

# 리액트 기초

1. HTML 문서를 리액트로 변환하기
2. 컴포넌트로 데이터 전달하기
3. 컴포넌트 분리하기
4. Props와 State 활용
5. render 함수
6. 이벤트 적용

## ■ 일반 HTML 문서 (public/pure.html)

```
<html>
  <body>
    <header>
      <h1>WEB</h1>
      World Wide Web!
    </header>

    <nav>
      <ul>
        <li><a href="1.html">HTML</a></li>
        <li><a href="2.html">CSS</a></li>
        <li><a href="3.html">JavaScript</a></li>
      </ul>
    </nav>

    <article>
      <h2>HTML</h2>
      HTML is HyperText Markup Language.
    </article>
  </body>
</html>
```

### WEB

World Wide Web!

- [HTML](#)
- [CSS](#)
- [JavaScript](#)

### HTML

HTML is HyperText Markup Language.

## ■ React 변환

- <header> → Subject class (Component)

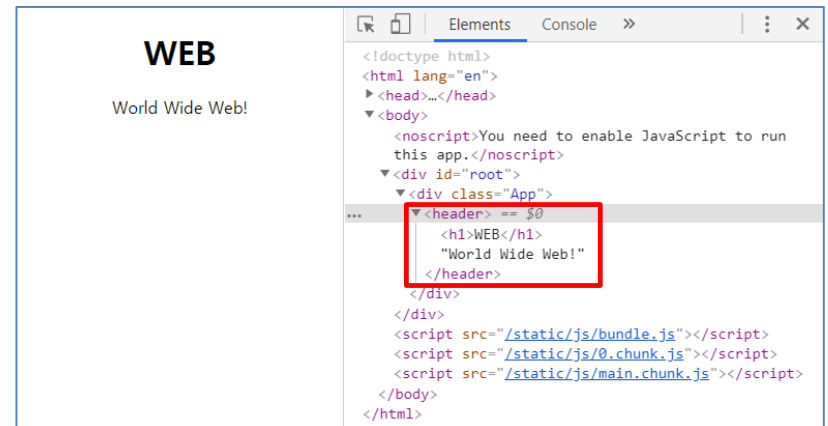
```
<header>
  <h1>WEB</h1>
  World Wide Web!
</header>
```



```
function Subject (){
  return (
    <header>
      <h1>WEB</h1>
      World Wide Web!
    </header>
  );
}
```

- Subject Component 적용

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



## ■ React 변환

- <nav> ➔ TOC(Table Of Contents) class (Component)

```
<nav>
  <ul>
    <li><a href="1.html">HTML</a></li>
    <li><a href="2.html">CSS</a></li>
    <li><a href="3.html">JavaScript</a></li>
  </ul>
</nav>
```

```
function TOC (){
  return (
    <nav>
      <ul>
        <li><a href="1.html">HTML</a></li>
        <li><a href="2.html">CSS</a></li>
        <li><a href="3.html">JavaScript</a></li>
      </ul>
    </nav>
  );
}
```

## ■ React 변환

### ● TOC Component 적용

```
function App () {  
  return (  
    <div className="App">  
      <Subject></Subject>  
      <TOC></TOC>  
    </div>  
  );  
}
```

**WEB**

World Wide Web!

- [HTML](#)
- [CSS](#)
- [JavaScript](#)

Elements Console >> X

```
<!doctype html>  
<html lang="en">  
  <head>...</head>  
  <body>  
    <noscript>You need to enable JavaScript to run  
    this app.</noscript>  
    <div id="root">  
      <div class="App">  
        <header>...</header>  
        <nav> == $0  
          <ul>...</ul>  
        </nav>  
      </div>  
    </div>  
    <script src="/static/js/bundle.js"></script>  
    <script src="/static/js/1.chunk.js"></script>  
    <script src="/static/js/main.chunk.js"></script>  
    <script src="/main.a4591ab...hot-update.js">  
  </script>  
</body>  
</html>
```

## ■ React 변환

- <article> ➔ Content class (Component)

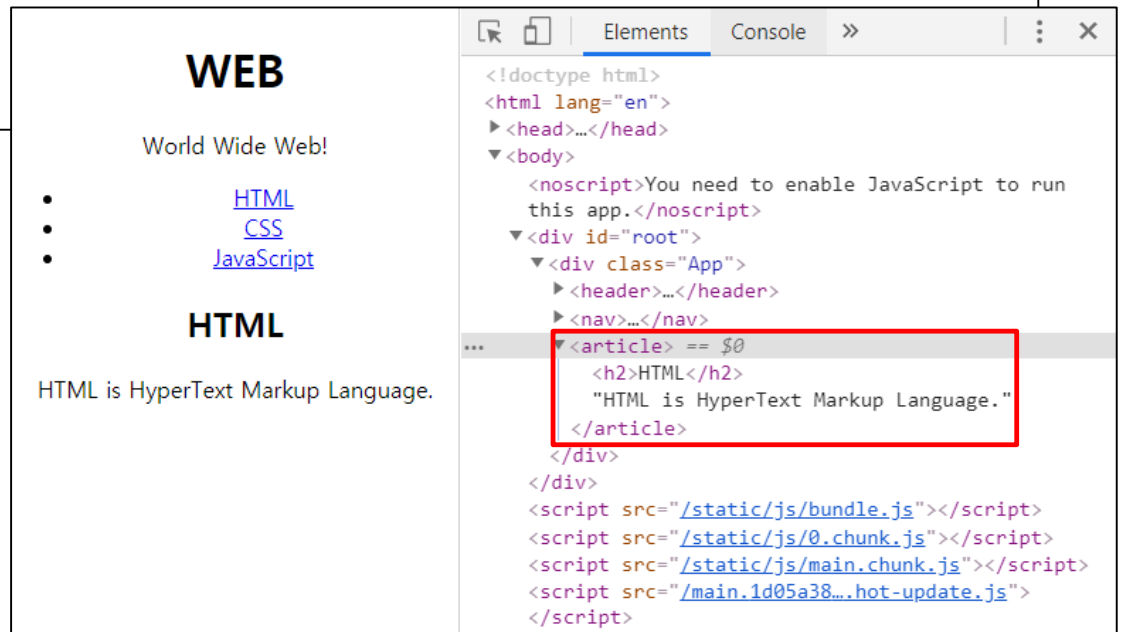
```
<article>
  <h2>HTML</h2>
  HTML is HyperText Markup Language.
</article>
```

```
function Content () {
  return (
    <article>
      <h2>HTML</h2>
      HTML is HyperText Markup Language.
    </article>
  );
}
```

## ■ React 변환

### ● Content Component 적용

```
function App () {  
  return (  
    <div className="App">  
      <Subject></Subject>  
      <TOC></TOC>  
      <Content></Content>  
    </div>  
  );  
}
```



**WEB**

World Wide Web!

- [HTML](#)
- [CSS](#)
- [JavaScript](#)

**HTML**

HTML is HyperText Markup Language.

