SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성청년 SW·AI아카데미

JavaScript

<알림>

본 강의는 삼성청년SW·AI아카데미의 컨텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사, 촬영,
녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day3-1. Array의 순회 Method

챕터의 포인트

- forEach, some, every
- find, findIndex
- map, filter, reduce
- 순회 메서드 연습
- 배열 고차함수 연습

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

forEach, some, every

함수를 정의하는 방법

- 일반 함수 : function ( ) { }
- 화살표 함수 : ( ) => { }

화살표 함수 : 익명 함수

```
1   <script>
2
3       let a = function () {
4           alert("AA");
5       }
6
7       let b = () => {
8           alert("BB");
9       }
10
11      a();
12      b();
13
14  </script>
```

화살표 함수는 익명 함수이다.

- 값을 전달하고, 리턴 가능

```
<script>

    let a = (a, b) => {
        alert(a + b);
    }

    a(3, 4);

</script>
```

a, b, 받고 합 출력

```
<script>

    let a = (a, b) => {
        return a + b;
    }

    let sum = a(3, 4);
    alert(sum);

</script>
```

a, b 받고 합 return 하기

콜백 함수는 두 가지 의미를 가진다.

- **이벤트 발생시 호출되는 예약 함수**
  - 사용자가 이벤트가 발생 시켰을 때, 호출 되는 예약된 함수
  - OS가 특정 행동을 할 때, 호출 되는 예약된 함수
- **일반 함수의 Parameter로 등록되는 함수**

```
1  <script>
2
3    function run(e) {
4      alert(e.keyCode);
5    }
6
7    document.addEventListener('keypress', run);
8
9  </script>
```

run은 event의 callback 함수이다.

```
const run = () => console.log("run함수실행");
const walk = () => console.log("walk함수실행");

const go = (action) => {
  action();
};

// go(run())
go(run);
go(walk);
```

run과 walk은 callback 함수로 쓰였다.

