모던 자바스크립트로 배우는 리액트 입문

내용

- 모던 자바스크립트 기초
  - DOM, 가상 DOM
  - 패키지 관리자
  - 모듈핸들러, 트랜스파일러
  - SPA와 기존 웹시스템의 차이
- 모던 자바스크립트 기능(문법적 요소)
  - 변수 선언 키워드: const, let
  - 템플릿 문자열
  - 화살표 함수 ()=>{}
  - 분할 대입 {} []
  - 디폴트값 =
  - 스프레드 구문
  - 객체 생략 표기법
  - map, filter
  - 삼항연산자/논리연산자 && ||
- 자바스크립트 DOM 조작

실습 환경

1. VS code: 코드 에디터, 통합개발환경
2. Node.js: 자바스크립트로 네트워크 앱 개발 가능하도록 지원 환경
    1. npm: node package manager
        1. node.js를 위한 패키지 매니저: 프로젝트에서 필요로 하는 다양한 외부 패키지들의 버전과 의존성 관리함
        2. Node.js 설치하면 자동 설치됨

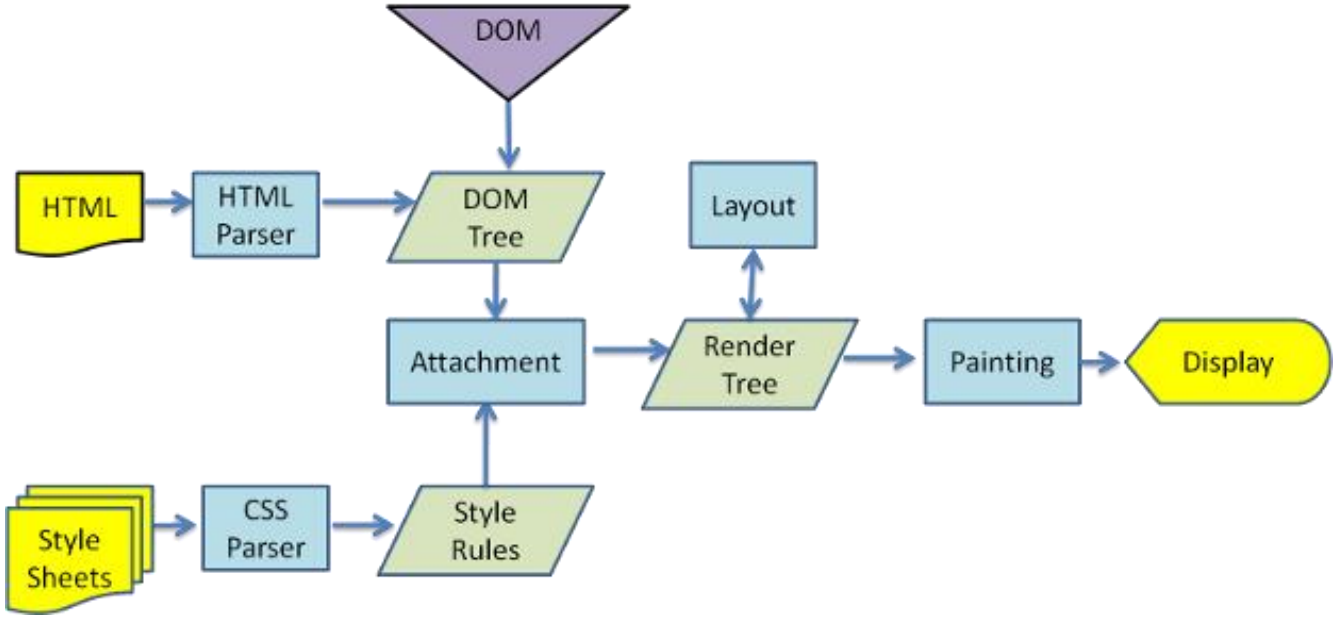
모던 자바스크립트란

- 최신 ECMAScript(ECMAScript 6(=ES2015=ES6) 이상) 사양을 준수하는 자바스크립트
  - **유럽 컴퓨터 제조업 협회 (European Computer Manufacturers Association, ECMA)**
- 특징
  - 리액트, 뷰, 앵귤러 등 **가상 DOM**을 이용하는 라이브러리/프레임워크를 사용
  - npm, yarn 등 패키지 관리자 사용
  - 웹팩 등 모듈 핸들러 사용
  - 바벨 등 트랜스파일러 사용
  - SPA로 작성

웹페이지 만들어지는 과정

웹페이지작성기술

웹페이지렌더링 과정



웹 페이지가 렌더링 되는 과정

1. HTML parser가 HTML을 바탕으로 DOM tree를 그린다.
2. CSS parser가 CSS를 바탕으로 CSSOM을 그린다.
3. DOM에 CSSOM을 적용하여 Render Tree를 그린다.
4. Render Tree를 바탕으로 Painting 하여 실제 화면에 렌더링 한다.

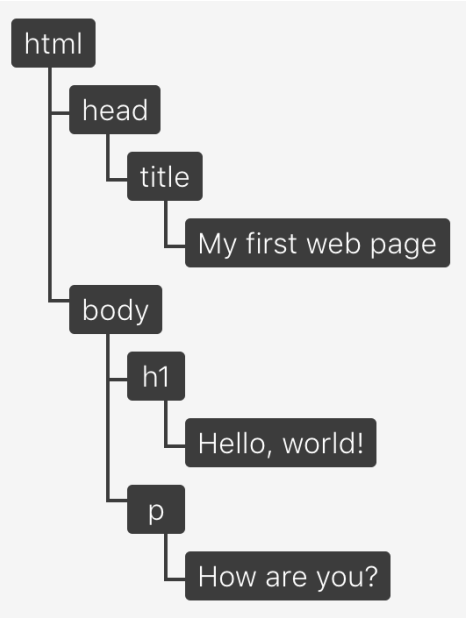
*HTML 코드를 읽어 내려가다가 <script></script> 태그를 만나면 파싱을 잠시 중단하고 js 파일을 로드한다.

