대학 주도형 아카데미 K-디지털 트레이닝
AI 데이터분석 풀스택 웹 개발자 양성과정

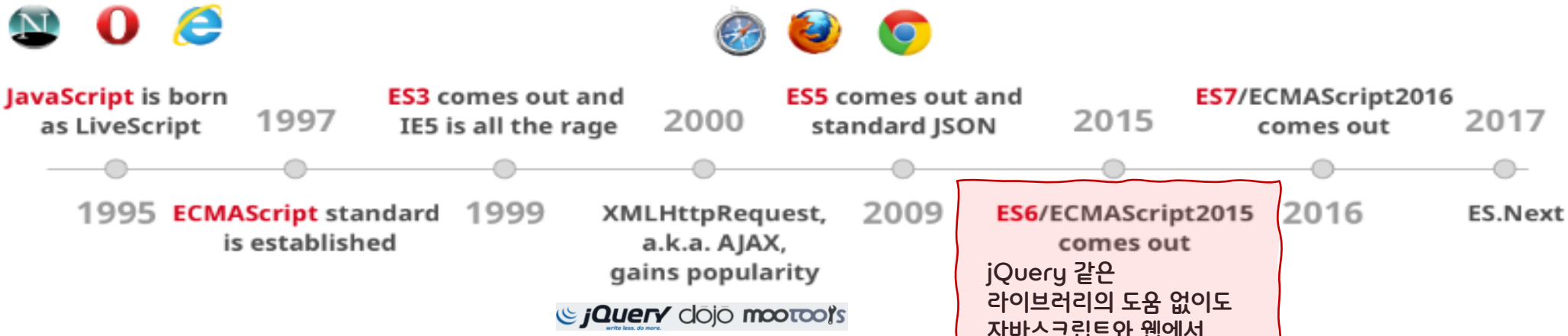
JavaScript

김경민

pnumin@pusan.ac.kr

JavaScript

- 객체(Object) 기반의 스크립트 언어로 웹 브라우저에서 해석되는 인터프리터 언어
  - 웹 브라우저에 인터프리터가 내장되어 주로 클라이언트 측 프로그래밍 작성
  - Node.js와 같은 프레임워크를 사용하면 서버 측 프로그래밍에서도 사용 가능



jQuery 같은 라이브러리의 도움 없이도 자바스크립트와 웹에서 제공하는 API들 만으로도 모든 웹에서 표준화된 웹사이트 개발

- 자바스크립트 버전
  https://www.w3schools.com/js/js_versions.asp

JavaScript

- 자바스크립트 엔진(JavaScript engine)
  - 자바스크립트 코드를 실행하는 데 사용되는 프로그램 또는 인터프리터
  - 자바스크립트 코드를 해석하고 실행하는 핵심 역할
  - 최근에는 성능 향상을 위해 JIT(Just-in-Time) 컴파일을 사용하는 엔진이 많이 사용되고 있음
- Just-in-time(JIT) 컴파일 방식
  - 프로그래밍 언어를 실행하는 동안, 즉 "실시간"으로 코드가 실행되기 직전에 컴파일하는 기술
  - 인터프리터와 전통적인 컴파일러의 장점을 결합한 것으로, 프로그램 실행 중 성능을 최적화하고 속도를 높이기 위해 사용
  - 구글의 V8 엔진이나 Mozilla의 SpiderMonkey 같은 엔진에서 널리 사용

