

[2022-11-22] 2반 실습 파일

학습 목표

- React의 Hook 활용하여 To-do 리스트를 만들기 + 적용된 css를 분석
- React의 스타일링 방법을 2가지(css파일 작성 및 css-in-js) 방식으로 학습하고, 각각 활용해 봅니다. + styled-components 라이브러리를 학습하고, 이를 활용하여 React 앱을 스타일링

문제 풀이

1. To-do list 앱: useState 사용하기

사전 지식

Mutate?

지금까지 저희는 setter 함수를 통해서만 state값을 업데이트 해주었습니다.

왜 직접 수정하면 안돼냐?!

쉽게 말씀 드리면, setter 함수를 통해서 상태값을 업데이트 시켜주게 되면 메시지를 줘요. 아 상태값이 업데이트되었구나 하고 상태값을 업데이트 되었을 때 개발자 도구로 확인해보면 그 상태값만 업데이트 되는 걸 볼 수 있어요.

setter 함수를 통해서가 아닌 직접적으로 상태값을 변경시켜주게 되면 그 메시지를 받지 못하게 되어서 렌더링이 일어나지 않게될 수 있어요.

상태를 “잘” 업데이트 시켜주기 위해 mutate라는 개념을 알아야해요. (다음주 redux 공부할 때도 나올 개념~)

mutate == 기존 배열 그 자체 업데이트, mutate가 아닌 것 === 업데이트가 반영된 새로운 배열을 리턴.

state는 read-only로 직접 변경을 해주면 안되므로 mutate가 아닌 방법을 사용해야겠죠?

리액트 공식 문서에서 권고하는 사항이 array 상태값을 업데이트해주는 방법이 있는데요,

(예제 후 10분 암기. 연습 kahoot: <https://create.kahoot.it/details/61dff3f4-fe35-4546-a95c-ae5a0d0893d2>)

→ 들어가서 닉네임 치기. 영광의 MVP 두구두구두구)

	avoid (mutates the array)	prefer (returns a new array)
adding	<code>push</code> , <code>unshift</code>	<code>concat</code> , <code>[...arr]</code>
removing	<code>pop</code> , <code>shift</code> , <code>splice</code>	<code>filter</code> , <code>slice</code>
replacing	<code>splice</code> , <code>arr[i] = ...</code>	<code>map</code>

(물론 위 방법 뿐 아니라 spread operator로 꺼내주어 그걸 mutate 한 뒤 그 값을 리턴해줘도 되긴 하지만... 굳이 지름길을 놔두고??!!)

```
// Adding

// push 뒤에
const myArray = [1]
myArray.push(2)
console.log(myArray) // [1, 2]

// unshift 앞에
const myArray1 = [1]
myArray1.unshift(-1, 0)
console.log(myArray1) // [-1, 0, 1]

// concat: concatenate 사슬 연결하다는 이미지. 새 배열을 반환
const myArray2 = [1]
const newArray_concat = myArray2.concat(3)
console.log(newArray_concat) // [1, 3]

// spread
```

```
const myArray3 = [1]
const newArray_spread = [...myArray3, 4]
console.log(newArray_spread) // [1,4]
```

```
// Removing

// pop
const arr = ['a', 'b', 'c', 'e', 'f'];
arr.pop(); //배열의 마지막요소 제거 'f' 제거
console.log(arr)

// shift
const arr1 = ['a', 'b', 'c', 'e', 'f'];
arr1.shift(); // 배열의 첫번째 요소를 제거
console.log(arr1)

// splice
const arr2 = ['a', 'b', 'c', 'e', 'f'];
arr2.splice(2, 1); // index 2 부터 1개의 요소('c')를 제거
console.log(arr2)

// filter
const arr3 = ['a', 'b', 'c', 'e', 'f'];
const new_arr = arr3.filter(el => el !== 'a'); // 특정 조건에 해당하는 원소를 제외하여 리턴. 'b'~'f'
console.log(new_arr)

// slice
const arr4 = ['a', 'b', 'c', 'e', 'f'];
const new_arr1 = arr4.slice(0,3); // index 범위에 해당하는 원소만 리턴. 여기서 index 0~2까지. 'a','b','c'
console.log(new_arr1)
```

```
// Replacing

// splice
const arr = ['a', 'b', 'c', 'e', 'f'];
arr.splice(3, 2, '토마토', '메롱'); // index 3 부터 2개의 원소를 각각 '토마토', '메롱'으로 변경
console.log(arr)

// arr[i]
const arr1 = ['a', 'b', 'c', 'e', 'f'];
arr1[2] = '토마토' // index 2 원소 변경
console.log(arr1)

// map
const arr2 = ['a', 'b', 'c', 'e', 'f'];
const newArr = arr2.map(el => el === 'a'? '토마토' : el);
console.log(newArr)
```