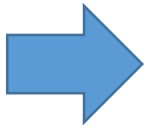
배열 메서드를 잘 사용해야하는 이유

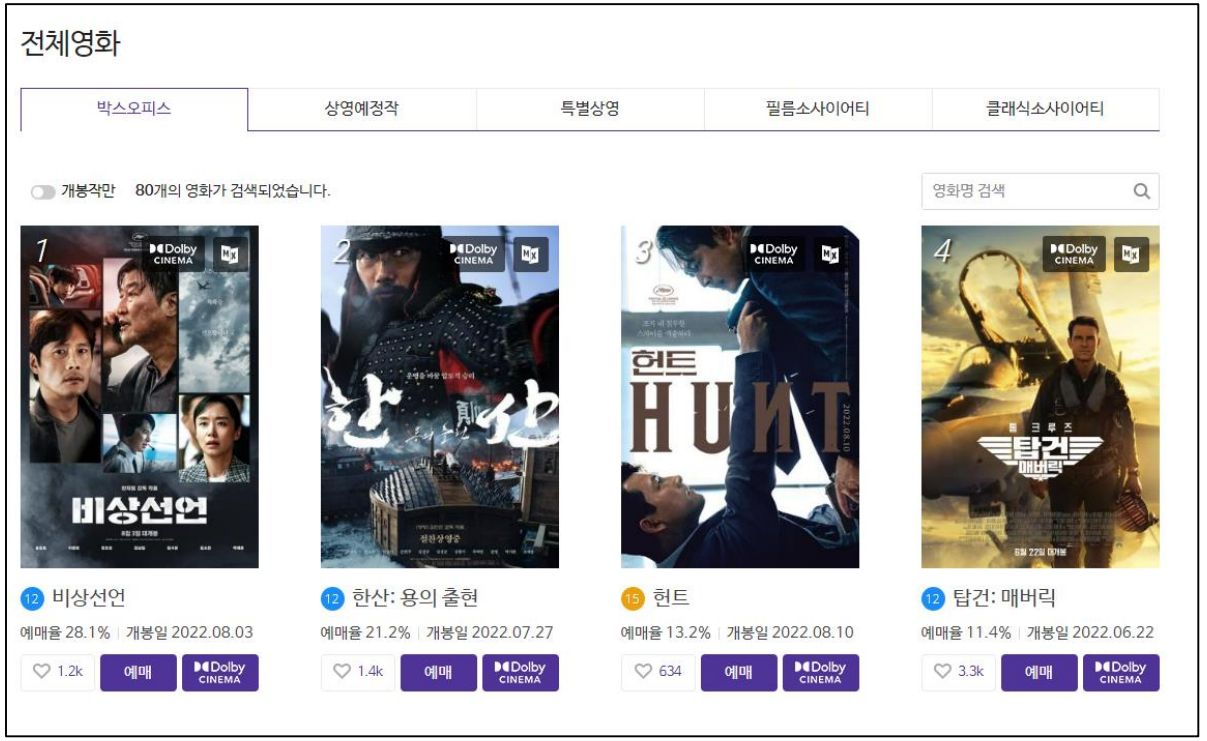
Confidential

배열 메서드

- 대부분의 데이터는 배열의 형태로 뿌려지고 있다.
- 해당 데이터를 잘 받아서 보기 좋게 뿌려주는 것이 프론트 엔드의 역할


```
movieList: [{movieNo: "21084100", boxoRank: 1, boxoAccmRank:
  0: {movieNo: "21084100", boxoRank: 1, boxoAccmRank: 14, me
  1: {movieNo: "22023000", boxoRank: 2, boxoAccmRank: 3, meg
  2: {movieNo: "22037300", boxoRank: 3, boxoAccmRank: 19, me
  3: {movieNo: "22018400", boxoRank: 4, boxoAccmRank: 2, meg
  4: {movieNo: "21020400", boxoRank: 5, boxoAccmRank: 7, meg
  5: {movieNo: "22040600", boxoRank: 6, boxoAccmRank: 21, me
  6: {movieNo: "22044900", boxoRank: 7, boxoAccmRank: 23, me
  7: {movieNo: "22022900", boxoRank: 8, boxoAccmRank: 6, meg
  8: {movieNo: "22034600", boxoRank: 9, boxoAccmRank: 8, meg
  9: {movieNo: "22032900", boxoRank: 10, boxoAccmRank: 13, m
  10: {movieNo: "22030100", boxoRank: 11, boxoAccmRank: 16,
  11: {movieNo: "22029600", boxoRank: 12, boxoAccmRank: 15,
  12: {movieNo: "22035000", boxoRank: 13, boxoAccmRank: 11,
  13: {movieNo: "22035700", boxoRank: 14, boxoAccmRank: 30,
  14: {movieNo: "22040400", boxoRank: 15, boxoAccmRank: 31,
  15: {movieNo: "22041200", boxoRank: 16, boxoAccmRank: 32,
  16: {movieNo: "22041100", boxoRank: 17, boxoAccmRank: 34,
  17: {movieNo: "22040800", boxoRank: 19, boxoAccmRank: 36,
  18: {movieNo: "22028200", boxoRank: 20, boxoAccmRank: 4, m
  19: {movieNo: "22041300", boxoRank: 21, boxoAccmRank: 40,
```



Array.forEach(Callback 함수)

- Arrary의 원소 개수만큼 Callback 함수가 호출된다.
- Callback 함수 한번 호출될 때 마다 원소 하나씩 Argument로 들어간다.

```
const test = (value) => {
  console.log(value);
};

const arr = [3, 5, 4, 2];
arr.forEach(test);
```

test가 총 네 번 호출 됨
3 / 5 / 4 / 2 가 출력 됨

```
const arr = [3, 5, 4, 2];
arr.forEach((value) => {
  console.log(value);
});
```

왼쪽과 같은 소스코드

[실습] 각 요소의 값 처리하기

[4, 3, 5, 1, 6, 5] 에서 홀수의 값이 몇 개인지 console.log

- forEach와 화살표 함수를 이용하여 표현하자.
- cnt 변수는 전역변수를 사용한다.

[실습] 각 요소에서 양수와 음수를 구분해서 넣기

[-5, 3, 4, 2, -7, -2, 7] array 에서 양수 음수를 구분해보자

- **pplus, mminus array를 만들어 두자.**
- **foreach와 화살표 함수를 이용하여 숫자를 분류하자**
  - pplus 배열에는 0 보다 큰 수를 넣기
  - mminus 배열에는 음수만 넣기

Array.some(Callback 함수)

- Arrary의 원소 개수만큼 Callback 함수가 호출된다.
- Callback 함수 한번 호출될 때 마다 원소 하나씩 Argument로 들어간다.
- 하나의 조건만 충족해도 true를 반환한다.

```
const datas = [1, 2, 3, 4, 5];
console.log(datas.some((data) => data > 4));
```

true 반환

```
const datas = [1, 2, 3, 4, 5];
console.log(datas.some((data) => data < 0));
```

false 반환

Array.every(Callback 함수)

- Arrary의 원소 개수만큼 Callback 함수가 호출된다.
- Callback 함수 한번 호출될 때 마다 원소 하나씩 Argument로 들어간다.
- 모든 조건을 충족해야 true를 반환한다

```
const datas = [1, 2, 3, 4, 5];
console.log(datas.every((data) => data > 0));
```

true 반환

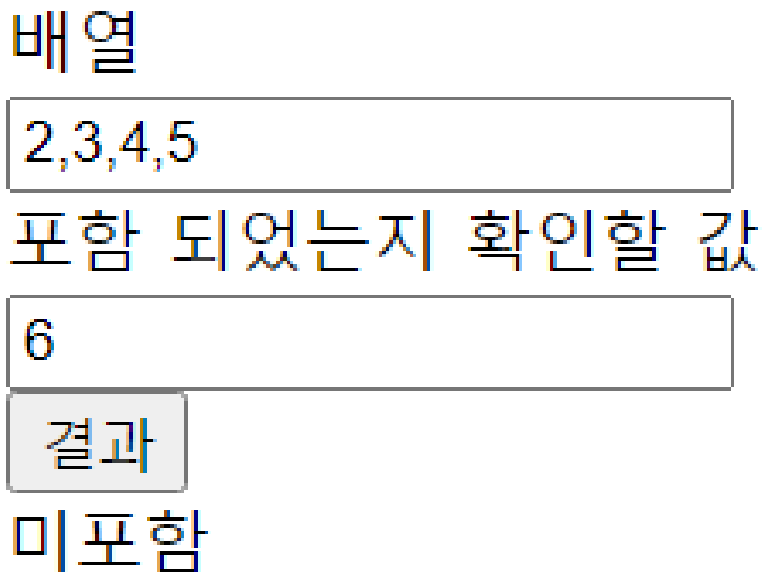
```
const datas = [1, 2, 3, 4, 5];
console.log(datas.every((data) => data > 3));
```

