

9/29 목요일 강의자료

1교시

2-02 map, filter, reduce 구현하기

▼ map, filter, reduce 내부 들여다 보기

map 메서드 내부

```
// Map 함수 구현 로직
if (!Array.prototype.map) {
  Array.prototype.map = function (callback, thisArg) {
    var T, A, k;
    if (this == null) {
      throw new TypeError(" this is null or not defined");
    }
    var O = Object(this); // 기존 array
    var len = O.length >>> 0; // 비트연산
    if (typeof callback !== "function") {
      throw new TypeError(callback + " is not a function");
    }
    if (arguments.length > 1) {
      T = thisArg;
    }
    A = new Array(len);
    k = 0;

    while (k < len) {
      var kValue, mappedValue; // kValue : 기존 배열 요소값, mappedValue : 새로운 배열 요소값
      if (k in O) {
        // 객체O 안에 속성의 이름이나 배열의 인덱스를 뜻하는 문자열 또는 수 값 K가 존재하는지 반환
        // in : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/in
        kValue = O[k];
        mappedValue = callback.call(T, kValue, k, O); // 함수에 재할당 (this값, 요소값, 요소idx, 배열)

        A[k] = mappedValue;
      }
      k++;
    }
    return A;
  };
}
```

filter 메서드 내부

```
// filter 함수 구현 로직
if (!Array.prototype.filter) {
  Array.prototype.filter = function (func, thisArg) {
    "use strict";
    if (!((typeof func === "Function" || typeof func === "function") && this))
      throw new TypeError();

    var len = this.length >>> 0,
```

```

    res = new Array(len), // preallocate array
    t = this,
    c = 0,
    i = -1;
  if (thisArg === undefined) {
    while (++i !== len) {
      // checks to see if the key was set
      if (i in this) {
        if (func(t[i], i, t)) {
          res[c++] = t[i];
        }
      }
    }
  } else {
    while (++i !== len) {
      // checks to see if the key was set
      if (i in this) {
        if (func.call(thisArg, t[i], i, t)) {
          res[c++] = t[i];
        }
      }
    }
  }

  res.length = c; // shrink down array to proper size
  return res;
};
}

```

reduce 메서드 내부

```

// reduce 함수 구현 로직
if (!Array.prototype.reduce) {
  Object.defineProperty(Array.prototype, "reduce", {
    value: function (callback /*, initialValue*/) {
      if (this === null) {
        throw new TypeError(
          "Array.prototype.reduce " + "called on null or undefined"
        );
      }
      if (typeof callback !== "function") {
        throw new TypeError(callback + " is not a function");
      }

      var o = Object(this);
      var len = o.length >> 0;
      var k = 0;
      var value;

      if (arguments.length >= 2) {
        value = arguments[1];
      } else {
        while (k < len && !(k in o)) {
          k++;
        }
        if (k >= len) {
          throw new TypeError(
            "Reduce of empty array " + "with no initial value"
          );
        }
        value = o[k++];
      }

      while (k < len) {
        if (k in o) {
          value = callback(value, o[k], k, o);
        }
      }
    }
  });
}

```

```

        k++;
    }

    return value;
  },
});
}

```

Object.defineProperty() : 정적 메서드는 객체에 새로운 속성을 직접 정의하거나 이미 존재하는 속성을 수정한 후, 해당 객체를 반환

Object.defineProperty(obj, prop, descriptor)

- **obj** : 속성을 정의할 객체
- **prop** : 새로 만들거나 수정하려는 속성의 이름
- **descriptor** : 객체에 정의하는 속성을 기술하는 객체

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty

2-03 Rest Operator 사용

▼ ... 문법 (rest parameter)

ES6에 나온 문법으로 함수가 정해지지 않은 수의 매개변수를 배열로 받을 수 있습니다.

여러가지 활용법은 실습코드를 통해 확인해보겠습니다!

+ rest parameter 와 arguments의 차이?

- **arguments** 객체는 **실제 배열이 아닙니다**. 그러나 나머지 매개변수는 **Array** 인스턴스이므로 **sort**, **map**, **forEach**, **pop** 등의 메서드를 직접 적용할 수 있습니다.

