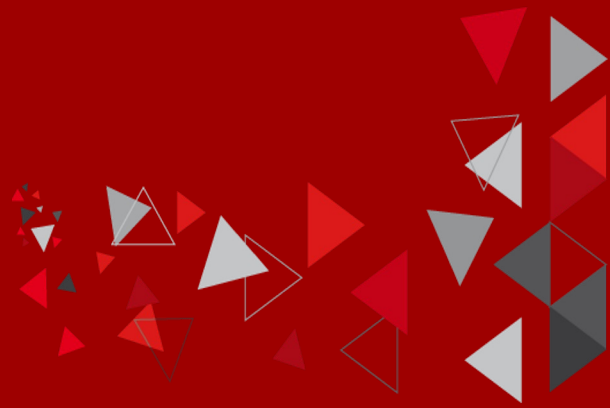
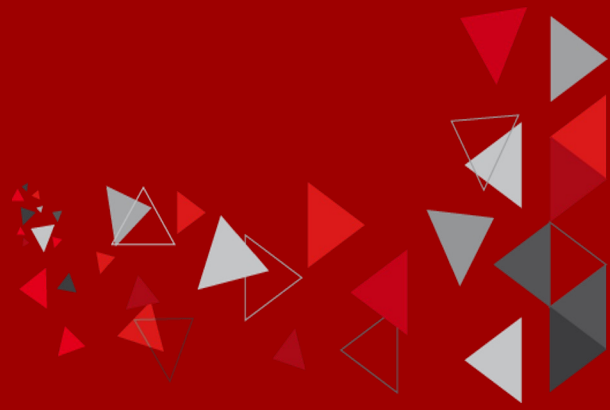
esprit
Se former autrement
HONORIS UNITED UNIVERSITIES

# Composant Fonctionnel & Hooks

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia
EUR-ACE® Délivrée par la Commission des Titres d'Ingénieur
CONFÉRENCE DES GRANDES ÉCOLES
cdio CONCEIVE DESIGN IMPLEMENT OPERATE



- **Composant Fonctionnel**
- **Hooks**
- **Hooks Personnalisés**



# Composant Fonctionnel

## « Functional component »

# Composant Fonctionnel

- Avec React, il est possible d'implémenter deux types de composants:
  - les composants de classes
  - les composants fonctionnels

- Les composant fonctionnels sont beaucoup plus simple à écrire et à comprendre que les composant de classes.
  - Les composants fonctionnels sont **généralement** les plus utilisés.

# Composant Fonctionnel

Un composant fonctionnel est tout simplement une **fonction javascript** qui renvoie du JSX.

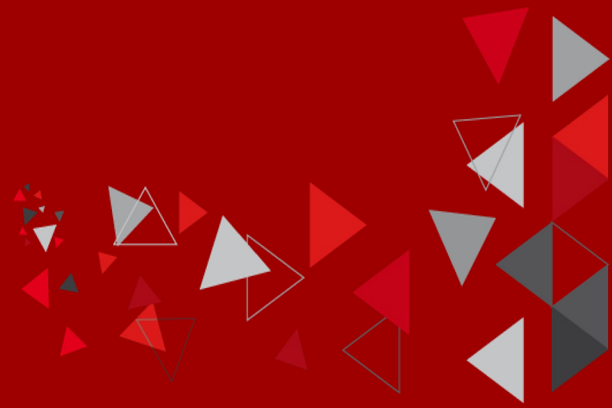
```
function ComposantFonctionnel() {
  return <h1>salut !!!</h1>;
}

const Cp2 = (props) => <h2> une autre ecriture {props.name}!!!</h2>;
```