웹페이지 만들어지는 과정

```html
<!doctype html>
<html land="ko">
    <head>
        <title>My first web page</title>
    </head>
    <body>
        <h1>Hello, world!</h1>
        <p>How are you?</p>
    </body>
</html>
```

DOM tree 구조(DOM 객체)로 바꿈

HTML Parser



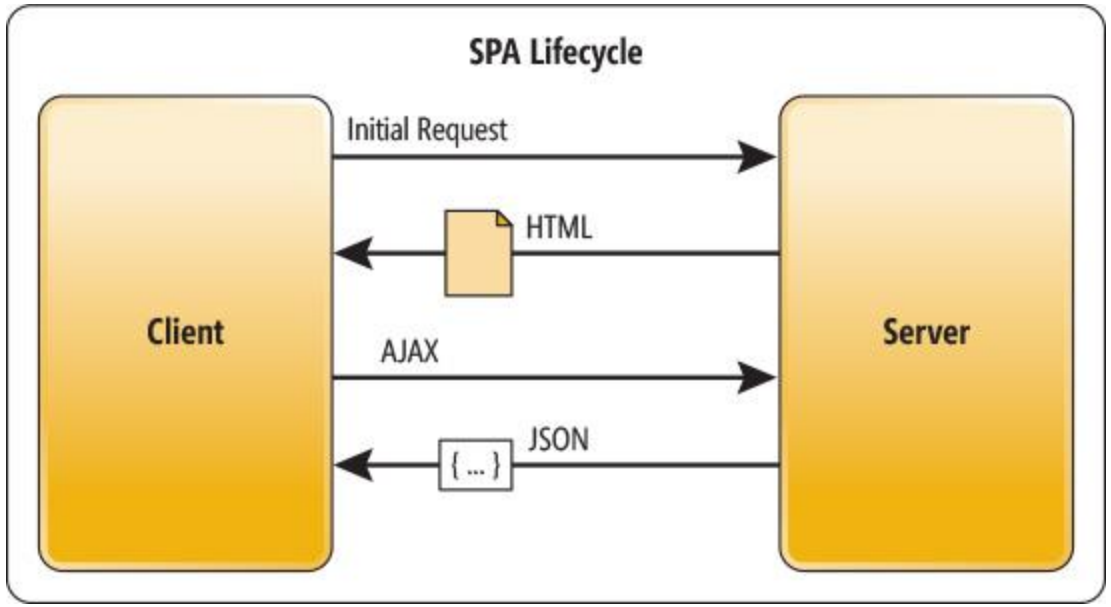
- **CSSOM(Cascading Style Sheets Object Model)**
  - DOM에 CSS를 입힌 것

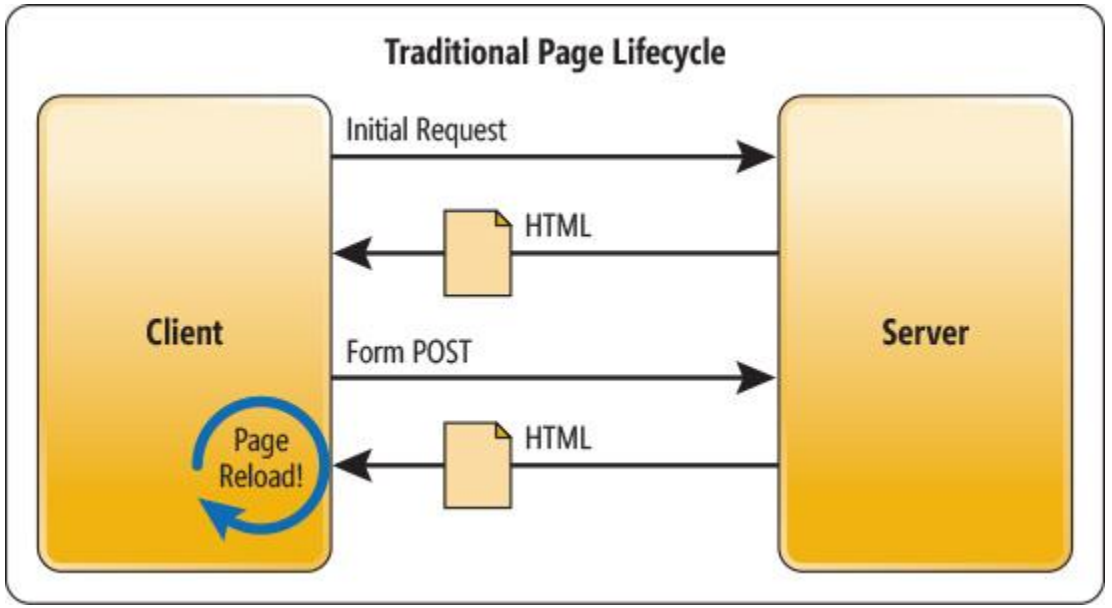
```html
<!doctype html>
<html land="ko">
  <head>
    <title>My first web page</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p style="display: none;">How are you?</p>
  </body>
</html>
```

```
html
└head
  └title
    └My first web page
body
└h1
  └Hello, world!
```

SPA와 기존 웹 시스템(MPA)의 차이

SPA(Single Page Application) VS MPA(Multi Page Application)