false 반환

[실습] 포함 여부 나타내기

input의 value를 통해 포함여부 나타내기

- input 1에는 배열안에 값들을 , 로 구분해서 넣어준다
- input 2에는 포함되어 있는지 여부를 나타나게 해주는 값
- 결과 버튼 클릭 시 미포함, 포함 출력 나타나게 하기



find, findIndex

Array.find(Callback 함수)

- Arrary의 원소 개수만큼 Callback 함수가 호출된다.
- Callback 함수 한번 호출될 때 마다 원소 하나씩 Argument로 들어간다.
- 해당 조건을 만족하는 첫번째 요소의 값을 반환한다.

```
const datas = [1, 2, 3, 4, 5];
datas.find((data) => data > 4);
```

5가 출력된다

```
const datas = [1, 2, 3, 4, 5];
datas.find((data) => data > 5);
```

undefined(값이 없을 경우)

Array.findIndex(Callback 함수)

- Arrary의 원소 개수만큼 Callback 함수가 호출된다.
- Callback 함수 한번 호출될 때 마다 원소 하나씩 Argument로 들어간다.
- 해당 조건을 만족하는 첫번째 요소의 인덱스를 반환한다.

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

console.log(datas.findIndex((data) => data > 4));
```

4 리턴

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

console.log(datas.findIndex((data) => data > 5));
```

-1 리턴(없는 경우)

배열 선언하기

- 닭다리가 존재하는 객체와 객체의 인덱스를 console.log로 나타내기
- find와 findIndex 사용

```
const chicken = [
  { name: "머리", quantity: 1 },
  { name: "날개", quantity: 2 },
  { name: "닭다리", quantity: 2 },
  { name: "닭가슴살", quantity: 1 },
  { name: "닭발", quantity: 2 },
];
```

map, filter, reduce

forEach

- 배열의 요소를 하나씩 탐색하며, Callback 함수를 수행

map

- 배열의 요소를 하나씩 탐색하며, Callback 함수의 return 값을 모아둔다.
- Array를 새롭게 만든다.

배열안의 각 요소를 변환할 때 사용

- 이 과정에서 새로운 배열이 만들어짐

```
const datas = [3, 5, 4, 2];
const newDatas = datas.map((data) => data + 1);
console.log(newDatas);
```

```
▶ (4) [4, 6, 5, 3]
```

[실습] map 활용하기

1. const arr = [1, 2, 3, 4, 5]
   - 제곱의 값을 모아 둔 Array 생성
   - Array 출력 (출력결과 : [1, 4, 9, 16, 25])

2. const arr2 = [“a”, “bcd”, “ef”, “g”]
   - 배열 요소의 길이를 모아 둔 Array 생성
   - Array 출력하여라 (출력결과 : [1, 3, 2, 1])

배열에서 특정 조건을 만족하는 값들만 따로 추출

- 이 과정에서 새로운 배열이 만들어짐

```
const datas = [3, 5, 4, 2];
const newDatas = datas.filter((data) => data > 3);
console.log(newDatas);
```

```
▶ (2) [5, 4]
```

[실습] filter 활용하기 1

1. const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 을 하드코딩 하고

   - arr1 에는 arr 의 값 중 홀수만 담아서 console.log로 출력하기
   - arr2 에는 arr 의 값 중 3보다 크고 9보다 작은 숫자들만 담아서 console.log로 출력하기

제한시간 (5분)

본인의 bucketList를 작성하고 done 속성이 false인 객체들만 새로운 Array에 저장 후, Array 출력하기

```
const bucketList = [
  {
    id: 1,
    text: "여행 가기",
    done: false,
  },
  {
    id: 2,
    text: "치킨 먹기",
    done: true,
  },
  {
    id: 3,
    text: "코딩 하기",
    done: true,
  },
  {
    id: 4,
    text: "요리 하기",
    done: false,
  },
];
```

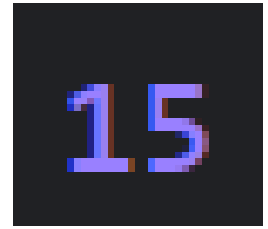
reduce

- **Arrary.reduce((acc, cur) -> {}, 초기값)**
  - 초기값은 생략이 가능하다.
  - 초기값에 숫자뿐만 아니라, 문자, 배열, 빈 객체도 넣을 수 있다.
- **첫 번째 인자(acc)는 accumulator(누산기를 의미한다)**
  - 결과 값을 해당 acc에 담아서 출력
- **두 번째 인자(cur)는 현재 값을 의미**
- **최종 결과 값이(total) accResult에 들어간다.**

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

const accResult = datas.reduce((acc ,cur) => acc + cur );

console.log(accResult);
```



reduce 활용

- reduce를 통해 map, filter, find.. 등등 거의 모든 메서드가 구현가능

```
// reduce를 활용해 제곱으로 표현하기
const array = [1, 2, 3, 4]

array.reduce((acc, cur) => {
    const data = cur*cur;
    acc.push(data);
    return acc;
}, [])
```

filter 또한 reduce로 구현이 가능하다.