++또 다른 필터링 방법

추가 추가 spread & rest 연습

spread (...) 연산자: 오브젝트 꺼내주기

rest (...rest): 명시한 키 제외 나머지 뭉뚱그려 표현

```
const aespa = { name: "에스파", age: 19, experiencesY: 5 }

const bts = { name: "방탄", age: 30, experiencesY: 15 }

const songkang = { name: "송강", age: 28, experiencesY: 10 }

const humans = [aespa, bts, songkang] // 각 원소의 이름을 제외한 나머지 값 리턴~

const getData = ({name, ...rest}) => rest;

const res = humans.map((human)=>getData(human))

console.log(res)
```

```
// 이후에 OOP 개념 대입 설명해보기
```

▼ 문제 정답

```
// App.js

//1. useState를 react에서 import하세요.
import React, { useState } from 'react';
import Todo from './todo';
import TodoForm from './TodoForm';

function App() {
  //2. useState가 반환하는 첫 번째 인자인 state와 두 번째 인자인 setState를 todos, setTodos로 변경하세요.
  const [todos, setTodos] = useState([
    //3. useState의 배열에 초기화면에 display할 원소를 작성하세요.이렇게하면 useState 는 배열로 초기화하는데, 개별 원소는 text(사용자가 입력한 todo)와 hasCompleted: false,
    {
      text: 'Upload the Vlog by tonight',
      hasCompleted: false,
    },

    {
      text: 'Read Book from page 51~220',
      hasCompleted: false,
    },

    {
      text: 'Finish Season 3 of La Casa De Papel',
      hasCompleted: false,
    },
  ]);

  //4. addTodo()와 completeTodo() 함수를 작성하세요.이때 사용자가 입력한 text 데이터를 array(newTodos)로 받아 text로 전달하세요.
  const addTodo = text => {
    const newTodo = { text: text, hasCompleted: false };
    // todo 추가 순서 바꾸기 전
    // 방법 1.
    const newTodos = [...todos, newTodo];
    // 방법1.1
    // const newTodos = [newTodo, ...todos]; // 새로운 todo가 뒤에서부터 추가되는게 거슬려...순서 바꿔줘!
    // 방법2.
    // const newTodos = todos.concat(newTodo)
    // 방법3. 굳이?!
    // const newTodos = [...todos]
    // newTodos.push(newTodo)
    // 방법4. 굳이?!
    // const newTodos = [...todos]
    // newTodos.unshift(newTodo)
    setTodos(newTodos);
  };
  // **코치님~ 왜 addTodo, completeTodo이런 함수들은 Todo컴포넌트에서 선언 안해주고 여기서 했어요?

  const completeTodo = index => {
    const newTodos = [...todos];
    newTodos[index].hasCompleted = true;
    // 더 좋은 방법 있으면 공유 ㄱㄱ
    setTodos(newTodos);
  };

  const removeTodo = index => {
    // update arrays in state
    // 방법 1. index로부터 1개 원소 제거
    const newTodos = [...todos];
    newTodos.splice(index, 1);

    // 방법 2. 필터
    // const newTodos = todos.filter((el, idx)=> idx !== index)

    setTodos(newTodos);
  };

  return (
    <div className="app">
      <div className="todo-list">
        <h1>TO DO LIST</h1>
      </div>
    </div>
  );
}
```

```

      <TodoForm addTodo={addTodo} />
      {todos.map((todo, index) => {
        return (
          <Todo
            key={index} // Each child in a list should have a unique "key" prop
            index={index}
            todo={todo}
            completeTodo={completeTodo}
            removeTodo={removeTodo}
          />
        );
      })}
    </div>
  </div>
);
}

export default App;

```

```

// TodoForm.js

import { useState } from 'react';
import React from 'react';

// Props 받아오는 방법. props or {<props name>}
function TodoForm({ addTodo }) {
  // 유저가 입력한 값
  const [value, setValue] = useState('');

  const handleSubmit = e => {
    // 있을 때 없을 때 차이
    // preventDefault: 기본 동작 해제
    // a 태그나 submit 태그는 누르게 되면 href 를 통해 이동하거나 , 창이 새로고침하여 실행됩니다.
    // preventDefault 를 통해 이러한 동작을 막아줄 수 있습니다.
    e.preventDefault();

    // 아무것도 입력하지 않은채로 제출 시 addTodo되는걸 방지
    // 함수 멈춰!
    if (!value) return;

    // 스페이스 값을 입력해도 입력되는 걸 방지
    // if (!value.trim()) { // trim()은 기존 값에 변형을 주지 않고 새로운 값 리턴
    //   return;
    // }

    // 아래 or if statement 혹시 되나 싶었는데 안된다고 함: https://stackoverflow.com/questions/9056112/javascript-shorthand-if-else-and-return
    // value || return // 앞이 null, undefined, 0, -0 ...이면 뒤에 실행

    addTodo(value);
    setValue('');
  };

  return (
    // input 태그가 하나인 경우, enter -> submit. 여러개면 엔터도 안먹혀!
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        className="input"
        autoFocus
        // 왜 value? default값을 넣을 수도, 아니면 제출 후 빈 배열로 바꿔줄 거라서
        value={value}
        placeholder="Enter todo item"
        onChange={e => setValue(e.target.value)}
      />
    </form>
  );
}

export default TodoForm;

