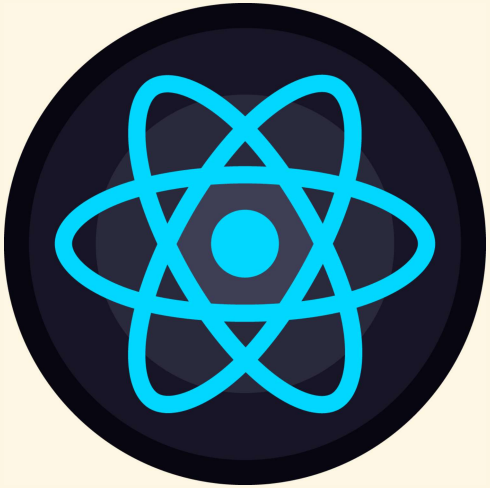
LES HOOKS



DÉFINITION

- Fonctions JavaScript
- Utilise des fonctionnalités de React sans avoir à écrire de classes
- Commencent tous par "use"

DÉFINITION (SUITE)

- Ne peuvent être utilisés que dans les composants fonctionnels.
- Ne peuvent être utilisés que dans le corps du composant.

DÉFINITION (FIN)

- Ne peut être appeler qu'au plus haut niveau (racine) de votre fonction.

```
// Ce code va fonctionner
function Good() {
  const [isOpen, setIsOpen] = useState(false);
  return <>Go</>;
}

// Le hook ne peut pas fonctionner et vous aurez une erreur
function Fail() {
  if (true) {
    const [isOpen, setIsOpen] = useState(false);
  }
  return <>Go</>;
}
```

DÉFINITION (FIN)

- Ne peut être appeler qu'au plus haut niveau (racine) de votre fonction.

```jsx
// Ce code va fonctionner
function Good() {
  const [isOpen, setIsOpen] = useState(false);
  return <>Go</>;
}

// Le hook ne peut pas fonctionner et vous aurez une erreur
function Fail() {
  if (true) {
    const [isOpen, setIsOpen] = useState(false);
  }
  return <>Go</>;
}
```

USESTATE

DÉFINITION

- Permet de gérer l'état d'un composant.
- Prend en entrée un paramètre initial pour l'état
- Retourne un tableau contenant deux éléments :
  - La valeur actuelle de l'état.
  - Une fonction qui permet de mettre à jour la valeur de l'état.

SYNTAXE

```
const maValeurInitiale = 0;
const [monEtat, setMonEtat] = useState(maValeurInitiale);
```

- **monEtat** : valeur actuelle de l'état
- **setMonEtat** : fonction qui permet de mettre à jour la valeur de l'état
- **maValeurInitiale** : valeur initiale de l'état

EXEMPLE

```jsx
// On importe useState depuis la librairie React
import React, { useState } from "react";

function Compteur() {
  // Déclare une nouvelle variable d'état, qu'on va appeler "count"
  const [count, setCount] = useState(0);

  // Fonction setCount, incrémente au clic la valeur de "count" de 1
  return (
    <div>
      <p>Le compteur est actuellement à : {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}
```

EXEMPLE

```jsx
// On importe useState depuis la librairie React
import React, { useState } from "react";

function Compteur() {
  // Déclare une nouvelle variable d'état, qu'on va appeler "count"
  const [count, setCount] = useState(0);

  // Fonction setCount, incrémente au clic la valeur de "count" de 1
  return (
    <div>
      <p>Le compteur est actuellement à : {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}
```

EXEMPLE

```jsx
// On importe useState depuis la librairie React
import React, { useState } from "react";

function Compteur() {
  // Déclare une nouvelle variable d'état, qu'on va appeler "count"
  const [count, setCount] = useState(0);

  // Fonction setCount, incrémente au clic la valeur de "count" de 1
  return (
    <div>
      <p>Le compteur est actuellement à : {count}</p>
      <button onClick={() => setCount(count + 1)}>Incrémenter</button>
    </div>
  );
}
```

USEEFFECT

DÉFINITION

- Permet de se positionner le cycle de vie d'un composant
- Par exemple
  - Récupérer des données à partir d'une API
  - Mettre à jour le DOM

SYNTAXE

- Le `useEffect` accepte 2 paramètres :
  - Une **fonction de rappel** (callback) qui sera exécutée pour l'effet que vous souhaitez "écouter"
  - Permet de définir l'état (state ou props) que l'on souhaite observer **(optionnel)**

```
useEffect(() => {
  // Code exécuté à chaque fois que le composant est rendu
  return () => {
    // Code exécuté à chaque fois que le composant est mis à jour ou démonte
  };
});
```