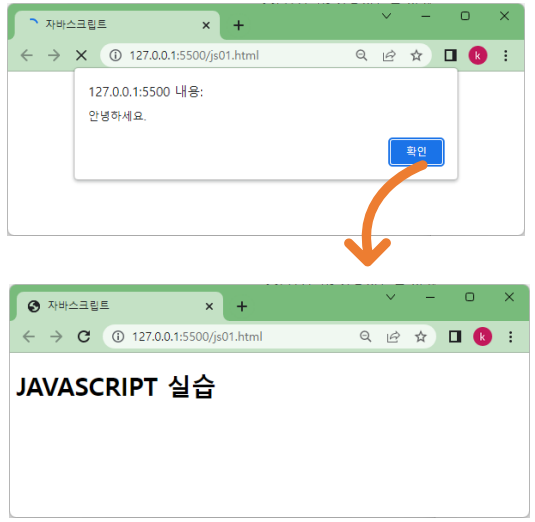
JavaScript

- 자바스크립트로 할 수 있는 일
  - HTML 및 스타일 조작
    - 웹 페이지에 새로운 HTML 요소를 추가하거나, 기존의 HTML 요소 및 스타일을 동적으로 수정
  - 사용자 인터랙션 처리
    - 마우스 클릭, 포인터 움직임, 키보드 입력 등과 같은 사용자 행동에 실시간으로 반응
  - 네트워크 통신
    - AJAX나 COMET 등의 기술을 사용해, 원격 서버에 요청을 보내거나 파일을 다운로드 및 업로드하는 작업을 수행
  - 쿠키 및 사용자 메시지 관리
    - 쿠키를 가져오거나 설정하고, 사용자에게 질문을 건네거나 메시지를 표시
  - 클라이언트 측 데이터 저장
    - 로컬 스토리지를 사용하여 클라이언트 측에 데이터를 저장하고, 이를 필요할 때 불러올 수 있음

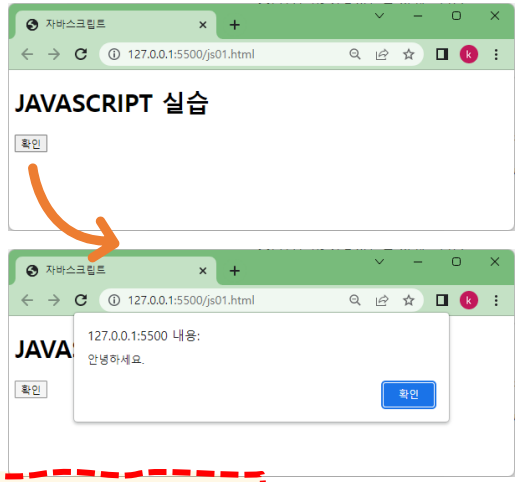
JavaScript 시작하기

- 내부 자바스크립트 코드로 작성
  - <script> 요소로 삽입

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>자바스크립트</title>
  <script>
    alert("안녕하세요.") ;
  </script>
</head>
<body>
  <h1>JAVASCRIPT 실습</h1>
</body>
</html>
```




```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>자바스크립트</title>
  <script>
    function hello() {
      alert("안녕하세요.") ;
    }
  </script>
</head>
<body>
  <h1>JAVASCRIPT 실습</h1>
  <input type="button"" value="확인" onclick="hello();">
</body>
</html>
```



HTML CSS JavaScript React JS

JavaScript 시작하기

- **외부 자바스크립트 파일 삽입하여 작성**
  - <script src=“”> 요소의 src 속성을 이용하여 외부 자바스크립트 파일 추가
  - 단, 작성된 외부 자바스크립트 파일의 확장자는 .js로 사용

```html
<!DOCTYPE html>
<html lang="ko">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>자바스크립트 기초</title>
  <!-- 외부 스크립트 참고 -->
  <script src='./03_1.js'></script>

  <!-- 외부 스타일 -->
  <link rel='stylesheet' type='text/css' href='../index.css'>
  <link rel='stylesheet' type='text/css' href='./03.css'>
</head>

<body>
  <main>
    <header>
      <h1>자바스크립트 기초</h1>
      <div>홈</div>
    </header>
    <section>
      <div>
        <button onclick='handleRand();'>랜덤수 생성</button>
      </div>
    </section>
  </main>
</body>

</html>
```

03_1.js

```js
const handleRand = () => {
  let n ;
  let s = '';

  for(let i=1; i < 8; i++) {
    n = Math.floor(Math.random()*45) + 1 ;
    s = i == 1 ? `${n}` : `${s},${n}` ;
  }

  alert(s) ;
}
```

화살표 함수

- **화살표 함수(Arrow function)**
  - ECMAScript 6(ES6)에서 도입
  - 자바스크립트에서 사용하던 기존의 함수 선언식이나 함수 표현식보다 좀 더 간결하며, 쉽게 사용하기 위해 도입
  - 화살표 함수만의 특별한 구문
    - **하나의 매개변수**인 경우 **소괄호 () 를 생략**할 수 있음
    - 함수 내부가 **단일 표현식(single expression)인 경우 return 키워드를 생략**할 수 있음
      - return 키워드를 생략할 경우 중괄호 {} 는 반드시 생략

