Make-Up

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

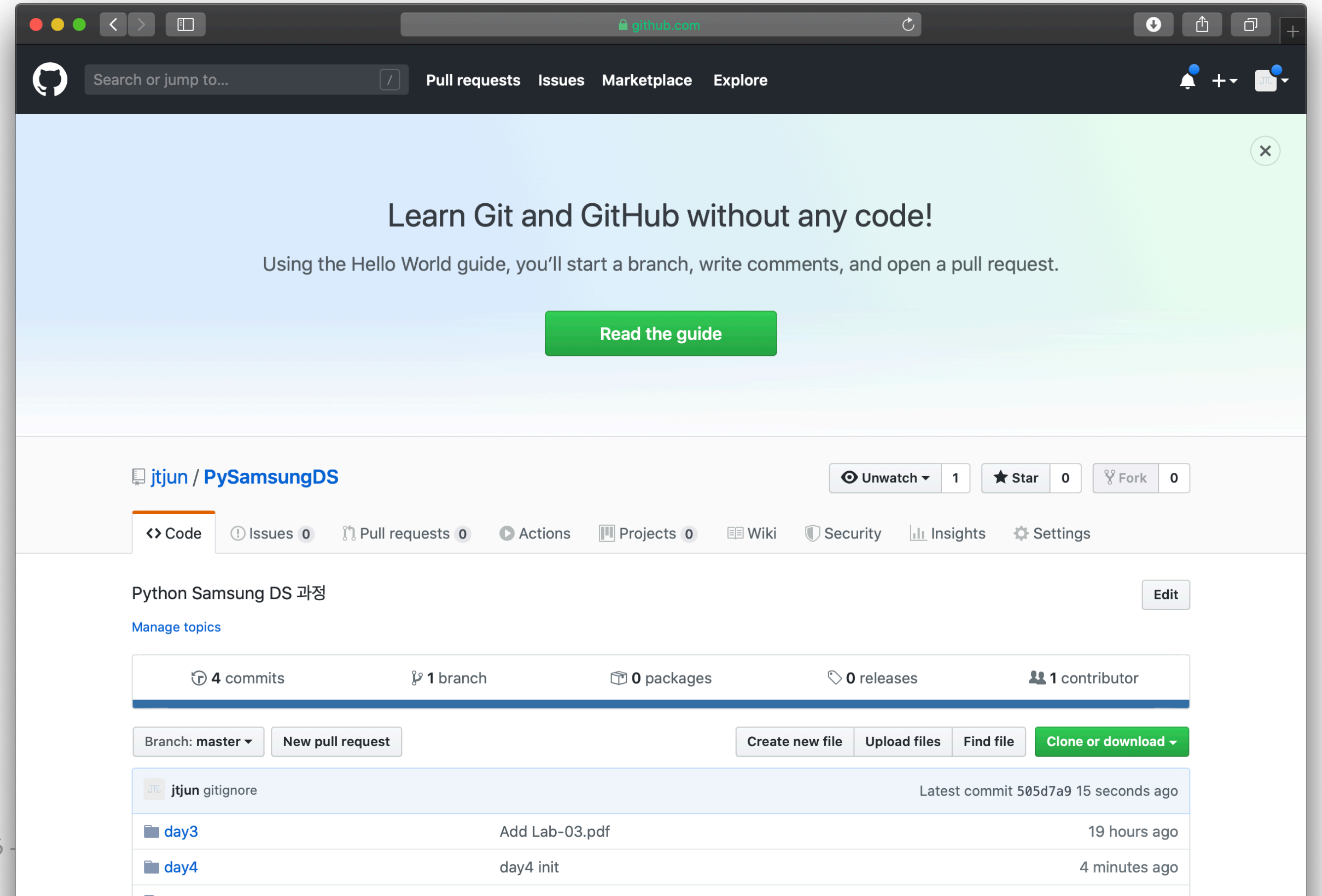
Systems Software &
Architecture Lab.