```
// reduce를 활용해 2보다 큰 경우만 배열에 담기
const array = [1, 2, 3, 4]

array.reduce((acc, cur) => {

    if(cur > 2){
        acc.push(cur)
    }
    return acc;
}, [])
```

배열의 중복된 개수 찾기

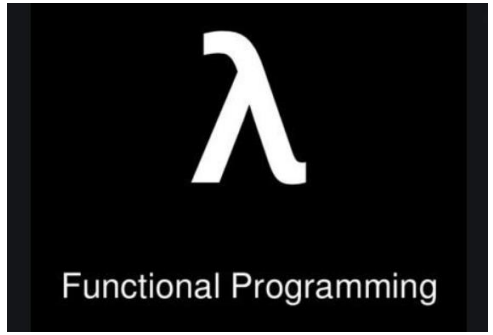
```
const arr = ["피카츄", "라이츄", "파이리", "꼬부기", "피카츄", "파이리"];

const result = arr.reduce((acc, cur) => {

    if(acc[cur]){
        acc[cur] = acc[cur] + 1;
    }else{
        acc[cur] = 1;
    }
    return acc;
},{});

result
▶ {피카츄: 2, 라이츄: 1, 파이리: 2, 꼬부기: 1}
```

reduce를 활용한 함수형 프로그래밍



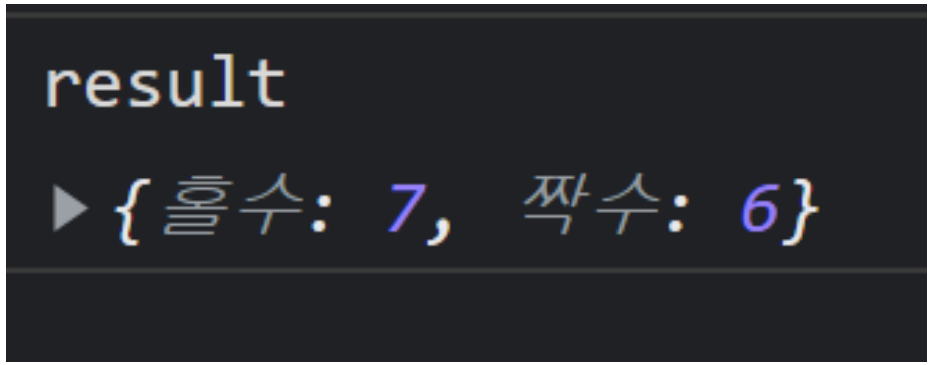

```
const increment = (input) => input + 1;
const decrement = (input) => input -1;
const squared = input => input* input;
const half = input => input/2;

const initValue = 1;
const pipeLine = [increment, decrement, increment, increment, decrement, squared, squared, half, increment];
const result = pipeLine.reduce((acc, cur) => cur(acc), initValue);
console.log(result);
```

```
9                                                                    VM11685:9
```

reduce 활용

- const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
  - reduce를 활용하여 홀수의 개수와 짝수의 개수를 나타내어라



SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

순회 메서드 연습

제한시간

1. **const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**
   - filter, map을 활용하여 짝수인 경우에만 10을 곱해서 출력하기
   - 출력결과 : [20, 40, 60, 80, 100]

2. **const arr2 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**
   - reduce를 활용하여 짝수인 경우에만 10을 곱해서 출력하기
   - 출력결과 : [20, 40, 60, 80, 100]

[실습] map, filter, reduce 활용 2

제한시간

1. count가 0 인 경우 메뉴에 보이지 않게 한다.
2. event가 true인 경우 saledPrice 에 10퍼센트 감면된 가격을 넣는다
3. map과 filter, reduce 버전 두가지로 구현한다.

```
const orderList = [
  {
    menu: "치킨",
    price: 17000,
    event: false,
    count: 50,
  },
  {
    menu: "돈까스",
    price: 8500,
    event: true,
    count: 99,
  },
  {
    menu: "마라탕",
    price: 8000,
    event: false,
    count: 100,
  },
  {
    menu: "쫄면",
    price: 6500,
    event: false,
    count: 0,
  },
  {
    menu: "짜장면",
    price: 5500,
    event: true,
    count: 30,
  },
];
```



```
▼(4) [{…}, {…}, {…}, {…}]
  ▼0:
      count: 50
      event: false
      menu: "치킨"
      price: 17000
    ▶__proto__: Object
  ▼1:
      count: 99
      event: true
      menu: "돈까스"
      price: 8500
      saledPrice: 7650
    ▶__proto__: Object
  ▼2:
      count: 100
      event: false
      menu: "마라탕"
      price: 8000
    ▶__proto__: Object
  ▼3:
      count: 30
      event: true
      menu: "짜장면"
      price: 5500
      saledPrice: 4950
    ▶__proto__: Object
    length: 4
  ▶__proto__: Array(0)
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

