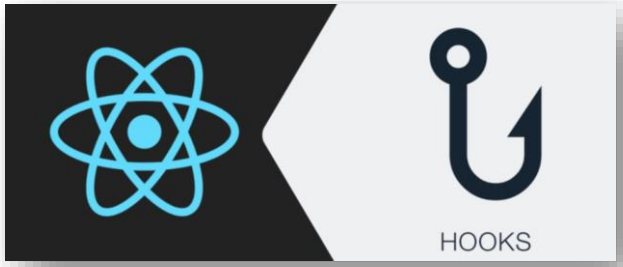
클래스 vs 함수

1. props 사용
2. state 사용
3. Life Cycle 이해 및 활용

※ Hooks

■ Class vs Function

- 리액트에서 **컴포넌트를 사용**할 때 지원하는 **2가지** 방법
- 클래스 방식은 리액트의 모든 기능 활용이 가능하지만
  함수 방식은 내부의 state 활용, life cycle 제어 등이 불가했지만
  hook 기능 도입으로 함수에서도 모든 기능을 활용 가능



class

```
import {Component} from 'react';

class ClassComponent extends Component {
  render() {
    return (
      <h1>Class Component</h1>
    )
  }
}
```

function

```
function FunctionComponent() {
  return (
    <h1>Function Component</h1>
  )
}
```

■ Class vs Function

● 프로젝트 생성

```
mkdir react-app-component
cd react-app-component
create-react-app .
```

```
D:₩study₩react>mkdir react-app-component

D:₩study₩react>cd react-app-component

D:₩study₩react₩react-app-component>create-react-app .

Creating a new React app in D:₩study₩react₩react-app-component.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[                    ] ₩ idealTree:react-app-component: sill idealTree buildDeps
```

■ Class vs Function

● 컴포넌트

App.js

```
function App() {
  return (
    <div className="container">
      <h1>Hello World</h1>
      <ClassComp></ClassComp>
      <FunctionComp></FunctionComp>
    </div>
  );
}

export default App;
```

App.css

```
.App {
  text-align: center;
}

.container {
  border: 5px solid red;
  margin: 5px;
}
```

index.js

```
import App from './App';

const root =
ReactDom.createRoot(document.getElement
ById('root'));

ReactDOM.render( <React.StrictMode>
    <App />
  </React.StrictMode> );
```

■ Class vs Function

● 컴포넌트

App.js

```
import './App.css';
import { Component } from 'react';

class ClassComp extends Component {
  render() {
    return (
      <div className="container">
        <h2>Class Style Component</h2>
      </div>
    )
  }
}

function FunctionComp() {
  return (
    <div className="container">
      <h2>Function Style Component</h2>
    </div>
  )
}
```

■ Class vs Function

● props

class style

```
class ClassComp extends Component {
  render() {
    const number = this.props.initNumber;
    return (
      <div className="container">
        <h2>Class Style Component</h2>
        <p>Number : { number }</p>
      </div>
    )
  }
}
```