```

```

// Todo.js

import React from 'react';

// PascalCasing Component a
function Todo({ todo, index, completeTodo, removeTodo }) {

```

```
// props == readOnly
return (
  <div
    style={{ textDecoration: todo.hasCompleted ? 'line-through' : 'none' }}
    className="todo"
  >
    {todo.text}
    <div>
      <button className="btn" onClick={() => completeTodo(index)}>
        <a href="#">Complete</a>
      </button>
      <button className="btn btn-remove" onClick={() => removeTodo(index)}>
        <a href="#">Remove</a>
      </button>
    </div>
  </div>
);
}

export default Todo;
```

▼ 문제 확장

**오브젝트 업데이트

```
// 상태가 state일 때 특정 key에 해당하는 value를 수정하고 싶은 경우!
const obj = { foo: "a", bar: "b" };
const updatedObj = { ...obj, foo: 1 };
```

**코치님~ 왜 addTodo, completeTodo, removeToDo 이런 함수들은 각자 사용되는 컴포넌트 내에서가 아니라 App 컴포넌트에 위치해있나요?

언급한 모든 함수들은 todo 상태값과 세터함수를 필요로 해요. 그 상태값과 세터함수 모두 props로 전달해준다면 각 컴포넌트 내에서 사용하는 함수가 될 수 있어요!

하지만 저희 props 를 == props는 read-only로 전달해주게되면 전달받은 컴포넌트는 그 값을 수정할 수 없게되어요!

만약 가능하다고 해도, 상태값을 업데이트 시키는 함수를 만들 때마다 props로 전달해줘야한다는 사실이 번거롭죠?

그래서 저희는 다음 주에 redux라는 툴을 배울 건데요, 바로바로 컴포넌트 레벨에 상관없이 언제든지 접근이 가능하도록 저장소를 따로 만들어서 그 곳에 상태값과 세터 함수를 넣어줄 수 있습니다.

2. To-do list 앱: CSS로 레이아웃 만들기 I

사전 지식

일반적인 CSS를 React에서는 어떻게 사용되는지 확인해봅시다

▼ 문제 정답

```
/* index.css */

.todo,
input {
  /* 변경 전 파일을 보고 결과물과 일치하게 만들어보자 */
  /* 하나씩 질문하면서 해보기... */
  /* 폰트도 바뀌어야할 것 같고... */
  /* 인풋 내 패딩도... */
}
```

```

/* border 둥근 정도도... */
/* 인풋 박스를 일직선으로... */
/* 배경색... */
/* 폰트패밀리...https://fonts.google.com/ */

/* 패딩으로 박스 사이즈 설정 오키 */
/* 아니면 박스 사이즈 미리 설정하고 내부 텍스트 크기 조절 */

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
  Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
font-weight: 600;
background: #fff;
/* https://html-css-js.com/css/generator/box-shadow/ */
box-shadow: 1px 1px 1px rgba(0, 0, 0, 0.15);
/* 상하 좌우 2px 4px */
padding: 3px 10px;
/* 폰트사이즈 화면 크기 + 미니멈 min(5px, 10vh) */
font-size: 12px;
/* 헤더(50px짜리)를 제외한 나머지 높이 전체: calc(100vh - 50px) */
margin: 10px;
border-radius: 3px;
display: flex;
justify-content: space-between;
align-items: center;
}

```

```

// todo.js
import React from 'react';

import React from 'react';

function Todo({ todo, index, completeTodo, removeTodo }) {
  // const {text, hasCompleted} = todo; // 구할당으로 분해해줄 수 있지만 여기서 그대로 해볼게요!
  // 방법 1. semantic 태그에 직접 오브젝트 형식의 {} 스타일링 넘겨주기
  // 방법 2. 우리가 동일한 클래스를 추가해 동일한 css를 공유했던 것처럼
  // 변수로 빼주면 컴포넌트 내에서 재사용할 수 있어
  const todoStyle = {
    textDecoration: todo.hasCompleted ? 'line-through' : 'none',
  };

  // <div style={{ ...todoStyle, color: 'red' }} className="todo">
  // ++ 스타일링을 더 추가하거나 오버라이딩하고 싶다면 { ...todoStyle, color: 'red' }

  return (
    <div style={todoStyle} className="todo">
      {todo.text}
      <div>
        <button className="btn" onClick={() => completeTodo(index)}>
          <a href="#"!>Complete</a>
        </button>
        <button className="btn btn-remove" onClick={() => removeTodo(index)}>
          <a href="#"!>Remove</a>
        </button>
      </div>
    </div>
  );
}

export default Todo;