Elements Console

```
<!doctype html>  
<html lang="en">  
  <head>...</head>  
  <body>  
    <noscript>You need to enable JavaScript to run  
    this app.</noscript>  
    <div id="root">  
      <div class="App">  
        <header>...</header>  
        <nav>...</nav>  
        <article> == $0  
          <h2>HTML</h2>  
          "HTML is HyperText Markup Language."  
        </article>  
      </div>  
    </div>  
    <script src="/static/js/bundle.js"></script>  
    <script src="/static/js/0.chunk.js"></script>  
    <script src="/static/js/main.chunk.js"></script>  
    <script src="/main.1d05a38...hot-update.js">  
    </script>
```

## ■ Component로 데이터 전달

- 컴포넌트 호출 시 attribute를 사용하여 표현하고자 하는 데이터 입력

```
function App (){  
  ...  
  <MyComponent data='Data!' number='2324'></MyComponent>  
  ...  
}
```

- 컴포넌트 내에서 props 라는 내부 객체를 통해 데이터 사용

```
const MyComponent = (props){  
  ...  
  <div>  
    <h1>{props.data}</h1>  
    <h1>{props.number}</h1>  
  </div>  
  ...  
}
```

**Data!**

**2324**

```
console.log(props)
```

```
▼ {data: "Data!", number: "2324"} ⓘ  
  data: "Data!"  
  number: "2324"  
  ► __proto__: Object
```



## ■ Component로 데이터 전달

### ● Subject

```
<Subject title='WEB' sub='World Wide Web! '></Subject>
```



```
function Subject(props){  
  return (  
    <header>  
      <h1>{this.props.title}</h1>  
      {this.props.sub}  
    </header>  
  );  
}
```

## ■ Component로 데이터 전달

### ● Content

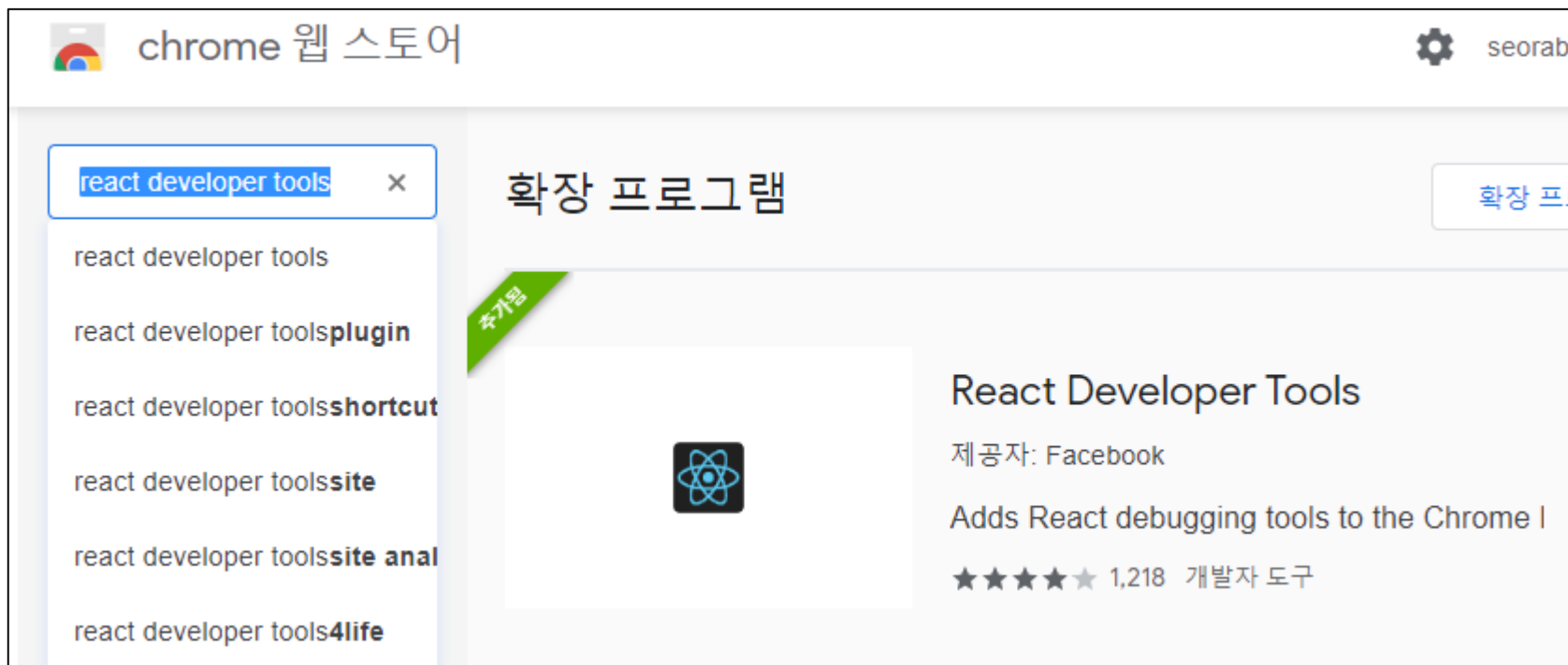
```
<Content title='HTML'  
        desc='HTML is HyperText Markup Language.'></Content>
```



```
function Content(props){  
  return (  
    <article>  
      <h2>{this.props.title}</h2>  
      {this.props.desc}  
    </article>  
  );  
}
```