여러가지 활용법은 실습코드를 통해 확인해보겠습니다!

arguments : <https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Functions/arguments>

참고자료 : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Functions/rest_parameters

▼ ... 문법 (Spread operator)

ES6에 나온 문법으로 기존 배열이나 객체의 전체 또는 일부를 다른 배열이나 객체로 빠르게 복사하여 사용할 수 있습니다.

여러가지 활용법은 실습코드를 통해 확인해보겠습니다!

참고자료 : https://www.w3schools.com/react/react_es6_spread.asp

2교시

2-01 This로 이벤트 접근하기

▼ this는 무엇일까요?

this는 자신이 속한 객체 또는 자신이 생성할 인스턴스를 가리키는 자기 참조 변수입니다.

따라서 어떤 환경에서 호출하는지에 따라 this는 다르게 작동합니다.

브라우저 환경 ⇒ window

```
> function myFn() {  
  return console.log(this);  
}  
myFn();  
  
▼Window {0: global, 1: global, 2: global, 3: global, window: Window, self: Window, document: document, name: '', location: Location, ...} 5  
  ▶ 0: global {window: global, self: global, location: {...}, closed: false, frames: global, ...}  
  ▶ 1: global {window: global, self: global, location: {...}, closed: false, frames: global, ...}  
  ▶ 2: global {window: global, self: global, location: {...}, closed: false, frames: global, ...}  
  ▶ 3: global {window: global, self: global, location: {...}, closed: false, frames: global, ...}  
  ▶ $: f (e,t)  
  ▶ DIV1: {item_count: '%d Item', items_count: '%d Items'}
```

node 환경 ⇒ global

```
[Running] node "/Users/jaehun/Developer/JS/elice_SW_track3/week3/9.29/testCode/this.js"  
<ref *1> Object [global] {  
  global: [Circular *1],  
  clearInterval: [Function: clearInterval],  
  clearTimeout: [Function: clearTimeout],  
  setInterval: [Function: setInterval],  
  setTimeout: [Function: setTimeout] {  
    [Symbol(nodejs.util.promisify.custom)]: [Getter]  
  },  
  queueMicrotask: [Function: queueMicrotask],  
  performance: Performance {  
    nodeTiming: PerformanceNodeTiming {  
      name: 'node',  
      entryType: 'node',  
      startTime: 0,  
      duration: 20.275166995823383,  
      nodeStart: 0.7313749939203262,  
      v8Start: 1.1219169944524765,  
      bootstrapComplete: 16.0394169986248,  
      environment: 9.40787499397993,  
      loopStart: -1,  
      loopExit: -1,
```

이전에 설명했던 화살표함수와 일반함수의 차이점으로 this가 있었죠?

실습을 통해 알아보겠습니다.

이처럼 일반함수의 this는 일반함수를 호출한 실행 컨텍스트를 가리키는 반면 화살표함수의this는 렉시컬 스코프를 따라 상위 스코프의 실행 컨텍스트를 가리킵니다.

```
> const fn = {
  title: "Hello World!",
  tags: [1, 2, 3, 4],
  func() {
    console.log(this.title);
  },
  funcThis: {
    title: "Hello elice",
    func() {
      console.log(this.title);
    },
    arrFunc: () => {
      console.log(this);
    },
  },
};
fn.func();
fn.funcThis.func();
fn.funcThis.arrFunc();
Hello World!
Hello elice
> Window {0: Window, 1: global, 2: global, 3: global, 4: global, window: Window, self: Window, document: document, name: '', location: Location, ...}
```

또한 일반함수는 this를 바인딩하며 변경할 수 있는 반면 화살표 함수는 this를 유지시켜 바꿀 수 없습니다.

따라서 this를 바꿔야 하는 상황일 때는 일반함수를 바뀌지 않았으면 하는 상황에서는 화살표함수를 사용하는 것을 권장드립니다.

▼ 스코프(scope)란?

