재귀

작성일시	@May 29, 2023 10:23 AM
자료	
# 주차	1

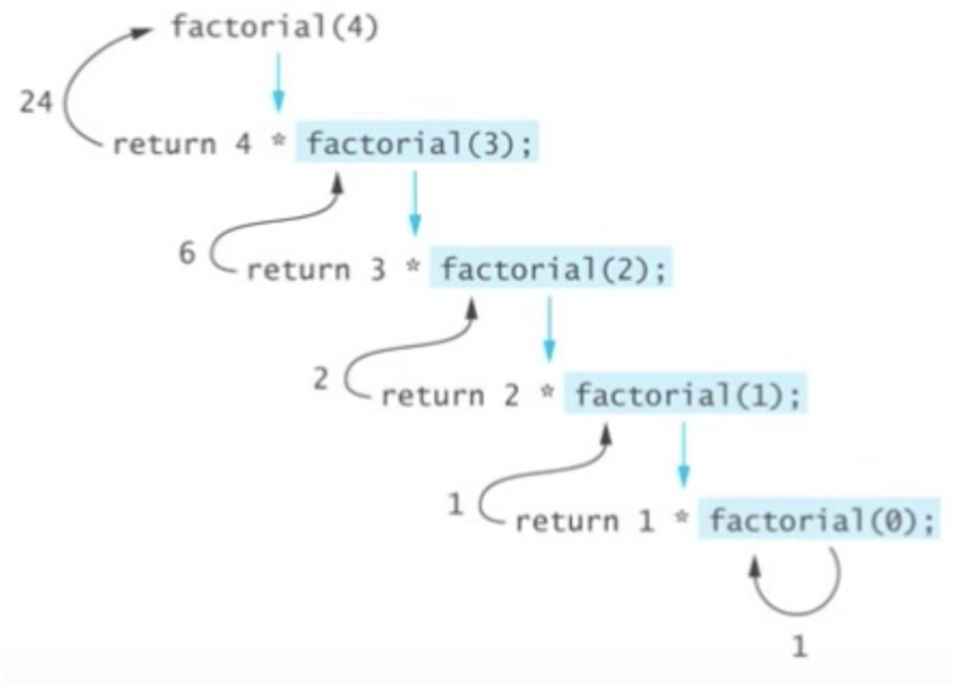
재귀함수

함수가 자기 자신을 호출하는 것을 재귀 호출(recursive call)이라 한다. 재귀 함수(recursive function)는 **자기 자신을 호출하는 행위**, 즉 재귀 호출을 수행하는 함수를 말한다.

재귀 함수를 사용하면 반복되는 처리를 반복문 없이 구현할 수 있다.

```
function factorial(n) {
  if(n <= 1) return 1;
  return n*factorial(n-1);
}

console.log(factorial(0)); // 0! = 1
console.log(factorial(1)); // 1! = 1
console.log(factorial(2)); // 2! = 2*1
console.log(factorial(3)); // 3! = 3*2*1
console.log(factorial(4)); // 4! = 4*3*2
console.log(factorial(5)); // 5! = 5*4*3*2*1
```



재귀함수는 자신을 무한 재귀 호출한다(무한 루프에 빠짐). 따라서 재귀 함수 내에는 재귀 호출을 멈출 수 있는 탈출 조건, 베이스 케이스(base case)를 반드시 만들어야 한다.

재귀함수는 반복문을 사용하는 것보다 재귀 함수를 사용하는 편이 더 직관적으로 이해하기 쉬울 때만 한정적으로 사용하는 것이 바람직하다.

재귀 vs 반복문

일반적으로 반복문이 실행 속도가 빠른데 재쉬함수를 사용하는 이유는 무엇일까? 재귀 함수로 표현하면 가독성이 좋아진다. 또한 변수 사용을 줄일 수 있다.

```
// for loop
function factorial(num) {
  let result = 1;
  for(let i=0; i<num; i++) {
    result *= (i+1);
  }
  return result;
}
```

위 예제와 같이 반복문을 사용해서 팩토리얼을 만든다면 이런 방식일 것이다. 함수에서 result, i라는 변수를 사용하고 있다.

```
function factorial(n) {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
}
```

하지만 재귀 함수로 만든다면 간결하게 코드를 작성할 수 있다.

재귀호출로 인한 stack overflow를 막을 수 있는 방법은?

만약 베이스 케이스를 깜빡해 재귀 호출이 무한히 계속된다면 어떤 일이 벌어질까? 실제로는 무한의 늪에 빠지기 전에 브라우저가 **스택 오버플로(stack overflow)** 에러를 던진다.

