

Var

변수를 선언하고, 선택적으로 초기화

```
Var x = 1;
if (x === 1) {
    console.log(x); //2
}
var x = 2;
console.log(x); //2
```

let 블록 스코프의 범위를 가리는 지역 변수 선언, 선택과 동시에

상위의 값으로 초기화

var은 블록을 고려하지 않고 전역 변수(또는 전역스코프)

마지막에 접근할 수 있는 변수 선언

```
let x = 1;
if (x === 1) {
    console.log(x);
}
let x = 2;
console.log(x); //2
}
```

Const

const 선언은 블록 범위의 상수를 선언

상위 값은 재할당 X 다시 선언 X

```
CONST number = 42; console.log(number); //42
try {
    number = 99;
} catch (err) { //Type error
    console.log(err);
}
```

Template Literal string 자료형을 조합할 때 + 연산자 사용

const hello = "Hello";

const name = "UnNamed";

const height = 175;

const text = hello + "!" + name + "He" + height + "cm";

console.log(text);

// `` (backtick) 앞의 변수 넣을 때 \${{place holder}} 사용

const newText = \${hello}! \${name} \${height};
console.log(Text);

Destructuring(전분해 할당)

구조 분해 할당 구문은 배열이나 객체의 속성을 편리하게

그 값을 개별 변수로 할당 가능하게 하는 Javascript 문장

```
const lab = {
    name: 'web',
    room: 205
};
const people = ['이상', '정우'];
var name = lab.name;
var room = lab.room;
var minsoo = people[0];
var chulsoo = people[1];
Var {name, room} = lab;
console.log(name); //web
const [minsoo, chulsoo] = people;
```

Array.prototype

• map, forEach 외에도 배열 자체에 사용

Array.prototype.forEach() 배열의 callback을 배열에 적용

각 요소마다 해당하는 순서로 실행

```
product.forEach(function (product, index) {
    console.log(index + 1) + '번재 품목';
    console.log(product);
});
product.map(function (product, index) {
    console.log(index) + '번재 품목';
    console.log(product);
});
```

Array.prototype.map() map()은 callback 함수를 갖지만

모든 예상한 순서로 실행되며 결과를 넣어

함수의 반환값으로 새로운 배열 생성

Lambda Function(익명함수)

function을 대체하는 Lambda Function (()=>) 형식 사용

```
const getName = () => {
    return '정우';
}
// lambda의 return은 생략 가능
const getName = (name) => `${name}입니다`
```

console.log((index)+1) + '번재 품목';

console.log(product);});

var productNames = product.map((productIndex) => {

return product.name;}); map()은 name을 배열로

console.log(productNames); ['정우', '정우'] 뿐만 아니라

상황에 따라

if-else 대체형태

condition ? exprTrue : exprFalse

(language == 'javascript') ? console.log('이된다') : console.log('아니다')

리액트 라이브러리 (React Library) → Node.js 파일

const isJavascript = language == 'javascript' ? true : false;

HTML 문서나 웹에서 실행되는 파일

https://unpkg.com/react@17/umd/react.development.js

/react-dom@17/umd/react-dom.development.js

/babel-stable@6/babel.min.js

루트 컨테이너 설정하기

```
<script type="text/babel">  
const component = <h1>리액트개발 </h1>  
const container = document.getElementById('root')  
//리액트 사용할 수 있는 코드  
ReactDOM.render(component, container)  
</script>
```

JSX(자바스크립트 확장문법) 웹 브라우저는 이러한 코드를 알지 못함

브라우저 JSX 코드를 알고 자바스크립트 문법으로 변환해줌

```
class component = /*#__PURE__*/ React.createElement('h1', null, "Hello B...");
```

Visual Studio에서 실행시켜 JSX 파일로 들어가서 script를 실행해

보면 브라우저에 변환되어 있음.

클래스 정의 사용자 정의 클래스 또는 함수를 통해 컴포넌트 만들수 있음.

```
class 정의하는 클래스 extends React.Component {  
render() {
```

return <h1>HelloWorld </h1> ;

```
} const container = document.getElementById('root')  
ReactDOM.render(<App/>, container)
```

State 속성에는 초기값을 넣을 것임

state 속성을 설정하는 메소드는 setState() 메소드 사용

값은 변화하면 컴포넌트는 render() 메소드를 통해서 화면에 반영되는 흐름

리액트를 활용한 현재 시간 출력 프로그램

```
<script type="text/babel">  
class App extends React.Component {  
constructor(props) {  
super(props)  
this.state = {  
time: new Date()  
}}}
```

render() {

```
return <h1>{this.state.time.toLocaleTimeString()}</h1>  
componentDidMount() {  
//컴포넌트가 화면에 출력되었을 때  
this.timerId = setInterval(() => {  
this.setState({  
time: new Date()  
}), 1000)
```

componentWillUnmount() {

//컴포넌트가 화면에서 제거될 때
clearInterval(this.timerId)

