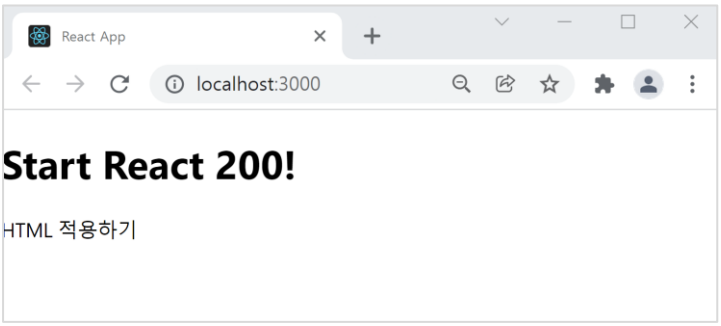
01장 입문 - React.js 시작하기

001 .jsx에 html 적용하기

- 학습 내용: react에서 html 코드를 적용하는 방법을 이해한다.
- 힌트 내용: jsx 소스에서 return() 안에 html 코드를 입력한다.
- App.js 파일
  - 05~08: 스타일이 적용되지 않은 기본 HTML 코드만 삽입된 상태이다.

[react200-lab/src/001/App.js]

```
01  import React from 'react';
02
03  function App() {
04    return (
05      <div>
06        <h1>Start React 200!</h1>
07        <p>HTML 적용하기</p>
08      </div>
09    );
10  }
11
12  export default App;
```



002 .jsx에 css 적용하기

- 학습 내용: react에서 css 파일로 html 코드에 스타일을 적용하는 방법을 이해한다.
- 힌트 내용: css 파일을 별도로 만든 후 jsx 파일에서 임포트(import)해 사용한다.
- App.css 파일

[react200-lab/src/002/App.css]

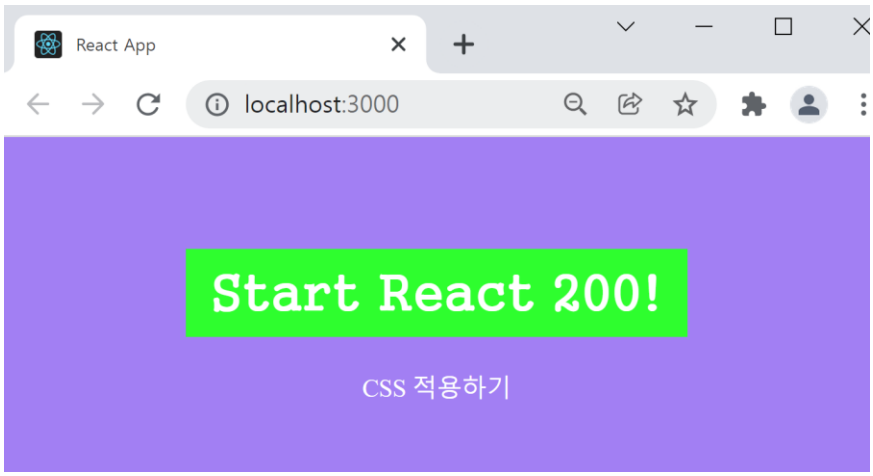
```
01  div {
02    background-color: rgb(162, 127, 243);
03    color: rgb(255, 255, 255);
04    padding: 40px;
05    font-family: 고딕;
06    text-align: center;
07  }
```

```
08
09  h1 {
10    color: white;
11    background-color: #2EFE2E;
12    padding: 10px;
13    font-family: 궁서;
14  }
```

■ App.js 파일

[react200-lab/src/002/App.js]

```
01  import React from 'react';
02  import './App.css';
03
04  function App() {
05    return (
06      <div>
07        <h1>Start React 200!</h1>
08        <p>CSS 적용하기</p>
09      </div>
10    );
11  }
12
13  export default App;
```



003 Component 사용하기

■ 학습 내용: react에서 component를 사용해 다른 파일에 있는 HTML 코드를 이식해 사용하는 방법을 이해한다.

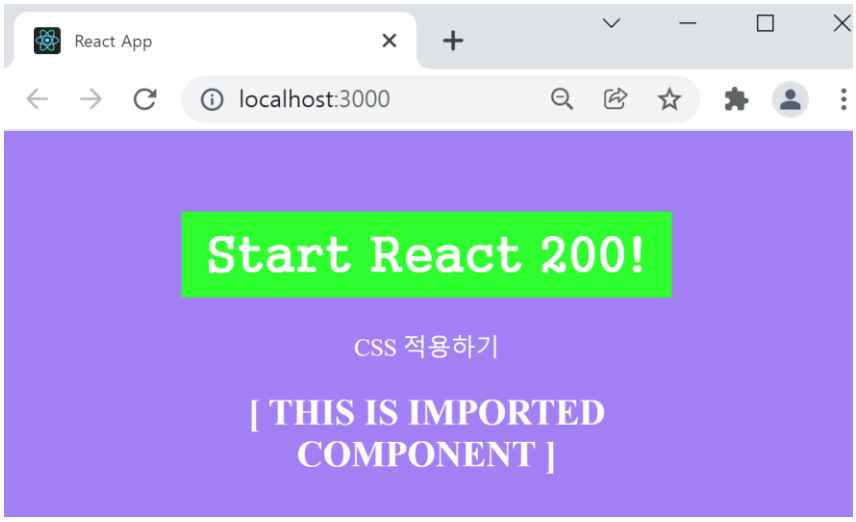
■ 힌트 내용: 다른 파일에서 작성한 html 코드를 component 단위로 임포트해 사용한다.

