `password` n'est pas affiché ! L'accessoire est utilisé pour *décider* de ce qui sera affiché. Il s'agit d'une technique courante.

exemple :

Gretting.js

```
import React from 'react';

function Greeting(props) {
  if (props.signedIn == false) {
    return <h1>Please login.</h1>;
  } else {
    return (
      <>
        <h1>Welcome back, {props.name}!</h1>
        <article>
          Latest Movie: A Computer Bug's Life
        </article>
      </>
    )
  }
}

export default Greeting;
```

Regardez dans la définition du composant de fonction. Vous pouvez voir que dans `Greeting` deux accessoires sont désormais attendus : `name` et `signedIn`

Notez que `props.signedIn` ne se trouve *pas* à l'intérieur d'une instruction `return`. Cela signifie que `Greeting` n'affichera jamais la valeur de `signedIn`. Mais `Greeting` *utilisera* cette valeur pour décider quel message afficher.

Regardez `Greeting` jusqu'à ce que vous ayez l'impression de comprendre comment cela fonctionne.

Dans App.js, donnons au composant `Greeting` un attribut supplémentaire `signedIn` avec la valeur `true`.

```
import React from 'react';
import Greeting from './Greeting';

function App() {
  return (
    <div>
      <h1>
        MovieFlix
      </h1>
      <Greeting name="Alison" signedIn={true}/>
    </div>
  );
}

export default App;
```

Voici le rendu du DOM :

MovieFlix

Welcome back, Alison!

Latest Movie: A Computer Bug's Life