## ■ React Developer Tools

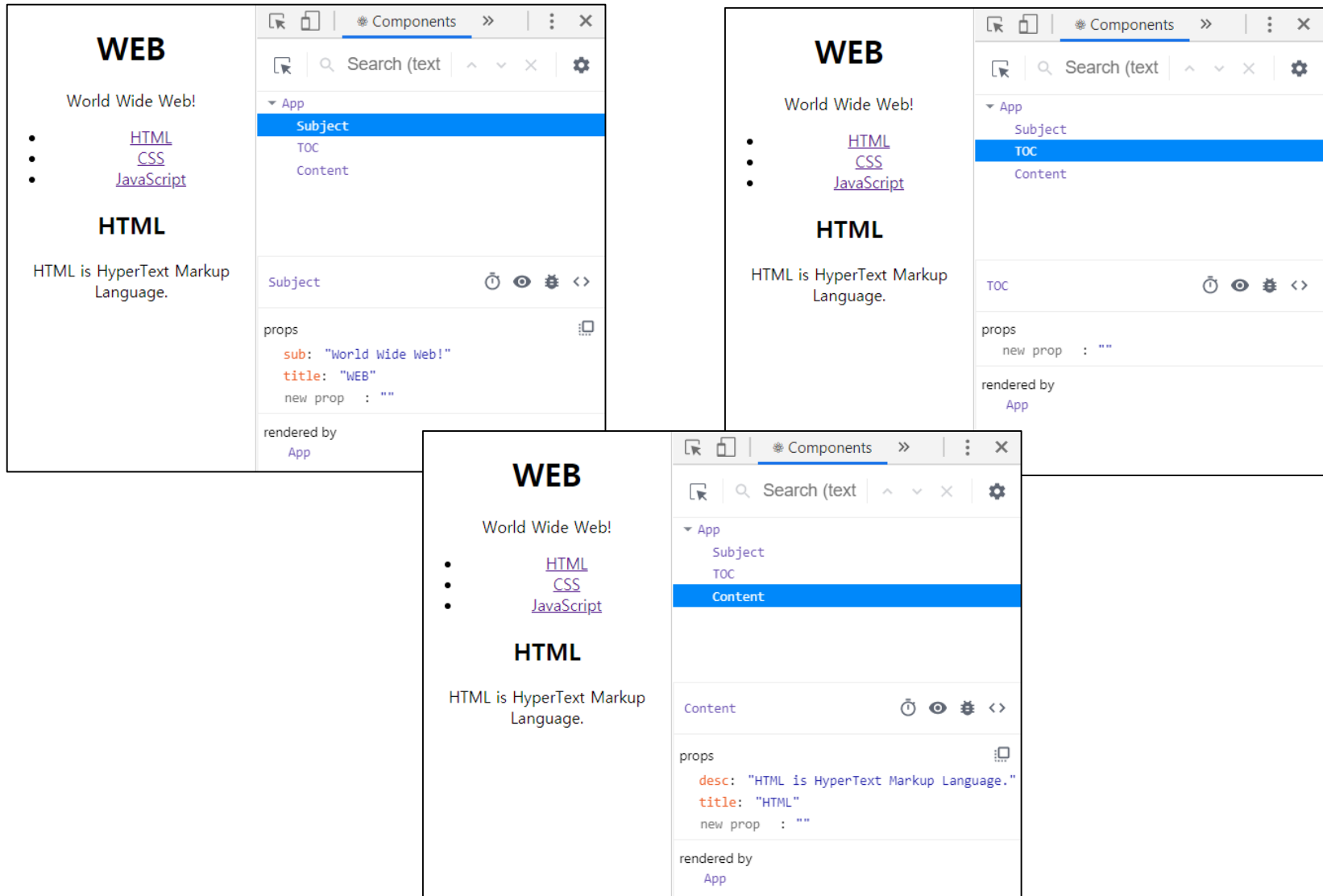
### ● chrome 웹 스토어 검색



## ■ React Developer Tools

● chrome 개발자 도구(F12 / 마우스 우클릭 → 검사)

- Components Tab



## ■ Component 분리 - Subject

### ● src/Subject.js 파일 생성

```
import React from 'react';

function Subject(props){
  return (
    <header>
      <h1>{props.title}</h1>
      {props.sub}
    </header>
  );
}

export default Subject;
```

### ● App.js 내용 수정

- Subject.js 가져오기

```
import Subject from './Subject';
```

## ■ Component 분리 - TOC

### ● src/TOC.js 파일 생성

```
import React from 'react';

function TOC(){
  return (
    <nav>
      <ul>
        <li><a href="1.html">HTML</a></li>
        <li><a href="2.html">CSS</a></li>
        <li><a href="3.html">JavaScript</a></li>
      </ul>
    </nav>
  );
}

export default TOC;
```

### ● App.js 내용 수정

- TOC.js 가져오기

```
import TOC from './TOC';
```

## ■ Component 분리 - Content

### ● src/Content.js 파일 생성

```
import React from 'react';

function Content(props){
  return (
    <article>
      <h2>{this.props.title}</h2>
      {this.props.desc}
    </article>
  );
}

export default Content;
```

### ● App.js 내용 수정

- Content.js 가져오기

```
import Content from './Content';
```

## ■ 연습문제

### ● [sample.html](#)의 HTML 코드를 React 프로젝트에서 동작할 수 있도록 변경하기

1. react-sample 프로젝트 생성
2. Header / Section / Article / Footer 컴포넌트 작성
3. App 클래스의 render 함수에서 위의 4개 컴포넌트 출력

```
<body>
  <header><h1>야구게임 방식</h1></header>
  <hr>
  <section>
    <h4>사용되는 숫자는 0에서 9까지 서로 다른 숫자이다.</h4>
    ...
  </section>
  <article>
    <table border="1">... </table>
  </article>
  <footer>
    <h5>1. 830 - 들어맞는 숫자가 아예 없으므로 아웃. ...</h5>
    ...
  </footer>
</body>
```

#### 야구게임 방식

사용되는 숫자는 0에서 9까지 서로 다른 숫자이다.

숫자는 맞지만 위치가 틀렸을 때는 볼.

숫자와 위치가 전부 맞으면 스트라이크.

숫자와 위치가 전부 틀리면 아웃. "틀렸다"는 게 중요하다.

물론 무엇이 볼이고 스트라이크인지는 알려주지 않는다.

횟수	숫자	판정
1	830	아웃
2	659	0S 1B
3	264	1S 1B
4	126	1S 2B
5	216	3S 0B

1. 830 - 들어맞는 숫자가 아예 없으므로 아웃. 이때부터 0, 3, 8이 후보에서 빠지므로 남은 숫자는 1, 2, 4, 5, 6, 7, 9다.

2. 659 - 6이 있지만 위치가 다르므로 1볼. 게임 상으로는 어떤 숫자가 맞는지 모르기 때문에 가장 난감하다.

3. 264 - 2가 있고 위치가 맞으며, 6이 있지만 위치가 다르므로 1스트라이크 1볼.

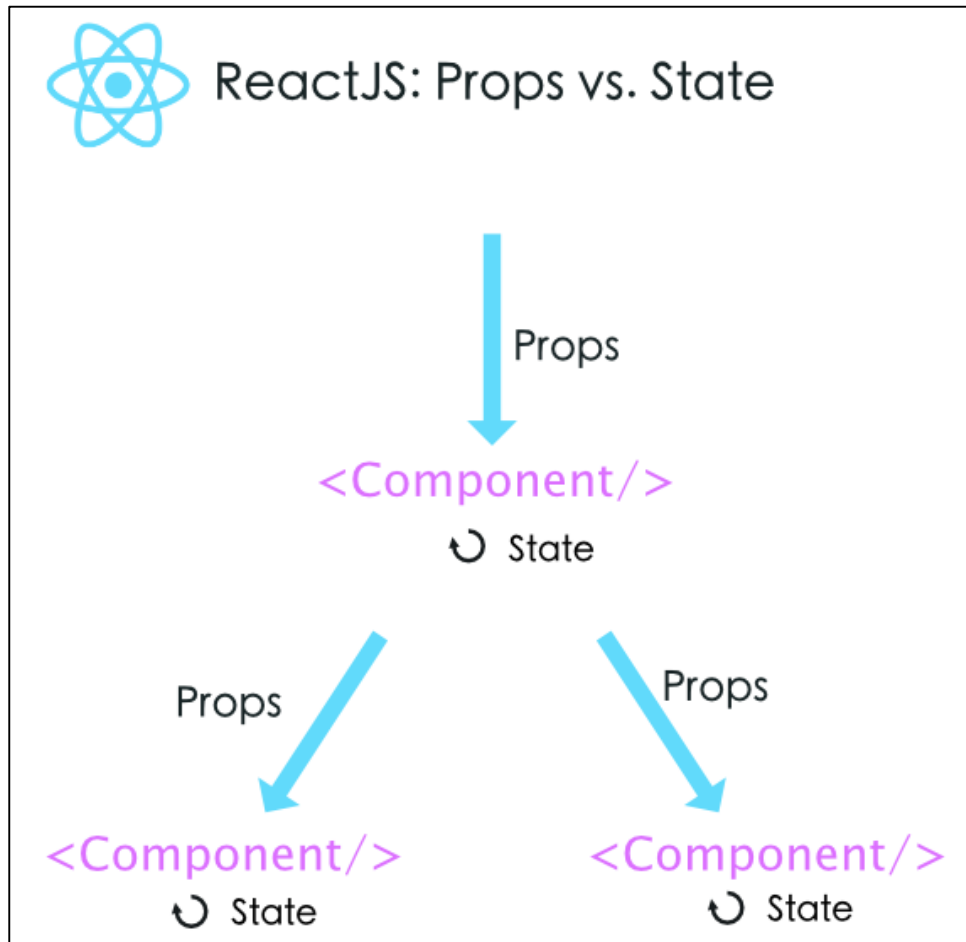
4. 126 - 숫자는 전부 맞지만 위치는 6만 맞고 나머지 둘은 다르므로 1스트라이크 2볼.

5. 216 - 전부 맞으므로 승리.