props 속성 사용

```
<ClassComp initNumber={2}></ClassComp>
```

function style

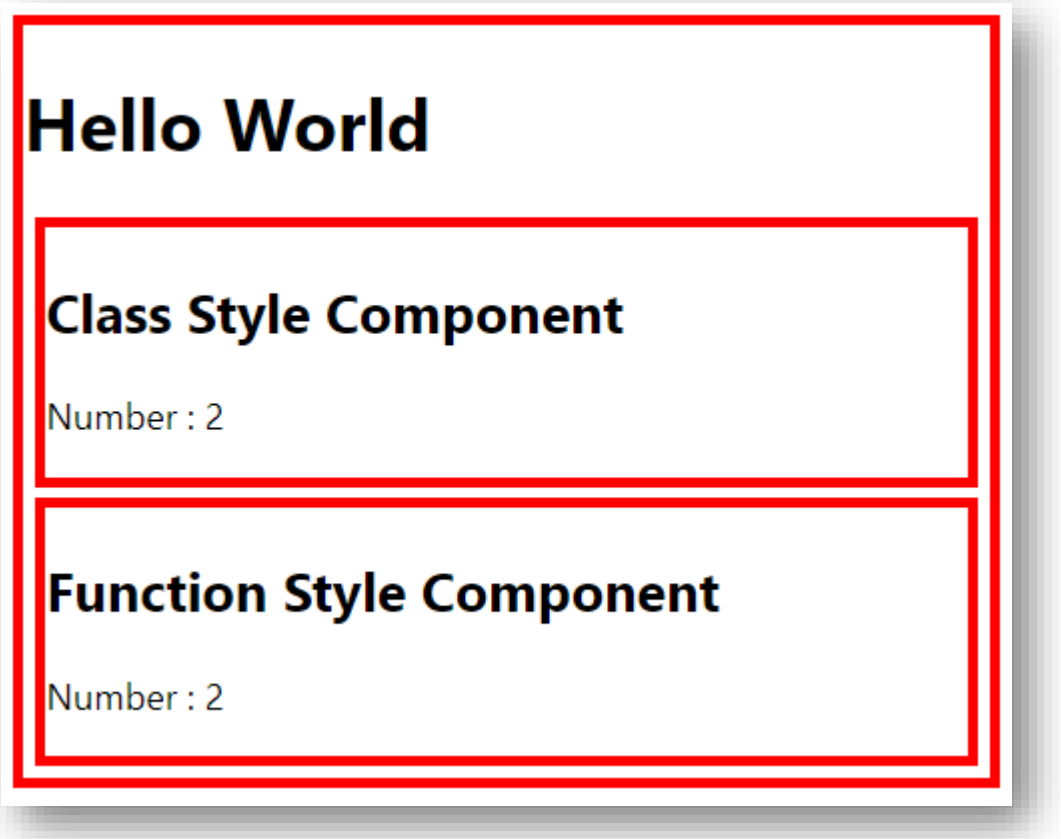
함수의 첫번째 인자 사용

```
function FunctionComp(props) {
  const number = props.initNumber;
  return (
    <div className="container">
      <h2>Function Style Component</h2>
      <p>Number : { number }</p>
    </div>
  )
}
```

```
<FunctionComp initNumber={2}></FunctionComp>
```

■ Class vs Function

● props



■ Class vs Function

● state

class style

```
class ClassComp extends Component {
  state = {
    number: this.props.initNumber
  }
  render() {
    return (
      <div className="container">
        <h2>Class Style Component</h2>
        <p>Number : { this.state.number }</p>
      </div>
    )
  }
}
```

