**1. 콜백함수를 코드예시를 들어 설명하시오**

**비동기 처리? 그게 뭔가요?**

자바스크립트의 비동기 처리란 특정 코드의 연산이 끝날 때까지 코드의 실행을 멈추지 않고 다음 코드를 먼저 실행하는 자바스크립트의 특성을 의미합니다.

Jquery로 작성된 ajax 코드를 잠깐 살펴보겠습니다.

function getData() {

var tableData;

$.get('https://domain.com/products/1', function(response) {

tableData = response;

});

return tableData;

}

console.log(getData()); *// undefined*

여기서 $.get()이 ajax 통신을 하는 부분입니다. https://domain.com 에다가 HTTP GET 요청을 날려 1번 상품(product) 정보를 요청하는 코드죠. 좀 더 쉽게 말하면 지정된 URL에 ‘데이터를 하나 보내주세요’ 라는 요청을 날리는 것과 같습니다.

그렇게 서버에서 받아온 데이터는 response 인자에 담깁니다. 그리고 tableData = response; 코드로 받아온 데이터를 tableData라는 변수에 저장합니다. 그럼 이제 이 getData()를 호출하면 어떻게 될까요? 받아온 데이터가 뭐든 일단 뭔가 찍혀야겠죠. 근데 결과는 맨 아래에서 보시는 것처럼 undefined입니다. 왜 그럴까요?

그 이유는 $.get()로 데이터를 요청하고 받아올 때까지 기다려주지 않고 다음 코드인 return tableData;를 실행했기 때문입니다. 따라서, getData()의 결과 값은 초기 값을 설정하지 않은 tableData의 값 undefined를 출력합니다.

이렇게 특정 로직의 실행이 끝날 때까지 기다려주지 않고 나머지 코드를 먼저 실행하는 것이 비동기 처리입니다. 자바스크립트에서 비동기 처리가 필요한 이유를 생각해보면, 화면에서 서버로 데이터를 요청했을 때 서버가 언제 그 요청에 대한 응답을 줄지도 모르는데 마냥 다른 코드를 실행 안 하고 기다릴 순 없기 때문입니다. 위에선 간단한 요청 1개만 보냈는데 만약 100개 보낸다고 생각해보세요. 비동기 처리가 아니고 동기 처리라면 코드 실행하고 기다리고, 실행하고 기다리고.. 아마 웹 애플리케이션을 실행하는데 수십 분은 걸릴 겁니다.

**콜백 함수로 비동기 처리 방식의 문제점 해결하기**

앞에서 자바스크립트 비동기 처리 방식에 의해 야기될 수 있는 문제들을 살펴봤습니다. 이러한 문제들은 어떻게 해결할 수 있을까요? 바로 콜백(callback) 함수를 이용하는 것입니다. 앞에서 살펴본 ajax 통신 코드를 콜백 함수로 개선해보겠습니다.

function getData(callbackFunc) {

$.get('https://domain.com/products/1', function(response) {

callbackFunc(response);

*// 서버에서 받은 데이터 response를 callbackFunc() 함수에 넘겨줌*

});

}

getData(function(tableData) {

console.log(tableData); *// $.get()의 response 값이 tableData에 전달됨*

});

이렇게 콜백 함수를 사용하면 특정 로직이 끝났을 때 원하는 동작을 실행시킬 수 있습니다.

**콜백 지옥 (Callback hell)**

콜백 지옥은 비동기 처리 로직을 위해 콜백 함수를 연속해서 사용할 때 발생하는 문제입니다. 아마 아래와 같은 코드를 본 적이 있을 겁니다.

$.get('url', function(response) {

parseValue(response, function(id) {

auth(id, function(result) {

display(result, function(text) {

console.log(text);

});

});

});

});

웹 서비스를 개발하다 보면 서버에서 데이터를 받아와 화면에 표시하기까지 인코딩, 사용자 인증 등을 처리해야 하는 경우가 있습니다. 만약 이 모든 과정을 비동기로 처리해야 한다고 하면 위와 같이 콜백 안에 콜백을 계속 무는 형식으로 코딩을 하게 됩니다. 이러한 코드 구조는 가독성도 떨어지고 로직을 변경하기도 어렵습니다. 이와 같은 코드 구조를 콜백 지옥이라고 합니다.

**콜백 지옥을 해결하는 방법**

일반적으로 콜백 지옥을 해결하는 방법에는 Promise나 Async를 사용하는 방법이 있습니다. 만약 코딩 패턴으로만 콜백 지옥을 해결하려면 아래와 같이 각 콜백 함수를 분리해주면 됩니다.

function parseValueDone(id) {

auth(id, authDone);

}

function authDone(result) {

display(result, displayDone);

}

function displayDone(text) {

console.log(text);

}

$.get('url', function(response) {

parseValue(response, parseValueDone);

});

위 코드는 앞의 콜백 지옥 예시를 개선한 코드입니다. 중첩해서 선언했던 콜백 익명 함수를 각각의 함수로 구분하였습니다. 정리된 코드를 간단하게 살펴보겠습니다. 먼저 ajax 통신으로 받은 데이터를 parseValue() 메서드로 파싱 합니다. parseValueDone()에 파싱 한 결과값인 id가 전달되고 auth() 메서드가 실행됩니다. auth() 메서드로 인증을 거치고 나면 콜백 함수 authDone()이 실행됩니다. 인증 결과 값인 result로 display()를 호출하면 마지막으로 displayDone() 메서드가 수행되면서 text가 콘솔에 출력됩니다.