[과제 2] 함수 작성법 연습 (10문제)

반드시 모든 문제는, function을 사용해 수행해야함.

1. 완전탐색

완전탐색(Exhaustive Search)이란, 가능한 모든 경우의 수를 탐색하여 답을 찾는 방법론을 말한다.

주어진 숫자들의 집합에서 원하는 숫자를 찾고자 할 때, 집합 내의 모든 원소를 하나씩 확인하면서 해당하는 수를 찾는 방법은 완전탐색이라 할 수 있다. 아래의 입출력 조건에 맞는 완전탐색 함수를 구현하시오.

입력1: 1에서 50까지 랜덤하게 선택된 20개의 수

입력2: 1에서 50까지의 수 중 사용자로부터 입력받는 하나의 수

출력: 입력 2에서 사용자로부터 받은 수가 입력 1에서 선택된 20개의 수에 포함되면 TRUE, 아니면 FALSE 출력

2. 거스름돈

고려대학교 전통 매점 또랑에서는 동전으로만 간식을 구매할 수 있다. 매점의 오랜 규칙은 간식을 구매할 때 최소한의 동전만으로 계산해야 한다는 것이다. 동전의 종류는 500원, 100원, 50원, 10원이 있다. 간식의 가격이 주어졌을 때 지불해야 하는 동전의 최소 개수를 출력하는 함수를 작성하시오.

입력: 간식의 가격 (10으로 나누어 떨어지는 임의의 수)
출력: 간식을 구매하는 데 필요한 최소한의 동전 개수

3. 영화 평점 분석

유명 영화 평론가 스미스는 매년 개봉하는 영화의 평점을 기반으로 내년 영화의 트렌드를 예측한다. 항상 최고의 자리를 유지하는 스미스만의 예측 기법은 다음과 같다. 한 해에 개봉하는 모든 영화 평점의 평균을 구한다. 그리고 그 해의 마지막 영화 평점에서 평점의 평균을 뺀다. 이 값을 다음 해 개봉하는 영화에 대한 평점 예측 평균으로 사용하며, 만약 값이 0보다 작을 경우 0으로 설정하고 10을 넘길 경우 10으로 설정한다. 한 해동안 개봉한 영화의 평점이 주어졌을 때 스미스의 기법으로 영화 평점을 예측하는 함수를 구현하시오.

입력: 0에서 10까지 랜덤하게 선택된 5개의 수

출력: 다음 해 개봉하는 영화의 평균 평점 예측값

4. 불량품 검출

A 공장에서는 어린이들을 위한 장난감을 생산한다. 장난감은 어린이들이 가지고 놀기 좋은 크기(10~30cm)로 생산되어야 하는데, 기계가 너무 노후화되어 지나치게 크거나 작은 장난감이 생산되는 문제가 있다. 장난감을 생산하는 기계는 원

재료의 크기가 입력으로 들어왔을 때, 랜덤한 숫자를 곱하여 장난감의 크기를 결정한다. 임의의 크기를 갖는 원재료가 입력으로 주어졌을 때, 장난감을 만드는 함수와 만들어진 장난감 중 불량품의 갯수를 출력하는 함수를 구현하시오.

[1] 함수 1 - 장난감 생산 기계 함수

입력: 랜덤하게 선택된 10개의 수(1 이상 30 이하)

출력: 입력값에 랜덤하게 선택된 수(1 이상 5 이하)를 곱한 결과

[2] 함수 2 - 불량품 검출 함수

입력: 랜덤하게 선택된 10개의 수(1 이상 30 이하)

출력: 크기가 불량인 장난감의 개수

주의사항: 불량품 검출 함수(함수 2) 내부에 장난감 생산 기계 함수(함수 1)를 사용하여 구현할 것

5. 계산기

사칙연산이 가능한 계산기를 함수를 통해 구현하시오. 구현하는 계산기는 숫자 2개와 연산자를 받아 연산이 가능해야 한다.

입력: 두 자연수 (number_1, number_2) / 연산자 ('+', '-', '*', '/')
출력: 두 자연수의 사칙연산 결과
ex) 입력으로 두 자연수 (10,3) / 연산자 ('*')가 들어가면 30을 출력

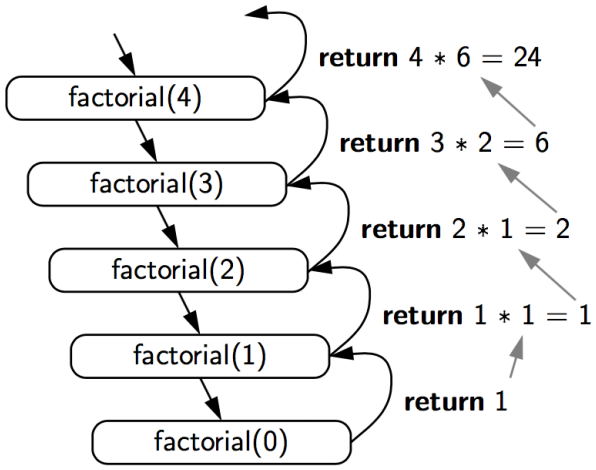
주의사항 1: print / sprintf 를 사용하여 각 과정의 결과를 표현할 것
ex) "10 * 3의 결과는 30 입니다."
주의사항 2: 나눗셈은 몫과 나머지를 분할해서 출력

```
> calculator(3, 4, '/')
[1] "3 / 4의 몫은 0 이며, 나머지는 3입니다."
```