state 속성 사용

```
▼ {number: 2}
    number: 2
  ▶ [[Prototype]]: Object
```

■ Class vs Function

● state

function style

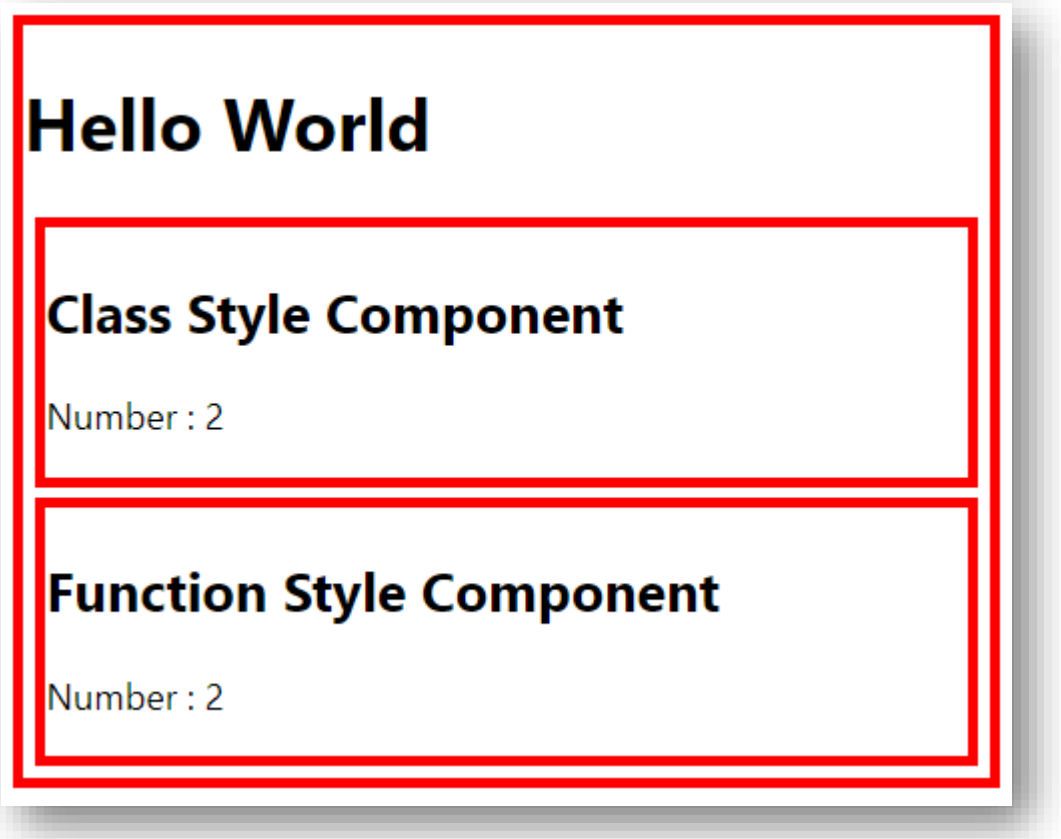
```
import { Component, useState } from 'react';

function FunctionComp(props) {
  const numberState = useState(props.initNumber);
  const number = numberState[0];

  return (
    <div className="container">
      <h2>Function Style Component</h2>
      <p>Number : { number }</p>
    </div>
  )
}
```

React Hook 요소 사용 - useState()

```
▼ (2) [2, ƒ]
    0: 2
  ▶ 1: ƒ ()
    length: 2
  ▶ [[Prototype]]: Array(0)
```

■ Class vs Function

- state



■ Class vs Function

● state 값 변경

class style

```
class ClassComp extends Component {
  state = {
    number: this.props.initNumber
  }
  render() {
    return (
      <div className="container">
        <h2>Class Style Component</h2>
        <p>Number : { this.state.number }</p>
        <button type='button' onClick={() => {

          this.setState({number: Math.random()})
```

setState 메소드 + 객체 사용

```
        }}>random</button>
      </div>
    )
  }
}
```

Class Style Component

Number : 2

random

■ Class vs Function

● state 값 변경

function style

```react/components/FunctionComp.js
import { Component, useState } from 'react';

function FunctionComp(props) {
  const numberState = useState(props.initNumber);
  const number = numberState[0];
  const setNumber = numberState[1];

  return (
    <div className="container">
      <h2>Function Style Component</h2>
      <p>Number : { number }</p>
      <button type='button' onClick={() => {

        setNumber(Math.random())

      }}>random</button>
    </div>
  )
}
```

```
const [number, setNumber] =
  useState(props.initNumber);
```

useState 의 두번째 인자 numberState[1] 사용

Function Style Component

Number : 2

random

■ Class vs Function

● state 값 변경

■ Class vs Function

● state 여러개의 값 변경

class style

```
class ClassComp extends Component {
  state = {
    number: this.props.initNumber,
    date: new Date().toString()
  }
  render() {
    return (
      <div className="container">
        ...
        <p>Date : { this.state.date }</p>
        <button type='button' onClick={() => {

          this.setState({date: new Date().toString()});

        }}>date</button>
      </div>
    )
  }
}
```