```

3. To-do list 앱: CSS로 레이아웃 만들기 I

사전 지식

주석 내 많은 정보가 담겨있습니다 꼭 읽어주세요

만약 안에 있는 원소들 크기 기준으로 바깥걸 만들고 싶으면 안에서 밖으로~

바깥 크기 정해놓고 안으로 가고 싶으면 밖에서 안으로~

그냥 하고싶은대로 해도 ㅇㅋ~

저는 여기서 큰걸 좀혀나가는 식으로 밖에서 안으로 가볼게요~

▼ 문제 정답

```
/* TodoListTemplate.css */

/*배웠던 내용을 활용하여 자유롭게 스타일링해보세요.*/

/* css selectors: https://www.w3schools.com/csSref/css_selectors.php */
/*배웠던 내용을 활용하여 자유롭게 스타일링해보세요.*/

:root {
  --boxShadow: 0 3px 6px rgba(0, 0, 0, 0.16);
  --purple: #6d32a5;
  --pink: #c55aa7;
}

/* reset css: https://meyerweb.com/eric/tools/css/reset/ reset css 키워드 구글링 고고 */

html,
body {
  margin: 0px;
  height: 100%;
  width: 100%;
  font-family: 'Noto Sans KR', sans-serif;
}

body {
  /* 시티팝 키워드 검색 */
  /* https://soundtrip.herokuapp.com/ 스크래핑 */
  background-image: url('https://images.unsplash.com/photo-1536768139911-e290a59011e4');
  background-size: cover;
}

.todo-list-template {
  /* 너비 responsive~ */
  width: 50%;
  min-width: 300px;
  /* 배경 background-color ㅇㅋ background 더 옵션 많이 줘 url */
  background: white;
  /* 자주 쓸 것 같은 것들 _variable.css 파일 만들어서 $boxShadow: 0 3px 6px rgba(0, 0, 0, 0.16); 불러올 때 var(--boxShadow)*/
  /* https://html-css-js.com/css/generator/box-shadow/ */
  box-shadow: 0 0 40px rgba(255, 255, 255, 0.5);
  /* margin - 남의 나라 침범 vs padding - 우리 나라 지키자 외벽을 쌓는다 */
  /* margin 위아래 좌우<- auto면 중앙정렬 가능 */
  margin: 20vh auto;
  /* 끝을 둥글게 만들어보자 */
  border-radius: 25px;
  overflow: hidden;
}

.title {
  padding: 2rem;
  /* 구글 폰트 */
  font-family: 'Secular One', sans-serif;
  font-size: 2.7rem;
  text-align: center;
  font-weight: 600;
  /* color picker extension */
  background: #6d32a5;
  color: white;
  background-image: linear-gradient(to right, var(--pink), var(--purple));
}

.form-wrapper {
  padding: 1rem;
  border-bottom: 1px solid var(--purple);
}

.todos-wrapper {
  /* 아무 투두가 없어도 */
  min-height: 5rem;
}
```

```

/* Form.css */

/*배웠던 내용을 활용하여 자유롭게 스타일링해보세요.*/

.form {
  display: flex;
}

/* form 안에 있는 최상단 input 태그 selector*/
.form input {
  /* Flexbox Froggy 게임: https://flexboxfroggy.com/ */
  flex: 1;
  /* == flex: 1 1 0% 부모 총 넓이 중에 나머지 스페이스 다 잡아먹음*/
  font-size: 1.05rem;
  outline: none;
  /* focus되었을 때 선 */
  border: none;
  /* 박스 선 */
  border-bottom: 1px solid var(--pink);
}

.create-button {
  padding: 0.5rem 1rem;
  margin-left: 1rem;
  background-color: var(--purple);
  opacity: 0.5;
  border-radius: 15px;
  color: white;
  font-weight: 600;
  cursor: pointer;
  transition: all 0.1s ease-in-out;
}

.create-button:hover {
  opacity: 0.8;
  scale: 1.1;
}

.todo-item {
  display: flex;
  align-items: center;
  padding: 1rem;
  cursor: pointer;
  /* https://developer.mozilla.org/en-US/docs/Web/CSS/user-select */
  user-select: none;
  transition: all 0.1s;
}

.todo-item:hover {
  background-color: var(--pink);
  opacity: 0.5;
  color: white;
}

/* 붙여쓰면 두 클래스에 모두 해당하는 것 띄어쓰면 뭐 안에있는 뭐 */
.todo-item.remove {
  opacity: 0;
}

.todo-item:hover.remove {
  opacity: 1;
}

/* 첫번째 자식 빼고 다 */
/* 만약 첫+마지막 빼고 하고 싶으면 .todo-item:not(:first-child):not(:last-child) */
.todo-item:not(:first-child) {
  border-top: 1px solid var(--purple);
}

.remove {
  margin-right: 1rem;
  font-weight: 600;
}

.todo-text {
  flex: 1;
  /* https://developer.mozilla.org/en-US/docs/Web/CSS/word-break */
  word-break: break-all;
}

```



```
.checked {
  text-decoration: line-through;
  opacity: 0.6;
}

.check-mark {
  margin-left: 1rem;
  font-weight: 800;
}
```