```
function handleRand() {
    let n ;
    let s = '';

    n = Math.floor(Math.random()*6) + 1 ;
    s = `발생한 랜덤수 : ${n}`

    alert(s) ;
}
```

```
const handleRand = () => {
    let n ;
    let s = '';

    n = Math.floor(Math.random()*6) + 1 ;
    s = `발생한 랜덤수 : ${n}`

    alert(s) ;
}
```

변수/상수 선언

- 선언
  - 변수 선언 : let
  - 상수 선언 : const
- 명명규칙
  - 변수명에는 오직 문자와 숫자, 그리고 기호 $와 _로 작성
  - 첫 글자는 숫자가 될 수 없음
  - 대소문자 구분
- 한번만 선언

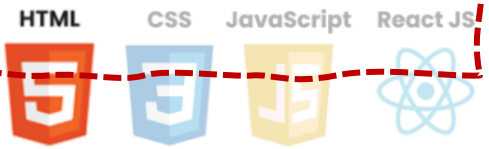
변수를 let으로 수정하면 오류

```
▶Uncaught                    js01.js:1
ReferenceError: Cannot access
'x' before initialization
    at js01.js:1:13
```

- 이전 변수 선언
  - 변수 선언 : var
- 호이스팅
  - 코드 실행전에 변수나 함수의 선언이 저장되어 선언 구문이 파일의 최상단으로 끌어올려져 선언문보다 참조나 호출이 먼저 나와도 동작할 수 있음

```
console.log(x)
var x = 1
console.log(x)
```

```
undefined                    js01.js:1
1                            js01.js:3
```



호이스팅(Hoisting)

- 자바스크립트 코드에서 변수와 함수의 선언이 코드의 상단으로 끌어올려지는 현상
  - 코드가 실행되기 전에 자바스크립트 엔진이 모든 변수와 함수의 선언을 해당 유효 범위의 최상단으로 이동
  - 자바스크립트 엔진은 코드를 실행하기 전에 실행 컨텍스트를 준비하는 과정에서 모든 선언문(예: var, let, const, function, class)을 스코프에 등록
  - 이로 인해 선언이 코드의 어느 위치에 있든 상관없이, 변수나 함수를 참조하거나 호출하는 코드가 앞서 있어도 오류 없이 동작

일시적 사각지대(Temporal Dead Zone, TDZ)

- 스코프의 시작부터 변수의 선언까지의 구간을 말하며, 이 구간에서는 변수를 참조할 수 없음
  - let, const, class 키워드를 사용한 선언은 호이스팅이 발생하지만 일시적 사각지대(Temporal Dead Zone; TDZ)에 빠지기 때문 호이스팅이 발생하지 않는 것처럼 동작
  - TDZ의 영향을 받지 않는 구문 : var, function, import 구문
    - var 키워드는 선언과 함께 undefined로 초기화되어 메모리에 저장되는데 let과 const는 초기화되지 않은 상태로 선언만 메모리에 저장

자료형

- 동적 타입(dynamically typed) 언어
  - 변수를 선언할 때 특정 자료형을 지정하지 않으며, 변수에 저장되는 값에 따라 자료형이 자동으로 결정
  - 변수에 저장된 값의 자료형이 언제든지 변경될 수 있음
- 자바스크립트의 주요 자료형
  - 숫자 : 정수, 부동 소수점 숫자 등의 숫자
  - 문자열 : 작은따옴표(')나 큰따옴표(")로 묶인 문자열
  - 불리언형 : true, false
  - 배열:대괄호로 묶이고 쉼표로 구분 된 여러 값을 포함하는 단일 객체
  - 객체:복잡한 데이터 구조를 표현
  - null : 알 수 없는 null 값을 위한 독립 자료형
  - undefined :할당되지 않은 undefined 값을 위한 독립 자료형
- 자료형 확인 : typeof(변수명)

연산자

비교연산자	이름	예제

===	엄격 일치 (정확히 같은가?)	5 === 2 + 4 // false
		'Chris' === 'Bob' // false
		5 === 2 + 3 // true
		2 === '2' // false
		숫자와 문자열은 다름 Copy to Clipboard

==	동등연산	0 == false // true
		'' == false // true
		다른 피연산자를 비교할 때 피연산자를 숫자형으로 바꾸기 때문에 발생

!==	불일치 (같지 않은가?)	5 !== 2 + 4 // true 'Chris' !== 'Bob' // true 5 !== 2 + 3 // false 2 !== '2' // true 숫자와 문자열은 다름 Copy to Clipboard
<	미만	6 < 10 // true 20 < 10 // falseCopy to Clipboard
>	초과	6 > 10 // false 20 > 10 // true

산술연산자	이름	예제
+	더하기	6 + 9
-	빼기	20 - 15
*	곱하기	3 * 7
**	거듭제곱	2 ** 4
%	나머지	10 % 2
/	나누기	10 / 5

- 일치연산자 ===를 사용하여 null과 undefined를 비교하면 거짓
- 동등연산자 ==를 사용하여 null과 undefined를 비교하면 참
- undefined를 다른 값과 비교하면 안됨

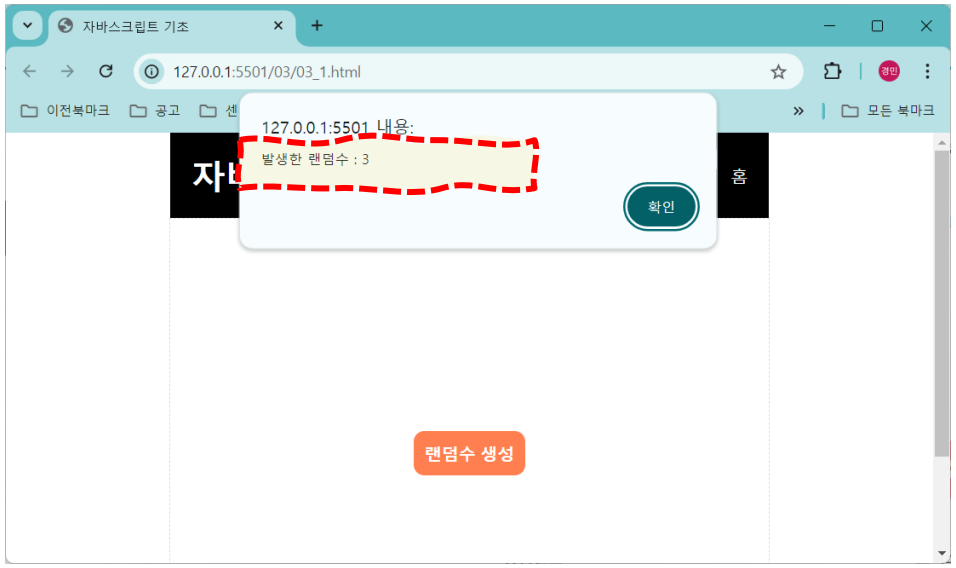
템플릿 문자열

- 백틱(`)으로 감싸면 문자열 안에 변수 사용 가능
  - ${변수}