## ■ Props / State

### ● Component의 상태와 데이터



출처 : <https://velog.io/@kyusung/리액트-교과서-React-컴포넌트와-상태-객체>

## ■ Props / State

- React의 컴포넌트는 상태 값을 이용해서 UI를 표현
- 컴포넌트 간의 단방향 데이터 전송으로 props를 사용하고  
컴포넌트 내부에서 사용하는 state가 존재
- props는 부모 컴포넌트에서 자식 컴포넌트로 데이터를 전달할 때 사용
  - 데이터 변경 불가 (readonly)
- state는 컴포넌트 내에서 관리하는 상태 값으로 데이터를 유동적으로  
다룰 때 사용
  - 데이터 변경 가능
  - 변경된 상태를 컴포넌트에게 알려주기 위해서 setState 함수 사용

## ■ Props 사용

- 부모 컴포넌트 (App.js) → 자식 컴포넌트 (Child.js)

```
class App extends Component {  
  render() {  
    return (  
      <div className="App">  
        <Child a='1' b='2'></Child>  
      </div>  
    );  
  }  
}
```

```
class Child extends Component {  
  render() {  
    return (  
      <div>  
        <p>{this.props.a}</p>  
        <p>{this.props.b}</p>  
      </div>  
    );  
  }  
}
```

## ■ State 활용

- 부모 컴포넌트 (App.js) ➔ 자식 컴포넌트 (Child.js)

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      child: {a: '1', b: '2'}
    }
  }

  render() {
    return (
      <div className="App">
        <Child a={this.state.child.a} b={this.state.child.b}>
        </Child>
      </div>
    );
  }
}
```

## ■ Props / State 적용

### ● Subject (title / sub)

- App.js

```
import ...

class App extends Component {
  constructor(props) {
    super(props);
    this.state = {
      subject: { title: 'WEB', sub: 'World Wide Web!' }
    }
  }
  render() {
    return (
      <div className="App">
        Hello, React
        <Subject title={ this.state.subject.title }
                  sub={ this.state.subject.sub }></Subject>
        ...
      </div>
    )
  }
}
```

## ■ Props / State 적용

### ● Subject (title / sub)

- Subject.js

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

class Subject extends Component {
  render() {
    return (
      <header>
        <h1>{this.props.title}</h1>
        {this.props.sub}
      </header>
    );
  }
}

export default Subject;
```

**WEB**

World Wide Web!

## ■ Props / State 적용

### ● TOC (contents)

- App.js

```
...
this.state = {
  ...
  contents: [
    {id: 1, title: 'HTML'},
    {id: 2, title: 'CSS'},
    {id: 3, title: 'JavaScript'}
  ]
}
}
render() {
  return (
    <div className="App">
      ...
      <TOC contents={ this.state.contents }></TOC>
      ...
    </div>
  )
}
```

## ■ Props / State 적용

### ● TOC (contents)

- TOC.js

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

class TOC extends Component {
  render() {
    return (
      <nav>
        <ul>
          <li>
            <a href={ this.props.contents[0].id + '.html'}>
              { this.props.contents[0].title }
            </a>
          </li>
          ...
        </ul>
      </nav>
    )
  }
}

export default TOC;
```

- [HTML](#)
- [CSS](#)
- [JavaScript](#)



## ■ Props / State 적용

### ● Content (title / desc)

- App.js

```
...
this.state = {
  ...
  contents: [
    {id: 1, title: 'HTML', desc: 'HTML is for information.'},
    {id: 2, title: 'CSS', desc: 'CSS is for design.'},
    {id: 3, title: 'JavaScript',
      desc: 'JavaScript is for interactive.'}
  ]
}
}
render() {
  return (
    <div className="App">
      ...
      <TOC contents={ this.state.contents }></TOC>
      ...
    </div>
  )
  ...
}
```

## ■ Props / State 적용

### ● Content (title / desc)

- Content.js

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

class Content extends Component {
  render() {
    return (
      <article>
        <h2>{this.props.title}</h2>
        {this.props.desc}
      </article>
    )
  }
}

export default Content;
```

## HTML

HTML is for information.

## ■ Props / State 적용

### ● Content (title / desc)

- App.js

```
constructor(props) {  
  ...  
  contents: [  
    {id: 1, title: 'HTML',  
      desc: 'HTML is for information.'},  
    {id: 2, title: 'CSS', desc: 'CSS is for design.'},  
    {id: 3, title: 'JavaScript',  
      desc: 'JavaScript is for interactive.'}  
  ]  
  ...  
  render() {  
    ...  
    <Content title={this.state.contents[0].title}  
      desc={this.state.contents[0].desc}></Content>  
    ...  
  }  
}
```

- Content.js

```
<article>  
  <h1>{this.props.title}</h1>  
  {this.props.desc}  
</article>
```

## ■ 연습문제

- react-sample 프로젝트의 코드 중 Header와 Footer 컴포넌트에 props와 state 기능 적용하기

1. 각 컴포넌트에 출력되는 내용을 부모 컴포넌트인 App에서 전달
2. Footer의 1 ~ 5 항목은 배열 객체를 사용하여 전달 및 출력

### 야구게임 설명

사용되는 숫자는 0에서 9까지 서로 다른 숫자이다.

숫자는 맞지만 위치가 틀렸을 때는 볼.

숫자와 위치가 전부 맞으면 스트라이크.

숫자와 위치가 전부 틀리면 아웃. "틀렸다"는 게 중요하다.

물론 무엇이 볼이고 스트라이크인지는 알려주지 않는다.

횟수	숫자	판정
1	8 3 0	아웃
2	6 5 9	0S 1B
3	2 6 4	1S 1B
4	1 2 6	1S 2B
5	2 1 6	3S 0B

1. 830 - 들어맞는 숫자가 아예 없으므로 아웃. 이때부터 0, 3, 8이 후보에서 빠지므로 남은 숫자는 1, 2, 4, 5, 6, 7, 9다.

2. 659 - 6이 있지만 위치가 다르므로 1볼. 게임 상으로는 어떤 숫자가 맞는지 모르기 때문에 가장 난감하다.

3. 264 - 2가 있고 위치가 맞으며, 6이 있지만 위치가 다르므로 1스트라이크 1볼.

4. 126 - 숫자는 전부 맞지만 위치는 6만 맞고 나머지 둘은 다르므로 1스트라이크 2볼.

5. 216 - 전부 맞으므로 승리.

## ■ render()

- state의 값이 바뀌게 되면 해당 컴포넌트의 render 함수 재호출
- state의 값을 직접 바꾸면 React가 감지하지 못하므로 setState 함수 이용
- 특정 값의 상태(조건)에 따라 다른 내용이 보여지도록 수정
  - mode 라는 값을 이용하여 처음 페이지에 들어온 상태인지 목록을 선택한 상태인지에 따라 Content 컴포넌트에 보여줄 내용 변경
  - App.js → constructor()

```
this.state = {  
  ...  
  mode: 'welcome',  
  welcome: {title: 'welcome', desc: 'Hello, React'}  
}
```