# Comparaison Composant Fonctionnel et Composants de classes

| Composant fonctionnel | Composant de classe |
|---|---|
| Une fonction Javascript, qui peut prendre des props comme arguments et retourne un élément React. | Une classe Javascript qui hérite de **React.Component.** Il crée une fonction de rendu qui retourne un élément React. |
| N'a pas besoin de la méthode **render()** | L'appel de la méthode **render()** est obligatoire pour afficher un rendu. |
| Certaines méthodes du cycle de vie de react ne peuvent pas y être utilisées. | Toutes les méthodes de cycle de vie peuvent y être utilisées. |
| N'a pas de constructeur. | Peut avoir un constructeur. |

**Différences entre les composants de classes et les composants fonctionnels**



# Hooks

# Hooks

- React hooks sont des fonctionnalités apparues avec la version 16.7 et introduites à React dans la version 16.8
- Les Hooks permettent d'utiliser toutes les fonctionnalités des classes React dans des composants fonctionnels.
- Il est donc possible de développer une application React uniquement avec les composants fonctionnels.

# Hooks

- Il est possible de passer d'un composant de classe à un composant fonctionnel au fur et à mesure dans votre application car les deux sont compatibles.
- Les hooks sont des fonction qui s'accrochent au états et cycle de vie de React.
- Fonction qui permet de bénéficier d'un état local et d'autres fonctionnalités de React sans le recours d'une classe.
- Par convention un Hook commence par **use**

# Hooks

## Introduction

**Il existe deux règles à suivre pour utiliser les Hooks :**

- Les hooks sont appelés **uniquement** dans des composants fonctionnels.

⇒ Ils ne fonctionnent pas dans une classe

- Les hooks sont appelés **au plus haut niveau** d'un composant fonctionnel.

=> On ne peut pas les appeler depuis une boucle, une condition, une fonction ou une sous-fonction.

# Hooks

## Introduction

**React Hooks nous offre la possibilité d'utiliser 10 Hooks :**

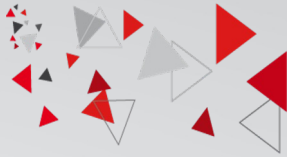
**Hooks Basique:**

- useState
- useEffect
- useContext

**Hooks Additionnels:**

- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

# Hooks



## useState

- **useState** est le hook qui va nous permettre de suivre l'état d'un composant fonctionnel.
- Cette fonction va prendre un état initial en paramètre et retourne un tableau:
  - Le premier paramètre est la valeur de l'état.
  - Le deuxième paramètre est une fonction qui va changer cet état, qui est similaire à **this.setState**.

# Hooks

## useState

**Exemple :**

```
const [state, setState] = useState(initialState);
```

- **state** représente la valeur de la variable courante.
- **setState** est la function qui vas pouvoir changer la valeur de state.
- **initialState** est la valeur initiale.

# Hooks

## useState

**Exemple :**

```
const [{color,background}, setColor] = useState({color:"red",background:"purple"});
return (
  <div className="App">
    <input onChange={e=>setColor(current => ({...current,color:e.target.value}))}/>
    <h1>My favorite color is {color}!</h1>
  </div>
);
```

***Remarque:*** **state** peut être un objet, mais il faudra utiliser l'opérateur de spread pour garder les autres valeurs qu'on ne va pas changer.

# Hooks

## useEffect

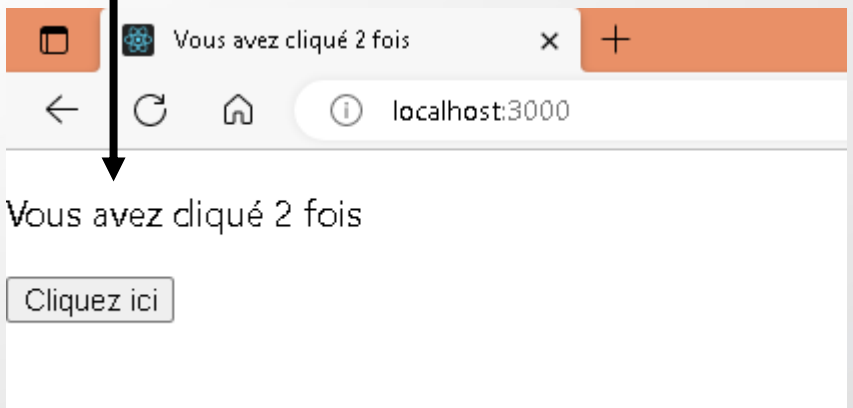
- Le Hook  permet de définir une action à effectuer dès que le composant est affiché ou mis à jour (tout comme les méthodes de cycle de vie des classes React).
- **useEffect**  est le hook qui va nous permettre de simuler le cycle de vie d'un composant React.
- La fonction prend en paramètre **une fonction et un tableau de dépendance** et retourne **une fonction.**