배열 고차함수 연습

[실습] 배열 고차함수 연습하기(1)

오른쪽과 같은 결과가 나오도록 우측 코드 완성하기

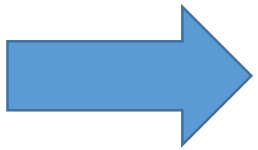
- render 함수를 통해 html 코드를 오른쪽과 같이 나타낼 수 있게 만들자

```
const bucketLists = [
    { id: 3, text: '여행가기', done: false },
    { id: 2, text: '치킨 먹기', done: true },
    { id: 1, text: '코딩 하기', done: false }
];

function render() {
    let html = '';

    return html;
}

console.log(render());
```



```
<li id="3"><label>
        <input type="checkbox">여행가기
        </label>
        </li><li id="2"><label>
        <input type="checkbox">치킨 먹기
        </label>
        </li><li id="1"><label>
        <input type="checkbox">코딩 하기
        </label>
        </li>
```

[실습] 배열 고차함수 연습하기(2)

오른쪽과 같은 결과가 나오도록 우측 코드 완성하기

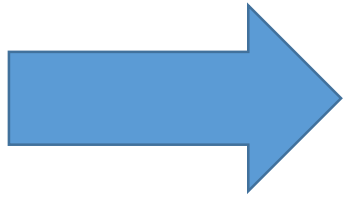
- 단, forEach나 for문은 사용하지 않도록 하자

```
const bucketLists = [
  { id: 3, text: '여행가기', done: false },
  { id: 2, text: '치킨 먹기', done: true },
  { id: 1, text: '코딩 하기', done: false }
];

  function getValues(key) {

  }

  console.log(getValues('id'));
  console.log(getValues('text'));
  console.log(getValues('done'));
```



```
▶ (3) [3, 2, 1]

▶ (3) ['여행가기', '치킨 먹기', '코딩 하기']
▶ (3) [false, true, false]
```

[실습] 배열 고차함수 연습하기(3)

오른쪽과 같은 결과가 나오도록 우측 코드 완성하기

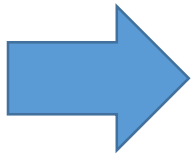
- **배열에 객체를 추가하는 함수를 만들자**

```
let bucketLists = [
    { id: 3, text: '여행가기', done: false },
    { id: 2, text: '치킨 먹기', done: true },
    { id: 1, text: '코딩 하기', done: false }
];
function addList(List) {

}

addList({ id: 4, content: '요리하기', completed: true });

console.log(bucketLists);
```



```
[
  { id: 4, content: '요리하기', completed: true },
  { id: 3, text: '여행가기', done: false },
  { id: 2, text: '치킨 먹기', done: true },
  { id: 1, text: '코딩 하기', done: false }
]
```

[실습] 배열 고차함수 연습하기(4)

오른쪽과 같은 결과가 나오도록 우측 코드 완성하기

- 배열의 객체를 제거하는 함수를 만들자
- id값을 기준으로 제거

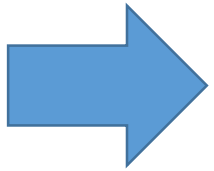
```
let bucketLists = [
    { id: 3, text: '여행가기', done: false },
    { id: 2, text: '치킨 먹기', done: true },
    { id: 1, text: '코딩 하기', done: false }
];


function removeList(id) {

}

removeList(2);

console.log(bucketLists);
```



```
[
  { id: 3, text: '여행가기', done: false },
  { id: 1, text: '코딩 하기', done: false }
]
```

[실습] 배열 고차함수 연습하기(5)

오른쪽과 같은 결과가 나오도록 우측 코드 완성하기

- id값을 기준으로 done을 toggle하는 함수를 만들자

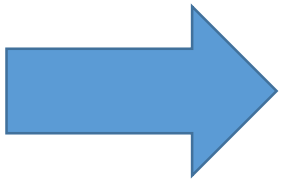
```
let bucketLists = [
  { id: 3, text: '여행가기', done: false },
  { id: 2, text: '치킨 먹기', done: true },
  { id: 1, text: '코딩 하기', done: false }
];


function toggle(id) {

}

toggle(2);

console.log(bucketLists);
```



```
[
  { id: 3, text: '여행가기', done: false },
  { id: 2, text: '치킨 먹기', done: false },
  { id: 1, text: '코딩 하기', done: false }
]
```

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Day3-2. 비동기 프로그래밍

챕터의 포인트

- 동기와 비동기
- Callback

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

동기와 비동기

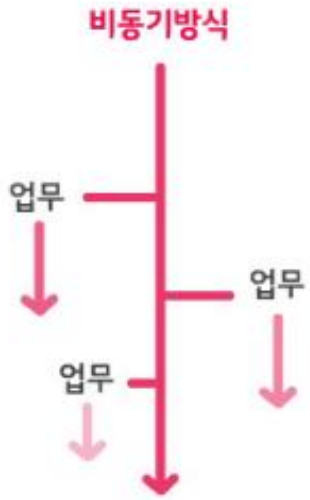
동기 방식

한 업무가 끝나야, 다음 업무를 시작한다.

