Utiliser JSX multiligne dans un composant

Dans cette leçon, vous apprendrez quelques façons courantes dont les composants JSX et React fonctionnent ensemble. Vous deviendrez plus à l'aise avec les composants JSX et React tout en apprenant de nouvelles astuces.

Jetez un œil à ce code JSX :

```
function QuoteMaker() {
    return (
      <blockquote>
        <p>
          The world is full of objects, more or less interesting; I do not wish to
add any more.
        </p>
        <cite>
          <a target="_blank"
            href="https://en.wikipedia.org/wiki/Douglas_Huebler">
            Douglas Huebler
          </a>
        </cite>
      </blockquote>
    );
};
```

Comment pourriez-vous faire en sorte qu'un composant React renvoie ce code JSX?

L'élément clé à remarquer dans `QuoteMaker` sont les parenthèses dans l'instruction `return`, aux lignes 4 et 16. Jusqu'à présent, les instructions `return` de vos composants de fonction ressemblaient à ceci, sans aucune parenthèse :

```
return <h1>Hello world</h1>;
```

Cependant, une expression JSX multiligne doit toujours être placée entre parenthèses ! C'est pourquoi l'instruction return de `QuoteMaker` est entourée de parenthèses.

Utiliser un attribut de variable dans un composant

Jetez un oeil à cet objet JavaScript nommé `redPanda`:

```
const redPanda = {
  src:
'https://upload.wikimedia.org/wikipedia/commons/b/b2/Endangered_Red_Panda.jpg',
  alt: 'Red Panda',
  width:  '200px'
};
```

Comment rendre un composant React avec une image `redPanda` et ses propriétés ?

```
function RedPanda(){
    return (
      <div>
        <h1>Cute Red Panda</h1>
        <img
          src={redPanda.src}
          alt={redPanda.alt}
          width={redPanda.width} />
      </div>
    );
}
```

Mettre la logique dans un composant de fonction

Un composant fonction doit avoir une instruction `return`. Cependant, ce n'est pas *tout* ce qu'il peut avoir.

Vous pouvez également effectuer des calculs simples qui doivent être effectués avant de renvoyer votre élément JSX dans le composant de fonction.

Voici un exemple de calculs pouvant être effectués à l'intérieur d'un composant de fonction :

```
function RandomNumber() {
  //First, some logic that must happen before returning
  const n = Math.floor(Math.random() * 10 + 1);
  //Next, a return statement using that logic:
  return <h1>{n}</h1>
}
```

Attention à cette erreur courante :

```
function RandomNumber() {
  return (
    const n = Math.floor(Math.random() * 10 + 1);
    <h1>{n}</h1>
  )
}
```

Dans l'exemple ci-dessus, la ligne avec la déclaration `const n` provoquera une erreur de syntaxe, car elle devrait précéder le `return` et non être dans le return.

Utiliser un conditionnel dans un composant de fonction

Comment pouvez-vous utiliser une instruction *conditionnelle* à l'intérieur d'un composant fonction ?

Sélectionnez **TodaysPlan.js** pour voir une façon de procéder.

```
function TodaysPlan(){
  let task;
  let apocalypse = true;
  if(!apocalypse){
    task = 'learn React.js';
  }else{
    task = 'run around';
  }
  return <h1> Today I am going to {task}...</h1>

  }
```

Notez que l' instruction `if` est située *à l'intérieur* du composant fonction, mais *avant* l'instruction `return`.

Écouteur d'événements et gestionnaires d'événements dans un composant

Vos composants de fonction peuvent inclure des gestionnaires d'événements. Avec les gestionnaires d'événements, nous pouvons exécuter du code en réponse aux interactions avec l'interface, comme un clic.

```
function MyComponent(){
  function handleHover() {
    alert('Stop it.  Stop hovering.');
  }
  return <div onHover={handleHover}></div>;
}
```

Dans l'exemple ci-dessus, le gestionnaire d'événements est `handleHover()`. Il est transmis comme accessoire à l'élément JSX `<div>`. Nous discuterons davantage des accessoires dans une leçon ultérieure, mais pour l'instant, comprenez que les accessoires sont des informations transmises à une balise JSX.

La logique de ce qui devrait se passer lorsque le survol est placé sur la souris est contenue à l'intérieur de la fonction `handleHover()`. La fonction est ensuite transmise à l' élément `<div>`.

Les fonctions de gestionnaire d'événements sont définies à l'intérieur du composant de fonction et, par convention, commencent par le mot « handle » suivi du type d'événement qu'elles gèrent.

Il y a une petite bizarrerie à laquelle vous devriez faire attention. Jetez à nouveau un œil à cette ligne :

```
return <div onHover={handleHover}></div>
```

La fonction `handleHover()` est transmise sans les parenthèses que nous verrions généralement lors de l'appel d'une fonction. En effet, le transmettre

comme `handleHover` indique qu'il ne doit être appelé qu'une fois l'événement survenu. Le transmettre comme `handleHover()`déclencherait la fonction immédiatement, alors soyez prudent !