스코프란 변수에 접근할 수 있는 범위로써 식별자(변수)를 찾기 위한 규칙으로 변수의 유효범위라고 할 수 있습니다.

스코프의 종류로는 2가지가 있습니다.

전역 스코프 (Global scope) : 코드 어디에서든지 참조할 수 있다.

지역 스코프 (Local scope or Function-level scope) : 함수 코드 블록이 만든 스코프로 함수 자신과 하위 함수에서만 참조할 수 있다.

JS는 다른 언어들이랑 다르게 함수레벨스코프를 따릅니다. 하지만 ES6문법인 const, let는 블록레벨스코프를 따릅니다.

위에서 언급한 스코프를 결정하는 방식도 두가지가 있습니다 .

동적 스코프(Dynamic scope) : 함수를 어디서 **호출**했는지에 따라 결정

렉시컬 스코프(Lexical scope) 또는 정적 스코프(Static scope) : 함수를 어디서 **선언**했는지에 따라 결정

JS는 렉시컬 스코프를 따릅니다.

이때 화살표 함수는 렉시컬 스코프를 통해 상위 스코프를 정하기때문에 this가 메서드를 호출한 객체가 아닌 전역 객체를 가리킵니다.

▼ 호이스팅과 TDZ란?

호이스팅은 변수의 선언문을 유효범위 즉, 스코프 최상단으로 올려주는 것입니다.

위의 예시로 var, function, import 같은 것들이 있었죠?

이때 TDZ은 Temporal Dead Zone 즉 일시적 사각지대 라는 뜻으로 변수 선언 및 초기화 하기 전 상태로 TDZ 상태에서 변수를 사용하려고 하면 ReferenceError가 나타나게 됩니다.

2-04 Closure 개념

▼ Closure란..?

함수와 함수가 사용하는 변수들을 저장한 공간을 클로저라고 하며 함수안에 함수를 정의했을때 내부함수에서 외부함수의 변수를 접근할 수있는 특성이 있습니다.

먼저 선행적으로 알아야 할 개념들이 있습니다.

실행 컨텍스트 : 우리가 작성한 코드가 실행되는 환경

- **글로벌 실행 컨텍스트(Global Execution Context)**

: 코드가 실행되기 전에 생성이 되며, 함수 내에 없는 코드는 모두 **전역 실행 컨텍스트 안에 존재 (오직 하나, 종료될때까지 유지)**

- **함수 실행 컨텍스트(Functional Execution Context)**

: 전역 실행 컨텍스트가 생성된 후, 함수가 실행(ex 호출) 될 때마다 **새로운 실행 컨텍스트가 작성**

스코프 체인 : 전역 객체와 중첩된 함수의 스코프의 레퍼런스를 차례로 저장하여 어떻게 연결 되어있는지 알려 줍니다.

실행 컨텍스트가 실행되면, 엔진이 스코프 체인을 통해 렉시컬 스코프를 먼저 파악합니다.

렉시컬 스코프 : 함수를 어디서 호출하는지가 아니라 어디에 **선언하였는지에 따라 결정되는 것 즉, 함수를 어디서 선언하였는지에 따라 상위 스코프를 결정**

클로저를 사용하는 이유는 **상태를 안전하게 은닉하고 보존시키기**위함입니다.

외부에서는 접근, 변경을 할 수 없지만 클로저 내부에서는 접근, 변경을 할 수 있는것이죠. 이전 시간에 class 수업을 하면서 getter,setter 배우신것과 비슷합니다!

이러한 특성을 이용해서 클로저를 통해 JAVA의 Private method를 흉내낼 수 있습니다.