DÉMO

```
// On importe useEffect depuis la librairie React
import { useEffect } from "react";

function MonComposant() {
  useEffect(() => {
    console.log("Je n'ai qu'un paramètre");
  });

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

QUAND ET COMMENT L'UTILISER ?

DURANT LE MONTAGE DU COMPOSANT

- Il vous faut ajouter le 2eme paramètre que peut accepter `useEffect`.
- Comme nous n'écoutons rien de particulier, nous allons lui fournir un tableau vide : `[]`

```
import { useEffect } from "react";

function MonComposant() {
  useEffect(() => {
    console.log("Je suis un composant prêt à être utilisé");
  }, []);
  return <>Hello world</>;
}
```

DURANT LA MISE À JOUR DU COMPOSANT

- Ex: mettre à jour le titre du document lorsque la valeur de la propriété **titre** d'un composant change

```
import { useEffect } from "react";

function MonComposant({ titre }) {
  useEffect(() => {
    document.titre = titre;
  }, [titre]);

  return <h1>{titre}</h1>;
}
```

DURANT LE DÉMONTAGE DU COMPOSANT

- Il vous faut retourner une fonction dans le callback de `useEffect`

```
useEffect(() => {
  // Ce que vous voulez
  return () => {
    // Fonction de rappelle qui s'exécute lors du démontage
  };
});
```

POURQUOI ?

```
function Counter() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    const countInterval = setInterval(() => {
      setCount((count) => count + 1);
    }, 1000);

    return () => {
      // Si vous ne le faites pas, vous générez une "fuite de mémoire"
      clearInterval(countInterval);
    };
  }, []);
  return <div style={{ fontWeight: "bold", fontSize: 90 }}>{count}</div>;
}
```

CYCLE DE VIE D'UN COMPOSANT FONCTIONNEL

DÉFINITION

- C'est le processus par lequel un composant est **créé**, **mis à jour** et **détruit** au cours de son utilisation.
- Il est composé de 3 phases :
  - Montage
  - Mise à jour
  - Démontage

MONTAGE

- Son contenu est généré en appelant sa fonction de rendu.
- Si le composant possède des enfants, leur cycle de vie est à nouveau lancé depuis le début.

MISE À JOUR

LORSQUE LES PROPS OU L'ÉTAT DU COMPOSANT SONT MIS À JOUR

On utilise useEffect pour exécuter des opérations en réponse à ces mises à jour

ex :

- mettre à jour des données asynchrones
- réagir à des changements d'état ou de props

> *Le composant est à nouveau rendu et son contenu est mis à jour.*

DÉMONTAGE

LORSQUE LE COMPOSANT EST RETIRÉ DE L'ARBRE DE RENDU

- On utilise `useEffect` avec un tableau vide pour effectuer des opérations de nettoyage avant la suppression du composant

DEMO

USEREF

DÉFINITION

- `useRef` permet de créer une référence mutable.
- Une référence mutable est un conteneur qui stocke une valeur mutable, similaire à une variable classique.
- La différence est que la valeur d'une référence mutable ne change pas lors des rendus.

UTILISATION

- `useRef` est souvent utilisé pour stocker une valeur qui doit être conservée entre les rendus.
- Pour stocker une référence à un élément DOM et y accéder dans un gestionnaire d'événement

SYNTAXE

```
const ref = useRef(maValeurInitiale);
```

- Prend en paramètre la valeur initiale de la référence.
- Retourne un objet contenant la propriété `current` qui contient la valeur de la référence.

> *La valeur de référence peut être modifiée en modifiant la propriété `current`.*

EXEMPLE

```jsx
import { useRef } from "react";

// fonction qui retourne un composant qui affice la valeur d'un compteur
// incrémenté à chaque clic sur le bouton
function Counter() {
  const count = useRef(0);
  const increment = () => {
    count.current += 1;
  };
  return (
    <>
      <p>Count: {count.current}</p>
      <button onClick={increment}>Increment</button>
    </>
  );
}
```

USEMEMO

DÉFINITION

- Permet de créer et conserver une valeur calculée.
- Évite de recalculer une valeur à chaque rendu.
- Souvent utilisé pour optimiser les performances.

SYNTAXE

```
const value = useMemo(() => computeValue(a, b), [a, b]);
```

- `useMemo` prend en paramètre une fonction et un tableau de dépendances.
- Retourne la valeur calculée.

EXEMPLE

```
import { useMemo } from "react";

function Example({ data }) {
  const processedData = useMemo(() => doExpensiveComputation(data), [data]);

  return <div>{processedData}</div>;
}
```

LA SUITE

Par ici !