6. 피보나치 수

수학에서, 피보나치 수(영어: Fibonacci numbers)는 첫째 및 둘째 항이 1이며 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열이다. 처음 여섯 항은 각각 1, 1, 2, 3, 5, 8이다. Fibonacci 수열을 재귀함수를 통해 구현하라.

재귀함수란 재귀적(Recursive)하게 작동하는 함수로서, 어떤 함수에서 자기 자신 함수를 다시 호출해 작업을 수행하는 방식의 함수를 말한다.



어느 한 컴퓨터공학과 학생이 유명한 교수님을 찾아가 물었다.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날에 산 꼭대기에 현자가 있었어. 질문엔 모두 지혜롭게 대답해 주었지.
그런데 어느날, 그 선인에게 한 선비가 찾아와서 물었어.
"재귀함수가 뭔가요?"
"잘 들어보게. 옛날에 산 꼭대기...

피보나치 수 문제와 비슷한 Factorial 함수의 알고리즘 / 재귀함수에 대한 이야기

입력: n번째 index
출력: Fibonacci n번째 수

```
> fibo(10)
[1] 55
```

7. *로 이루어진 삼각형

'*'로 구성된 직각삼각형을 그리시오. while과 for를 개별적으로 사용하여 함수 두개를 만드시오.

입력: 삼각형 층 수
출력: 직각 삼각형

```
> triangle_while(10)
[1] "*"
[1] "**"
[1] "***"
[1] "****"
[1] "*****"
[1] "******"
[1] "*******"
[1] "********"
[1] "*********"
[1] "**********"
[1] "***********"
```

```
> triangle_for(10)
[1] "*"
[1] "**"
[1] "***"
[1] "****"
[1] "*****"
[1] "******"
[1] "*******"
[1] "********"
[1] "*********"
[1] "**********"
[1] "***********"
```