■ R001_ImportConponent.js 파일

[react200-lab/src/003/R001_ImportConponent']

```
01  import React, { Component } from 'react';
02
03  class R001_ImportConponent extends Component {
```

```
04    render () {
05      return (
06        <h2>[ THIS IS IMPORTED COMPONENT ]</h2>
07      )
08    }
09  }
10
11  export default R001_ImportConponent;
```

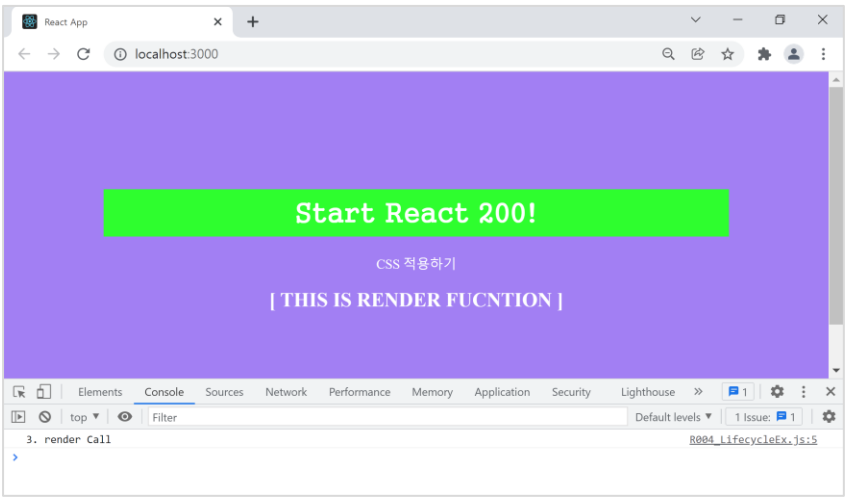


004 생명주기 함수 render() 사용하기

- 학습 내용: 컴포넌트의 생명주기 함수 중 render()에 대해 이해한다.
- 힌트 내용: render() 함수가 실행될 때 로그를 출력하고 콘솔에서 로그를 확인한다.
- R004_LifecycleEx.js 파일
  - 04: render()는 return되는 html 형식의 코드를 화면에 그려주는 함수다. 화면 내용이 변경돼야 할 시점에 자동으로 호출된다.

[react200-lab/src/004/R004_LifecycleEx.js]

```
01  import React, { Component } from 'react';
02
03  class R004_LifecycleEx extends Component {
04    render() {
05      console.log('3. render Call');
06      return (
07        <h2>[ THIS IS RENDER FUCNTION ]</h2>
08      )
09    }
10  }
11
12  export default R004_LifecycleEx;
```

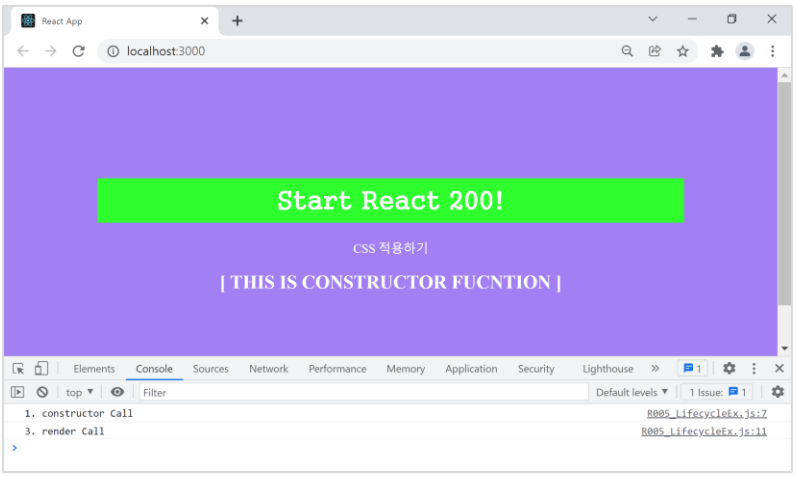


005 생명주기 함수 constructor(props) 사용하기

- 학습 내용: 컴포넌트의 생명주기 함수 중 constructor()에 대해 이해한다.
- 힌트 내용: 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.
- R005_LifecycleEx.js 파일
  - 04~08: constructor(props) 함수는 생명주기 함수 중 가장 먼저 실행되며, 처음 한 번만 호출된다. component 내부에서 사용되는 변수(state)를 선언하고 부모 객체에서 전달받은 변수(props)를 초기화할 때 사용한다. super() 함수는 가장 위에 호출해야 한다.

[react200-lab/src/005/R005_LifecycleEx.js]

```
01  import React, { Component } from 'react';
02
03  class R005_LifecycleEx extends Component {
04    constructor(props) {
05      super(props);
06      this.state = {};
07      console.log('1. constructor Call');
08    }
09
10    render() {
11      console.log('3. render Call');
12      return (
13        <h2>[ THIS IS CONSTRUCTOR FUCNTION ]</h2>
14      )
15    }
16  }
17
18  export default R005_LifecycleEx;
```



006 생명주기 함수 static getDerivedStateFormProps(props, state) 사용하기

- 학습 내용: 컴포넌트의 생명주기 함수 중 getDerivedStateFormProps()에 대해 이해한다.
- 힌트 내용: 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.
- App.js 파일
  - 10~12: R006_LifecycleEx 컴포넌트로 prop_value라는 변수를 전달한다.