▼ 문제 확장

전반적인 코드 훑어보기.

```
// App.js

import React, { useState } from 'react';
import TodoListTemplate from '../components/TodoListTemplate';
import Form from '../components/Form';
import TodoItemList from '../components/TodoItemList';

function App() {
  // **코치님??!! const는 불변값, let은 변하는 값 아닌가요?
  let [state, setState] = useState({
    input: '',
    todos: [
      { id: 1, text: '점심약속', checked: false },
      { id: 2, text: '운동', checked: true },
      { id: 3, text: '낮잠', checked: false },
      { id: 4, text: '리액트 실시간 강의', checked: false },
    ],
  });

  function handleChange(e) {
    setState(state => ({
      ...state,
      input: e.target.value, // input의 다음 바뀔 값
    })); // 기존값을 참조해온 딕셔너리 상태 업데이트!!
  }

  function handleCreate() {
    const { input, todos } = state; // 구할당
    setState({
      input: '', // 인풋 다시 비움
      // concat 을 사용하여 배열에 추가. concat은 원소 추가! 배열을 반환!
      todos: todos.concat({
        id: state.todos.length + 1,
        text: input,
        checked: false,
      }),
    });
  }

  function handleKeyPress(e) {
    // 클릭된 키가 Enter 일 경우 handleCreate 호출
    if (e.key === 'Enter') {
      handleCreate();
    }
  }

  function handleToggle(id) {
    const { input, todos } = state;

    // 방법 1.
    const index = todos.findIndex(todo => todo.id === id); // 짝 라인을 줄 투두 인덱스
    const selected = todos[index]; // 짝 라인을 줄 투두
    const nextTodos = [...todos];
    nextTodos[index] = {
      // 인덱스 직접 불러 그 투두의 checked 키 업데이트
      ...selected,
      checked: !selected.checked,
    };

    // 방법 2.
    // const nextTodos = [...todos]; // 깊은 복사
```

```

    // nextTodos.find((todo, idx) => {
    //   // find는 조건에 만족하는 아이템 찾으면 다음은 실행이 멈춰서 선택! 외 다른 forEach 함수로도 구현 가능할듯
    //   if (todo.id === id) {
    //     const checked = nextTodos[idx].checked;
    //     nextTodos[idx].checked = !checked;
    //   }
    // });

    setState({ input: input, todos: nextTodos });
  }

  function handleRemove(id) {
    const { todos } = state;
    let nextTodos = todos.filter(todo => todo.id !== id);
    setState({ ...state, todos: nextTodos });
  }

  // 이벤트는 다 외울 수 없어...그때 그때 찾아보기
  // 모든 events 를 react에 쓸 수 있습니까? 노습
  // 아직 계속 지원하려고 작업하는게 있다고는 함
  // DOM events supported by React 키워드 검색 고고
  // 자바스크립트 이벤트 잘 정리된 사이트: https://inpa.tistory.com/entry/JS-%F0%9F%93%9A-%EC%9D%B4%EB%B2%A4%ED%8A%B8-%F0%9F%92%AF-%EC%B4%9D-%E
  // 자바스크립트 이벤트 닥스 > Event listening 탭 확인: https://developer.mozilla.org/en-US/docs/Web/Events#event\_listing

  // <Component>여기 위치한 계</Component> 바로바로 children prop
  // ***자 그럼 언제 children 언제 일반 props 쓰나요?
  return (
    <TodoListTemplate
      form={
        <Form
          value={state.input}
          onPress={handleKeyPress} // keypress - 키를 누르고 뺏을 때
          onChange={handleChange} // change - 워든 변경이 있을 때
          onCreate={handleCreate} // 커스텀 이벤트. 일관성 유지를 위해 이벤트 핸들러 프롭스 on-식으로 명명합니다
        />
      }
    >
    <TodoItemList
      todos={state.todos}
      onToggle={handleToggle} // 커스텀 이벤트
      onRemove={handleRemove} // 커스텀 이벤트
    />
    </TodoListTemplate>
  );
}

export default App;

```

```

// TodoItem.js

import React, { Component } from 'react';
import './Form.css';

function TodoItem(props) {
  const { text, checked, id, onToggle, onRemove } = props;

  // preventDefault 는 기본으로 정의된 이벤트를 작동하지 못하게 하는 메서드
  // stopPropagation: 버블링 막음. 자식 이벤트가 부모한테까지 전달.
  return (
    <div
      className="todo-item"
      onClick={() => {
        // console.log('부모가 클릭되었'); // 이걸 막으려고 stopPropagation메소드 사용
        onToggle(id);
      }}
    >
    <div
      className="remove"
      onClick={e => {
        console.log('자식 클릭');
        e.stopPropagation();
        onRemove(id);
      }}
    >
    &times;
    </div>
    // && 와 ||
    <div className={`todo-text ${checked && 'checked'}}`>
      <div>{text}</div>
    </div>
  );
}

```

```

    </div>
    {checked && <div className="check-mark">&#x2713;</div>}
  </div>
  );
}

export default TodoItem;

```

**깊은 복사 Deep-copy vs 얕은 복사 Shallow-copy

원시값? number, string, boolean, null, undefined 등. 변수에 실제 데이터값이 저장.