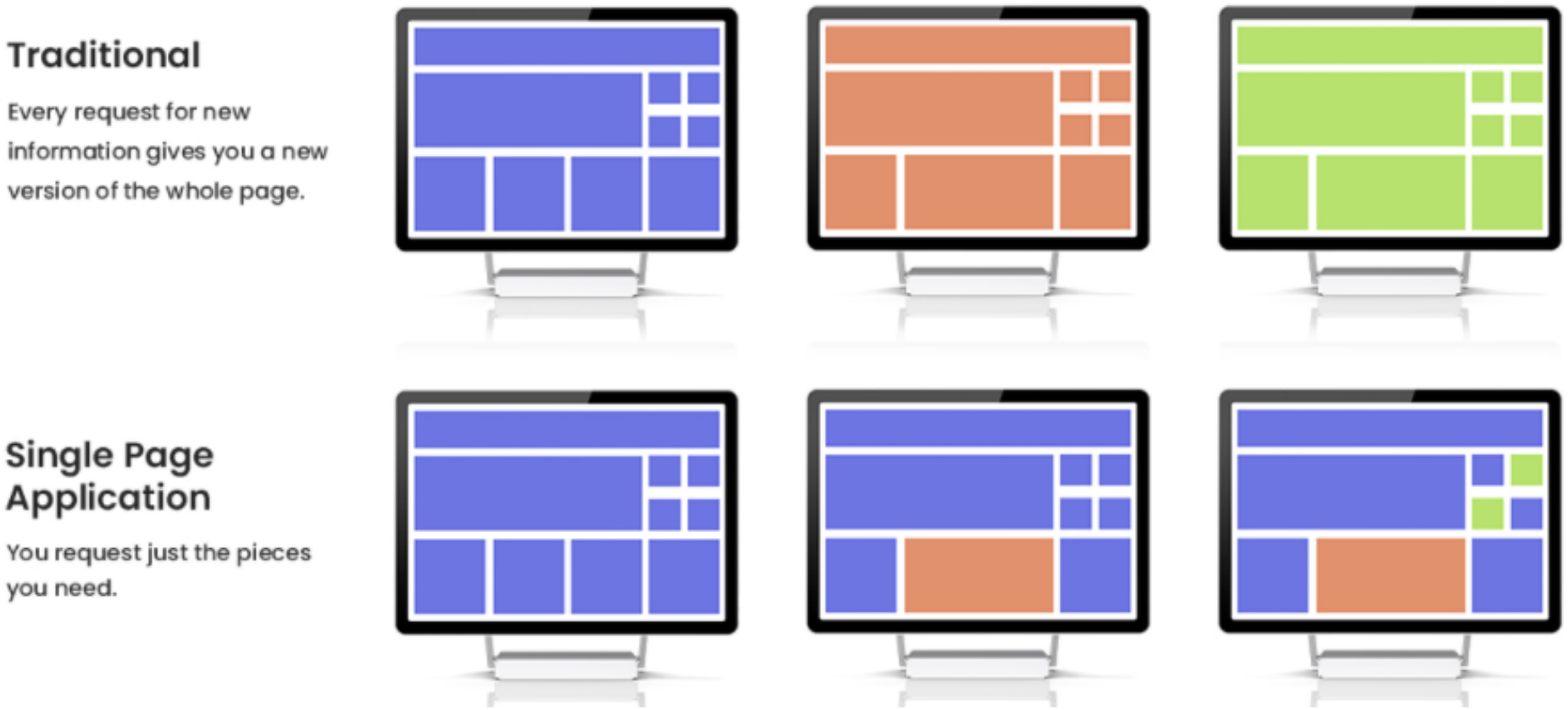


MPA는 전통적인 웹 애플리케이션 개발 방식(jsp, php 등의 웹 서버 언어로 구축된 웹사이트)

SPA와 기존 웹 시스템(MPA)의 차이

SPA(Single Page Application) 장점

- 사용자 경험 향상
  - 쇼핑사이트 등에서는 표시 속도 매출과 직결됨
  - 표시 속도가 0.1초 느려질 때 매출이 1% 감소, 1초 빨라질 때 10% 증가
- 컴포넌트 분리가 쉬워져 개발 효율 향상
  - 화면 요소들을 컴포넌트로 정의하여 재사용함



가상 DOM의 필요성과 리액트

- DOM 구조 변경시마다 새로 렌더링해야 하므로 비용 발생
- 화면 구조가 복잡할 경우 코드가 비대해져 문제 해결이 어려움 발생
  - 가상 DOM을 이용하면 변경된 부분만 실제 DOM에 반영함

