**콜백**

**함수 다시 집고 넘어가기**

* 매개변수란 무엇인가 ?
* 전달인자란 무엇인가 ?
* 함수 선언식으로 2개의 전달인자를 받아 두개의 값을 더해 출력하는 함수를 만드세요
* 함수 표현식으로 2개의 전달인자를 받아 두개의 값을 곱하여 출력하는 함수를 만드세요. multiplication
* Return 이라고 불리우는 반환이란 키워드는 무엇인가요 ? 함수가 값이 되는 것.
* 기명함수와 익명함수의 차이는 무엇인가요 ?
* IIFE 즉시 실행함수를 하나 만들어 보세요.
* 스코프란 무엇인가요 ?
* 클로저란 무엇인가요 ?
* 클로저를 이용하여, globalFunc 변수를 localVar 변수에 접근 하도록 해보시오 .
* 블럭과 객체리터럴의 차이는 ?

**함수는 일급 객체이다.**

자바스크립트에서 함수는 일급 객체입니다. 일급 객체란 말은 영국의 컴퓨터 과학자 크리스토퍼 스트래치가 처음 언급한 내용입니다.

일급 객체란 다음과 같은 특성을 가진 객체입니다.

1. 모든 요소는 함수의 실제 매개변수가 될 수 있다.
2. 모든 요소는 함수의 반환 값이 될 수 있다.
3. 변수나 데이터 구조 안에 담을 수 있다.
4. 런타임에 생성될 수 있다.

자바스크립트의 함수는 이 모든 조건을 만족 하기에 자바스크립트의 함수는 일급 객체라고 할 수 있습니다.

**콜백**

콜백은 함수를 함수의 전달 인자로 사용하는 기법으로 이를 이용하여, 제이쿼리나 여러 통신 지연 시간 문제를 해결 할 수 있습니다.

콜백 함수는 함수형 프로그래밍에서 발생한 것으로, 함수형 프로그래밍은 함수를 인자처럼 사용하는 것인데, 이는 소수의 전문가들만이 이해하고 사용하는 기법이었지만 지속적으로는 누군가에 의해 설명되고 분석 되어 오면서 우리가 쉽게 이해하고 사용 할 수 있는 수준이 되었습니다.

그 중에서 가장 일반적인 기술 중 하나가 바로 콜백 함수 입니다.

**콜백의 예**

콜백은 간단하게 함수의 전달 인자로 전달하는 함수를 말한다

function print(num\_1,num\_2,compare){

if(compare(num\_1,num\_2)){

return `${num\_1} 이 큽니다.`;

}else{

return `${num\_2} 이 큽니다.`;

}

}

function compare(num\_1,num\_2){

console.log(`${num\_1} 과 ${num\_2}를 비교 중 입니다.`);

if(num\_1 > num\_2){

return true;

}else{

return false;

}

}

console.log(print(10,20,compare));

*//콘솔 로그로 print함수를 준 것도 콜백 패턴을 이용한 것이다.*

**함수형 프로그래밍 간단한 예**

*//check loto*

function makeCoffe(){

console.log("1번 절차 : 원두를 그라인더에 간다. 이때 그라인더가 정상인지 확인한다.");

let blender = Math.floor(Math.random() \* 10);

if( blender % 2 === 0){

return "실패";

}

let tamping = Math.floor(Math.random() \* 10);

console.log("2번 절차 : 포터필터 옮겨 담아 적절한 힘으로 탬핑을 한다. 이 때 중심이 흐트러지면 부시고 다시 탬핑한다.");

while(tamping % 2 === 0){

console.log("부시고 다시하는 중입니다.");

tamping = Math.floor(Math.random() \* 10);

}

console.log("3번 절차 : 커피 머신에 포터필터를 장착한 후 버튼을 누릅니다. 이때 25초 ~ 40초 사이에 나와야 함 아니라면 실패");

let out = Math.floor(Math.random() \* 10);

if( out % 9 === 0){

return "실패";

}

console.log("4번 절차 : 커피를 뜨거운 물에 붙는다. ");

return "커피"

}

이 함수는 유연하지 않다. makeCoffe함수가 모든 동작이 makeCoffe한 함수에서 다 정의되어 이 함수의 동작 중 연속적인 부분에서 오류가 생기는 경우 모든 동작에 영향을 주어 원하는 결과가 안 생길 위험도 크고, 설사 틀린 부분을 고친다고 해도 그 부분이 다른 곳에 어떤 영향을 주는지 쉽게 알기 어렵다. 또한 한개의 함수에서 모든 변수 동작을 관리하다 보니 같은 스코프내에서 변수명이 겹치는 위험도 크다. 그래서 기능 별로 함수를 만들어 makeCoffe함수에 전달 하면 어느 함수에서 고장 났는지도 쉽게 알 수 있고, 스코프 겹침에 의한 변수명 충돌도 막을 수 있다. 또한 새로운 내용을 도입할 시, 새로만든 함수를 전달 하기만 하면 되니 유지보수도 쉬워졌다.

function blender(){

console.log("1번 절차 : 원두를 그라인더에 간다. 이때 그라인더가 정상인지 확인한다.");

let blender = Math.floor(Math.random() \* 10);

if( blender % 2 === 0){

console.log("그라인더가 정상입니다.")

}else{

console.log("그라인더가 비 정상입니다.")

}

}

function tamping(){

let tamping = Math.floor(Math.random() \* 10);

console.log("2번 절차 : 포터필터 옮겨 담아 적절한 힘으로 탬핑을 한다. 이 때 중심이 흐트러지면 부시고 다시 탬핑한다.");

while(tamping % 2 === 0){

console.log("부시고 다시하는 중입니다.");

tamping = Math.floor(Math.random() \* 10);

}

}

function out(){

console.log("3번 절차 : 커피 머신에 포터필터를 장착한 후 버튼을 누릅니다. 이때 25초 ~ 40초 사이에 나와야 함 아니라면 실패");

let out = Math.floor(Math.random() \* 10);

if( out % 9 === 0){

console.log("추출에 실패했습니다.");

}else{

console.log("추출에 성공 했습니다.");

}

}

*//check loto*

function makeCoffe(tamping, blender, out){

blender()

tamping()

out()

console.log("4번 절차 : 커피를 뜨거운 물에 붙는다. ");

return "커피"

}

makeCoffe(tamping,blender,out);