참조값? 배열, 오브젝트, 함수... 변수에 값이 저장된 메모리 공간의 위치 값을 저장.

오브젝트를 얕은 복사를 하게되면 주소 값을 복사하는 것이고, 깊은 복사를 하면 실제 값을 새로운 메모리 공간에 그 값을 복사하는 것이 됨.

예제로 알아보기

```

function fn(x,y){
  if (x === y){ // == : 형변환해줌 , ===: 타입까지 일치해야함
    return true
  } else {
    return false
  }
}

const a = [1,2,3]
const b = a // 얕은 복사 a의 메모리 주소값이 저장

console.log("a,b", fn(a,b)) // 같은 값을 참조해서 true

a.pop()

console.log("a", a, "b", b) // 둘다 같은 주소를 참조하고 있어 둘 다 변형된 값 출력

const c = [1,2,3] // 새로 생긴 변수 다른 주소값 저장

console.log("a,c", fn(a,c)) // 다른 값을 참조해서 false

const d = [...a] // 깊은 복사~ 다른 주소값 저장

console.log("a,d", fn(a,d)) // 다른 값을 참조해서 false

let e = 5
let f = 5 // 숫자의 경우 그 값 자체가 저장
console.log("e,f", fn(e,f)) // 값 비교 동일 true

```

**코치님 const는 불변값, let이 변하는 값 아닌가요?!

사실 그렇지 않습니다~! 처음 배우실 때 헷갈리실까봐 상수/변수로 나누어 설명드렸었지만 익숙해졌으니 언제 쓰이는지 다시 알아보아요 위에서 두가지 데이터 타입의 값(원시값, 참조값)에 대해서 살펴봤습니다.

원시값(숫자, 문자열 등) - 변경할 수 없어요 (immutable)

모든 오브젝트(배열, 오브젝트) - 변경 가능 (mutable).

즉 const는 스코프 내에서 새로운 값이 assign될 순 없지만, 참조 데이터 타입의 경우는 예외적으로 변경이 가능해요!

```

const temp = [1,2,3]

temp.pop() // okey update
temp = [4] // 나ㄴ assign. assign와 update를 구분해주세요

const num = 12

num += 2 // 나ㄴ assign. 풀어쓰면 num = num + 2.

```

**이전 오브젝트 상태값 참조 업데이트~

```
function handleChange(e) {
  setState(state => ({
    ...state, // spread operator로 오브젝트를 밖으로 꺼내주고
    input: e.target.value, // 그 중 input이라는 Key값을 e.target.value로 업데이트해라
  }));
}
```

**키보드 관련 이벤트~

```
window.addEventListener("keydown", (event) => {
  if (event.code === 'Space'){
    영상플레이버튼.click();
  }
  if (event.key === 'm' || event.key === 'M'){
    영상유트버튼.click();
  }
  if (event.key === 'f' || event.key === 'F'){
    풀스크린버튼.click();
  }
});
```

**&&와 ||

앞 && 뒤> && 연산자는 앞이 true여야 뒤가 리턴

앞 || 뒤> || 연산자는 앞이 false(undefined, null, 0...)여야 뒤가 리턴

**언제 children? 언제 보통 props를 사용하나요?

children prop의 장점을 살펴보면 언제 어떤 걸 쓸지 명확해집니다.

children prop 장점

- 어떠한 children도 nesting(둥지처럼 안에 쌓고쌓고)할 수 있음
- 무엇이 들어올지 모를 때 이름을 고민할 필요가 없음

요로코롬 아래처럼 모든 페이지에 같은 레이아웃을 주고 싶은 경우 쓸 수 있겠죠?

(codesandbox에서 실행)

```
// App.js

import React from "react"; // 17? 버전부터 React 임포트 안해주도 바로 사용 가능
import "./style.css";

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

```

// components > layout > Footer.js
import React from "react";
const Footer = () => {
  return(
    <footer>
      Footer
    </footer>
  );
}
export default Footer;

// components > layout > Header.js
import React from "react";
const Header = () => {
  return(
    <header>
      Header
    </header>
  );
}
export default Header;

// components > layout > Layout.js
import React from "react"
import Header from "../Header"
import Footer from "../Footer"
const Layout = (props) => {
  return (
    <>
      <Header />
      <main>{props.children}</main>
      <Footer />
    </>
  )
}
export default Layout;

