-javascript: Callback 함수

콜백(Callback)함수란?

파라미터로 함수를 전달하는 함수이다.

구조는 다음과 같다.

```
let number = [1, 2, 3, 4, 5];

number.forEach(x => {
    console.log(x * 2);
});

<output>
2
4
6
8
10
```

위 코드의 예로는 forEach함수의 경우 함수안에 익명의 함수를 넣어서 forEach문을 동작시킨다.

콜백함수의 사용 원칙으로는

익명의 함수 사용

함수의 이름 넘기기

전역,지역변수 파라미터 전달기능이 있다.

익명의 함수 사용으로는 위 사진과 같은 구조로 사용된 것과 동일한 방식이며
화살표함수에서 일반 함수로만 바꾸어준다면 다음과 같다.

익명의 함수 사용

```
let number = [1, 2, 3, 4, 5];

number.forEach(function(x) {
    console.log(x * 2);
});
```

함수의 이름 넘기기는 다음과 같은 구조를 가지고있다.

함수의 이름(만) 넘기기

```
function whatYourName(name, callback) {
    console.log('name: ', name);
    callback();
}

function finishFunc() {
    console.log('finish function');
}

whatYourName('miniddo', finishFunc);

<output>
name: miniddo
finish function
```

말 그대로 매개변수자리에 함수이름을 넣어 코드상 매개변수로
입력받은 함수를 다시 호출하는식으로 출력이 이루어진다.

추가로 javascript는 null과 undefind타입을 제외하고는 모든 것을 객체로 다룬다.
함수를 변수나 다른 함수의 변수처럼 사용할 수 있으며 함수를 콜백함수로 사용 할 경우,
매개변수로 넘겨주기만 하면 된다.
그렇기 때문에 위 매개변수자리에는 callback 뒤에 ()가 없는 것을 확인할 수 있다.

전역,지역 변수 콜백함수의 전달기능으로는

전역변수, 지역변수 콜백함수의 파라미터로 전달 가능

- 전역변수(Global Variable) : 함수 외부에서 선언된 변수
- 지역변수(Local Variable) : 함수 내부에서 선언된 변수

```
let fruit = 'apple';    // Global Variable

function callbackFunc(callback) {
    let vegetable = 'tomato';   // Local Variable
    callback(vegetable);
}

function eat(vegetable) {
    console.log(`fruit: ${fruit} / vegetable: ${vegetable}`);
}

callbackFunc(eat);

<output>
fruit: apple / vegetable: tomato
```

다음과같이 함수내부에서 넘겨진 변수의 출력문을 출력할 때 전역변수와 넘겨받은 지역변수를 각각 출력하는것으로
이것이 가능하다는 것을 알고 넘어갈 수 있다.

콜백함수를 사용하며 주의할 점으로는
This를 사용한 콜백함수, 콜백지옥이 있다.

this를 사용할 때는 다음과같은 문제점이 있다.

```
let userData = {
    signUp: '2020-10-06 15:00:00',
    id: 'minidoo',
    name: 'Not Set',
    setName: function(firstName, lastName) {
        this.name = firstName + ' ' + lastName;
    }
}

function getUserName(firstName, lastName, callback) {
    callback(firstName, lastName);
}

getUserName('PARK', 'MINIDDO', userData.setName);

console.log('1: ', userData.name);
console.log('2: ', window.name);

<output>
1: Not Set
2: PARK MINIDDO
```

다음과같이 1번출력은 not Set이 나오게된다. 코드의 로직을 따라가보면 getusername함수가 실행됨으로써 userData.setName callback함수가 작동하여 Park MINIDDO 가 출력될 것 같았으나
getUserName()은 전역함수이기때문에 의도대로 출력문이 출력되지않았다
즉 setName에서 사용된 객체가 window라는 글로벌객체를 가리키고있기 때문에 this를 보호하도록 콜백함수를 구성해야한다.

해결방안으로는 call, apply를 이용하는 방법이 존재하며 다음과 같다.

> 해결 방안 : `call()` 과 `apply()` 를 사용하여 `this` 를 보호할 수 있다.