## ■ render()

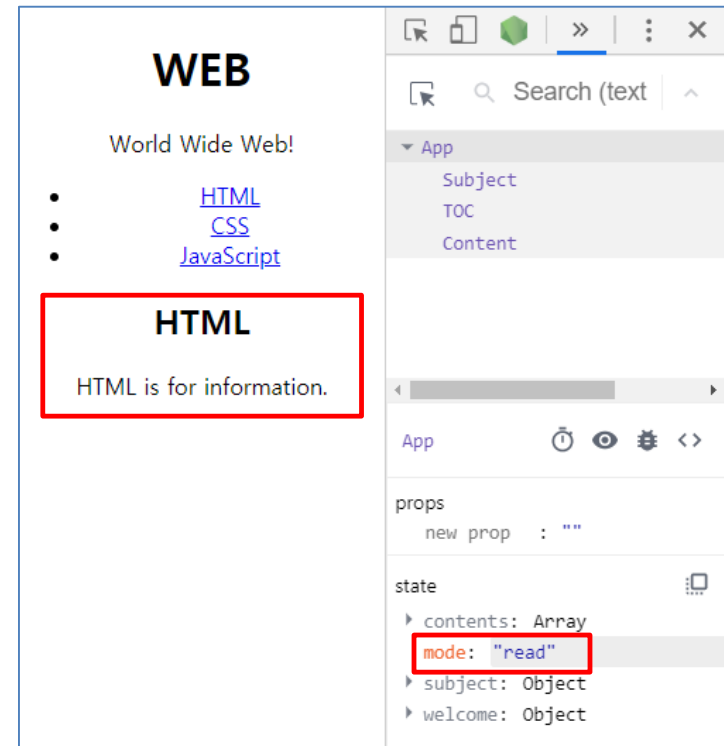
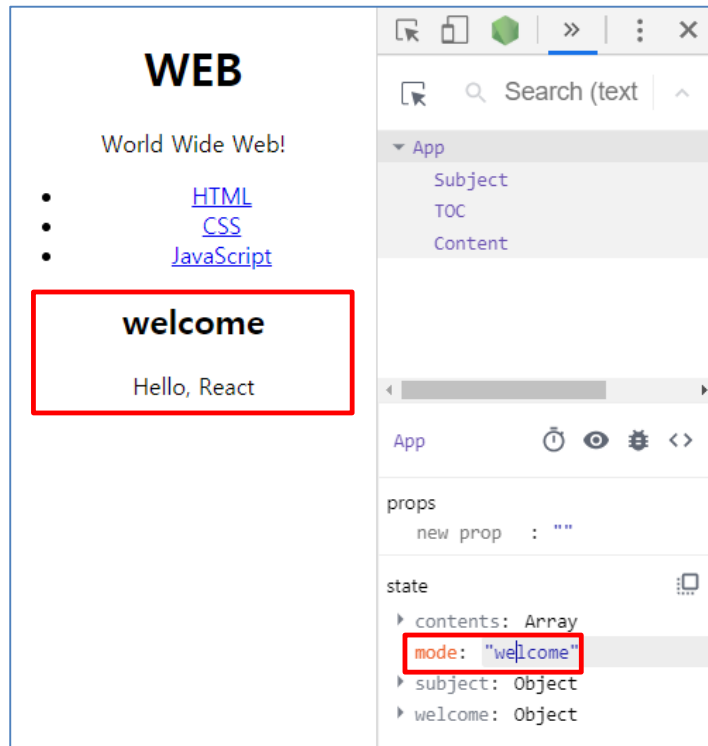
- state의 값이 바뀌게 되면 해당 컴포넌트의 render 함수 재호출
- state의 값을 직접 바꾸면 React가 감지하지 못하므로 setState 함수 이용
- 특정 값의 상태(조건)에 따라 다른 내용이 보여지도록 수정
  - mode 라는 값을 이용하여 처음 페이지에 들어온 상태인지 목록을 선택한 상태인지에 따라 Content 컴포넌트에 보여줄 내용 변경
  - App.js ➔ render()

```
render() {  
  let title, desc;  
  if(this.state.mode === 'welcome') {  
    title = this.state.welcome.title;  
    desc = this.state.welcome.desc;  
  } else if(this.state.mode === 'read') {  
    title = this.state.contents[0].title;  
    desc = this.state.contents[0].desc;  
  }  
  return (  
    <div className="App">  
      ...  
      <Content title={title} desc={desc}></Content>  
    </div>  
  );  
}
```

## ■ render()

● chrome 개발자 도구(F12 / 마우스 우클릭 → 검사)

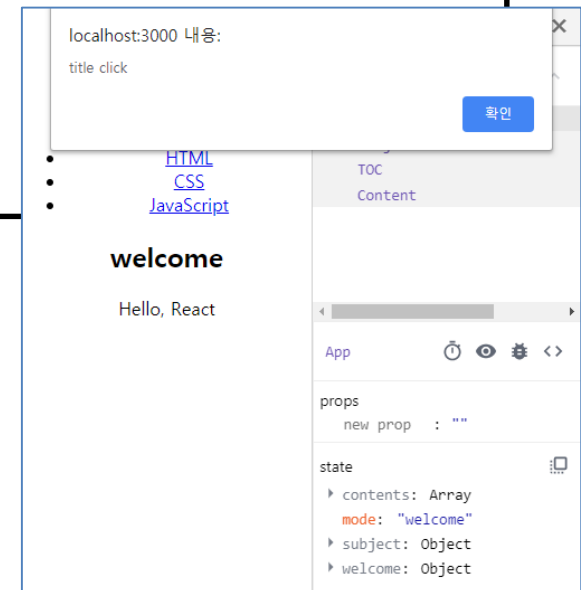
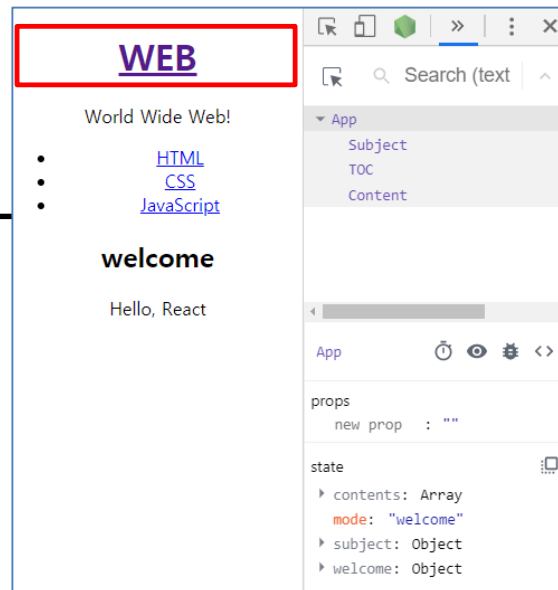
- Components Tab → mode 값 변경 (read)



## ■ Event

- 마우스 클릭, 키보드 키 입력 등의 행위가 발생되었을 때 동작 정의
- 행위(Listener - onClick, onKeyDown 등)와 동작(function)에 대해 작성

```
class Subject extends Component {  
  render() {  
    return (  
      <header>  
        <h1><a href="/" onClick={function() {  
          alert('title click');  
        }}>{this.props.title}</a></h1>  
        {this.props.sub}  
      </header>  
    );  
  }  
}
```

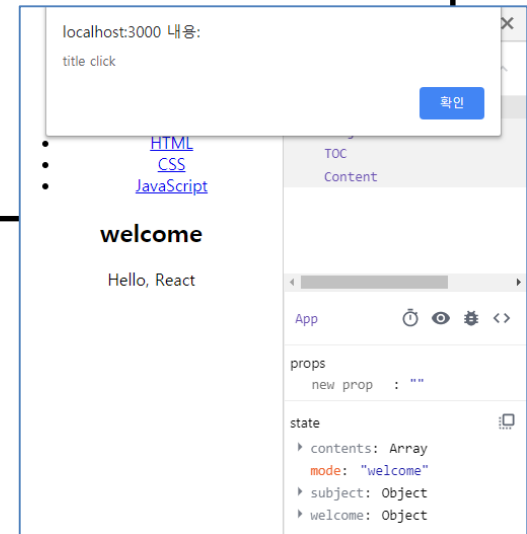
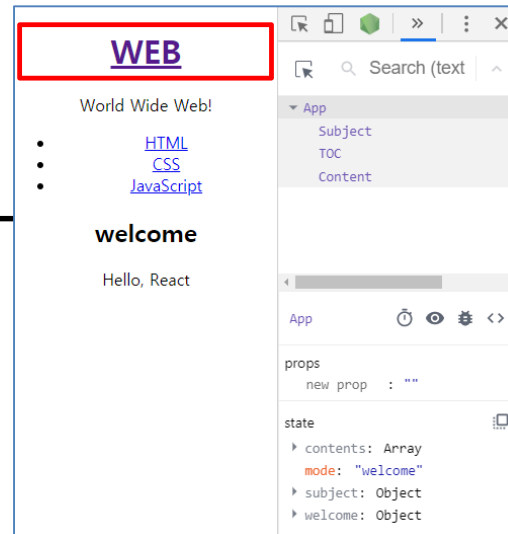