setState 메소드 + 객체 사용

■ Class vs Function

● state 여러개의 값 변경

function style

```
function FunctionComp(props) {
  ...
  const dateState = useState(new Date().toString());
  const date = dateState[0];
  const setDate = dateState[1];

  return (
    <div className="container">
      ...
      <p>Date : { date }</p>
      <button type='button' onClick={() => {

        setDate(new Date().toString());

      }}>date</button>
    </div>
  )
}
```

```
const [date, setDate] =
  useState(new Date().toString());
```

useState 의 두번째 인자 dateState[1]사용

Function Style Component

Number : 2

random

Date : Tue Nov 16 2021 09:26:13 GMT+0900 (한국 표준시)

date

■ Class vs Function

- state 여러개의 값 변경

Hello World

Class Style Component

Number : 2

random

Date : Tue Nov 16 2021 09:26:13 GMT+0900 (한국 표준시)

date

Function Style Component

Number : 2

random

Date : Tue Nov 16 2021 09:26:13 GMT+0900 (한국 표준시)

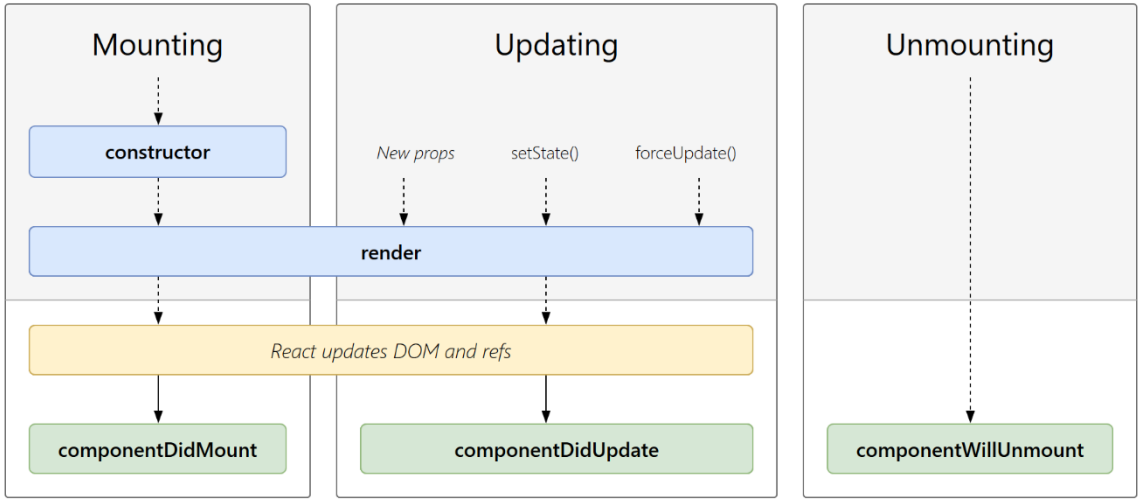
date

■ Class vs Function

● Life Cycle

- 3개의 상황으로 구분
- Mount(생성) / Update(업데이트) / Unmount(제거)
- Update는 다시 4개의 상황으로 구분

1. props 변경
2. **<u>state 변경</u>**
3. 부모 컴포넌트 Rerending
4. forceUpdate 메소드로 강제 업데이트 실행



■ Class vs Function

● Life Cycle

[Mounting]

constructor ➔ ~~*WillMount*~~ ➔ **render** ➔ DidMount

[Updating]

shouldComponentUpdate ➔ ~~*WillUpdate*~~ ➔ **render** ➔ DidUpdate

[Unmounting]

WillUnmount

```
componentDidCatch?        (method) React.ComponentLifecycle<any, any, any>.componentDidCa…
componentDidMount?
componentDidUpdate?
componentWillUnmount?
abc Component
~~componentWillMount?~~
~~componentWillReceiveProps?~~
~~componentWillUpdate?~~
shouldComponentUpdate?
~~UNSAFE_componentWillMount?~~
~~UNSAFE_componentWillReceiveProps?~~
~~UNSAFE_componentWillUpdate?~~
```