}

//출력하기

```
const container = document.getElementById('root')  
ReactDOM.render(<App/>, container)  
</script>
```

이벤트에 연결하기

1) 메소드 선언 2) 메소드 this를 바인드(bind)

3) render() 메소드는 태그에 이벤트 속성에 메소드를 입력해서 이벤트 연결.

props 속성은 정의된 속성에만 가능. 정의된 속성을 설정할 때 사용하는 예

```
class App extends React.Component {  
constructor(props) {  
super(props)  
this.handleClick = this.handleClick.bind(this)  
}  
render() { return <h1 onClick={this.handleClick}>?=this.handleClick=?</h1>  
}  
}  
//이벤트가 발생해 실행되는 예
```

<script type="text/babel">

```
//애플리케이션 클래스 생성하기  
class App extends React.Component {  
constructor(props) {  
super(props)  
this.state = {  
count: 0  
}}  
this.countUp = this.countUp.bind(this)  
}  
render() {  
return <div>  
<h1>클릭한 횟수: {this.state.count}</h1>  
<button onClick={this.countUp}>클릭</button>  
</div>  
}  
countUp(event) {  
this.setState({  
count: this.state.count + 1  
})  
}  
//출력하기  
const container = document.getElementById('root')  
ReactDOM.render(App, container)  
</script>
```

클래스의 메소드 정의하기

```
class App extends React.Component {  
constructor(props) {  
super(props);  
}  
render() {  
}  
}
```

componentDidMount() { 컴포넌트가 화면에 출력될 때 초기화

componentWillUnmount() { 컴포넌트가 화면에서 제거될 때 초기화

~~이벤트 연결하기~~ 입력 양식 사용하기

컴포넌트 배열

```

<script type="text/babel">
//애플리케이션 클래스 생성하기
class App extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      text: " // 작은 따옴표 2개
    }
    this.handleChange = this.handleChange.bind(this)
  }
  render () {
    return <div>
      <input value = {this.state.text}
        onChange = {this.handleChange} />
      <h1>(this.state.text)</h1>
    </div>
  }
  handleChange (event) {
    this.setState({
      text: event.target.value
    })
  }
}
//출력하기
const container = document.getElementById('root')
ReactDOM.render(<App/>, container)
</script>

```

JSX 를 토대로 같은 내용을 사용하여 한 번에 여러 개의 컴포넌트를 출력
 //애플리케이션 클래스 생성하기

```

<script type = "text/babel">
//애플리케이션 클래스
class App extends React.Component {
  render () {
    const list = [
      <li>사과</li>
      <li>바나나</li>
      <li>배</li>
      <li>귤</li>
    ]
    return <ul>{list}</ul>
  }
}
const container =
document.getElementById('root')
ReactDOM.render(<App/>, container)
</script>

```

this.state의 같은 배열을 만들고 render() 메소드 안에서 map() 메소드를 사용하여 이를 JSX로 바꿔서 브라우저에 출력하는 코드

생성하기
 class App extends React.Component {
 render () {
 const list = [
 사과
 바나나
 배
 귤
]
 return {list}
 }
 Constructor (props) {
 super(props);
 Super(props)
 this.state = ?
 fruits: ['사과', '바나나', '배', '귤'] ??
 render () { // 항목 출력
 const list = this.state.fruits.map((item) => ?
 return {item} ?
 // 항목 출력
 return {list} ?

스타일 지정하기

~~render() }~~
~~const style = {}~~

return <h1 style=?style?>문자</h1>

제작상태 | 대기 | 소설가작

```
<script type="text/babel">
//애플리케이션 클래스 생성하기
class App extends React.Component {
  constructor (props) {
    super(props)
    this.state = {
      checked: false
    }
    this.handleClick = this.handleClick.b
```

```
render () {
  const textStyle = {
    color: this.state.checked ? 'blue' : 'red'
  }
  return <div>
    <input type="checkbox"
      onClick={this.handleClick}/>
    <h1 style={textStyle}>글자</h1>
  </div>
}
handleClick (event) {
  this.setState({
    checked: event.target.checked
  })
}
```

```
        }
    }
//줄력하기
const container = document.getElementById('root')
ReactDOM.render(App, container)
```

리액트와 테이터 여러가지의 경우 노트 사용하기

Item 커스텀화를 만들고 사용하는 코드

```
<script type = "text/babel">
```

//애플리케이션 클래스 생성하기

```
class App extends React.Component {  
  render() {  
    return <ul>  
      <Item value="Item 1"/>  
      <Item value="Item 2"/>  
      <Item value="Item 3"/>  
    </ul>  
  }  
}
```

```
class Item extends React.Component {
```

```
    render() {
        return <li>Item 컴포넌트</li>
    }
}
```

```
    }  
    // 출력하기  
    const container = document.getElementById('root')  
    ReactDOM.render(<App />, container)  
  </script>
```

- 블록에서 주식 State 속성 변경하기

부모 컴포넌트에서 자식 컴포넌트로 어떤 데이터를 전달할 때는 속성(`this.props`)을 사용

자식으로 어떤 텍스트를 전달해 두 화면 내용을 변경할 때도 속성(this.props) 사용

1991-1992 - 100% of students at grade level (California and state)

한국어는 영어의 영향을 매우 받았습니다.

