Notions

- **State** : les « states » permettent de stocker, véhiculer et manipuler des données dans une application REACT au travers des différents composants. Dès lors qu'un state est modifié, tous les composants utilisant l'information sont impactés et rafraichis.
- **setState** : la fonction setState permet de mettre à jour les données dans le « state » et c'est l'écriture à utiliser pour s'assurer que les composants impactés soient rafraichis. Par ailleurs il faudra toujours mettre en application le concept d'immutabilité et faire en sorte de créer de nouvelles variables / constantes pour modifier la valeur d'un state.
- **Cycle de vie** : des fonctions sont disponibles pour suivre l'évolution d'un composant :
  - **componentDidMount** : lancé après le « montage »
  - **componentDidUpdate** : lancé après une mise à jour
  - **componentWillUnmount** : lancé avant le « démontage »

Exemples

```
class Horloge extends Component {
    state = {
        date: new Date(),
        compteur : 1
    }

    tick = () => {
        this.setState((oldState, props) => {
                return {
                    date : new Date(),
                    compteur : oldState.compteur+1 //dépend de l'ancienne val
                }
            }
        );
    }
    tick2 = () => {
        this.setState({
            date: new Date() //ne dépend pas de l'ancienne valeur du state
        })
    }

    componentDidMount() {
        this.timerId = setInterval(
        () => this.tick()
        ,1000);
    }
    componentWillUnmount(){
        clearInterval(this.timerId);
    }

    render () {
        return (
            <>
                <h2>Horloge : {this.state.date.toLocaleTimeString()}</h2>
                <div>Compteur : {this.state.compteur}</div>
            </>
        );
    }
}
```

Fonction

Objet

Horloge : 10:01:41

Compteur : 229