비동기 방식

다른 업무를 기다리지 않고, 진행한다.

동시에 진행되는 것이 아니다!



화면에 출력될 코드를 예상해보자

```
1   console.log("1")
2   console.log("2")
3   console.log("3")
```

```
console.log("1")
console.log("2")
console.log("3")
```

```
1
```

```
2
```

```
3
```

우리가 기존에 작성했던 코드는 “동기”이다.

- 소스 코드의 흐름은 위에서 아래로, 좌에서 우로 진행된다.
- 순서가 보장되며, 위의 코드가 오래 걸리면 아래 코드는 진행되지 않는다.

비동기 코드 예상하기

비동기코드는 어떻게 동작할지 예상해보자.

- setTimeout : 비동기 함수로, 두번째 매개변수 시간(ms) 만큼 시간이 지난 후, 첫번째 매개변수를 실행

```
console.log('1')

setTimeout(() => {
    console.log('2')
}, 1000);
console.log('3');
```

```
1
3
2
```

앞으로 비동기에 대해서 배우게 되면

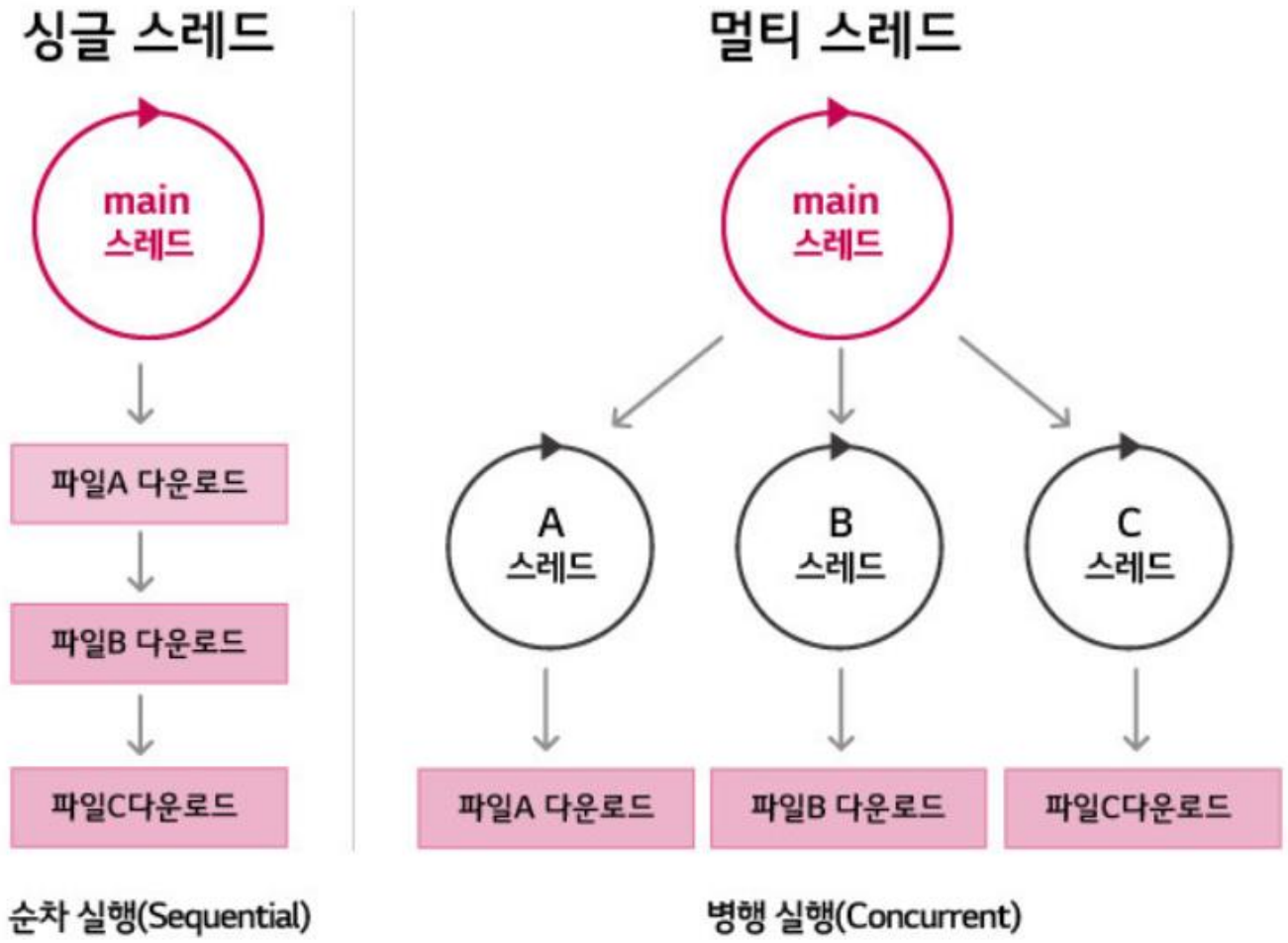
- 소스 코드의 흐름이 위에서 아래, 좌에서 우를 보장하지 않는다.
- 순서가 보장되지 않는 대신 오래 걸리는 작업 대신, 다른 작업을 먼저 진행할 수 있다.