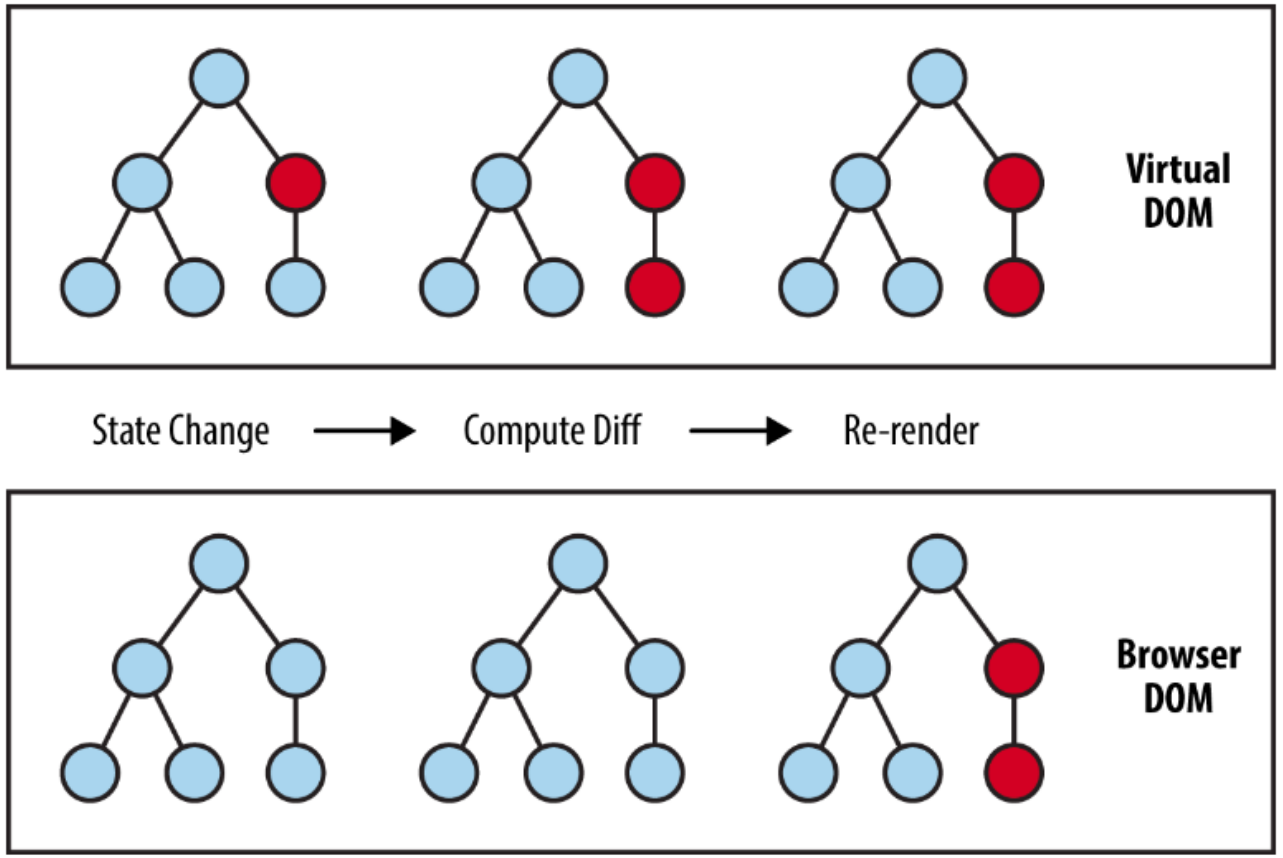
- 리액트는 새로 그려야 될 때 변경 부분만 새로 그린다.

React는 다음과 같은 경우에 리렌더링한다.
1. Props가 변경되었을 때
2. State가 변경되었을 때
3. forceUpdate() 를 실행하였을 때.
4. 부모 컴포넌트가 렌더링되었을 때

가상 DOM의 필요성과 리액트

- 리액트에서의 가상 DOM

  만약 상태나 속성값이 변경된 경우, 변경된 값으로 React는 가상의 돔을 그린다. 새로 그린 **Virtual DOM**과 **Real DOM**을 비교하여 변경된 사항만 반영하여 해당 내용을 실제 돔에서 수정한 이후 새로운 화면을 렌더링 한다.



패키지 관리자의 필요성

- JS에서 JS를 불러와 사용하므로 의존관계 발생, 개발 효율 저하 및 유지 보수 어려움
  - 해결: npm, yarn 등의 패키지 관리자 이용
    - 패키지 관리, 설치, 업그레이드 전담함
    - npm install [패키지명]
    - yarn add [패키지명]
    - package.json, package-lock.json(yarn.lock)

모듈 핸들러, 트랜스파일러

- 모듈 핸들러
  - 개발할 때에는 파일 나누어 작성
  - 모듈 핸들러는 프로덕션용으로 빌드할 때 사용.
    - 개발 할 때 여러 개의 파일로 나누어 작성된 것을 파일 하나로 모으는 과정에서의 의존 관계를 해결하는 기능 제공함
    - 예) 웹팩
- 트랜스파일러
  - 자바스크립트 표기법으로 브라우저에서 실행할 수 있는 형태로 변환하는 기능
  - 리액트는 js 파일에 jsx 표기법이라 부르는 특수한 규칙 적용하여 코드 기술
  - 이러한 코드를 브라우저가 인식할 수 있는 형태로 자동 변환함
  - 예) 바벨

웹페이지 동적(Dynamic)으로 만들기-DOM API

- 웹을 동적으로 제어한다는 것은 DOM을 제어한다는 것
  - DOM API 를 통해서 DOM 조작: DOM에 요소 추가/수정/삭제

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

웹페이지 동적(Dynamic) 만들기

- Document Object Model
  - HTML을 해석해서 트리 구조로 나타낸 것.
  - DOM 예제

```
//순수한 자바스크립트 코드:  id=popcorn을 가진 요소 아래에 Hello World!!라고 설정한 p 태그를 삽입하기
        var pElement = document.createElement("p");

        pElement.textContent = "Hello World!!";

        var divElement = document.getElementById("popcorn");

       divElement.appendChild(pElement);
```

```
// 제이쿼리: id=popcorn을 가진 요소 아래에 Hello World!!라고 설정한 p 태그를 삽입하기

const pElement = $('<p>').text('Hello World!!');

$('#popcorn').append(pElement);
```