- `call()` : 첫 번째 인자로 `this` 객체 사용, 나머지 인자들은 `,` 로 구분
- `apply()` : 첫 번째 인자로 `this` 객체 사용, 나머지 인자들은 배열 형태로 전달

```
// call

...

function getUserName(firstName, lastName, callback, obj) {
    callback.call(obj, firstName, lastName);    - (1)
}

getUserName('PARK', 'MINIDDO', userData.setName, userData);     - (2)

console.log(userData.name);

<output>
PARK MINIDDO
```

위 코드에서 (2)에 마지막 인자인 userData는 (1)에서 call함수의 첫번째 인자로 사용한다. 즉 call에 의해 userData에
this객체가 매핑되는 구조이다.

다음으로 apply이다.

```
// apply

...

function getUserName(firstName, lastName, callback, obj) {
    callback.apply(obj, [firstName, lastName]);
}

getUserName('PARK', 'MINIDDO', userData.setName, userData);

console.log(userData.name);

<output>
PARK MINIDDO
```

apply도 역시나 userData를 넘겨주고 첫번째 인자로 이용되었으며 처음 코드의 의도대로 PARKMINIDDO가 호출되는것을 확인할 수 있다.

다음으로 콜백지옥에대한 문제이다.
비동기 호출시 자주 일어나는 프로그램의 경우 '콜백 지옥'이 발생한다.
함수의 매개변수로 넘겨지는 콜백함수가 반복되어 코드의 들여쓰기 수준이 감당하기 힘들어질 정도로 깊어지는 현상이다.

```
function add(x, callback) {
    let sum = x + x;
    console.log(sum);
    callback(sum);
}

add(2, function(result) {
    add(result, function(result) {
        add(result, function(result) {
            console.log('finish!!');
        })
    })
})

<output>
4
8
16
finish!!
```

해결방안으로는 Promise를 사용하여 콜백지옥을 탈출할 수 있다.

```
function add(x) {
    return new Promise((resolve, reject) => {
        let sum = x + x;
        console.log(sum);
        resolve(sum);
    })
}

add(2).then(result => {
    add(result).then(result => {
        add(result).then(result => {
            console.log('finish!!');
        })
    })
})

<output>
4
8
16
finish!!
```

다음과 같이 Promise는 정상 수행 후 resolve, 실패 후 reject가 실행된다.
callback을 사용했던 것과 마찬가지로 resolve에 값을 담아 전달한다.

하지만 이러한 방법도 좋은 방법이 아니며 결국 콜백지옥처럼 들여쓰기 수준을 감당하기 힘들어진다.

마지막해결방안으로 Promise의 return을 사용하여 콜백지옥을 탈출하는방법이 있다.
프로미스를 사용하여 비동기메서드지만 동기메서드와같이 값을 반환한다. 다만 최종결과를 반환하지는 않고, 대신 프로미스를 반환해서 미래의 어떤 시점에 결과를 제공한다.
다음과같이 resolve를 통해 전달받은 값을 반환하여 사용해야 한다.