■ Life Cycle

● Mounting

class style

```
class ClassComp extends Component {
  constructor(props) {
    super(props);
    console.log('Class Comp => constructor');  ①
  }

  componentWillMount() {  // Deprecated
    console.log('Class Comp => componentWillMount');  ②
  }

  componentDidMount() {
    console.log('Class Comp => componentDidMount');  ④
  }

  render() {
    console.log('Class Comp => render');  ③
    ...
```

```
Class Comp => constructor
Class Comp => componentWillMount
Class Comp => render
Class Comp => componentDidMount
>
```

■ Life Cycle

● Updating

class style

```
class ClassComp extends Component {
  ...

  shouldComponentUpdate(nextProps, nextState) {
    console.log('Class Comp => shouldComponentUpdate');  ①
    return true;
  }
  componentWillUpdate(nextProps, nextState) {  // Deprecated
    console.log('Class Comp => componentWillUpdate');  ②
  }
  componentDidUpdate(nextProps, nextState) {
    console.log('Class Comp => componentDidUpdate');  ④
  }

  render() {
    console.log('Class Comp => render');  ③
    ...
```

```
Class Comp => shouldComponentUpdate
Class Comp => componentWillUpdate
Class Comp => render
Class Comp => componentDidUpdate
>
```

■ Life Cycle

● Unmounting

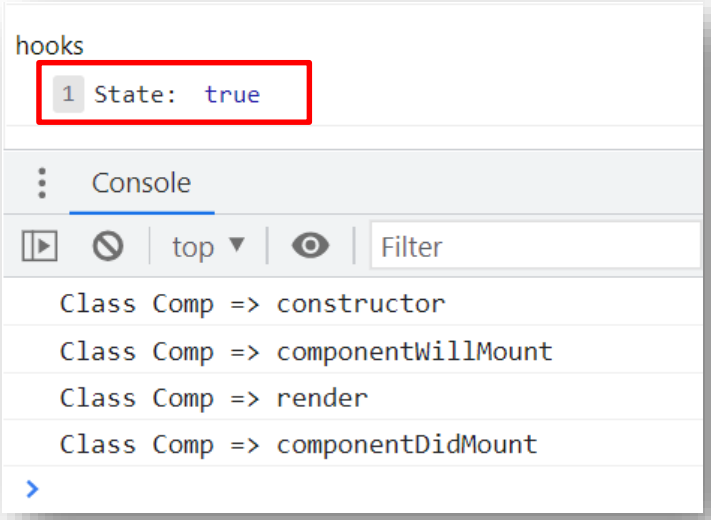
class style

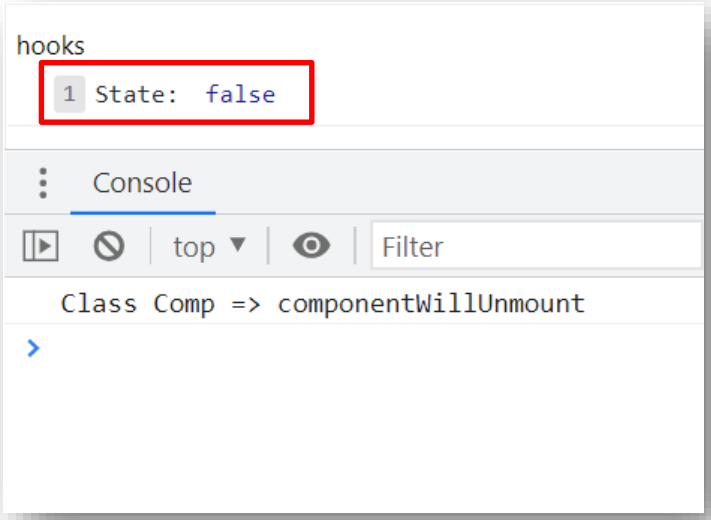
```
class ClassComp extends Component {
  ...

  componentWillUnmount() {
    console.log('Class Comp => componentWillUnmount');
  }

  ...

function App() {
  const [isShow, setShow] = useState(true);

  return (
    <div className="container">
      <h1>Hello World</h1>
      {
        isShow ? <ClassComp initNumber={2}></ClassComp> : null
      }

    ...
```