첫번째 예제: 자바스크립트로 “Hello World!” 화면 출력

```
<body>
  <!-- id가 'popcorn' 인 div 요소 →
  <div id="popcorn"></div>

  <!- 자바스크립트 코드를 포함하는 스크립트 →
  <script>
    // DOMContentLoaded 이벤트를 사용하여 DOM이 완전히 로드된 후에 코드를 실행합니다.
    document.addEventListener("DOMContentLoaded", function () {
      // 1. <p> 요소를 생성합니다.
      var pElement = document.createElement("p");

      // 2. <p> 요소에 텍스트 내용을 추가합니다.
      pElement.textContent = "Hello World!!";

      // 3. id가 'popcorn'인 <div> 요소를 찾습니다.
      var divElement = document.getElementById("popcorn");

      // 4. <div> 요소 안에 <p> 요소를 추가합니다.
      divElement.appendChild(pElement);
    });
  </script>
</body>
```

첫번째 예제: 자바스크립트로 “Hello World!” 화면 출력

```
<body>
  <!-- id가 'popcorn' 인 div 요소 →
  <div id="popcorn"></div>

  <!- 자바스크립트 코드를 포함하는 스크립트 →
  <script>
    // DOMContentLoaded 이벤트를 사용하여 DOM이 완전히 로드된 후에 코드를 실행합니다.
    document.addEventListener("DOMContentLoaded", function () {
      // 1. <p> 요소를 생성합니다.
      var pElement = document.createElement("p");

      // 2. <p> 요소에 텍스트 내용을 추가합니다.
      pElement.textContent = "Hello World!!";

      // 3. id가 'popcorn'인 <div> 요소를 찾습니다.
      var divElement = document.getElementById("popcorn");

      // 4. <div> 요소 안에 <p> 요소를 추가합니다.
      divElement.appendChild(pElement);
    });
  </script>
</body>
```

script.js 파일로 옮기기

모던 자바스크립트 문법

모던 자바스크립트 기능

- var 이용한 변수 선언의 문제점

```
<script>
  var val1 = "var 변수";
  console.log(val1);

  val1 = " var 변수 덮어쓰기";
  console.log(val1);

  var val1 = "변수 재선언 가능";
  console.log(val1);
</script>
```

모던 자바스크립트 기능

- let 이용한 변수 선언

```
<script>
    let val2 = "let 변수";
    console.log(val2);

    val2 = " let 변수 덮어쓰기";  //덮어쓰기 가능
    console.log(val2);

    let val2 = "변수 재선언";    //재선언 불가능
    console.log(val2);          //SyntaxError: Identifier 'val2' has already been declared
</script>
```

모던 자바스크립트 기능

- const 이용한 변수 선언

```
<script>
   const val3 = "const 변수";
   console.log(val3);

   val3 = " const 변수 덮어쓰기"; //TypeError: Assignment to constant variable.
   console.log(val3);

   const val3 = “const 변수 재선언"; //에러, 재선언 불가능
   console.log(val3);
 </script>
```

모던 자바스크립트 기능

- const로 정의한 변수

- 프리미티브(primitive) 타입의 데이터는 덮어쓰기 불가능함
  - 논리값, 수치, 문자열 등
- 객체(object) 타입의 데이터는 const로 정의해도 값 변경 가능함.
  - 배열, 객체, 함수 등

예) 객체 속성값 변경 및 추가

```
<script>
  //객체 정의
  const obj1 = {
    name: "popcorn",
    age: 24,
  };
  console.log(obj1);

  //속성값 변경
  obj1.name = "팝콘";
  console.log(obj1);

  //속성 추가
  obj1.address = "Seoul";
  console.log(obj1);
</script>
```

모던 자바스크립트 기능

예) 배열값 변경 및 추가

```
<script>
    //배열 정의
    const arr1 = ["dog", "cat"];
    console.log(arr1);

    //첫번째 값 변경
    arr1[0] = "bird";
    console.log(arr1);

    //값 추가
    arr1.push("monkey");
    console.log(arr1);
</script>
```

템플릿 문자열

- 문자열 안에서 변수를 전개하기 위한 새로운 표기법
- 기존 문자열과 변수 결합: + 연산자 이용
- 템플릿 문자열 이용: 문자열을 ``(역따옴표)로 문자열 감싸고, ${}에 자바스크립트 입력

```
<script>
  //기존의 문자열과 변수 결합 방법
  const name = "팝콘";
  const age = 24;

  //내이름은 팝콘입니다. 나이는 24세입니다. 를 출력할 경우
  const message =
    "내이름은 " + name + "입니다. 나이는 " + age + "세 입니다.";
  console.log(message);

  //템플릿 문자열 이용
  const message2 = `내이름은 ${name}입니다. 나이는 ${age}세 입니다.`;
  console.log(message2);
</script>
```

템플릿 문자열

```
<script>
  //함수 호출과 계산 실행
  function sayHello() {
    return "안녕하세요";
  }

  const month = 1;

  const message3 = `여러분 ${sayHello()}! 오늘부터 ${month * 8}월입니다.`;
  console.log(message3);
</script>
```

재사용할 수 있는 함수

- 함수: function 키워드로 정의
  - 예1. 두 수를 전달받아 더한 결과를 함수안에서 결과 출력하기

```
<script>
  //addNums(a, b)
  function addNums(a, b){
    console.log(a + b);
  }
  addNums(2, 3);
</script>
```

  - 예2. 두수를 전달받아 더한 결과를 리턴하기

```
  //addNums(a, b)
  function addNums(a, b){
    return a+b;
  }
  console.log(addNums(2, 3));
```

재사용할 수 있는 함수

- 매개변수에 기본값 지정하기
  - 예1.

```
function multiple(a, b=5, c=10){
  return a*b+c;
}
console.log(multiple(2, 3, 4));
console.log(multiple(2, 3 ));
console.log(multiple(2));
```