## ■ Event

- HTML 요소의 기본 동작을 막기 위해서 event 객체 사용
- `preventDefault()`

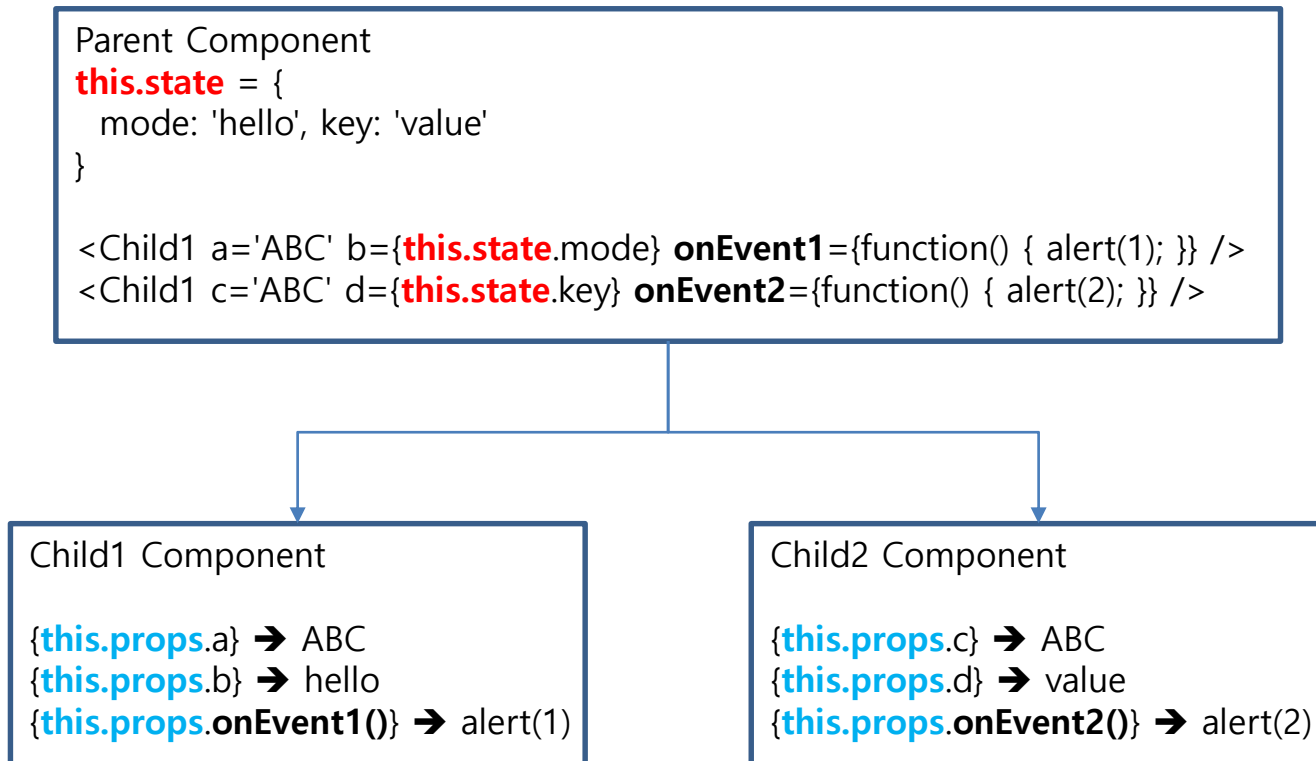
```
class Subject extends Component {  
  render() {  
    return (  
      <header>  
        <h1><a href="/" onClick={function(e) {  
          alert('title click');  
          e.preventDefault();  
        }}>{this.props.title}</a></h1>  
        {this.props.sub}  
      </header>  
    );  
  }  
}
```



## ■ Event

### ● 하위 컴포넌트에서 state의 값 변경

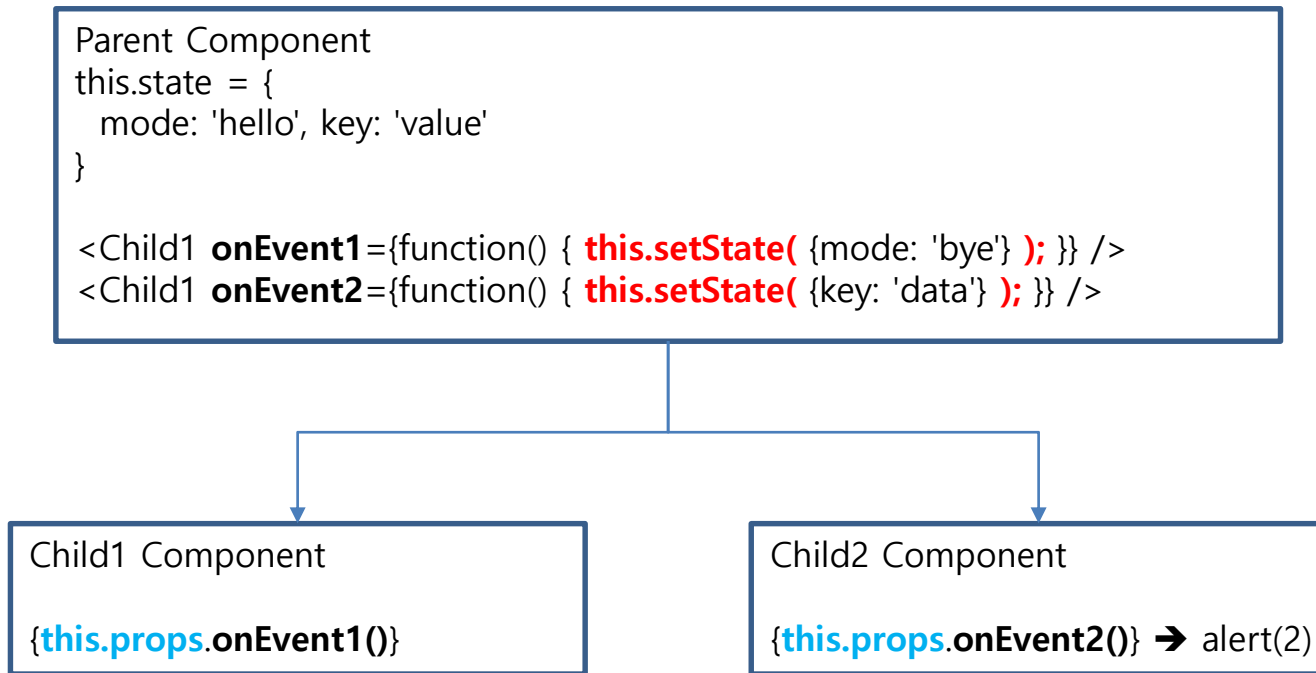
- 하위 컴포넌트에서 상위 컴포넌트의 state 변경 불가
- Parent의 state를 변경하기 위해서 임의의 이벤트 생성 후 호출