3교시

2-05 String 바꾸기

▼ split() 정리

str.split([separator] [, limit])

: 문자열str안에 separator값을 기준으로 나눠 arr로 반환해 줍니다.

split() : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/split

▼ reverse() 정리

array.reverse()

: 배열의 순서를 반전시켜 줍니다.

reverse():

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/reverse

▼ join() 정리

arr.join([separator])

: 배열의 값 사이에 인자값을 넣은 문자열로 반환해줍니다.

join() : https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/join

▼ toLocaleString() 정리

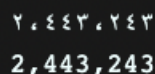
.toLocaleString([locales[, options]])

: Number, Date, Array, Object에 대해서 사용할 수 있는 메서드 인데요 특정 자료가 들어왔을 때 이를 현지화 시켜준다고 생각하시면 편합니다.

ex :

ar-EG :아랍어(이집트)

```
function groupByCommas(n) {  
  console.log(n.toLocaleString("ar-EG"));  
  return n.toLocaleString('ko-KR');  
}  
  
console.log(groupByCommas(2443243));
```



나라별 언어 : <http://www.lingoes.net/en/translator/langcode.htm>

추가적인 설명 : <https://blog.munilive.com/posts/javascript-localization-with-toLocaleString.html>

2-06 삼각형 일까요?

▼ sort() 사용

JavaScript에도 배열을 정렬해주는 sort()함수가 있습니다.

`arr.sort([compareFunction])`

옵셔널 파라미터 값으로 compareFunction이 있는데요 이 친구가 없다면 배열의 요소들을 문자열로 취급하여 유니코드 값 순서대로 정렬이 됩니다. 우리가 원하는 방법과 다르게 정렬될 수 있죠

```
const numbers = [10, 3, 8, 4, 1];
numbers.sort();

console.log(numbers);
// 결과 : [ 1, 10, 3, 4, 8 ]
```

위의 예시를 보면 10이 3보다 앞에 위치하는게 보이시죠? 바로 값을 비교할때 숫자 타입을 문자타입으로 형변환하여 비교하기 때문인데요 그러면 형변환 이후 첫번째 문자 기준으로 비교 하기 때문에 1이 3보다 작다고 보고 10을 3 앞에 두게 됩니다.

유니코드란?

각 나라별 언어를 모두 표현하기 위해 나온 코드 체계가 유니코드(unicode)입니다. 유니코드는 사용중인 운영 체제, 프로그램, 언어에 관계없이 문자마다 고유한 코드 값을 제공하는 새로운 개념의 코드로 언어와 상관없이 모든 문자를 16비트로 표현해요

https://ko.wikipedia.org/wiki/유니코드_0000~FFFF

그러면 우리가 원하는 방식으로 정렬하기 위해서는 어떻게 해줘야 할까요? compareFunction을 이용하면됩니다.

여기서 기억하실 것은

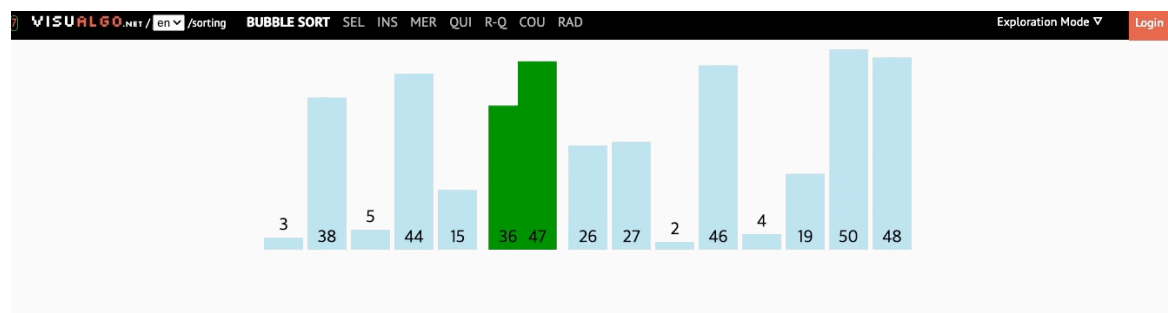
- 반환 값 < 0 : a 가 b보다 앞에 있어야 한다.
- 반환 값 = 0 : a와 b의 순서를 바꾸지 않는다.
- 반환 값 > 0 : b가 a보다 앞에 있어야 한다.

이 세가지를 생각하시며 비교함수를 작성하시면 됩니다.