[react200-lab/src/006/App.js]

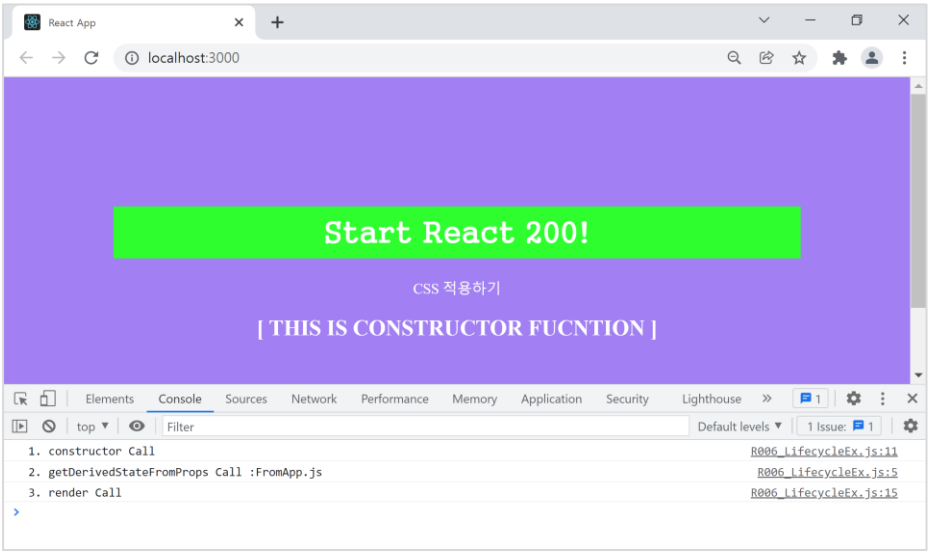
```
01  import React from 'react';
02  import './App.css';
03  import LifecycleEx from './R006_LifecycleEx'
04
05  function App() {
06    return (
07      <div>
08        <h1>Start React 200!</h1>
09        <p>CSS 적용하기</p>
10        <LifecycleEx
11          prop_value = 'FromApp.js'
12        />
13      </div>
14    );
15  }
16
17  export default App;
```

- R006_LifecycleEx.js 파일
  - 04~07: getDerivedStateFromProps(props, state) 함수는 constructor() 함수 다음으로 실행된다.

[react200-lab/src/006/R006_LifecycleEx.js]

```
01  import React, { Component } from 'react';
02
03  class R006_LifecycleEx extends Component {  static getDerivedStateFromProps(props, state) {
```

```
04      console.log('2. getDerivedStateFromProps Call :'+props.prop_value);
05      return {};
06    }
07    constructor(props) {
08      super(props);
09      this.state = {};
10      console.log('1. constructor Call');
11    }
12
13    render() {
14      console.log('3. render Call');
15      return (
16        <h2>[ THIS IS CONSTRUCTOR FUCNTION ]</h2>
17      )
18    }
19  }
20
21  export default R006_LifecycleEx;
22
```



007 생명주기 함수 componentDidMount() 사용하기

■ 학습 내용: 컴포넌트의 생명주기 함수 중 componentDidMount()에 대해 이해한다.

■ 힌트 내용: 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.

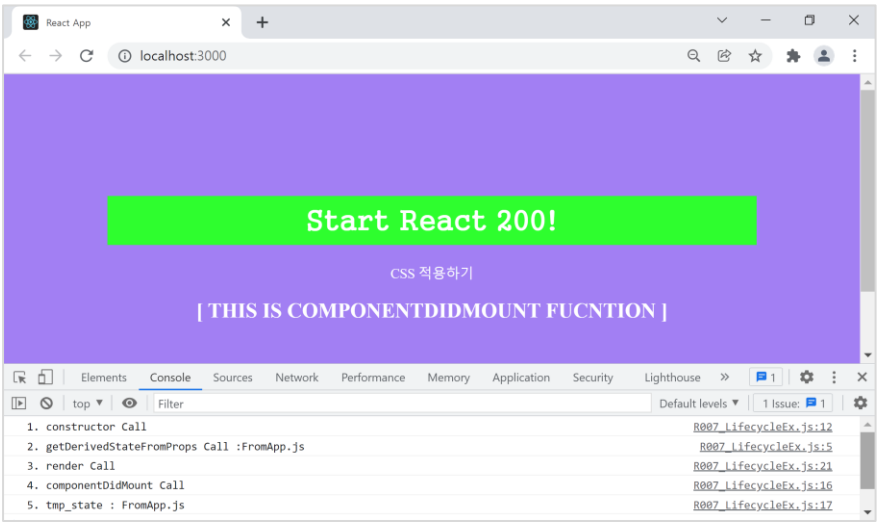
■ R007_LifecycleEx.js 파일

- 15~18: componentDidMount() 함수는 작성한 함수들 중 가장 마지막으로 실행된다. render() 함수가 return되는 html 형식의 코드를 화면에 그려준 후 실행된다. 화면이 모두 그려진 후에 실행돼야하는 이벤트 처리, 초기화 등 가장 많이 활용되는 함수다.

[react200-lab/src/007/R007_LifecycleEx.js]

```
01  import React, { Component } from 'react';
02
03  class R007_LifecycleEx extends Component {
04    static getDerivedStateFromProps(props, state) {
05      console.log('2. getDerivedStateFromProps Call :'+props.prop_value);
```

```
06      return {tmp_state:props.prop_value};
07    }
08
09    constructor(props) {
10      super(props);
11      this.state = {};
12      console.log('1. constructor Call');
13    }
14
15    componentDidMount() {
16      console.log('4. componentDidMount Call');
17      console.log('5. tmp_state : '+this.state.tmp_state);
18    }
19
20    render() {
21      console.log('3. render Call');
22      return (
23        <h2>[ THIS IS COMPONENTDIDMOUNT FUCNTION ]</h2>
24      )
25    }
26  }
27
28  export default R007_LifecycleEx;
```