```
const handleRand = () => {
  let n ;
  let s = '';

  n = Math.floor(Math.random()*6) + 1 ;
  s = `발생한 랜덤수 : ${n}`

  alert(s) ;
}
```



HTML CSS JavaScript React JS

문자열 기본 속성과 변환

- 문자열 길이
  - 문자열.length
- 대소문자 변환:
  - 문자열.toUpperCase(): 모든 문자를 대문자로 변환
  - 문자열.toLowerCase(): 모든 문자를 소문자로 변환
- 문자열에서 숫자 변환:
  - isNaN(): 유효한 숫자인지 검사
    - 숫자가 아니면 true를 반환하고 숫자이면 false
  - parseInt(): 문자열을 정수로 변환

```
//문자열
let str ;

//문자열 길이
str = "Hello";
console.log(str.length); // 5

//문자열 변환
str = "aBc" ;
console.log(str.toLowerCase()) ; //abc
console.log(str.toUpperCase()) ; //ABC

str = "123";
console.log(parseInt(str)); // 123
console.log(isNaN(str)); // false
```

문자열 순회

- 특정 문자 추출
  - 문자열[인덱스]
    - 인덱스를 사용하여 문자열에서 특정 문자를 추출
  - 문자열.charAt(인덱스)
    - 인덱스를 사용하여 특정 문자를 반환
- 문자열 분리
  - 문자열.split(“)
    - Split의 구분자를 이용하여 분리하면 배열 생성
- 문자열 순회
  - for (let c of 문자열)
    - 문자열의 각 문자를 순회

```
str = "Hello"

//인덱스를 활용한 문자열 추출
console.log(str[0]);                //H
console.log(str.charAt(0));         //H

//문자열 순회
for(let c of str) {
  console.log(c);                   // H, e, l, l, o
}

//문자열 인덱스와 값 순회
for(let [i,c] of str.split('').entries()) {
  console.log(i, c);                // 0 H, 1 e, 2 l, 3 l, 4 o
}
```

문자열 검색

- indexOf() 메서드
  - 문자열 내에서 특정 문자열이 처음으로 나타나는 위치를 반환
  - 만약 찾는 문자열이 없다면 -1을 반환
- includes() 메서드
  - 문자열이 포함되어 있는지 여부를 true 또는 false로 확인