함수 표현식

- 익명 함수 선언: 함수 이름이 없다. 함수 자체가 식(expression)이다.

```
function (a,b){
  return a*b;
}
```

- 함수를 변수에 할당, 다른 함수의 매개변수로 사용 가능

```
const mul = function (a,b){
  return a*b;
}
console.log(mul(2, 3));
```

함수 표현식

- 화살표 함수(=> 표기): ES6 버전부터 사용. 익명함수에서만 사용 가능.

```
형식:  (매개변수) => { 함수 내용 }
```

- 매개변수 없을 경우

```
const hi = function() {
   return “안녕하세요”;
}
```

```
const hi = () => {
   return “안녕하세요”;
}
```

```
//함수 실행문이 한 개일 경우, 중괄호와 리턴키워드 생략가능
const hi = () => “안녕하세요”;
```

함수 표현식

- 매개변수가 1개인 경우

```
const hi = function(user) {
  return `${user}님 안녕하세요`;
}
```

```
const hi = user => {   //매개변수의 괄호 생략 가능
   return `${user}님 안녕하세요`;
}
```

```
const hi = user => “안녕하세요”;
```

함수 표현식

- 매개변수가 2개 이상인 경우

```
const sum = function(a, b) {
  return a + b;
}
```

```
const sum = (a, b) => {
   return a+b;
}
```

```
const sum = (a, b) => a+b;
```

화살표 함수 ()=>{}

1. 다음과 같은 익명 함수가 정의될 경우 박스에 들어갈 코드는?

```
//화살표 함수 정의
const func3 = function (value) => {
  return value;
};

                                    //콘솔 출력    "func3입니다."
```

2. 위의 함수를 화살표 함수로 재정의 하시오.

```
//화살표 함수 정의
const func3 =
```

3. 아래 코드를 작성하여 실행했을 때 오류가 발생한다. 오류 부분을 해결하시오.

```
const func5 = value1, value2 => {
     value1+value2;
};

console.log(func5(2,3));  //출력 5
```

화살표 함수 ()=>{}

- 반환값이 여러행일 경우에는 ()로 감싼뒤 단위 행과 같이 모아서 반환할 수 있다.

```
//()를 이용해 한행으로 모으기
const func9 = (val1, val2) => ({
  name: val1,
  age: val2,
});

console.logfunc9("팝콘", 24));
```

- 다음과 같은 출력 결과가 나오도록 코드 작성하시오.

```
팝콘님의 나이는 24입니다.
```

분할 대입 {} []

- 분할 대입은 객체나 배열로부터 값을 추출하기 위한 방법

```
<script>
  //분할 대입을 이용하지 않고 문자열을 출력
  const myProfile = {
    name: "팝콘",
    age: 24,
  };

  const message = `내 이름은 ${myProfile.name}입니다. 나이는 ${myProfile.age}세 입니다.`;

  console.log(message);
</script>
```

위의 예와 같이 분할 대입을 이용하지 않을 경우, 객체 변수의 경우 myProfile.name, … 등과 같이 객체 변수 참조로 출력함. 객체 변수명이 많아질 경우 변수 참조하기가 번거롭게 된다. 이 때, 분할 대입을 이용한다.

분할 대입 {} []

- 분할 대입 이용

```
<script>
  const myProfile = {
    name: "팝콘",
    age: 24,
  };

  //분할 대입 이용
  const { name, age } = myProfile;
  const message2 = `내 이름은 ${name}입니다. 나이는 ${age}세 입니다.`;
  console.log(message2);
</script>
```

분할 대입 {} []

- 분할 대입 이용

```
<script>
  const myProfile = {
    name: "팝콘",
    age: 24,
  };

  //일부만 추출
  //const { age } = myProfile;
  //console.log(`나이는 ${age}세 입니다.`);

  //순서를 바꾸어 추출할 수 있다.
  const { age, name } = myProfile;
  console.log(`내 이름은 ${name}입니다. 나이는 ${age}세 입니다.`);

  //추출한 속성에 별명 지정할 수 있다.
  //콜론으로 다른 변수명 이용
  const { name: newName, age: newAge } = myProfile;
  console.log(`내 이름은 ${newName}입니다. 나이는 ${newAge}세 입니다.`);

</script>
```

배열 분할 대입 {} []

```
<script>
  const myProfile = ["팝콘",24];
  //인덱스로 배열의 각 요소 접근
  const message = `내 이름은 ${myProfile[0]}입니다. 나이는 ${myProfile[1]}세 입니다.`);
  console.log(message);
  //배열에 분할 대입
  //변수 선언부에 [] 사용하여 배열에 저장된 순서에 임의의 변수명 설정. 순서 변경할 수 없음.
  const [name, age] = myProfile;
  const message = `내 이름은 ${name}입니다. 나이는 ${age}세 입니다.`);
  console.log(message);
  //배열에서 필요한 요소만 추출
  //첫번째 요소만 필요한 경우
  const [name] = myProfile;
</script>
```

디폴트값=

- 디폴트값은 함수의 인수나 객체를 분할 대입할 경우 설정하여 사용
- 값이 존재하지 않을 경우, 초기값 설정하므로 안전한 처리 가능.