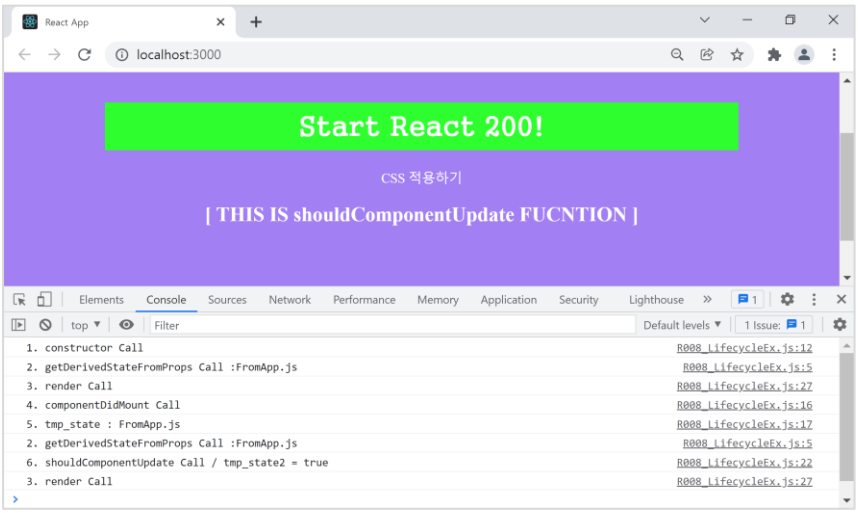


008 생명주기 함수 shouldComponentUpdate() 사용하기

- 학습 내용: 컴포넌트의 생명주기 함수 중 shouldComponentUpdate()에 대해 이해한다.
- 힌트 내용: 생명주기 함수들이 실행될 때 로그를 출력하고 실행 순서를 확인한다.
- R008_LifecycleEx.js 파일
  - 18: state의 변경이 발생하기 때문에 '변경' 단계의 생명주기 함수 shouldComponentUpdate()가 실행된다.
  - 21~24: shouldComponentUpdate() 함수는 component의 변경 과정에 속한다. 여기서 변경이란 props나 state의 변경을 말한다.

[react200-lab/src/008/R008_LifecycleEx.js]

```
01  import React, { Component } from 'react';
02
03  class R008_LifecycleEx extends Component {
04    static getDerivedStateFromProps(props, state) {
05      console.log('2. getDerivedStateFromProps Call :'+props.prop_value);
06      return {tmp_state:props.prop_value};
07    }
08
09    constructor(props) {
10      super(props);
11      this.state = {};
12      console.log('1. constructor Call');
13    }
14
15    componentDidMount() {
16      console.log('4. componentDidMount Call');
17      console.log('5. tmp_state : '+this.state.tmp_state);
18      this.setState({tmp_state2 : true});
19    }
20
21    shouldComponentUpdate(props, state) {
22      console.log('6. shouldComponentUpdate Call / tmp_state2 = ' + state.tmp_state2);
23      return state.tmp_state2
24    }
25
26    render() {
27      console.log('3. render Call');
28      return (
29        <h2>[ THIS IS shouldComponentUpdate FUCNTION ]</h2>
30      )
31    }
32  }
33
34  export default R008_LifecycleEx;
```



009 템플릿 문자열 사용하기

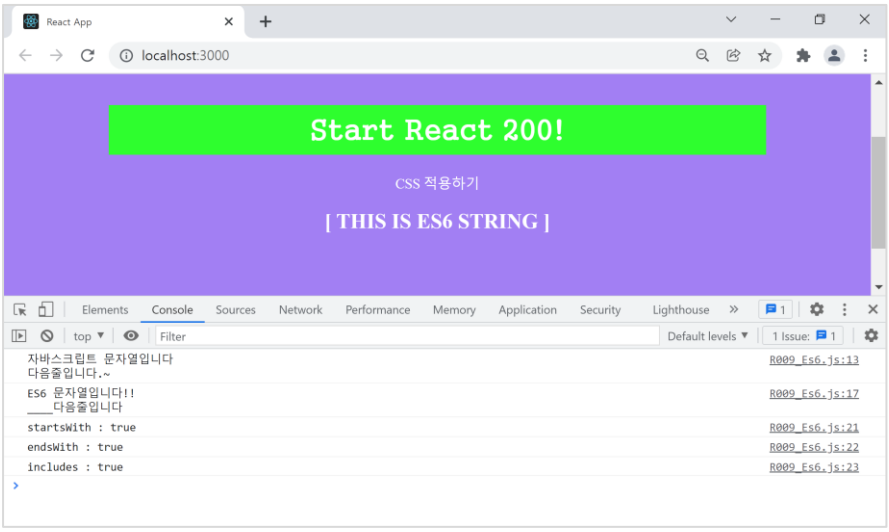
- 학습 내용: ES6 문자열의 사용 방법에 대해 이해한다.
- 힌트 내용: 기존 자바스크립트 문자열과 비교하고 추가된 함수를 확인한다.

■ R009_Es6.js 파일

- 13: 문자열과 변수를 합치기 위해서는 문자열을 작음 따옴표로 감싸고 +로 연결해야 한다.
- 17: 따옴표(')가 아닌 백틱(`)으로 전체 문자열과 변수를 묶어 사용한다.