비동기 프로그래밍을 사용하는 이유

자바스크립트는 싱글 스레드 기반 비동기 프로그래밍

- **따라서 멀티 스레드 방식이 아닌 비동기 프로그래밍을 해야 한다.**

Single Thread, Multi Thread 요약 비교



출처: https://blog.lgcns.com/1084

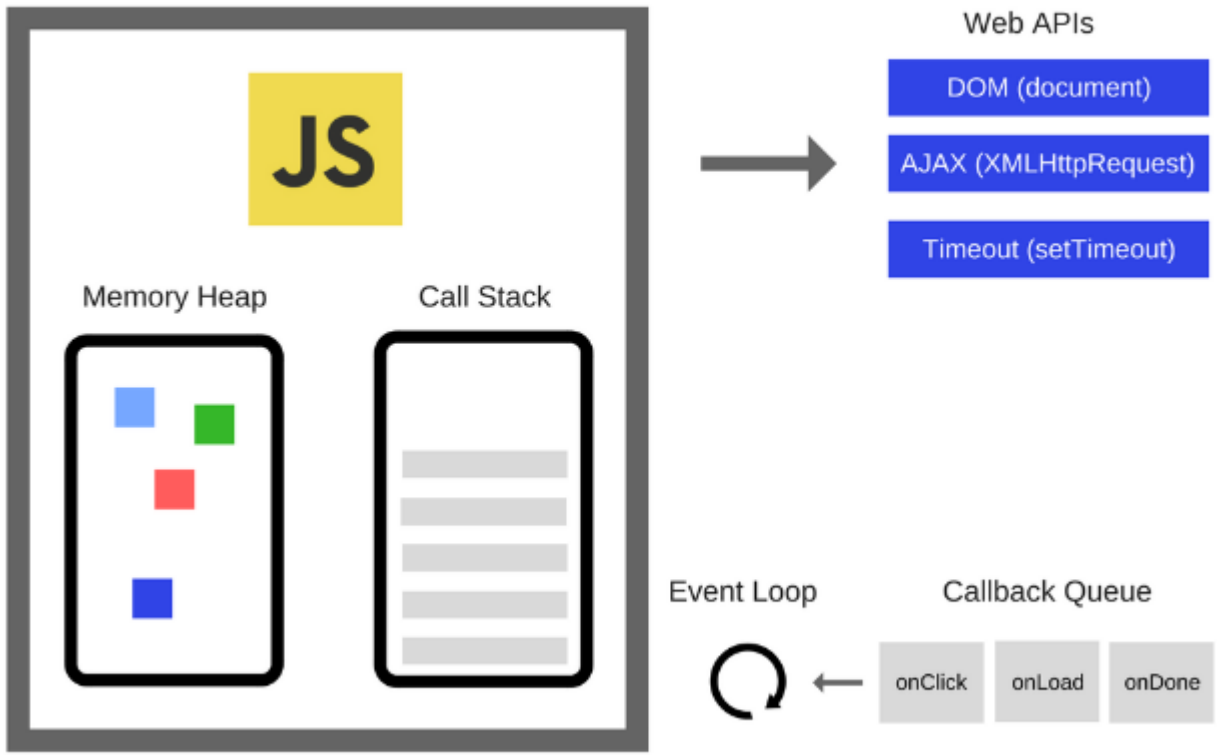
Multi Thread (멀티 스레드 방식)

- 여러 함수들을 동시에 수행할 수 있다.

Asynchronous (비동기 방식)

- 싱글 스레드 기반의 한계를 극복하기 위해 나온 프로그래밍
- 시간이 걸리는 함수를 잠시 보류하고, 다른 함수부터 수행한다.

비동기 통신 동작 방식



동기 방식의 sleep 함수 구현하기

sleep 함수 구현

- 시작 시간을 저장 해 둠
- Time Limit 시간을 계산
- 현재 시간이 Time Limit 시간이 될 때까지 무한 Loop

```
1  function sleep(delay){
2      const start = Date.now();
3      let limit = start + delay;
4      while(Date.now() < limit){}
5  }
6
7  console.log("빨래");
8  sleep(3000);
9  console.log("설거지");
```

3,000ms 동안 다른 작업을 수행 할 수 없다.

setTimeout 함수

1. 콜백함수를 등록 해 둠
2. 특정 시간이 지나면 콜백함수가 호출 됨

```
1   setTimeout(() => {
2       console.log("빨래")
3   }, 3000)
4
5   console.log("설거지")
6
7   // setTimeout(() => {
8   //     console.log("설거지")
9   // }, 1000)
```

3,000ms 동안 다른 작업을
수행이 가능하다.

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Callback

비동기와 콜백 함수의 관계

setTimeout 비동기 함수를 학습하면서 첫번째 매개변수에 함수를 넣었다. 비슷한 방식을 이미 학습하였다.

- addEventListener의 두번째 매개변수에 콜백함수를 넣었었다.

```
const body = document.querySelector('body');

body.addEventListener('click',  function(){
    console.log("hi")
});
```

콜백 함수의 두 가지 의미

‘등록’되는 함수를 Callback 함수라고 한다.