// index.js

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import Layout from './components/Layout/Layout';
ReactDOM.render(
  <Layout>
    <App />
  </Layout>,
  document.getElementById('root')
);

```

4. To-do list 앱: Styled-Components로 레이아웃 만들기 III

사전 지식

CSS-in-JS.

등장배경을 봐봅시다.

리액트는 JS 안에서 HTML 엘리먼트를 생성할 수 있게 해주어 독립적인 컴포넌트 단위의 개발이 가능해졌죠?

그런데 CSS-in-JS 가 등장하기 전에는 별도의 css 파일을 작성해야했기에,

- 진정한 캡슐화?!에 어울리지 않습니다

- 모든 스타일에 별도의 class 명명 규칙을 적용해야하는 문제... 창의적인 클래스명 떠올리기 쉽지 않죠... 너무많은 클래스 사용하다 보면 이름이 요오오오_로오오오오_코_름 길어질 수 있습니다.
- css 파일 간의 의존성 관리 문제 (한 태그에 여러 클래스...css 오버라이딩 문제.. 어떤 스타일이 다른 스타일을 덮는 경우)
- 기능을 업데이트 함에 있어 불필요해진 CSS를 다시 찾아 제거하기 어렵..
- css 로드 순서에 따라서 스타일 우선순위가 달라져 (non-deterministic resolution, 경우에 따라 같은 인풋이어도 다른 결과가 나오는 문제)

그런데...! 이제 CSS-in-JS 덕분에,

- class명이 빌드 시 유니크한 해시값으로 변경 → 저희의 작명 실력이 두각을 나타낼 일이 없죠
- 스타일링 단위가 컴포넌트 단위로 나뉘어 css 파일간 의존성을 신경쓰지 않아도 되게 됩니다
- 컴포넌트와 css가 같은 구조로 관리되어 업데이트 함에 있어 수정 및 삭제가 쉽습니다
- state 값을 공유할 수 있게 되죠(Sharing constants). 뭐가 좋으나? 상태값을 조건부로 다양한 스타일링이 가능해집니다
- css가 컴포넌트 스코프에서만 적용되어 우선순위에 따른 문제가 발생하지 않습니다

단점도 있다면...

css-in-js 파일은 컴파일될 때 css로 변환해주는 장치가 필요...그러면 번들의 크기가 커지겠죠?!

특히 css-in-js는 js가 모~두 로딩된 후에 css 코드가 생성되니까 눈에 보이는 초기 진입이 느려지겠죠?!

무조건 뭐가 좋다는 답은 없거니와...웹서비스의 용도에 따라 고려해봅니다!

개발 효율성이 중요하나?! 컴포넌트 위주 프로젝트인가?! ⇒ css-in-js 고려! 말고도 추가 고려 사항이 있을 수 있습니다...

그러기 위해서는 추가로 익숙해져야할 개념이 있는데 (SSR, CSR) 그건 다음 강의 때 언급하도록 하고!

Styled-components 실습해보기

Step1.

```
npm i styled-components
vscode-styled-components extension 설치
```

++Auto Import - ES6, TS, JSX, TSX extension 설치

아 이거 불편한데 싶으면 찾아보면 툴 있음 불편함거 참지마용

Step 2.

```
import styled from "styled-components"
```

Step 3. 코딩

상속, 확장, 조건,

실습예제 1. Props 받아오기. 성공 (초록) 실패 (빨강) 버튼을 만들어 보겠습니다.

(심화예제는 다음 강의에)

```
// App.js

import './App.css';
import { CustomButton } from './components/Button.styles';
// import 한 라인으로 쓰기엔 버튼이 너무 많아...하면 import * as B from "..."
```

```
// 후에 <B.CustomButton></B.CustomButton>로도 사용 가능
// 같은 디자인을 적용하되 시맨틱 태그 자체를 바꿔주고 싶다면 as="뭐시기"

function App() {
  // 그냥 succeed 넘겨주면 succeed = true가 넘겨짐
  return (
    <div className="App">
      <CustomButton succeed>Custom succeeded</CustomButton>
      <br></br>
      <CustomButton>Custom failed</CustomButton>
      <br></br>
      <CustomButton as="a">Home</CustomButton>
    </div>
  );
}
export default App;
```

```
// Button.styles.js

import styled, {keyframes} from 'styled-components'

// css 에서는
// @keyframes slidein {
//   ...
// }

// animation [시간 남으면 진행]

const rotate = keyframes`
  from {
    transform: rotate(0deg); // 시작을 여기서
  } to {
    transform: rotate(360deg) // 마지막을 여기로
  }
`;

const rainbow = keyframes`
  0% {background-color: red;} // 시작과 끝을 퍼센티지로 무지개 만들어 보자
  25% {background-color: yellow;}
  50% {background-color: green;}
  75% {background-color: blue;}
  100% {background-color: purple;}
`;

// props 받아오기
export const CustomButton = styled.button`
  background-color: ${({props}) => (props.succeed ? "green": "tomato")};
  color: white;
  outline: none;
  border: none;
  // pseudo class
  &:hover{
    opacity: 0.6;
  }
  animation: ${rotate} infinite 2s linear;
`;
```

▼ 문제 정답

```
// Form.jsx

import React from "react";
import styled from "styled-components";

const FormStyled = styled.div`
  display: flex;
  & input {
    flex: 1;
    font-size: 1.25rem;
    outline: none;
    border: none;
  }
`;
```

```

        border-bottom: 1px solid #c5f6fa;
    }
`;

const CreateButton = styled.div`
padding: 0.5rem 1rem;
margin-left: 1rem;
background: 3px solid #5426cc;
border-radius: 3px;
color: #cef1ee;
font-weight: 600;
cursor: pointer;