[react200-lab/src/009/R009_Es6.js]

```
01  import React, { Component } from 'react';
02
03  class R009_Es6 extends Component {
04
05    constructor(props) {
06      super(props);
07      this.state = {};
08    }
09
10    componentDidMount() {
11      var jsString1 = '자바스크립트'
12      var jsString2 = '입니다\n다음줄입니다.'
13      console.log(jsString1+' 문자열'+jsString2+'~');
14
15      var Es6String1 = 'ES6'
16      var Es6String2 = '입니다'
17      console.log(`${Es6String1} 문자열${Es6String2}!!
18  ____다음줄입니다`);
19
20      var LongString = "ES6에추가된String함수들입니다.";
21      console.log('startsWith : '+LongString.startsWith("ES6에추"));
22      console.log('endsWith : '+LongString.endsWith("함수들입니다."));
23      console.log('includes : '+LongString.includes("추가된String"));
24    }
25
26    render() {
27      return (
28        <h2>[ THIS IS ES6 STRING ]</h2>
29      )
30    }
31  }
32
33  export default R009_Es6;
```

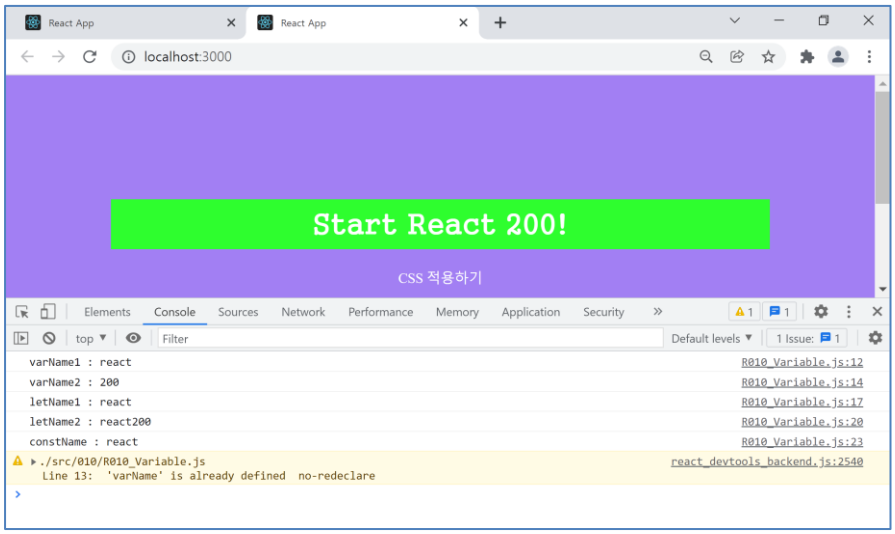


010 var, let, const 사용하기

- 학습 내용: ES6의 변수 선언 방식인 let과 const에 대해 이해한다.
- 힌트 내용: 기존 var 변수와 비교해 변수의 재선언, 재할당이 기능한지 확인한다.
- ES5에서 사용하던 var는 유연한 방식으로 변수를 재선언, 재할당할 수 있다. 이런 특징으로 인해 변수의 사용 범위가 불확실해지거나 의도하지 않은 변수값 변경이 발생할 수 있다. 이러한 var의 단점을 보완하기 위해 ES6에서 let와 const가 추가되었다.
- R010_Variable.js 파일

[react200-lab/src/010/R010_Variable.js]

```
01  import React, { Component } from 'react';
02
03  class R010_Variable extends Component {
04
05    constructor(props) {
06      super(props);
07      this.state = {};
08    }
09
10    componentDidMount() {
11      var varName = 'react'
12      console.log('varName1 : '+varName)
13      var varName = '200' // 'varName' is already defined  no-redeclare
14      console.log('varName2 : '+varName)
15
16      let letName = 'react'
17      console.log('letName1 : '+letName)
18      // let letName = '200' // Parsing error: Identifier 'letName' has already been declared
19      letName = 'react200'
20      console.log('letName2 : '+letName)
21
22      const constName = 'react'
23      console.log('constName : ' + constName)
24      // const constName = '200' // Parsing error: Identifier 'constName' has already been declared
25      // constName = 'react200' // Uncaught TypeError: Assignment to constant variable.
26    }
27
28    render() {
29      return (
30        <h2>[ THIS IS Variable ]</h2>
31      )
32    }
33  }
34
35  export default R010_Variable;
```



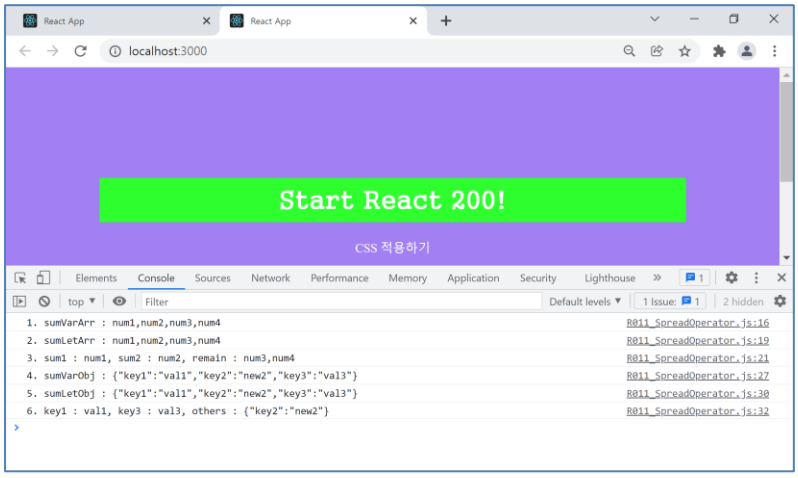
011 전개 연산자 사용하기

- 학습 내용: 전개 연산자를 통해 배열과 객체의 데이터를 합치거나 추출하는 방법을 이해한다.
- 힌트 내용: 기존 ES5 문법과 비교해 ES6 전개 연산자의 장점과 사용법을 확인한다.
- 전개 연산자는 배열이나 객체 변수를 좀 더 직관적이고 편리하게 합치거나 추출할 수 있게 도와주는 문법이다. 변수 앞에 ...(마침표 3개)를 입력해 사용한다.
- R011_SpreadOperator.js 파일
  - 26: 기존 ES5에서 객체 2개를 합치기 위해서는 Object.assign() 함수를 이용해야 한다. 첫 번째 인자 {}는 함수의 return 값이고 뒤의 인자에 객체들을 콤마(,)로 연결해 나영하면 여러 개의 객체를 합칠 수 있다.
  - 29: ES6에서는 ...(마침표 3개)을 객체명 앞에 붙여 여러 개의 객체를 합칠 수 있다.
  - 31: sumLetObj 객체의 키와 값을 추출해 키와 동일한 명칭의 개별 변수에 넣는다. 나머지는 마지막에 전개 연산자 처리된 ...others 변수에 넣는다.