컴포넌트 제거를 위한 State 사용

■ Life Cycle

● Unmounting





■ Life Cycle

● Mounting

function style

```
import { Component, useState, useEffect } from 'react';

function FunctionComp(props) {
  console.log('Function Comp => render');

  ...

  useEffect(function() {
    console.log('Function Comp => useEffect');
  });

  ...
```

①

②

```
Function Comp => render
Function Comp => useEffect
>
```

componentDidMount 의 역할과 동일

■ Life Cycle

● Updating

function style

```
import { Component, useState, useEffect } from 'react';

function FunctionComp(props) {
  console.log('Function Comp => render');  ①

  ...

  useEffect(function() {
    console.log('Function Comp => useEffect');  ②
  });

  ...
```

```
Function Comp => render
Function Comp => useEffect
>
```

componentDidUpdate 의 역할과 동일

■ Life Cycle

● Unmounting

function style

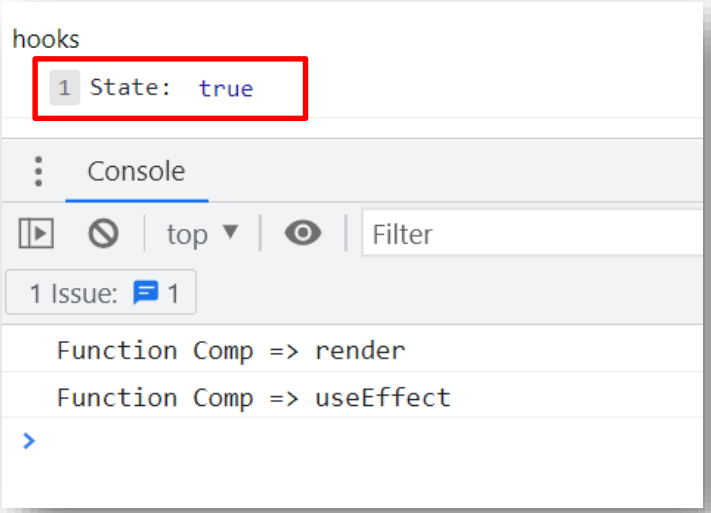
```
function FunctionComp(props) {
  console.log('Function Comp => render');
  ...
  useEffect(function() {
    console.log('Function Comp => useEffect');

    return function() {
      console.log(`Function Comp => useEffect return`);
    }
  });
  ...

function App() {
  ...
  return (
    <div className="container">
      ...
      {
        isShow ? <FunctionComp initNumber={2}></FunctionComp> : null
      }
      ...
```

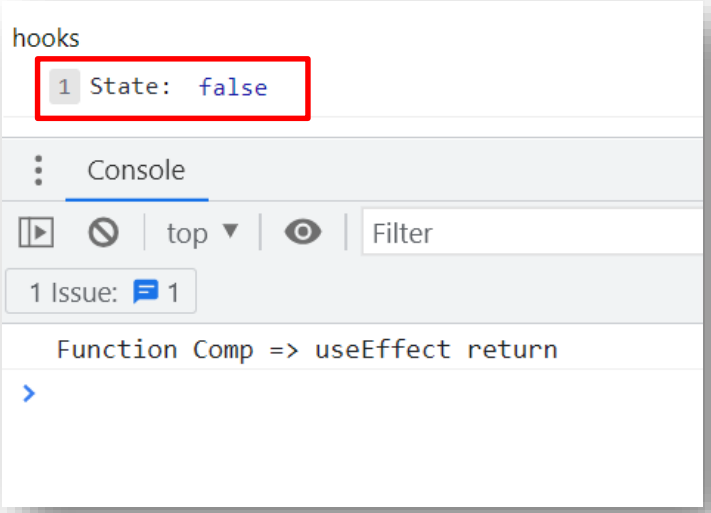
Clean-up 함수

컴포넌트 제거를 위한 State 사용

■ Life Cycle

● Unmounting





■ Life Cycle

● 클래스 생명주기 메소드 / useEffect() 참고사항

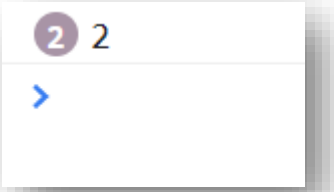
- 클래스 : 최신 state 상태 사용
- useEffect() : 렌더링 당시의 state 상태 사용