=> On peut avoir plusieurs **useEffect** dans un composant et vont être exécutés dans l'ordre.

# Hooks

## useEffect

```
import React, { useState, useEffect } from 'react'
function Example() {
const [count, setCount] = useState(0);
// Similaire à componentDidMount et componentDidUpdate :
useEffect(() => {
// Met à jour le titre du document via l'API du navigateur
document.title = `Vous avez cliqué ${count} fois`;
});
return (<div>
<p>Vous avez cliqué {count} fois</p>
<button onClick={() => setCount(count + 1)}> Cliquez
ici </button>
</div>);
}
export deafault Example ;
```



# Hooks

## useEffect

### Passage d'un tableau vide

```
useEffect(() => {
//Runs only on the first render
}, []);
```

```
function Timer() {
const [count, setCount] = useState(0);
useEffect(() => {
setTimeout(() => {
setCount((count) => count + 1);
}, 1000);
}, [] ); //Exécuter un effet une seule fois
(au
montage puis au démontage)
return <h1>I've rendered {count} times!
</h1>;
}
```

# Hooks

## useEffect

### Passage de state ou props

```
function Example() {
const [count, setCount] = useState(0);
const [calculation, setCalculation] =
useState(0);
useEffect(() => {
setCalculation(() => count * 2);
}, [count]); // N'exécute l'effet que si
count a changé
return (<>
<p>Count: {count}</p>
<button onClick={() => setCount((c) => c +
1)}>+</button>
<p>Calculation: {calculation}</p>
</>);
}
```

```
useEffect(() => {
//Runs on the first render
//And any time any dependency value changes
}, [prop, state]);
```

# Hooks

## useEffect

**Exemple :**

```
const [{color,background}, setColor] = useState({color:"red",background:"purple"});

useEffect(() => {
  console.log('cette fonction va etre execute la premiere fois et a chaque modification de la variable color ');
  console.log('car le deuxieme argument est color ');
  return () => {
    console.log('cette partie va etre execute pour nettoyer et  lors de loperation unmounting ');
  }
}, [color])

useEffect(() => {
  console.log('cette fonction va etre execute une fois seulement ');
  console.log('car le deuxieme argument est un tableau vide ');
  return () => {
    console.log('cette partie va etre execute seulement lors de loperation unmounting ');
  }
}, [])

useEffect(() => {
  console.log('cette fonction va etre execute chaque re-render');
  console.log('car pas de deuxieme argument passer');
})
```

# Hooks

## useContext

- **useContext** est le hook qui va nous permettre de passer des valeurs d'un composant parent à ces fils sans avoir utiliser les props.
- Ceci nous permet d'avoir un code beaucoup plus propre.
  - Pour créer un context on utilise la syntaxe ci-dessous

```
import React from "react";

const ThemeContext = React.createContext();
export default ThemeContext ;
```

  - Cette déclaration doit se faire en dehors d'une fonction.