[react200-lab/src/011/R011_SpreadOperator.js]

```
01  import React, { Component } from 'react';
02
03  class R011_SpreadOperator extends Component {
04
05    constructor(props) {
06      super(props);
07      this.state = {};
08    }
09
10    componentDidMount() {
11      //javascript Array
12      var varArray1 = ['num1', 'num2'];
13      var varArray2 = ['num3', 'num4'];
14      var sumVarArr = [varArray1[0], varArray1[1], varArray2[0], varArray2[1]];
15      // var sumVarArr = [].concat(varArray1, varArray2);
16      console.log('1. sumVarArr : '+sumVarArr)
17      //ES6 Array
18      let sumLetArr = [...varArray1, ...varArray2];
```

```
19      console.log('2. sumLetArr : '+sumLetArr)
20      const [ sum1, sum2, ...remain] = sumLetArr;
21      console.log('3. sum1 : '+sum1+', sum2 : '+sum2+ ', remain : '+remain)
22
23      var varObj1 = { key1 : 'val1', key2 : 'val2' }
24      var varObj2 = { key2 : 'new2', key3 : 'val3' }
25      //javascript Object
26      var sumVarObj = Object.assign({}, varObj1, varObj2)
27      console.log('4. sumVarObj : '+JSON.stringify(sumVarObj))
28      //ES6 Object
29      var sumLetObj = {...varObj1, ...varObj2}
30      console.log('5. sumLetObj : '+JSON.stringify(sumLetObj))
31      var {key1, key3, ...others} = sumLetObj;
32      console.log('6. key1 : '+key1+', key3 : '+key3+', others : '+JSON.stringify(others));
33    }
34
35    render() {
36      return (
37        <h2>[ THIS IS SpreadOperator ]</h2>
38      )
39    }
40  }
41
42  export default R011_SpreadOperator;
```



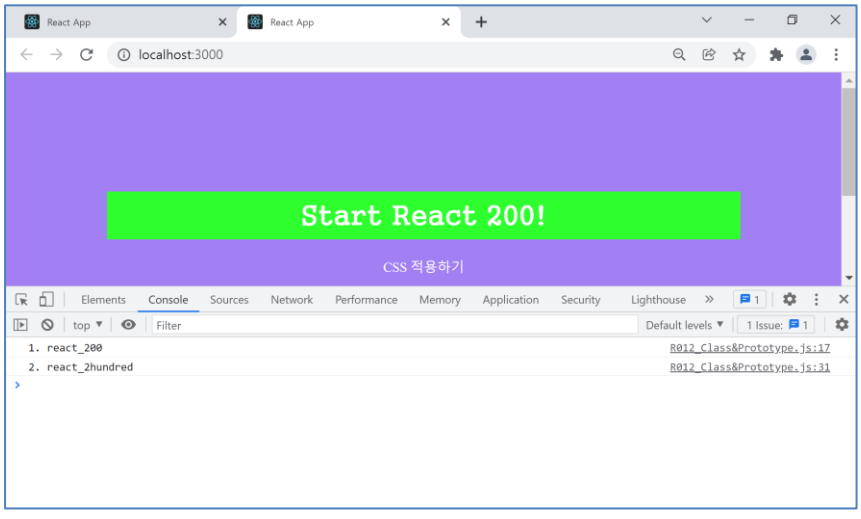
012 class 사용하기

- 학습 내용: 코드를 객체 단위로 재사용하는 class에 대해 이해한다.
- 힌트 내용: 기존 ES5의 prototype과 비교해 ES6 class의 장점과 사용법을 확인한다.
- 기존 ES5 자바스크립트에서는 객체를 구현하기 위해 prototype을 사용한다. ES6에서 등장한 class는 prototype과 같은 개념인데, 쉽게 읽고 표현하기 위해 고안된 문법이다.
- R012_Class&Prototpye.js 파일

[react200-lab/src/012/R012_Class&Prototype.js]

```
01  import React, { Component } from 'react';
02
03  class ClassPrototype extends Component {
```

```
04
05    constructor(props) {
06      super(props);
07      this.state = {};
08    }
09
10    componentDidMount() {
11      //ES5 prototype
12      var ExamCountFunc = (function () {
13        function ExamCount(num) {
14          this.number = num;
15        }
16        ExamCount.prototype.showNum = function () {
17          console.log('1. react_' + this.number);
18        };
19        return ExamCount;
20      }());
21
22      var cnt = new ExamCountFunc('200');
23      cnt.showNum();
24
25      //ES6 class
26      class ExamCountClass {
27        constructor(num2) {
28          this.number2 = num2;
29        }
30        showNum() {
31          console.log(`2. react_${this.number2}`);
32        }
33      }
34
35      var cnt2 = new ExamCountClass('2hundred');
36      cnt2.showNum();
37    }
38
39    render() {
40      return (
41        <h2>[ THIS IS Class ]</h2>
42      )
43    }
44  }
45
46  export default ClassPrototype;
```