&:hover {
    background: #3bc9db;
}
`;

const Form = ({ value, onChange, onCreate, onKeyPress }) => {
    return (
        <FormStyled>
            <input value={value} onChange={onChange} onKeyPress={onKeyPress} />
            <CreateButton onClick={onCreate}>추가</CreateButton>
        </FormStyled>
    );
};

export default Form;

```

```

// TodoItem.jsx

import React from "react";
import styled from "styled-components";

const TodoItemStyled = styled.div`
padding: 1rem;
display: flex;
align-items: center;
cursor: pointer;
transition: all 0.15s;
user-select: none;
&:hover {
    background: #cef1ee;
}
&:hover .remove {
    opacity: 1;
}
& + & {
    border-top: 1px solid #cef1ee;
}
`;

const RemoveStyled = styled.div`
margin-right: 1rem;
color: #e64980;
font-weight: 600;
opacity: 0;
`;

const TodoTextStyled = styled.div`
flex: 1;
word-break: break-all;
text-decoration: ${props => (props.checked ? "line-through" : "none")};
color: ${props => (props.checked ? "#4c445f" : "#000")};
`; // 문자열 리터럴 활용 넘겨준 props값을 리턴하는 형태로 받아올 수 있음

const CheckMark = styled.div`
font-size: 1.5rem;
line-height: 1rem;
margin-left: 1rem;
color: #cef1ee;
font-weight: 800;
`;

function TodoItem(props) {
    const { text, checked, id, onToggle, onRemove } = props;

    return (

```



```

<TodoItemStyled onClick={() => onToggle(id)}>
  <RemoveStyled
    className="remove"
    onClick={(e) => {
      e.stopPropagation();
      onRemove(id);
    }}
  >
    &times;
  </RemoveStyled>
  <TodoTextStyled checked={checked}>
    <div>{text}</div>
  </TodoTextStyled>
  {checked && <CheckMark>&#x2713;</CheckMark>}
</TodoItemStyled>
);
}

export default TodoItem;

```

```

// TodoListTemplate.jsx

import React from "react";
import styled from "styled-components";

const TodoListTemplateStyled = styled.main`
  background: white;
  width: 512px;
  box-shadow: 0 3px 6px rgba(0, 0, 0, 0.16), 0 3px 6px rgba(0, 0, 0, 0.23);
  margin: 4rem auto 0;
`;

const TitleStyled = styled.div`
  padding: 2rem;
  font-family: "Malgun Gothic";
  font-size: 2.7rem;
  text-align: center;
  font-weight: normal;
  background: #5426cc;
  color: white;
`;

const FormWrapperStyled = styled.section`
  padding: 1rem;
  border-bottom: 1px solid #5426cc;
`;

const TodosWrapperStyled = styled.section`
  min-height: 5rem;
`;

const TodoListTemplate = ({ form, children }) => {
  return (
    <TodoListTemplateStyled>
      <TitleStyled>Things to do</TitleStyled>
      <FormWrapperStyled>{form}</FormWrapperStyled>
      <TodosWrapperStyled>{children}</TodosWrapperStyled>
    </TodoListTemplateStyled>
  );
};

export default TodoListTemplate;

```

▼ 문제 확장

****JS inline func**

() 로 감싸졌거나 없거나> 싱글 값 리턴 eg. () => ()
 {}: 여러 실행문 실행> eg. () => {return ...}

****Styled components + Material UI**

<https://www.youtube.com/watch?v=or3np70c7zU&t=386s> Material UI~

Tailwind CSS

<https://www.youtube.com/watch?v=bxmDnn7lrnk&list=PL4cUxeGkcC9gpXORIEHjc5bgpli5HEGhw> Tailwind CSS

****styled component 안에 그래도 클래스를 넣어줄 수 있나요?**

네! 일반 css와 동일하게 특정 엘리먼트를 지칭해야하는 상황에 사용할 수 있어요.

실습 예제처럼요.

```
const TodoItemStyled = styled.div`
  padding: 1rem;
  display: flex;
  align-items: center;
  cursor: pointer;
  transition: all 0.15s;
  user-select: none;
  &:hover {
    background: #cef1ee;
  }
  &:hover .remove { // 애 또한 위 &:hover 안에 위치시켜도 동일. 그러나 이게 더 직관적임.
    opacity: 1;
  }
  & + & {
    border-top: 1px solid #cef1ee;
  }
`;

function TodoItem(props) {
  const { text, checked, id, onToggle, onRemove } = props;

  return (
    <TodoItemStyled onClick={() => onToggle(id)}>
      <RemoveStyled
        className="remove"
        onClick={e => {
          e.stopPropagation();
          onRemove(id);
        }}
      />
      &times;
    </RemoveStyled>
    <TodoTextStyled checked={checked}>
      <div>{text}</div>
    </TodoTextStyled>
    {checked && <CheckMarkStyled>&#x2713;</CheckMarkStyled>}
  </TodoItemStyled>
  );
}
```