1. 인수의 디폴트값

```
<script>
  //메시지 출력 함수 예
  const sayHello = (name) => console.log(`${name}님, 안녕하세요`);

  sayHello("팝콘"); //팝콘님 안녕하세요

  //실행시 인수가 전달되지 않을 경우
  sayHello(); //undefiend님 안녕하세요

  //디폴트값 지정하여 의미를 알 수 없는 메시지 출력을 방지한다.
  const sayHello2 = (name="게스트") => console.log(`${name}님, 안녕하세요`);
  sayHello2(); //게스트님 안녕하세요
  sayHello2("팝콘"); //팝콘님 안녕하세요
</script>
```

디폴트값=

2. 객체 분할 대입의 디폴트값

```
<script>
    //객체 분할 대입의 디폴트값 예
    const myProfile = {
      age: 24,
    }

    //존재하지 않는 name 변수
    const {name} = myProfile;
    const message = `${name}님 안녕하세요`;
    console.log(message);  //undefined님 안녕하세요

    //분할 대입의 디폴트값을 설정
    //const { name="게스트" } =  myProfile;
    //const message = `${name}님 안녕하세요`;
    //console.log(message);  //게스트님 안녕하세요
</script>
```

스프레드 연산자(Spread Operator): ...

- 배열이나 객체에 이용할 수 있는 표기법

1. 요소 전개: 배열에 적용-내부 요소를 순차적으로 전개한다.

```
<script>
  //스프레드 연산자 ...
  const arr1 = [1, 2];
  console.log(arr1); //[1,2]
  console.log(...arr1); //1 2

  //일반적인 함수와 스프레드 구분 비교
  const summaryFunc = (num1, num2) => num1 + num2;

  //함수 호출시 일반적인 방식의 배열값 전달
  console.log(`배열값 전달 방식: ${summaryFunc(arr1[0], arr1[1])}`);

  //스프레드 구문 방법으로 배열값 전달
  console.log(`스프레드 구문 전달: ${summaryFunc(...arr1)}`);
</script>
```

스프레드 연산자(Spread Operator): ...

2. 요소 모으기

```
const arr2 = [1, 2, 3, 4, 5];

//배열의 분할 대입시 남은 요소를 '모은다'
const [num1, num2, ...arr3] = arr2;
console.log(`num1: ${num1}`);  //1
console.log(`num2: ${num2}`);  //2
console.log(`arr3: ${arr3}`);  //[3,4,5]
```

3. 요소 복사와 결합
: 배열에 적용

```
const arr4=[10,20];
const arr5=[30,40];

//스프레드 구문 이용한 배열 복사
const arr6 = [...arr4];
console.log(arr4);
console.log(arr6);

//스프레드 구문 이용한 배열 결합
const arr7 = [...arr4, ...arr5];
console.log(arr7);
```

스프레드 구문...

3. 요소 복사와 결합
   : 객체에 적용

```
const obj4 = { val1: 10, val2: 20 };
const obj5 = { val3: 30, val4: 40 };

//스프레드 구문 이용한 객체 복사
const obj6 = { ...obj4 };
console.log(obj6);

//스프레드 구문 이용한 객체 결합
const obj7 = { ...obj4, ...obj5 };
console.log(obj7);
```

오브젝트 타입 변수의 등호 복사 주의사항

- 오브젝트 타입 변수의 경우, 등호를 이용한 복사는 **주소 복사**가 된다.
- 따라서 복사된 배열의 요소에 조작이 일어날 경우, 원래 배열도 바뀐다.

```
//스프레드 구문 이용한 객체 결합
const arr4 = [10, 20];

//등호로 배열 복사
const arr8 = arr4;
console.log(arr8);

//배열 원소 수정
arr8[0] = 100;
console.log(arr4);  //[100, 20]
console.log(arr8);  //[100, 20]
```

```
const arr4 = [10, 20];

//스프레드 구문 이용한 객체 복사
const arr9 = [...arr4];
arr9[0] = 100;
console.log(arr4);  //[10, 20]
console.log(arr8);  //[100, 20]
```

- 스프레드 구문 복사의 경우 완전히 새로운 배열을 만들게 된다.

객체 생략 표기법

- 쇼트핸드(shorthand: 생략표현)
  - 객체의 속성명과 설정할 변수명이 같으면 생략 가능
  - 예)

```
//속성명과 변수명이 같은 경우
const name = "팝콘";
const age = 24;

//user 객체 정의(속성은 name, age)
const user = {
  name: name,
  age: age,
};
console.log(user); //{name: "팝콘", age: 24}

//속성명과 변수명이 같은 경우- 생략표기법(콜론 생략)
const user1 = {
  name,
  age,
};
console.log(user1); //{name: "팝콘", age: 24}
```

map함수, filter함수

- **map()** 함수는
- 배열을 순회하며 지정된 콜백 함수를 적용하여 각 요소를 변환하고, 그 변환된 값을 모아서 새로운 배열로 반환하는 역할을 수행한다.
- **map()** 함수 매개변수: **value, index, array**

1. map() 함수 활용 예

map()함수를 호출할 때, 지정된 함수는 배열 내의 모든 요소 각각에 대하여 2배하여 리턴, map함수는 리턴된 값들을 모아서 **새로운 배열로 반환**.

```
const array1 = [1, 4, 9, 16];

const map1 = array1.map((x) => x * 2);     //x 는 array1의 요소값을 갖는 매개변수
console.log(map1);
```

map, filter

2. map() 함수 활용 예

- map() 함수를 사용하여 객체 배열에서 특정 속성이나 값만 추출하여 새로운 배열을 생성할 수 있다.

```
const users = [
    {id: 1, name: "Alice"},
    {id: 2, name: "Bob"},
    {id: 3, name: "Charlie"}
];

const names = users.map((user) => {
    return user.name;
});

console.log(names);
```