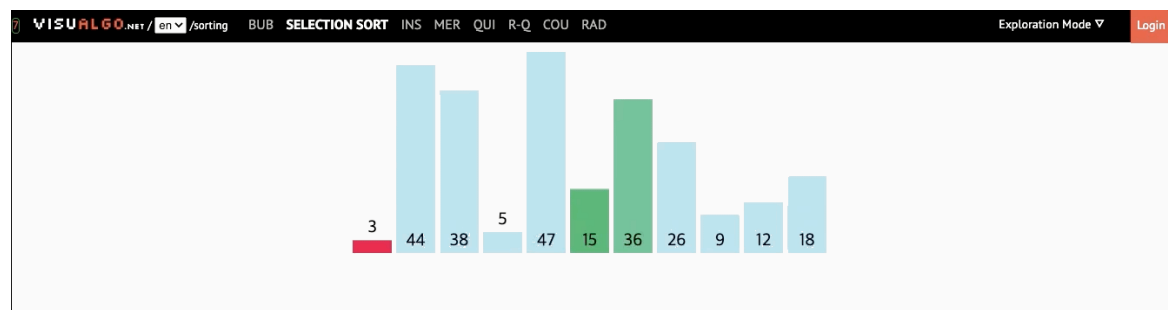
예를들어 오름차순으로 정렬하고 싶다면 (a,b)⇒ a-b 를 넣어주면

▼ 정렬하는 알고리즘들은 뭐가 있을까?

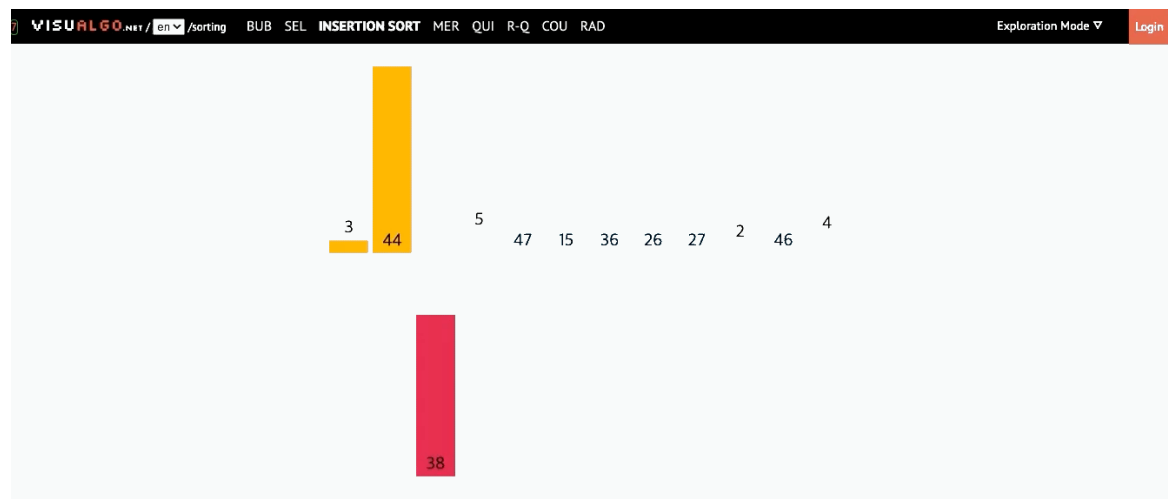
버블 정렬 (Bubble Sort)



선택 정렬 (Selection Sort)



삽입 정렬 (Insertion Sort)



여러가지 정렬 법 시각화로 보기 : <https://visualgo.net/en/sorting?slide=1>

2-07 제곱 리스트를 찾아라

▼ sqrt()

Math.sqrt(x)

: x 값의 제곱근을 반환해줍니다. 만약 음수라면 NaN을 반환합니다.

```
console.log(Math.sqrt(9)); // 3
console.log(Math.sqrt(2)); // 1.414213562373095
console.log(Math.sqrt(1)); // 1
console.log(Math.sqrt(0)); // 0
console.log(Math.sqrt(-1)); // NaN
```

Math.sqrt(x):

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Math/sqrt

▼ isInteger()

Number.isInteger(value)

: value 값이 정수인지 아닌지 True/False로 반환해줍니다.