Les hooks:

Quelques Règles concernant les looks:

- On ne déclare pas de hooks dans une boucle ou une condition

useState:

Permet d'ajouter des états à un composant.

L'appel au hook se présente de cette façon:
const [state, setState] = useState(initialState);

state: immutable, on ne peut pas le modifier en faisant "state =" ou "state.x = "

setState: permet de modifier state. Il peut prendre en paramètre un valeur ou une fonction.
S'il prend une fonction, celle-ci se présente de la façon suivante:
(étatPrécédent) => {votre code pour définir le nouvel état}

À savoir: après un appel à *setState*, *state n'est* ***mise à jour*** *qu'au* ***prochain cycle d'affichage(render).***

Illustration :

```
const [name, setName] = useState('Taylor')
function handleClick() {
setName('Robin');
console.log(name); // Toujours "Taylor"!
}
```

La valeur de name changera au prochaine cycle d'affichage.
De la même façon, des appels consécutifs à *setState* ne changeront la valeur de *state* qu'une fois:

```
const [age, setAge] = useState(42)
function handleClick() {
setAge(age + 1); // setAge(42 + 1)
setAge(age + 1); // setAge(42 + 1)
setAge(age + 1); // setAge(42 + 1)
}
```

Pour changer la valeur d'age à chaque appel, il faut utiliser la fonction updater qui prend en paramètre l'état précédent:

```
function handleClick() {
setAge(a => a + 1); // setAge(42 => 43)
setAge(a => a + 1); // setAge(43 => 44)
setAge(a => a + 1); // setAge(44 => 45)
}
```

Pour mettre à jour un état qui est un objet, il faut utiliser la déstructuration:

```
// initialisation de l'état
const [person, setPerson] = useState({
name: 'Niki de Saint Phalle',
artwork: {
title: 'Blue Nana',
city: 'Hamburg',
image: 'https://i.imgur.com/Sd1AgUOm.jpg',
}
```

```
});

// modification de l'objet
function handleNameChange(newName) {
setPerson({
...person,
name: newName
});
}
```

Quelques exemples d'utilisation de useState:
https://react.dev/reference/react/useState#examples-basic

Pour aller plus loin: https://react.dev/reference/react/useState

useEffect:

Permet de synchroniser le composant.

L'appel au hook se présente de cette façon:
useEffect(func, [dépendances1, dépendance2, ...])

func: la fonction qui va être appelée quand une des dépendance va changer.

À savoir:

- Le hook useEffect est appelé en dernier à la fin de chaque cycle d'affichage(render)
- Si aucune dépendance n'est spécifiée, la fonction *func* ne sera appelée qu'au moment ou le composant est monté pour la première fois.
- ce hook est appelé au début de la fonction, hors de la partie JSX de celui-ci

Exemple d'utilisation:

```
function ChatRoom({ roomId }) {
const [serverUrl, setServerUrl] = useState('https://localhost:1234');

useEffect(() => {
const connection = createConnection(serverUrl, roomId);
connection.connect();
return () => {
connection.disconnect();
};
}, [serverUrl, roomId]); // si serverUrl ou roomId changent, la fonction ci-dessus
sera appelée à la fin du cycle d'affichage en cours
// ...
}
```

Quelsques exemple d'utilisation de useEffect:
https://react.dev/reference/react/useEffect#examples-connecting

Il est possible de séparer la logique de useEffect en faisant un hook custom:

```
function useChatRoom({ serverUrl, roomId }) {
useEffect(() => {
const options = {
```

```
serverUrl: serverUrl,
roomId: roomId
};
const connection = createConnection(options);
connection.connect();
return () => connection.disconnect();
}, [roomId, serverUrl]);
}
```
Que l ‘on appellera ensuite de la façon suivante dans son comopsant:
```
useChatRoom({
roomId: roomId,
serverUrl: serverUrl
});
```
Pour aller plus loin: https://react.dev/reference/react/useEffect

useReducer:

Tout comme useState, il permet d’ajouter des états à un composant. Néanmoins il s’utilise de façon différente:

```
const [state, dispatch] = useReducer(reducer, initialArg)
```
state: l’état que retourne la fonction
dispatch: la fonction qui applique les modification possibles sur l'état

reducer: la fonction qui référence les modifications possibles sur l’état
initialArg: la valeur initiale de l’état

Illustration:

```
function reducer(state, action) {
switch (action.type) {
case 'incremented_age': {
return {
name: state.name,
age: state.age + 1
};
}
case 'changed_name': {
return {
name: action.nextName,
age: state.age
};
}
}
throw Error('Unknown action: ' + action.type);
}

function Form() {
const [state, dispatch] = useReducer(reducer, { name: 'Taylor', age: 42 });
function handleButtonClick() {
dispatch({ type: 'incremented_age' });
}

function handleInputChange(value) {
dispatch({
type: 'changed_name',
```

```
nextName: value
});
}
// …
}
```

Lorsque je veux modifier un état, je dois utiliser la déstructuration.

Erreur

```
function reducer(state, action) {
switch (action.type) {
case 'incremented_age': {
// 🚩 Don't mutate an object in state like this:
state.age = state.age + 1;
return state;
}
```

Bonne pratique:

```
function reducer(state, action) {
switch (action.type) {
case 'incremented_age': {
// ✅ Instead, return a new object
return {
...state,
age: state.age + 1
};
}
```

Quelques cas pratiques:
https://react.dev/reference/react/useReducer#examples-basic

Pratique: Création des composant de l'application de recette et gestion de leurs états