## ■ Event

### ● 하위 컴포넌트에서 state의 값 변경

- state의 값을 직접 변경하면 React가 감지할 수 없음
- setState() 를 이용하여 변경 (state 변경 시 각 컴포넌트의 render() 호출)



## ■ Event

- 함수 내에서 this에 저장되어 있는 데이터를 활용하기 위해서 bind 필요
- bind 함수 사용, arrow 함수 사용 등

```
const a = function() {  
  console.log(this);  
}  
a();
```

undefined App.js:31

```
const b = function() {  
  console.log(this);  
}.bind(this);  
b();
```

▶ App App.js:35

```
const c = () => {  
  console.log(this);  
};  
c();
```

▶ App App.js:39

## ■ Event - Subject

- App.js 의 state mode 값을 'read' 로 변경
- Subject.js 의 title 클릭 시 mode를 'welcome' 으로 변경
  - App.js ➔ constructor()

```
this.state = {  
  ...  
  mode: 'read',  
  ...  
}
```

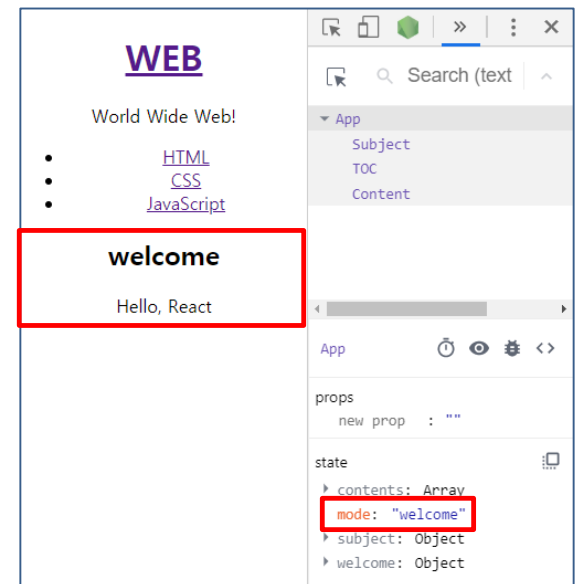
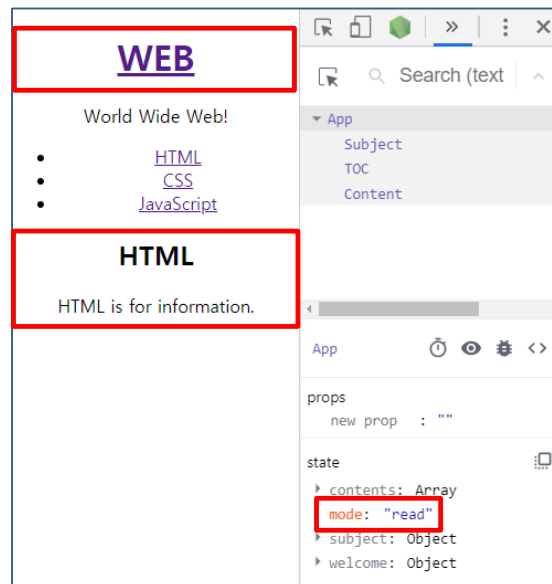
- App.js ➔ render()

```
<Subject title={this.state.subject.title}  
  sub={this.state.subject.sub}  
  onChangePage={function() {  
    this.setState({mode: 'welcome'})  
  }.bind(this)}></Subject>
```

## ■ Event - Subject

- App.js 의 state mode 값을 'read' 로 변경
  - Subject.js 의 title 클릭 시 mode를 'welcome' 으로 변경
- Subject.js

```
<h1>
  <a href="/" onClick={function(e) {
    this.props.onChangePage();
    e.preventDefault();
  }.bind(this)}>{this.props.title}</a>
</h1>
```



## ■ Event - TOC

- TOC.js 의 목록 클릭 시 mode를 'read' 로 변경, 선택 번호 저장

- App.js ➔ constructor()

```
this.state = {  
  ...  
  selected_content_id: 1  
}
```

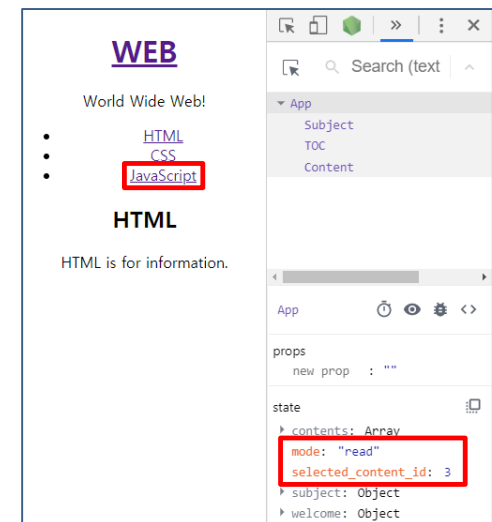
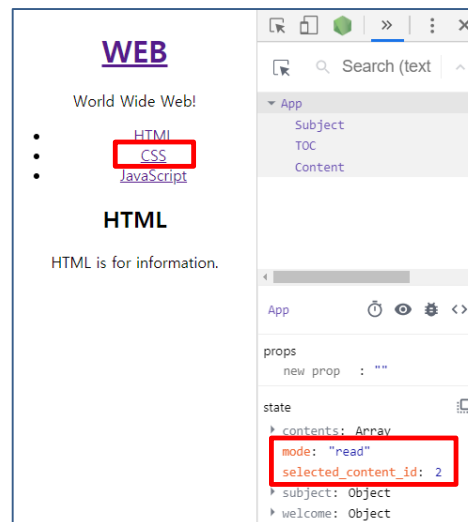
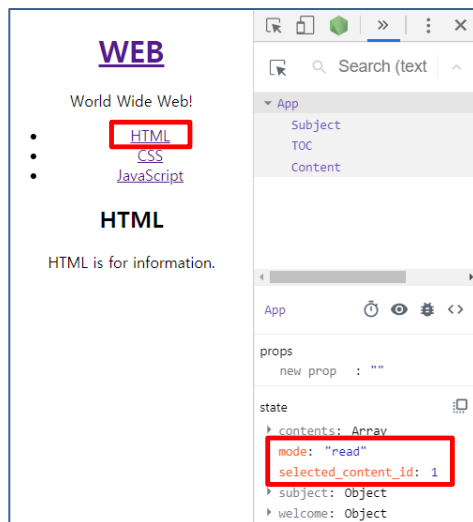
- App.js ➔ render()

```
<TOC contents={this.state.contents}  
  onSelect={function(id) {  
    this.setState({  
      mode: 'read',  
      selected_content_id: id  
    })  
  }}  
  >.bind(this)></TOC>
```

## ■ Event - TOC

- TOC.js 의 목록 클릭 시 mode를 'read' 로 변경, 선택 번호 저장
  - TOC.js

```
<li>
  <a href={this.props.contents[0].id}
    onClick={function(e) {
      e.preventDefault();
      this.props.onSelect(this.props.contents[0].id)
    }.bind(this)}>{this.props.contents[0].title}</a>
</li>
...
...
```