- **이벤트 발생시 호출되는 예약 함수**

  클릭 이벤트시 호출 될 함수를 등록한다. (= 콜백함수 등록)

  그리고, 클릭 이벤트 발생시 등록된 콜백함수가 호출 된다.

- **일반 함수의 Parameter로 등록되는 함수**

```
function run(e){
    alert(e.code);
}

document.addEventListener('keyup', run);
```

run은 event의 callback 함수이다.

비동기의 단점

다음 코드의 결과를 예상해보자.

```
setTimeout(() => {
    console.log("첫번째 일")
},5000)
setTimeout(() => {
    console.log("두번째 일")
},3000)
setTimeout(() => {
    console.log("세번째 일")
},1000)
setTimeout(() => {
    console.log("네번째 일")
},2000)
setTimeout(() => {
    console.log("다섯번째 일")
},4000)
```

결과

- 순서가 보장이 안되어서 어떤 일이 먼저 끝날지 보장할 수 없다

세번째 일	test.js:8
네번째 일	test.js:11
두번째 일	test.js:5
다섯번째 일	test.js:14
첫번째 일	test.js:2

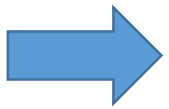
- 그렇다면 비동기 함수로 순서를 보장 시킬 수 없을까?

비동기 방식의 순서 보장

비동기 함수의 순서 보장이 가능하다.

- 첫번째 비동기 함수의 뒤에, 두번째 비동기 함수를 넣으면 된다.

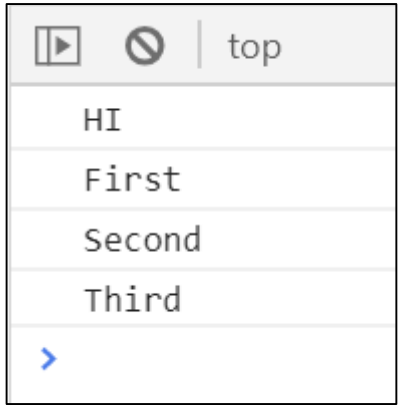
```
setTimeout(() => {
    console.log("첫번째 일")
    setTimeout(() => {
        console.log("두번째 일")
    },3000)
},5000)
```

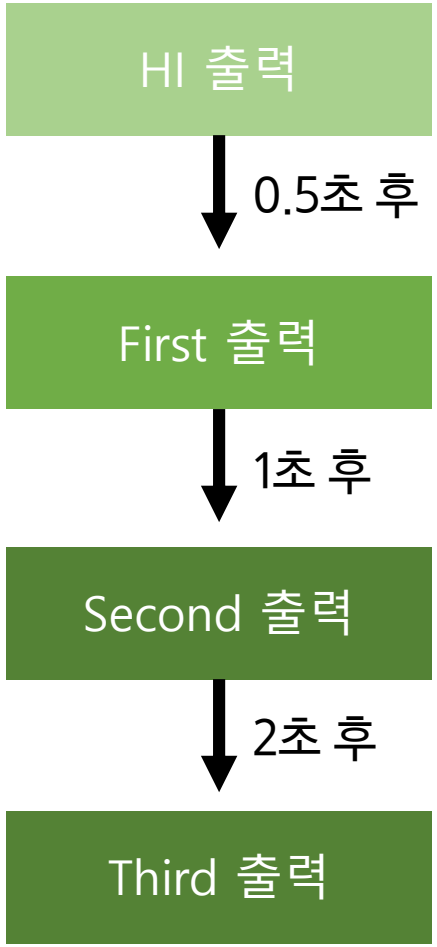


첫번째 일
두번째 일

setTimeout 을 사용하여 Sync(동기) 동작 구현

1. 웹페이지가 로딩 되자 마자 console.log 에 HI 출력
2. 0.5 초 후 First가 콘솔에 출력
3. 1 초 후 Second가 콘솔에 출력
4. 2 초 후 Third가 콘솔에 출력





비동기의 순서보장의 단점

다음 소스코드를 이해 해 보자.

- 비동기 함수로 동기 동작 구현하는 코드

```
<script>

  console.log("HI");

  setTimeout(()=>{
    console.log("one");
    setTimeout(() => {
      console.log("two");
      setTimeout(() => {
        console.log("three");
        setTimeout(() => {
          console.log("Four");
          setTimeout(() => {
            console.log("Five");
          }, 1000);
        }, 800);
      }, 600);
    }, 400);
  }, 200);

</script>
```

Async 함수들로 Sync 동작을 구현하는 코드 = Callback Hell

- **이해하기 힘들 정도로 가독성이 떨어져, 유지보수가 힘들다.**

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

출처 : https://medium.com/@quyetvv/async-flow-from-callback-hell-to-promise-to-async-await-2da3ecfff997

SAMSUNG SW AI ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

내일 방송에서 만나요!

삼성청년SW·AI아카데미

오늘의 과제

과제1(23p)
map 활용하기

과제2(26p)
filter 활용하기 2

과제3(37p)
배열 고차함수 연습하기(1)

과제4(38p)
배열 고차함수 연습하기(2)

과제5(39p)
배열 고차함수 연습하기(3)

과제6(40p)
배열 고차함수 연습하기(4)

과제7(41p)
배열 고차함수 연습하기(5)