class style

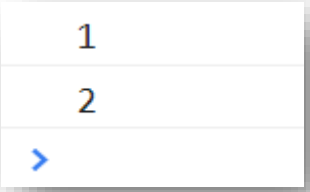
```
<button onClick={() => {
  this.state.count++;
  this.setState(
    {
      state: this.state.count
    }
  );
}}>click</button>

...

setTimeout(() => {
  console.log(`${this.state.count}`);
}, 1000);
```

function style

```
<button onClick={() => {

  setCount(count + 1);

}}>click</button>

...

const [count, setCount] = useState(0);
useEffect(function() {
  setTimeout(() => {
    console.log(`${count}`);
  }, 1000);
});
```



버튼 두번 연속 클릭



■ Life Cycle

● 클래스 생명주기 메소드 / useEffect() 참고사항

class style

```
class ClassCounter extends Component {
  state = { count: 0 }
  componentDidUpdate() {
    setTimeout(() => {
      console.log(`${this.state.count}`);
    }, 1000);
    return true;
  }

  render() {
    return (
      <>
        <button onClick={() => {
          this.state.count++;
          this.setState({state: this.state.count});
        }}>click</button>
      </>
    )
  }
}
```

```
function App() {
  ...
  return (
    <div className="container">
      ...
      <ClassCounter></ClassCounter>
    </div>
  );
}
```

■ Life Cycle

● 클래스 생명주기 메소드 / useEffect() 참고사항

function style

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

  useEffect(function() {
    setTimeout(() => {
      console.log(`${count}`);
    }, 1000);
  });

  return (
    <>
      <button onClick={() => {
        setCount(count + 1);
      }}>click</button>
    </>
  )
}
```

```
function App() {
  ...
  return (
    <div className="container">
      ...
      <FunctionCounter></FunctionCounter>
    </div>
  );
}
```

■ Life Cycle

● useEffect() 에서 최신 state 값 사용하기

function style

```
import React, { Component, useState, useEffect, useRef } from 'react';

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

  const lastestCount = useRef();

  useEffect(function() {

    lastestCount.current = count;

    setTimeout(() => {
      console.log(`${lastestCount.current}`);
    }, 1000);
  });

  return (
    <>
      <button onClick={() => {
        setCount(count + 1);
      }}>click</button>
    </>
  )
}
```

■ Life Cycle

● 컴포넌트 생성 시 한번만 useEffect() 실행하기 - Mounting

function style

```
import React, { Component, useState, useEffect, useRef } from 'react';

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

  const lastestCount = useRef();

  useEffect(function() {

    lastestCount.current = count;

    setTimeout(() => {
      console.log(`${lastestCount.current}`);
    }, 1000);
  });

  useEffect(function() {
    console.log(`FunctionCounter Mount`);
  }, []);

  ...
```

deps array

■ Life Cycle

● 지정된 값 변경 시 useEffect() 실행하기 - Updating

function style

```
import React, { Component, useState, useEffect, useRef } from 'react';

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

  const lastestCount = useRef();

  useEffect(function() {

    lastestCount.current = count;

    setTimeout(() => {
      console.log(`${lastestCount.current}`);
    }, 1000);
  });

  useEffect(function() {
    console.log(`FunctionCounter Mount`);
  }, [count]);

  ...
```

deps array

■ Life Cycle

● **Clean-up 함수**는 Unmounting과 Updating 수행 시 동작

- 동작 순서

1. props 또는 state 업데이트
2. render() - 컴포넌트 Rerendering
3. **useEffect() return 함수**
4. useEffect()