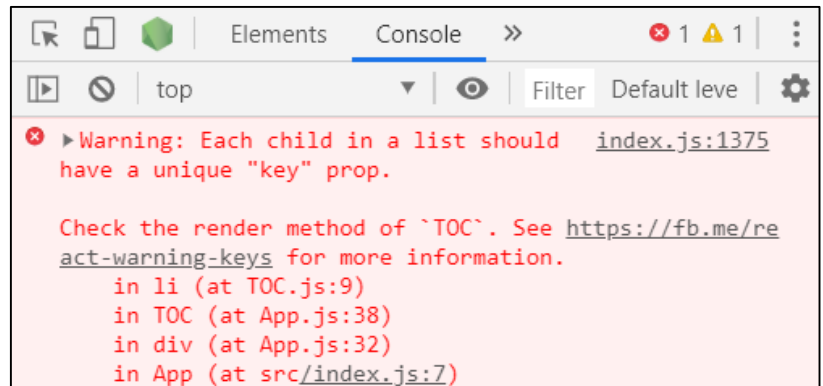


## ■ Event - TOC

### ● TOC.js 코드 수정 (Refactoring)

- 반복문 사용

```
render() {  
  const list = [];  
  for(let i = 0; i < this.props.contents.length; i++) {  
    const content = this.props.contents[i];  
    list.push(  
      <li>  
        <a href={'/content/' + content.id}  
          onClick={function(e) {  
            e.preventDefault();  
            this.props.onSelect(content.id)  
          }.bind(this)}>{content.title}</a>  
      </li>  
    )  
  }  
  return (  
    <nav>  
      <ul>{list}</ul>  
    </nav>  
  )  
}
```

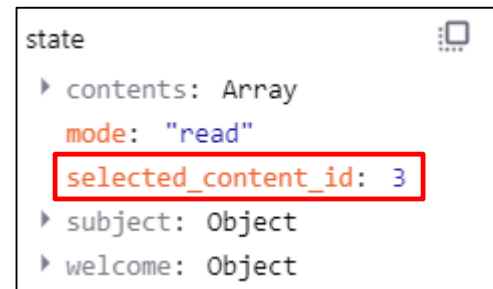


## ■ Event - TOC

### ● TOC.js 코드 수정 (Refactoring)

- 반복문을 이용하여 HTML 요소 생성 시 key 속성 추가

```
render() {  
  const list = [];  
  for(let i = 0; i < this.props.contents.length; i++) {  
    const content = this.props.contents[i];  
    list.push(  
      <li key={content.id}>  
        <a href={'/content/' + content.id}  
          onClick={function(e) {  
            e.preventDefault();  
            this.props.onSelect(content.id)  
          }.bind(this)}>{content.title}</a>  
      </li>  
    )  
  }  
  return (  
    <nav>  
      <ul>{list}</ul>  
    </nav>  
  )  
}
```



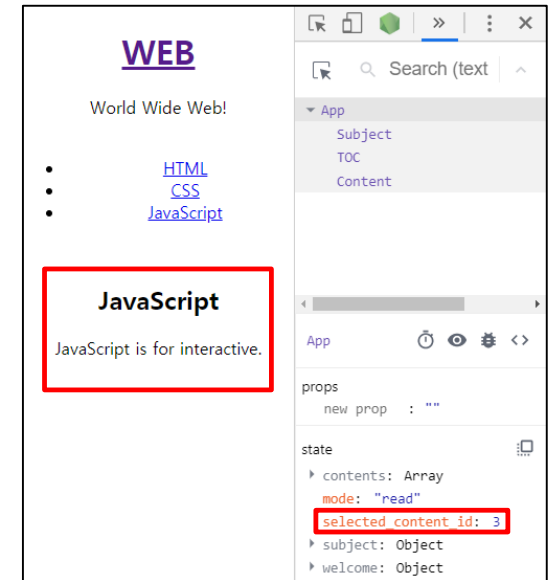
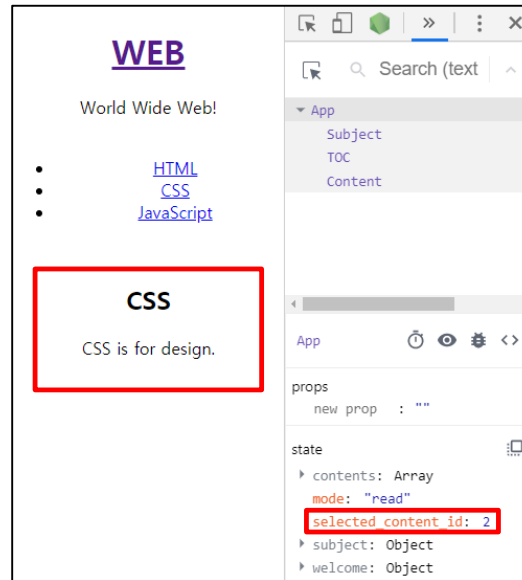
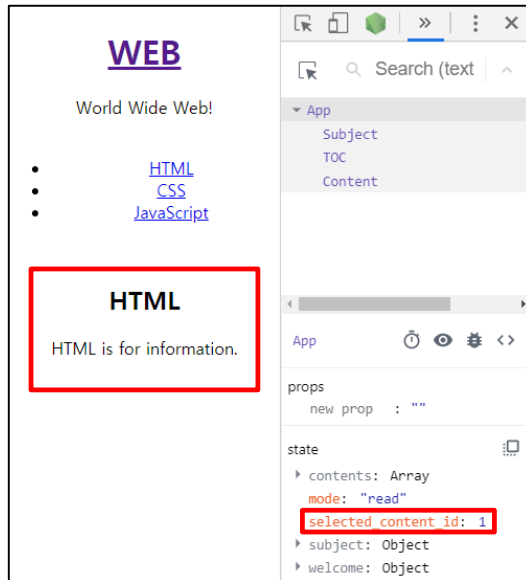
## ■ Event - Content

- selected\_content\_id의 값으로 Content에 보여질 데이터 가져오기
  - findContentById 함수 작성

```
constructor(props) { ... }  
findContentById(id) {  
  let content;  
  for(let i = 0; i < this.state.contents.length; i++) {  
    if(id === this.state.contents[i].id) {  
      content = this.state.contents[i];  
      break;  
    }  
  }  
  return content;  
}  
render() {  
  let title, desc;  
  ...  
} else if(this.state.mode === 'read') {  
  const content =  
    this.findContentById(this.state.selected_content_id);  
  title = content.title;  
  desc = content.desc;  
}
```

## ■ Event - Content

- selected\_content\_id의 값으로 Content에 보여질 데이터 가져오기
  - findContentById 함수 작성



## ■ 연습문제

- react-sample 프로젝트의 코드 중 Footer 컴포넌트에서 반복문 또는 배열 API를 사용하여 항목들을 출력하고 각 항목에 클릭 이벤트 설정하기
  1. 반복문 사용 시 key 속성 사용
  2. 클릭된 항목(문장)은 밑줄과 배경색 변경

밑줄, 배경색 변경 코드

```
(e) => {  
  e.target.style.backgroundColor = 'beige';  
  e.target.style.textDecoration = 'underline';  
}
```

### 실행 결과

1. 830 - 들어맞는 숫자가 아예 없으므로 아웃. 이때부터 0, 3, 8이 후보에서 빠지므로 남은 숫자는 1, 2, 4, 5, 6, 7, 9다.
2. 659 - 6이 있지만 위치가 다르므로 1볼. 게임 상으로는 어떤 숫자가 맞는지 모르기 때문에 가장 난감하다.
3. 264 - 2가 있고 위치가 맞으며, 6이 있지만 위치가 다르므로 1스트라이크 1볼.
4. 126 - 숫자는 전부 맞지만 위치는 6만 맞고 나머지 둘은 다르므로 1스트라이크 2볼.
5. 216 - 전부 맞으므로 승리.