Seoul National University

Dec 16 – 20, 2019

Lab 4. 다운로드 안내

- https://github.com/jtjun/PySamsungDS
- https://github.com/jtjun/PySamsungDS/archive/master.zip



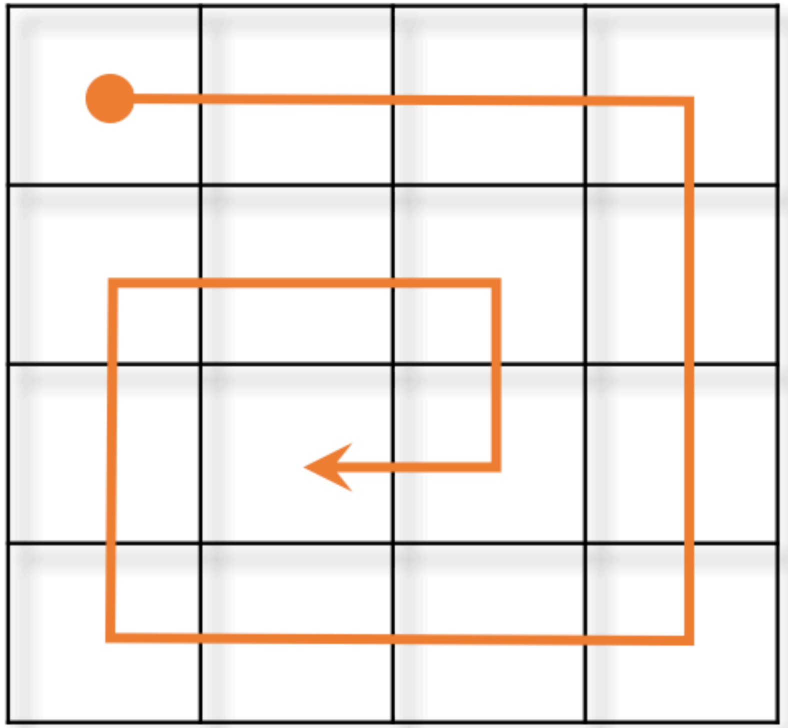
Make-Up

Lab 4-1. 달팽이 배열

- N*N 배열
  - 1부터 N^2 까지 수를 넣음
  - 달팽이 껍질 모양으로 숫자를 넣음

- 언제 진행 방향을 바꾸는가?
  - 배열 끝에 도달
  - 앞에 이미 다른 숫자 있을 때

→ 어떻게 프로그래밍할 수 있을까?



Lab 4-1. 달팽이 배열

- 언제 진행 방향을 바꾸는가?
  - 배열 끝에 도달
  - 앞에 이미 다른 숫자 있을 때

방향은 총 4 종류
오른쪽, 아래쪽, 왼쪽 위쪽 순서
→ 0, 1, 2, 3 에 대응시키기?

오른쪽, 왼쪽은 행에서 이동
위쪽, 아래쪽은 열에서 이동
→ 이중 배열에서 어떻게 표현?

```
board = [[0,1,2],
         [3,4,5],
         [6,7,8]]

# 4 : board[1][1]
5 == # 4's right
7 == # 4's below
3 == # 4's left
1 == # 4's up
```

Lab 4-1. 달팽이 배열

Skeleton
http://bitly.kr/6dP9qDuq

- 입출력 예시
  - 리스트 길이를 입력 받음
  - Hint: &   &

```
$ python snail.py
크기를 입력하세요. : 5↲
 1  2  3  4  5
16 17 18 19  6
15 24 25 20  7
14 23 22 21  8
13 12 11 10  9
```

```
$ python snail.py
크기를 입력하세요. : 3↲
1 2 3
8 9 4
7 6 5
```