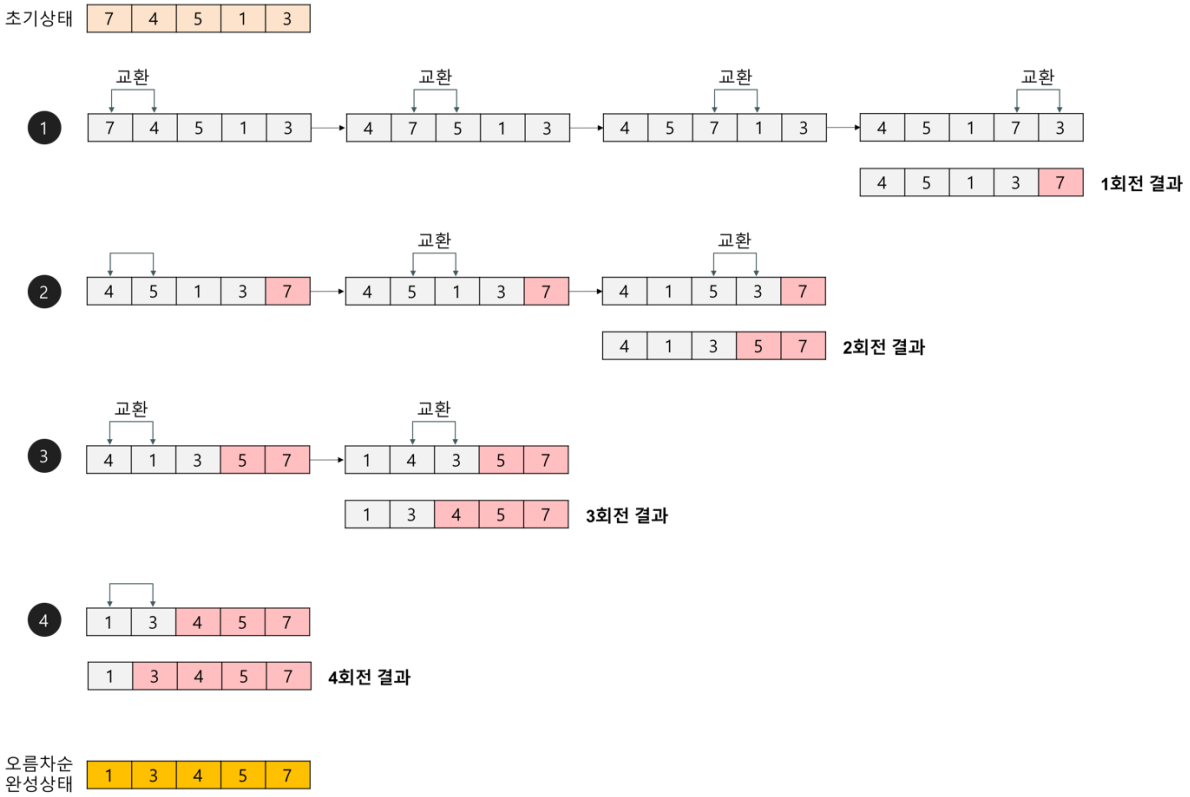
8. AI와의 가위바위보 대결

현재 우리는 제 4차 산업혁명, AI의 시대에 살고 있다. 고려대학교 학생, 영탁이는 AI와 가위바위보를 했을 때 몇 번이나 이길 수 있을 지 고민하고 있다.
"AI 가위바위보" function을 만들어 매 단계마다 승,패, 무승부를 표기하고 결과를 보고하는 함수를 제작하라. (말은 AI지만, 영탁이와 AI 모두 자신이 내는 손(가위,바위,보)을 Random을 사용하여 함수를 구성하라)

입력: 라운드 수
출력: 각 라운드 승, 패, 무승부와 최종 결과 보고

```
> AI_rps(10)
[1] "승"
[1] "무승부"
[1] "패"
[1] "무승부"
[1] "무승부"
[1] "무승부"
[1] "패"
[1] "패"
[1] "승"
[1] "패"
[1] "영탁군은 AI에 대항하여 2 승, 4 패, 4 무승부를 기록하였습니다."
```

9. BubbleSort



BubbleSort(버블 정렬)란 자료의 정렬 기법 중 하나이다. 서로 인접한 두 원소 간의 크기 비교를 통해 크거나 작은 순서대로 정렬하게 되는데, 정렬 과정이 거품이 올라오는 모양새와 비슷하여 다음와 같은 이름이 지어지게 되었다.

버블 정렬은 다음의 단계들로 이루어진다.

- 첫 번째 원소와 두 번째 원소의 크기 비교
- 두 원소가 정렬되어 있지 않다면 두 원소의 위치를 변경하여 정렬한 후, 그 다음 원소와 크기 비교
- 두 원소가 정렬되어 있다면, 두 번째와 세 번째 원소의 크기 비교
- 1회전을 완료한 경우 가장 마지막에는 가장 큰 원소가 위치
- 모든 리스트가 정렬될 때까지 반복

아래의 자료를 살펴보면, 좀 더 구체적인 BubbleSort 알고리즘을 알 수 있다.

참고 동영상: https://www.youtube.com/watch?v=lyZQPjUT5B4

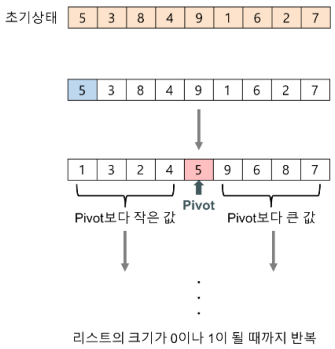
참고 블로그: https://gmlwjd9405.github.io/2018/05/06/algorithm-bubble-sort.html

BubbleSort를 함수로 구현하고 소요 시간을 확인해보시오. (함수의 시작 전과 후에 sys.time() 함수를 사용하여 시간을 체크할 수 있음)

입력: 1에서 1000까지 랜덤하게 선택된 100개의 수

출력: 크기의 오름차순으로 정렬된 수

10. QuickSort



QuickSort란 자료의 정렬 기법 중 하나이다. 하나의 리스트를 피벗(pivot)을 기준으로 두 개의 비균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 하는 방법이다.

퀵 정렬은 다음의 단계들로 이루어진다.

- 분할(Divide): 입력 배열을 피벗을 기준으로 비균등하게 2개의 부분 배열(피벗을 중심으로 왼쪽: 피벗보다 작은 요소들, 오른쪽: 피벗보다 큰 요소들)로 분할한다.
- 정복(Conquer): 부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복 방법을 적용한다.
- 결합(Combine): 정렬된 부분 배열들을 하나의 배열에 합병한다.
- 순환 호출이 한번 진행될 때마다 최소한 하나의 원소(피벗)는 최종적으로 위치가 정해지므로, 이 알고리즘은 반드시 끝난다는 것을 보장할 수 있다.

아래의 블로그를 살펴보면, 좀 더 구체적인 QuickSort 알고리즘을 알 수 있다.

참고 동영상: https://www.youtube.com/watch?v=ywWBy6J5gz8

참고 블로그: https://gmlwjd9405.github.io/2018/05/10/algorithm-quick-sort.html

입력: List

출력: Sorted List

QuickSort를 함수로 구현하고 소요 시간을 확인해보시오. (함수의 시작 전과 후에 sys.time() 함수를 사용하여 시간을 체크할 수 있음)

입력: 1에서 1000까지 랜덤하게 선택된 100개의 수

출력: 크기의 오름차순으로 정렬된 수