```
//문자열 검색
str = "Hello World";

console.log(str.indexOf("World")); // 6
console.log(str.indexOf("world")); // -1

console.log(str.includes("Hello")); // true
console.log(str.includes("hello")); // false
```

문자열 특정 부분 추출

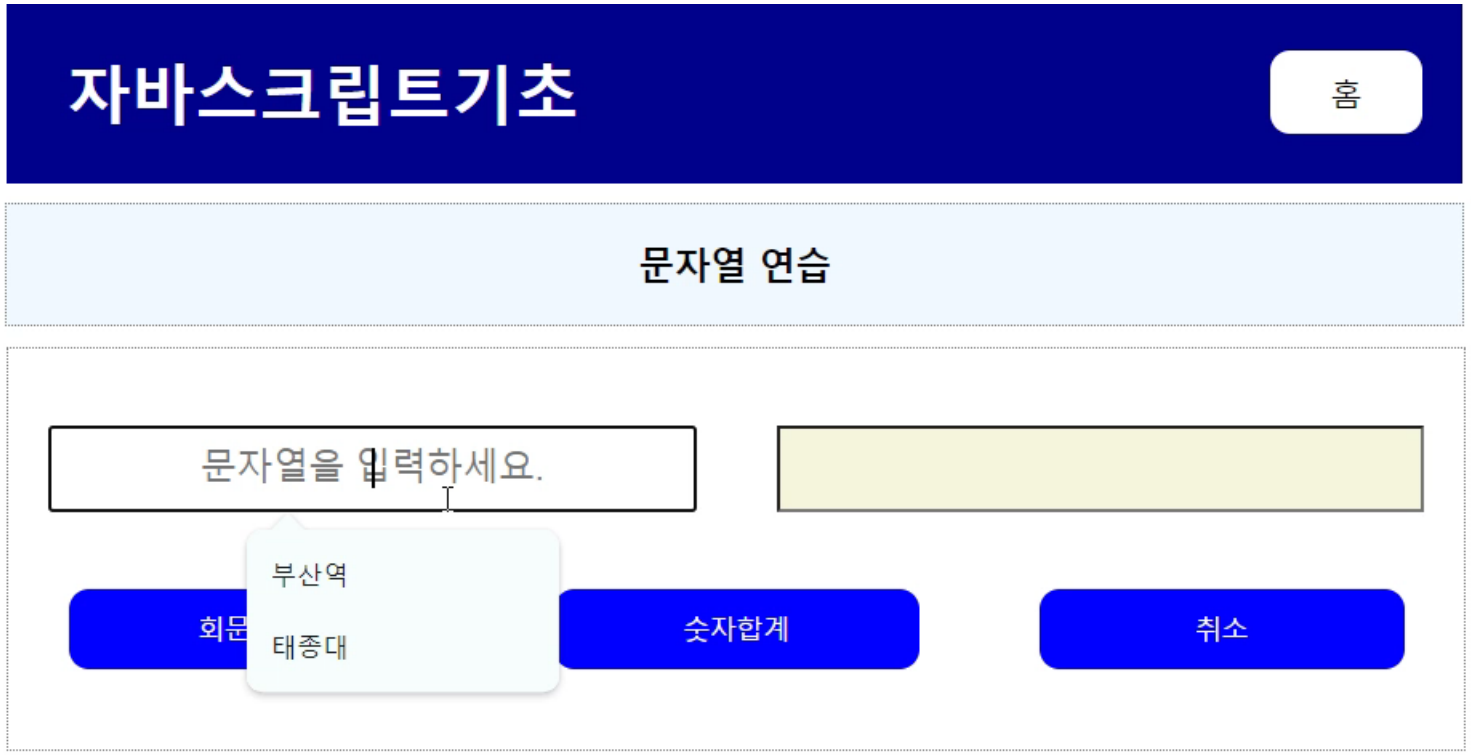
- slice(startIndex, endIndex) 메서드
  - 시작 인덱스(startIndex)와 종료 인덱스(endIndex)를 사용하여 문자열의 부분을 추출
  - endIndex 생략시 문자열 끝까지 추출
  - startIndex가 endIndex보다 크면 빈 문자열을 반환
  - 음수 인덱스를 허용하며, 음수 인덱스는 문자열의 끝에서부터 계산
- substring(startIndex, endIndex)
  - 시작 인덱스(startIndex)와 종료 인덱스(endIndex)를 사용하여 문자열의 부분을 추출
  - endIndex 생략시 문자열 끝까지 추출
  - startIndex가 endIndex보다 크면 두 값을 자동으로 교환