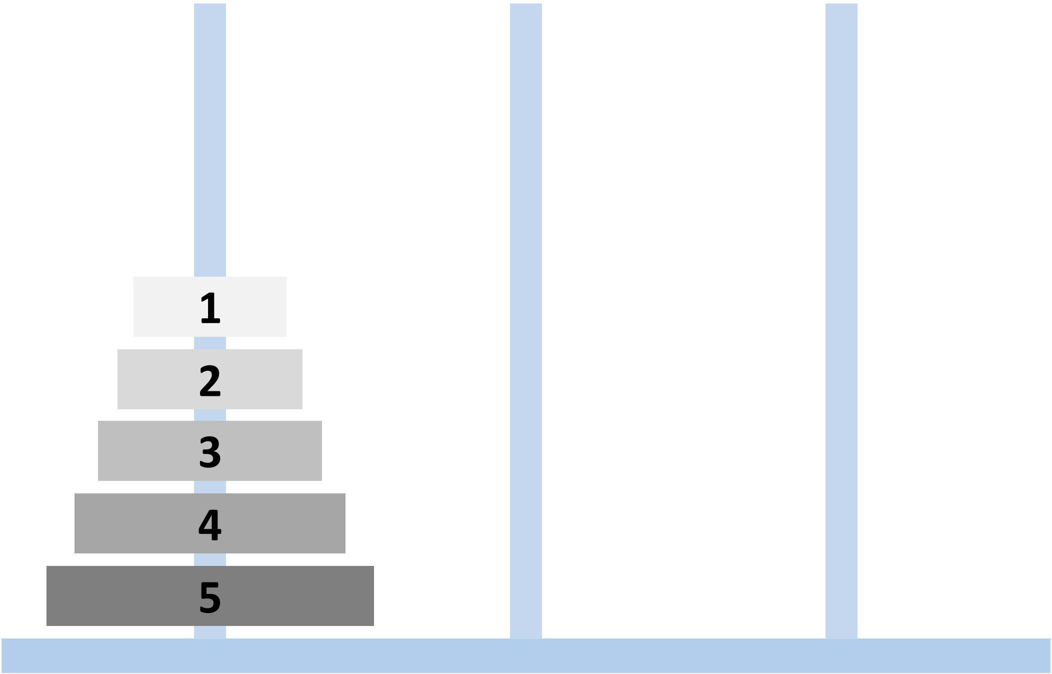
Lab 4-2. 하노이의 탑 2

- 수학적 귀납법의 아이디어를 이용하여 재귀적으로 문제 해결하기
  - **조건 1** : N=1 일 때, 문제는 자명하게 해결된다.
  - **조건 2**
    - N=k일 때 문제를 해결하는 방법을 알고 있다고 가정
    - 해당 가정 밑에서 N=k+1 일 때도 문제를 해결할 수 있음을 증명