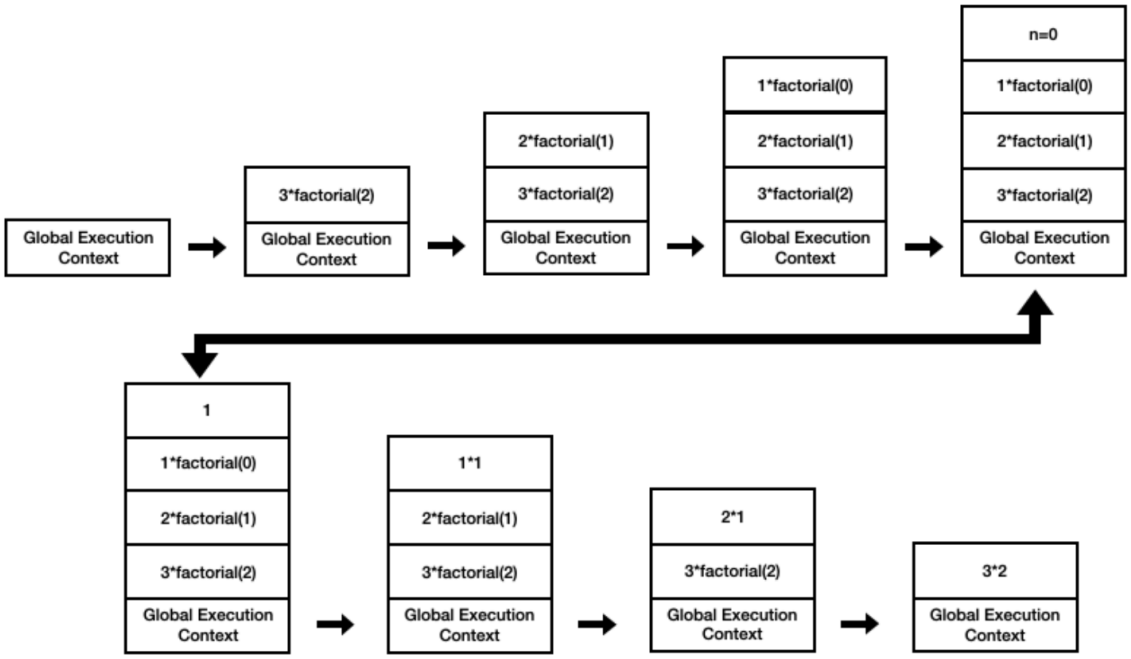
간단히 말해서 recursiveFn을 한 번 실행할 때마다 실행 컨텍스트가 생성되기 때문에 실행 컨텍스트 스택에 계속 쌓인다.

재귀호출의 한계량은 브라우저 마다 다르다. 다음 코드를 돌려보고 브라우저 종류별 한계량을 체크해보자.

```
var i = 0;
function recursiveFn() {
    i++;
    recursiveFn();
}
try{
    recursiveFn();
} catch(ex) {
    alert(`i: ${i}, error: ${ex})`)
}
```

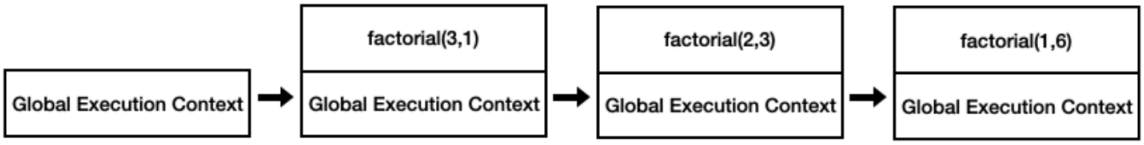
그렇다면 어떻게 해야할까?

꼬리 호출 최적화(Tail call optimization)



위의 그림은 factorial(3)을 실행했을 때 콜스택이다.

반환부에 연산이 있으므로 실행 컨텍스트가 종료되지 못하고 계속 쌓이게 된다. 따라서 반환부에 연산을 넣지 않아 실행 컨텍스트가 쌓이지 않도록 해야한다.



```
function factorial(n, acc=1) {
  if (n <= 1) return acc;
  return factorial(n - 1, n*acc);
}
```

factorial(3)을 한다면

n	acc	return
3	1	factorial(2, 3*1)
2	3*1	factorial(1, 3*2*1)
1	3*2*1	acc