map, filter

3. map() 함수 활용 예

- 조건에 따라 배열의 각 요소를 대체 값으로 변경하여 새로운 배열을 생성하는 경우 map() 함수를 활용할 수 있다.

```
const numbers = [1, 2, 3, 4, 5];

const modifiedNumbers = numbers.map(function(number) {
    if (number % 2 === 0) {
        return "Even";
    } else {
        return "Odd";
    }
});

console.log(modifiedNumbers);
```

map, filter

- map 함수 이용한 인덱스 다루기

```
const nameArr = ["팝콘", "뷰", "앤트"];

//map 함수의 인수 이용하여 요소 순서대로 추출
nameArr.map((name, index) =>
  console.log(`${index + 1}번째 손님: ${nameArr[index]} `)
);
```

```
1번째 손님: 팝콘
2번째 손님: 뷰
3번째 손님: 앤트
```

map, filter

- 실습: map() 함수를 활용하여 주어진 성적 배열에서 60점 이상인 학생의 명단 배열을 만드시오.

```
const users = [
  { name: "홍길동", score: 88 },
  { name: "강감찬", score: 90 },
  { name: "이순신", score: 55 },
  { name: "황미나", score: 75 },
  { name: "길동이", score: 92 },
  { name: "정철", score: 35 },
];
```

1단계: users 배열에서 score가 60이상인 pass 배열 생성

2단계: pass 배열 항목이 not null 또는 true 이면 passList 배열에 추가

3단계: passList 출력

map, filter

- 실습: map() 함수의 value와 index 매개변수를 활용하여 주어진 성적 배열에서 60점 이상인 학생의 이름과 성적을 출력하시오.

```
const users = [
  { name: "홍길동", score: 88 },
  { name: "강감찬", score: 90 },
  { name: "이순신", score: 55 },
  { name: "황미나", score: 75 },
  { name: "길동이", score: 92 },
  { name: "정철", score: 35 },
];
```

1단계: users 배열에서 score가 60이상인 indexes 배열 생성 users.map((user, index)=> { });

2단계: indexes 배열 항목이 not null 이면 users 항목 출력

map, filter

- filter 함수 이용
  - map 함수와 이용법 유사. return 뒤에 조건식 입력해서 일치하는 것만 반환함.

예) 배열에서 홀수만 추출하기

```
const numArr = [1, 2, 3, 4, 5];

//배열.filter() 이용, 홀수(2로 나누어 나머지가 1인 경우)만 추출
const numArr2 = numArr.filter((num) => {
  return num % 2 === 1;
});

console.log(numArr2); [1,3,5]
numArr.filter((num) => console.log(num % 2 == 1)); //true, false, true, false, true
```

map, filter

실습: filter()와 map() 함수를 활용하여 주어진 성적 배열에서 60점 이상인 학생 명단을 배열로 만들고 출력하시오.

```
const users = [
  { name: "홍길동", score: 88 },
  { name: "강감찬", score: 90 },
  { name: "이순신", score: 55 },
  { name: "황미나", score: 75 },
  { name: "길동이", score: 92 },
  { name: "정철", score: 35 },
];
```

1단계: filter()함수 이용하여 users 배열에서 score가 60이상인 항목에 대한 배열 생성

users.filter((user)=> { });

2단계: 1단계에서 생성된 배열에 map()함수 이용하여 name 항목만 배열로 생성후 출력

삼항연산자

- 리액트에서 자주 사용하는 연산자
- 문법: **조건 ?** 조건이 true일 때의 처리 : 조건이 false 일 때의 처리

```
//예시 1
const val1 = 1 > 0 ? "true입니다." : "false입니다.";
console.log(val1); //true입니다.


//예시 2
const result = (num) => (num % 2 === 0 ? "even" : "odd");
console.log(result(3));
```

삼항연산자

```
//예시 3 - 입력값에 대한 메시지 출력
//입력값이 숫자인 경우 세 자리마다 콤마로 구분한 표기로 변환 출력,
//숫자가 아닌 경우에는 주의 메시지 출력하는 함수
const printFormattedNum = (num) => {
  const formattedNum =
    typeof num === "number"
      ? num.toLocaleString()
      : "숫자를 입력하십시오.";
  console.log(formattedNum);
};

printFormattedNum(1300);
printFormattedNum("1300");
```

```
1,300
숫자를 입력하십시오.
```

삼항연산자

실습: 2개의 정수의 합이 100을 넘는지를 판단하는 코드 작성

```
//두 인수의 합이 100을 넘는지 판정하는 함수

console.log(checkSumOver100(50, 40));   //허용범위 안입니다.
console.log(checkSumOver100(50, 70));   //100을 넘었습니다.
```

논리연산자 &&, ||

- 논리연산자 이용한 조건 분기
  - || 논리 연산자- 왼쪽이 false라고 판정하면 오른쪽을 반환한다.
  - null, undefined, 0 등은 자바스크립트에서 false로 판정함.

```
//논리연산자 이용한 조건 분기
const flag1 = true;
const flag2 = false;

if (flag1 || flag2) {
  console.log("두 플래그 중 어느 하나는 true입니다.");
}
if (flag1 && flag2) {
  console.log("두 플래그 모두 true입니다.");
}
//null 설정할 경우의 출력 결과는?
const num = null;
const fee = num || "금액을 설정하지 않았습니다.";
console.log(fee);   //출력?

const num2 = 100;
const fee2 = num2 || "금액을 설정하지 않았습니다.";
console.log(fee2);   //출력?
```

논리연산자 &&, ||

- 논리연산자 이용한 조건 분기
  - && 논리 연산자- 왼쪽이 true라고 판정하면 오른쪽을 반환한다.

```
//논리연산자 이용한 조건 분기
const num3 = 100;
const fee3 = num2 && "무언가가 설정되었습니다.";
console.log(fee3);    //출력?
```