```
//문자열 추출
str = "Hello World";

console.log(str.slice(0, 5));       // Hello
console.log(str.slice(5, 0));       //
console.log(str.slice(-11, -6));    // Hello

console.log(str.substring(0, 5));       // Hello
console.log(str.substring(5, 0));       // Hello
console.log(str.substring(-11, -6));    //
```

해결문제



HTML CSS JavaScript React JS

배열

- 여러 개의 데이터를 하나의 변수에 저장할 수 있는 자료구조
  - 배열의 각 요소는 인덱스를 통해 접근
  - 순서가 있는 자료 저장
- 배열 생성
  - 리터럴 표기법
    - let arr = [1, 2, 3, 4, 5];
  - Array 생성자 사용
    - let arr = new Array(5);
    - let arr2 = new Array(1, 2, 3, 4, 5);
- 배열의 크기
  - 배열명.length
  - arr.length = 0;을 사용해 아주 간단하게 배열을 비울 수 있음

```
//배열 생성
let arr = [1, 2, 3, 4, 5];
console.log(arr)                        // [1, 2, 3, 4, 5]

//Array 생성자 사용
let arr1 = new Array(5);                // 길이가 5인 빈 배열 생성
let arr2 = new Array(1, 2, 3, 4, 5);    // 요소가 있는 배열 생성
console.log(arr1)                       // [empty × 5]
console.log(arr2)                       // [1, 2, 3, 4, 5]

//배열 크기
console.log(arr.length)                 // 5
arr.length = 0 ;
console.log(arr)                        // []
```

배열 자료 추가 삭제

- push(element)
  - 배열의 끝에 새로운 요소를 추가
- pop()
  - 배열의 마지막 요소를 제거하고, 그 값을 반환
- shift()
  - 배열의 첫 번째 요소를 제거하고, 그 값을 반환
- unshift(element)
  - 배열의 처음에 새로운 요소를 추가

```
//배열 요소 추가
arr.push(6);
console.log(arr)                    // [1, 2, 3, 4, 5, 6]

//배열 요소 삭제
arr.pop();
console.log(arr)                    // [1, 2, 3, 4, 5]

//배열 처음 요소 추가
arr.unshift(6);
console.log(arr)                    // [6, 1, 2, 3, 4, 5]

//배열 요소 삭제
arr.shift();
console.log(arr)                    // [1, 2, 3, 4, 5]
```

배열 순회

- 배열의 요소 접근
  - 배열명[인덱스]
- 배열 순회
  - 기본 for 루프, for...in 루프, for...of 루프, forEach() 메서드를 사용하여 순회

```
//배열 순회1
for(let i=0 ; i < arr.length ; i++) {
  console.log(arr[i]);
}

//배열 순회2
for(let i in arr) {
  console.log(arr[i]);
}

//배열 순회3
for(let item of arr) {
  console.log(item);
}

for(let [i,item] of arr.entries()) {
  console.log(i, item);
}

//배열 순회3
arr.forEach((item, i) => {
  console.log(i, item);
})
```

반복 for

- for
  - ES1 버전 부터 있었던 가장 전통적인 반복문
- for in
  - Object의 key를 순회하기 위해 사용되는 반복문
  - 배열의 반복을 위해서는 추천되지 않음
- forEach(함수)
  - Array를 순회하는 데 사용되는 Array의 메소드
  - 배열의 요소와 인덱스 모두에 접근
  - await을 루프 내부에 쓸 수 없음
  - 중간에 루프를 탈출할 수 없음
- for of 구문
  - ES6에 나온 가장 최신 기능
  - break continue를 사용 가능
  - 반복 가능한(iterable, 이터러블) 객체 접근
  - entries() 메소드로 인덱스와 값 모두 접근 가능

```
//전통적인 반복문
console.log("전통적인 반복문")
for(let i=0; i < bt4.length; i++) {
 console.log(bt4[i]);
}

//Object의 key를 순회하기 위해 사용되는 반복문
console.log("Object의 key를 순회")
for(let i in bt4) {
 console.log(bt4[i]);
}

//Array를 순회하는 데 사용
console.log("Array를 순회")
bt4.forEach((item, idx) => console.log(idx, item));