# Hooks

## useContext

- Après la déclaration, on encapsule les composants React où on veut accéder à la valeur du contexte

***Remarque :*** On pourra aussi accéder au contexte depuis les fils de ces composant.

```
<ThemeContext.Provider value={{ color, background }}>
<Colors />
</ThemeContext.Provider>
```

# Hooks

## useContext

- Pour récupérer notre contexte, on utilisera la fonction **useContext** avec le nom **«ThemeContext»** dans notre exemple.

```
function Colors() {
    const {color,background} = useContext(ThemeContext)
    return  <h2 style={{color:color,backgroundColor:background}}>Hi, I am a Text!</h2>;
  }
```

# Hooks

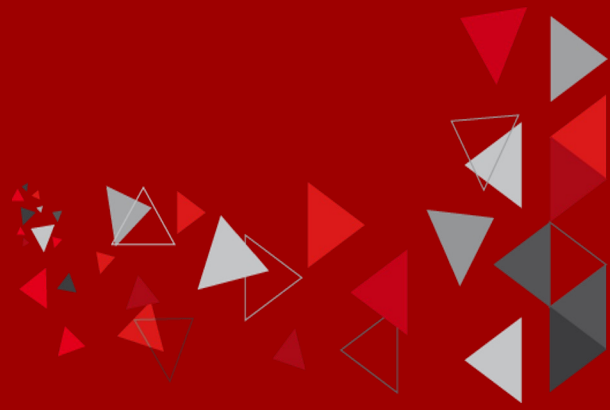
## useRef

- **useRef** se comporte comme un **useState** mais ne va pas engendrer un re-render.
  - Son utilisation principale va être de récupérer des éléments du DOM.

```
const refInput = useRef();
```

```
<input type="text" ref={refInput} />
```

```
refInput.current.onchange = (e) => {
  setColor((current) => ({ ...current, background: e.target.value }));
};
```



# Hooks personnalisés

# Hooks Personnalisés

## Exemple 1 Hook personnalisé

Un Hook personnalisé est une fonction définie par le développeur cette fonction doit impérativement commencer par le mot clé **use**.

elle extrait la logique réutilisable par plusieurs composant.

# Hooks Personnalisés

## Exemple 1 : Hook personnalisé

Ici, on va déclarer notre Hook personnalisé qui retournera **la largeur et l'hauteur** de notre fenêtre .
Nous pouvons constater qu'on a utilisé **useState** et **useEffect.**

```
1   import { useEffect, useState } from "react";
2
3   function useWindowWidth() {
4     const [width, setWidth] = useState(window.innerWidth);
5     const [height, setHeight] = useState(window.innerHeight);
6
7     useEffect(() => {
8       const handleResize = () => {
9         setWidth(window.innerWidth);
10        setHeight(window.innerHeight);
11      };
12      window.addEventListener("resize", handleResize);
13      return () => {
14        window.removeEventListener("resize", handleResize);
15      };
16    }, []);
17
18    return { width, height };
19  }
20
21  export default useWindowWidth;
22
```

# Hooks Personnalisés

## Exemple 1 : Hook personnalisé

- On va déclarer notre custom personnalisé comme n'importe quel autre hook :

```
const { width, height } = useWindowWidth();
```

- Afficher directement le résultat :

```
<h1>width : {width}</h1>
<h1>height : {height}</h1>
```

# Hooks Personnalisés

## Exemple 2 Hook personnalisé

- Ce Hook nous permett de vérifier l'état de notre connexion.

```
const isOnline = useIsOnline();
```

```
<h1>Online : {"" + isOnline + ""}</h1>
```

```
1  import { useEffect, useState } from "react";
2
3  function useIsOnline() {
4    const [online, setOnline] = useState(navigator.onLine);
5    useEffect(() => {
6      setOnline(navigator.onLine);
7      const offlineHandler = () => {
8        setOnline(false);
9      };
10     const onlineHandler = () => {
11       setOnline(true);
12     };
13     window.addEventListener("online", onlineHandler);
14     window.addEventListener("offline", offlineHandler);
15     return () => {
16       window.removeEventListener("online", onlineHandler);
17       window.removeEventListener("offline", offlineHandler);
18     };
19   }, []);
20   return online;
21 }
22 export default useIsOnline;
23
```

# Merci pour  votre attention