마지막으로 변경 가능한 프로퍼티 (mutableProps)

`prevProps` 속성 값과 현재 (`this.props`) 속성 값을 비교해서 변경이 있을 경우

부모에서 자식의 state 속성 변경하기
Item 컴포넌트에 속성 전달하기

```
<script type="text/babel">
//애플리케이션 클래스 생성하기
class App extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      time: new Date()
    }
  }
  componentDidMount () {
    //컴포넌트가 화면에 출력되었을 때
    this.timerId = setInterval(() => {
      this.setState({
        time: new Date()
      })
    }, 1000)
  }
  componentWillUnmount () {
    //컴포넌트가 화면에서 제거될 때
    clearInterval(this.timerId)
  }
  render () {
    return <ul>
      <li value={this.state.time.toLocaleString()} />
      <li value={this.state.time.toLocaleString()} />
      <li value={this.state.time.toLocaleString()} />
    </ul>
  }
}
class Item extends React.Component {
  constructor (props) {
    super(props)
    this.state = {
      value: props.value
    }
  }
  componentDidUpdate (prevProps) {
    if( prevProps.value !== this.props.value) {
      this.setState({
        value: props.value
      })
    }
  }
  render() {
    return <li>{this.state.value}</li>
  }
}
//출력하기
const container = document.getElementById('root')
ReactDOM.render(<App />, container)
</script>
```

자식에서 부모의 state 속성 변경하기
반대로 자식 컴포넌트에서 부모 컴포넌트의 상태를 변경할 때는
메소드를 사용

부모 컴포넌트에서 자식(부모)의 속성을 변경하는 메소드를 부모에게
전달한 뒤, 자식에게 이를 활용하도록 한다.

1) 메소드 선언 2) 메소드에 this 바인드

3) render() 메소드에서 출력하는 테그의 속성에 이벤트를
설정해나 시벤트를 연결

```
<script type="text/babel">
```

//애플리케이션 클래스 생성하기

```
class App extends React.Component {
  constructor (props) {
    super(props)
    this.state = {
      value:''
    }
    this.changeParent
    =this.changeParent.bind(this)
  }
  render() {
    return <div>
      <CustomInput
        onChange={this.changeParent} />
      <h1>{this.state.value}</h1>
    </div>
  }
  changeParent (event) {
    this.setState({
      value: event.target.value
    })
  }
}
```

```
class CustomInput extends
React.Component {
  render () {
    return <div>
      <input
        onChange={this.props.onChange} />
    </div>
  }
}
// 출력하기
const container =
document.getElementById('root')
ReactDOM.render(<App />, container)
</script>
```

//주차 2 앤드

import 기존에 html에서는 <script> 태그를 활용해서

자바스크립트 파일(라이브러리)을 불러옵니다

//외부 라이브러리를 불러옵니다

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

//hello.js를 불러옵니다.

```
<script src="./hello.js"></script>
```

ES6에서는 모듈을 불러오는 방식으로 import 사용

```
import axios from 'axios'; // 다운받은 axios 라이브러리를 불러옴
```

```
import App from './App.js' // App.js 파일을 불러옴
```

```
import Potato from './main' // import를 할 때 모듈명은 자유롭게 설정이 가능
```

export 모듈을 내보내서 다른 곳에서 사용할 수 있게 만들 수 있음.

기본적으로 import 하기 위해서는 export가 되어있어야함.

//App.js 파일

```
function App(){  
    return <h1>hello</h1>  
}
```

```
export default App; // App함수를 외부에서 사용할 수 있도록 export 하는 것
```

import 할 때는 {}를 이용

```
import {a, b, c} from './App.js'
```

export 모듈을 내보내서 다른 곳에서 사용할 수 있게 만들 수도 있음.

```
function funcA(){
```

```
    //...
```

```
}
```

```
function funcB() {
```

```
    //...
```

```
}
```

```
const varC = 'hello';
```

```
export {
```

```
    a : funcA,
```

```
    b : funcB,
```

```
    c : varC
```

```
}
```

```
import {a, b, c} from './App.js'
```

React

React library
주로 자바스크립트 코드로 작성
html CSS javascript
3 번째 버전

npx create-react-app huts

cd huts

npm start

node_modules: 라이브러리 모듈 풀더

public 폴더는 static 파일 보관함.

src 소스 보관함.

<div id="root"></div>

작성한 코드를 넣는다

APP.js 파일로 제공하는 웹페이지 index.js

```
function App() {
  return (
    <div className="App">
      <h1>Hello world!</h1>
    </div>
  );
}

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
  document.getElementById('root'));

```