//키만 접근하거나, 혹은 키와 값 모두 접근하거나 하는 것이 모두 가능
console.log("객체 순회")
for(let i of bt4) {
 console.log(i);
}
console.log("객체 순회2 : 인덱스접근")
for(let [idx, i] of bt4.entries()) {
 console.log(idx, i);
}
```

배열 변형과 정렬

- **map()**
  - 배열의 각 요소에 대해 주어진 함수를 호출하고, 결과를 새 배열로 반환
- **filter()**
  - 배열에서 주어진 조건을 만족하는 요소들만으로 새 배열을 생성
- **sort()**
  - 배열을 정렬
  - 기본적으로 요소를 문자열로 취급하여 정렬
  - 숫자 정렬에는 주의가 필요
    - arr.sort((a, b) => a - b); // 오름차순 정렬
    - arr.sort((a, b) => b - a); // 내림차순 정렬

```
arr = [1,4,2,3,5]

//배열 변경
arr2 = arr.map(i => i *2 ) ;
console.log(arr2);                    // [2, 8, 4, 6, 10]

arr2 = arr.filter(i => i %2 == 0 ) ;
console.log(arr2);                    // [4, 2]

//숫자 정렬
arr.sort((a, b) => a - b);
console.log(arr);                     // [1, 2, 3, 4, 5]
```

배열 결합과 분해

- concat():
  - 두 개 이상의 배열을 결합하여 새 배열
- join():
  - 배열의 모든 요소를 문자열로 결합
- slice(start, end)
  - 시작 인덱스부터 종료 인덱스까지 배열의 부분을 추출하여 새 배열
- splice(start, deleteCount, ...items)
  - 배열의 요소를 삭제하거나 교체하고, 필요에 따라 새 요소를 추가

```
//배열 결합
arr1 = [1,3,5]
arr2 = [2,4]

arr = arr1.concat(arr2);
console.log(arr);                    // [1, 3, 5, 2, 4]

arr = arr1.join();
console.log(arr);                    // 1,3,5
arr = arr1.join('');
console.log(arr);                    // 135

//배열 분해
arr = [1,2,3,4,5];

arr1 = arr.slice(2, 5);
console.log(arr1);                   // [3, 4, 5]

arr.splice(2,2,'a','b');
console.log(arr);                    // [1, 2, 'a', 'b', 5]
```

배열 요소 찾기

- includes(element, start);
  - 배열 또는 문자열이 특정 요소나 부분 문자열을 포함하고 있는지 여부를 확인
  - 해당 요소가 존재하면 true, 그렇지 않으면 false를 반환
  - element 배열에서 찾고자 하는 요소
  - start (선택 사항) 검색을 시작할 인덱스, 기본값은 0
- indexOf(element, start)
  - 배열 또는 문자열에서 특정 요소나 부분 문자열이 처음으로 나타나는 인덱스를 반환
  - 요소가 존재하지 않으면 -1을 반환
  - element 배열에서 찾고자 하는 요소
  - start (선택 사항) 검색을 시작할 인덱스, 기본값은 0
  - 존재하지 않으면 -1 반환

```
//배열 요소 찾기
arr =  [1, 2, 'a', 'b', 5]

console.log(arr.includes('1'));      // false
console.log(arr.includes(1));        // true

console.log(arr.indexOf('1'));       // -1
console.log(arr.indexOf(1));         // 0
```

배열 효율적인 코드 작성

- Trailing 쉼표 (Trailing Comma)
  - 객체 리터럴, 배열 리터럴, 함수 매개변수 목록 등에서 마지막 요소 뒤에 추가되는 쉼표
  - 마지막 요소 뒤에 쉼표를 허용하여 코드의 가독성을 높이고, 요소 추가 시 편리하게 유지보수
- 전개 연산자 (Spread Operator ...)
  - 배열이나 객체의 요소를 개별적인 요소로 확장할 때 사용
- 구조 분해 할당 (Destructuring Assignment)
  - 배열이나 객체의 값을 개별 변수로 추출하여 할당하는 간결한 문법

```
//배열에 trailing 쉼표
arr1 = [1, 2, 3,];
arr2 = [4, 5] ;

//전개 연산자 (Spread Operator ...)
arr = [...arr1];
console.log(arr);                    // [1, 2, 3]

arr = [...arr1, ...arr2];
console.log(arr);                    // [1, 2, 3, 4, 5]

[x, y] = arr2 ;
console.log(x, y);                   // 4 5
```

실습과제

HTML CSS JavaScript React JS

객체(Object)

- 키-값 쌍으로 구성되며 속성(프로퍼티)과 메서드를 가지는 데이터 타입
- 구성 요소
  - 키(Key): 문자열 또는 Symbol 타입으로, 객체 내에서 값을 참조하기 위한 식별자
  - 값(Value): 키에 연관된 데이터
- 객체 생성 방법
  - 리터럴 표기법 : let obj = {};
  - new 키워드 사용: let obj = new Object();