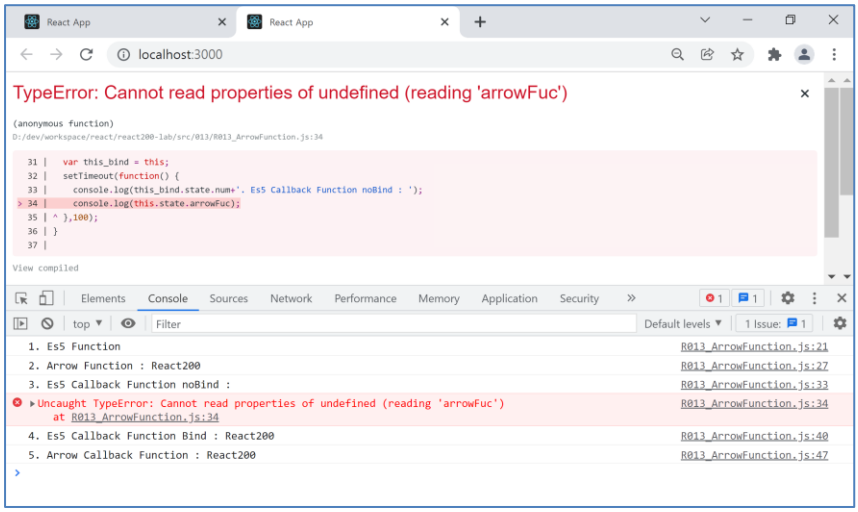
013 화살표 함수 사용하기

- 학습 내용: ES6에 추가된 화살표 함수의 사용 방법을 이해한다.
- 힌트 내용: 기존 ES5의 기본 함수 대비 ES6 화살표 함수의 장점과 사용법을 확인한다.
- ES6에서 등장한 화살표 함수는 ‘function’ 대신 ‘=>’ 문자열을 사용하며 ‘return’ 문자열을 생략할 수도 있다. 따라서 ES5 함수보다 간략하게 선언할 수 있다. 또한 화살표 함수에서는 콜백 함수에서 this를 bind해야 하는 문제도 발생하지 않는다.
- R013_ArrowFunction.js 파일
  - 34: 콜백 함수 내부에서 this는 window 객체이기 때문에 this로 state 변수에 접근하면 undefined 에러가 발생한다.
  - 40~41: 콜백 함수에 함수 밖의 this를 bind해주면, this를 컴포넌트로 사용할 수 있다.

[react200-lab/src/013/R013_ArrowFunction.js]

```
01  import React, { Component } from 'react';
02
03  class R013_ArrowFunction extends Component {
04
05    constructor(props) {
06      super(props);
07      this.state = {
08        arrowFuc: 'React200',
09        num: 3
10      };
11    }
12
13    componentDidMount() {
14      Function1(1);
15      this.Function2(1,1);
16      this.Function3(1,3);
17      this.Function4();
18      this.Function5(0,2,3);
19
20      function Function1(num1) {
21        return console.log(num1+'. Es5 Function');
22      }
23    }
24
25    Function2 = (num1, num2) => {
26      let num3 = num1 + num2;
27      console.log(num3+'. Arrow Function : '+this.state.arrowFuc);
28    }
29
30    Function3() {
31      var this_bind = this;
32      setTimeout(function() {
33        console.log(this_bind.state.num+'. Es5 Callback Function noBind : ');
34        console.log(this.state.arrowFuc);
35      },100);
36    }
37
38    Function4() {
39      setTimeout(function() {
40        console.log('4. Es5 Callback Function Bind : '+this.state.arrowFuc);
41      }.bind(this),100);
42    }
```

```
43
44    Function5 = (num1, num2, num3) => {
45      const num4 =  num1 + num2 + num3;
46      setTimeout(() => {
47        console.log(num4+'. Arrow Callback Function : '+this.state.arrowFuc);
48      }, 100);
49    }
50
51    render() {
52      return (
53        <h2>[ THIS IS ArrowFunction ]</h2>
54      )
55    }
56  }
57
58  export default R013_ArrowFunction;
```



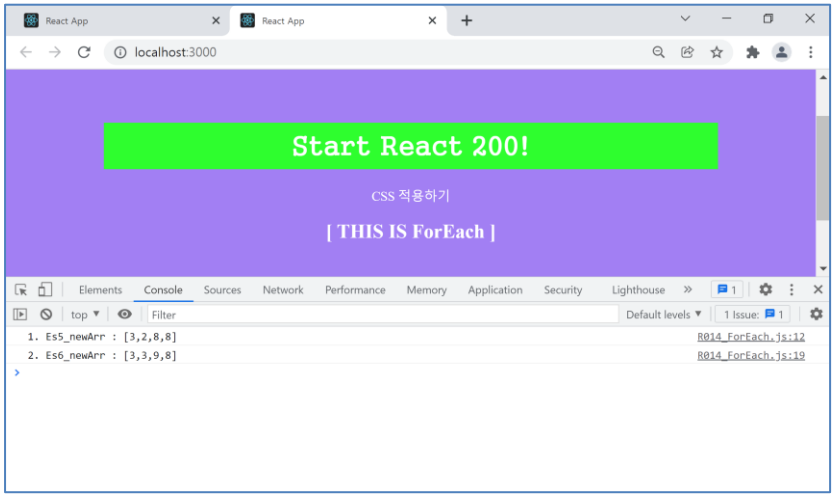
014 forEach() 함수 사용하기

- 학습 내용: 배열 함수인 forEach()의 사용 방법을 이해한다.
- 힌트 내용: For문 대비 forEach() 함수의 장점과 사용법을 확인한다.
- 배열 함수 forEach()는 for문에서 사용하던 순번과 배열의 크기 변수를 사용하지 않는다. 배열의 처음부터 마지막 순번까지 모두 작업하는 경우 forEach()문을 사용하는 것이 간편하다. 하지만 특정 순번에서만 배열 값을 사용하거나 변경해야 하는 상황이라면 for문을 사용해야 한다.
- R014_ForEach.js 파일