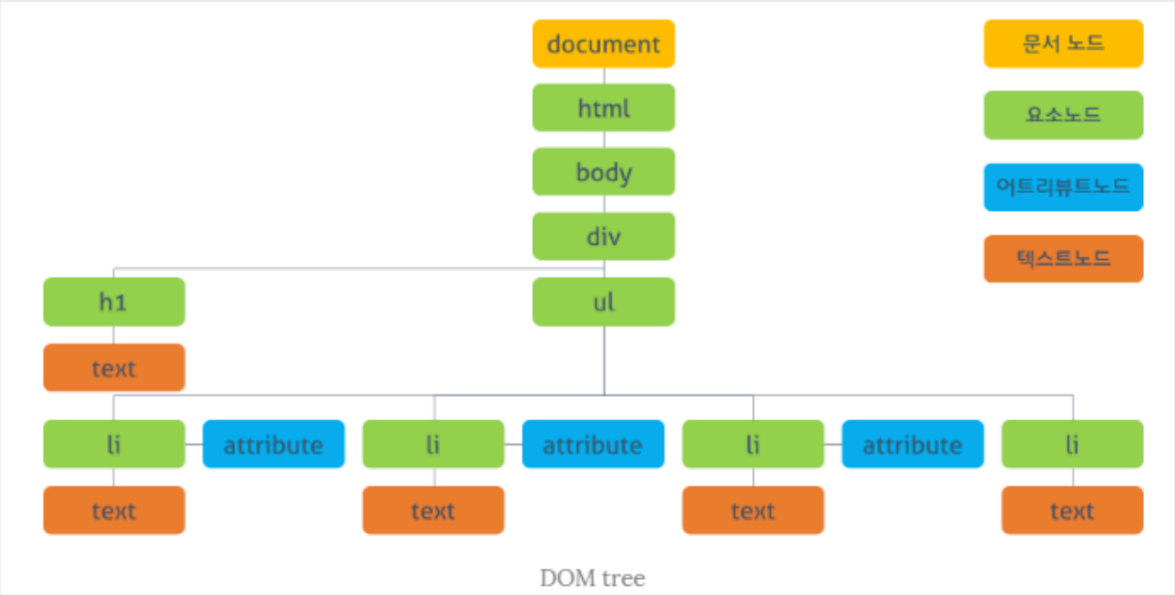
```
function add(x) {
    return new Promise((resolve, reject) => {
        let sum = x + x;
        console.log(sum);
        resolve(sum);
    })
}

add(2).then(result => {
    return add(result);
}).then(result => {
    return add(result);
}).then(result => {
    console.log('finish!!');
})

<output>
4
8
16
finish!!
```

결론적으로 자바스크립트를 사용하며 이러한 콜백지옥과같은 문제는 Promise와 async/await을 사용하여 비동기메서드가 많아지고 복잡해지는 함수구성방식을 최대한피하며 코드를 작성해 나가야 할 것이다.

DOM(Document Object Model)은 문서 객체 모델을 의미한다.
즉 객체지향 모델로써 구조화된 문서를 표현하는 형식이다.
쉽게말해 텍스트파일로 구성된 웹 문서를 브라우저에 렌더링하는과정에서 브라우저가 이해할 수 있는 구조로 메모리에 올려야 한다. 즉 앞서 공부하였던 HTML구조의 body부분을 명확하게 구성하여야 브라우저가 그 코드를 이해할 수 있게되고 이런 방식으로 구성된 것을 DOM이라 부른다. 또 모든 요소들과의 관계를 부자 관계로 표현할 수 있는 트리구조로 구성되어있으며 이 DOM은 자바스크립트를 통해 동적으로 변경할 수 있다.

-javascript: BOM, DOM

다음은 DOM Tree 이다.



DOM tree

사진과 같이 트리구조로 연결되어있으며 부모와 자식 관계가 형성되어 있음을 알 수 있다.
이러한 트리구조에서 노드는 9종류이며 주로쓰는 노드는 4종류이다.
4종류의 노드는 다음과 같다.

문서노드(Document Node)	트리의 최상위에 존재하며 각각의 하위요소들(엘리먼트, 어트리뷰트, 텍스트 노드)에 접근하려면 문서노드를 통해야 한다. 즉, 시작점이다.
요소노드(Element Node)	쉽게 말해 태그이다. <p> 등..
어트리뷰트노드(Attribute Node)	<input> 태그 안에는 name, value 등의 속성을 사용할 수 있는데 이러한 속성들을 가리키는 노드이다.
텍스트 노드(Text Node)	태그 내 텍스트를 표현한다. 텍스트 노드는 엘리먼드 노드의 자식이며 자신의 자식 노드를 가질 수 없기 때문에 돔 트리의 최종단이다. 안녕일 경우 텍스트 노드는 '안녕'이다.

BOM(Browser Object Model): 브라우저 객체 라고부르며 자바스크립트가 브라우저와 소통하기위해 만들어진 모델이다.
브라우저 대부분이 자바스크립트와의 상호작용에 있어서 비슷한 메소드와 속성으로 동작하기에 이와 같은 메소드들을 통칭하여 BOM이라 말한다.

navigator	브라우저명과 버전정보를 속성으로 가짐
window	최상위 객체로, 각 프레임별로 하나씩 존재
document	현재 문서에 대한 정보
location	현재 URL에 대한 정보 브라우저에서 사용자가 요청하는 URL
history	현재의 브라우저가 접근했던 URL history
screen	브라우저의 외부환경에 대한 정보를 제공