```
//오브젝트
//tailing 쉼표 사용가능
let obj = {apple:'🍎', carrot:'🥕', banana:'🍌',} ;
console.log("오브젝트 :", obj) ;
```

```
오브젝트 : ▶ {apple: '🍎', carrot: '🥕', banana: '🍌'}
```

객체 추가 및 수정

- **객체 접근 및 추가**
  - 점 표기법 : obj.키
  - 대괄호 표기법 : obj[키]
- **객체 추가**
  - obj[키] = 추가할 값
- **객체 수정**
  - obj[키] = 수정할 값
- **객체 삭제**
  - delete obj[삭제할키]

```
//객체 값추가
obj.orange = '🍊' ;

console.log(`오브젝트 orange 접근 : ${obj.orange}`);
console.log(`오브젝트 🍊 접근 : ${obj['orange']}`);

//객체 값수정
obj.orange = '🔴' ;
console.log(`오브젝트 orange 값 수정 : ${obj['orange']}`);

//객체 값삭제
console.log("삭제전 오브젝트 :", obj) ;
delete obj.orange ;
console.log("삭제후 오브젝트 :", obj) ;
```

```
오브젝트 orange 접근 : 🍊
오브젝트 🍊 접근 : 🍊
오브젝트 orange 값 수정 : 🔴
삭제전 오브젝트 : ▶ {apple: '🍎', carrot: '🥕', banana: '🍌', orange: '🔴'}
삭제후 오브젝트 : ▶ {apple: '🍎', carrot: '🥕', banana: '🍌'}
```

객체 추출 및 순회

- **키-값 쌍 추출**
  - Object.keys(): 객체의 모든 키를 배열로 반환
  - Object.values(): 객체의 모든 값을 배열로 반환
  - Object.entries(): 객체의 키-값 쌍을 배열의 배열로 반환
- **객체 순회**
  - 반복문 활용

```
//객체 순회1
console.log("for...in으로 순회") ;
for(let k in obj) {
  console.log(`키: ${k}, 값: ${obj[k]}`) ;
}

//객체 순회2
console.log("for...of로 순회") ;
for(let [k, v] of Object.entries(obj)) {
  console.log(`키: ${k}, 값: ${v}`) ;
}
```

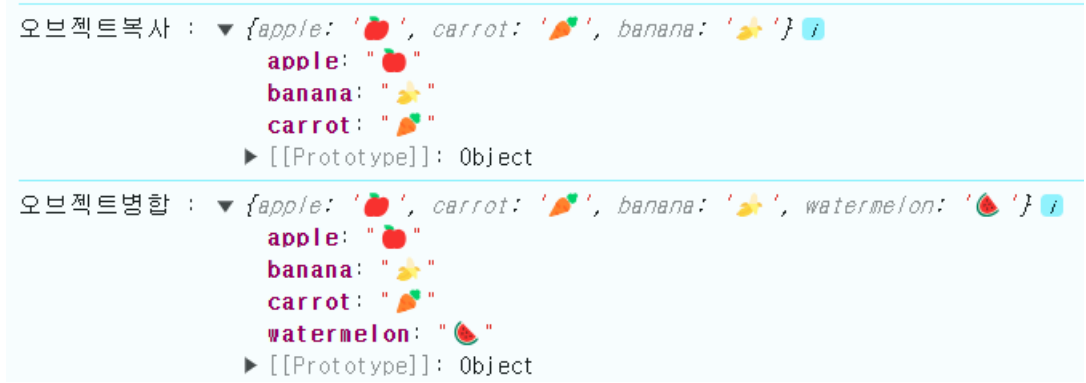
```
for...in으로 순회
키: apple, 값: 🍎
키: carrot, 값: 🥕
키: banana, 값: 🍌
for...of로 순회
키: apple, 값: 🍎
키: carrot, 값: 🥕
키: banana, 값: 🍌
```

객체 복사 및 병합

- 전개연산자(…)을 사용하여 복사 및 병합

```
//객체 복사
obj1 = {...obj} ;
console.log("오브젝트복사 :", obj1) ;

//객체 병합
obj1 = {watermelon:'🍉'}
obj2 = {...obj, ...obj1}
console.log("오브젝트병합 :", obj2) ;
```



HTML CSS JavaScript React JS