[react200-lab/src/014/R014_ForEach.js]

```
01  import React, { Component } from 'react';
02
03  class R014_ForEach extends Component {
04
05    componentDidMount() {
06      var Es5_Arr = [ 3, 2, 8, 8 ]
07      var Es5_newArr = []
```

```
08
09      for (var i = 0; i < Es5_Arr.length; i++) {
10        Es5_newArr.push(Es5_Arr[i])
11      }
12      console.log("1. Es5_newArr : ["+Es5_newArr+"]")
13
14      var Es6_Arr = [ 3, 3, 9, 8 ]
15      var Es6_newArr = []
16      Es6_Arr.forEach((result) => {
17        Es6_newArr.push(result)
18      })
19      console.log("2. Es6_newArr : ["+Es6_newArr+"]")
20    }
21
22    render() {
23      return (
24        <h2>[ THIS IS ForEach ]</h2>
25      )
26    }
27  }
28
29  export default R014_ForEach;
```



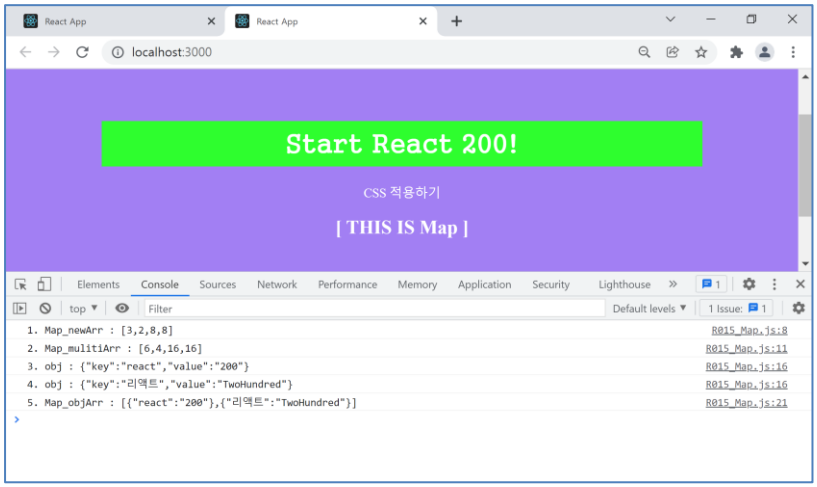
015 map() 함수 사용하기

- 학습 내용: 배열 함수인 map() 함수의 사용 방법을 이해한다.
- 힌트 내용: map() 함수의 특징과 사용법을 확인한다.
- 배열 함수 map()은 forEach()와 마찬가지로 for문에서 사용하던 순번과 배열의 크기 변수를 사용하지 않는다. 차이점은 map()은 forEach()와 달리 return을 사용해 반환 값을 받을 수 있다는 것이다.
- R015_Map.js 파일

[react200-lab/src/015/R015_Map.js]

```
01  import React, { Component } from 'react';
02
03  class R015_Map extends Component {
```

```
04
05    componentDidMount() {
06      var Map_Arr = [ 3, 2, 8, 8 ]
07      let Map_newArr = Map_Arr.map(x => x)
08      console.log("1. Map_newArr : ["+Map_newArr+"]")
09
10      let Map_mulitiArr = Map_Arr.map(x => x * 2)
11      console.log("2. Map_mulitiArr : ["+Map_mulitiArr+"]")
12
13      var ObjArray = [{key:'react', value:'200'},
14                      {key:'리액트', value:'TwoHundred'}];
15      let Map_objArr = ObjArray.map((obj, index) => {
16        console.log((index+3)+". obj : "+JSON.stringify(obj))
17        var Obj = {};
18        Obj[obj.key] = obj.value;
19        return Obj;
20      });
21      console.log("5. Map_objArr : "+JSON.stringify(Map_objArr))
22    }
23
24    render() {
25      return (
26        <h2>[ THIS IS Map ]</h2>
27      )
28    }
29  }
30
31  export default R015_Map;
```



016 jquery 사용하기

- 학습 내용: react에서 jquery 사용 방법을 이해한다.
- 힌트 내용: jquery를 임포트하는 방법과 기본 문법을 확인한다.
- jquery는 가장 인기 있는 자바스크립트 라이브러리다. 이벤트 처리, 애니메이션 등 자바스크립트의 기능을 간단하고 빠르게 구현하 수 있도록 지원해준다. jquery를 사용하기 위해 cmd 창을 열어 다음과 같이 npm으로 jquery를 설치한다.

```
D:\dev\workspace\react\react200-lab>npm install jquery
```

■ R016_Jquery.js 파일

[react200-lab/src/016/R016_Jquery.js]

```
01  import React, { Component } from 'react';
02  import $ from 'jquery';
03
04  class R016_Jquery extends Component {
05
06    input_alert = (e) => {
07      var input_val = $('#inputId').val();
08      alert(input_val);
09    }
10
11    render() {
12      return (
13        <div>
14          <h2>[ THIS IS Jquery ]</h2>
15          <input id="inputId" name="inputName"/>
16          <button id="buttonId" onClick={e => this.input_alert(e)}>Jquery Button</button>
17        </div>
18      )
19    }
20  }
21
22  export default R016_Jquery;
```