  - **조건 1과 조건 2를 만족하면, 모든 자연수 N에 대해 문제를 해결할 수 있다!**

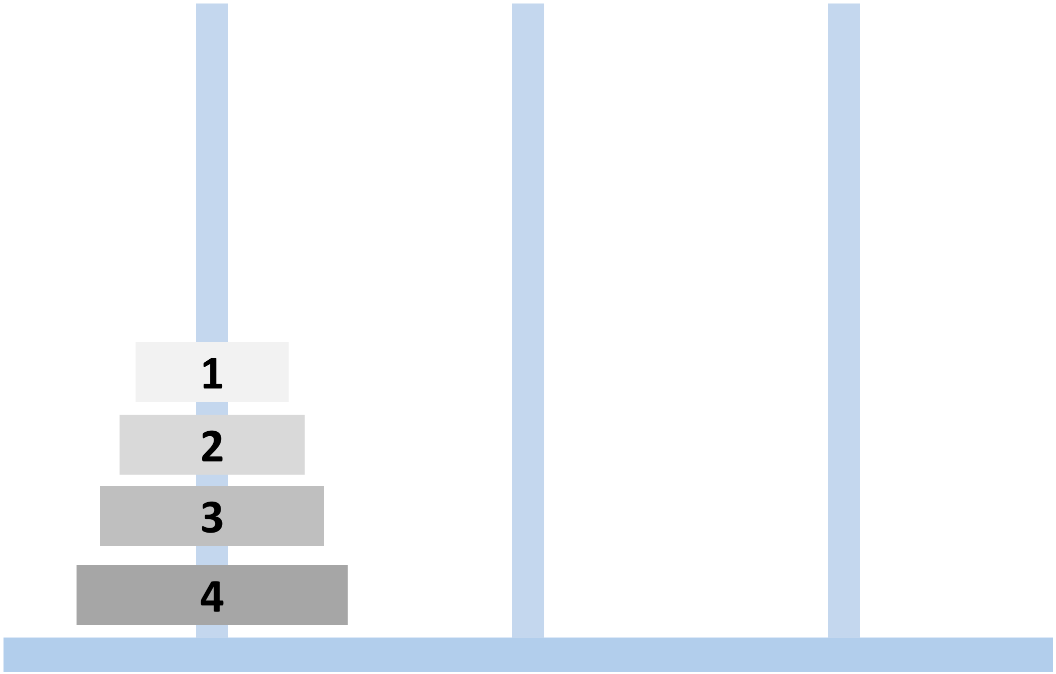
Lab 4-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - N = 4 하노이 탑 문제를 해결할 수 있다고 가정



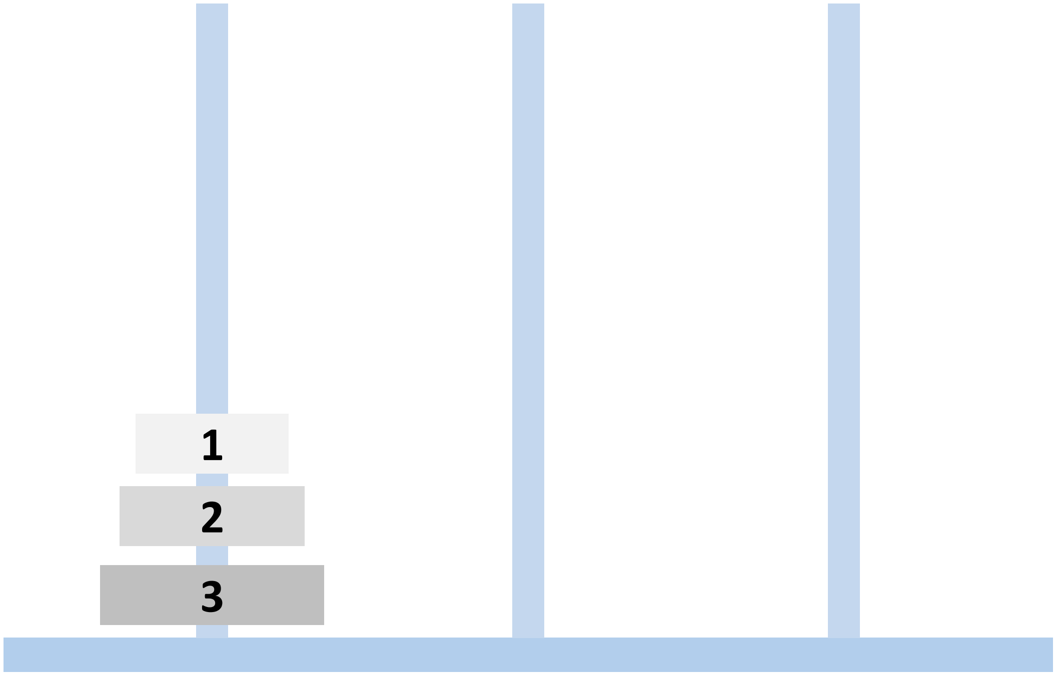
Lab 4-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - 그럼, N = 4 하노이 탑 문제는 어떻게 풀 건데?
    - N = 3일 때 문제를 해결할 수 있다고 가정~



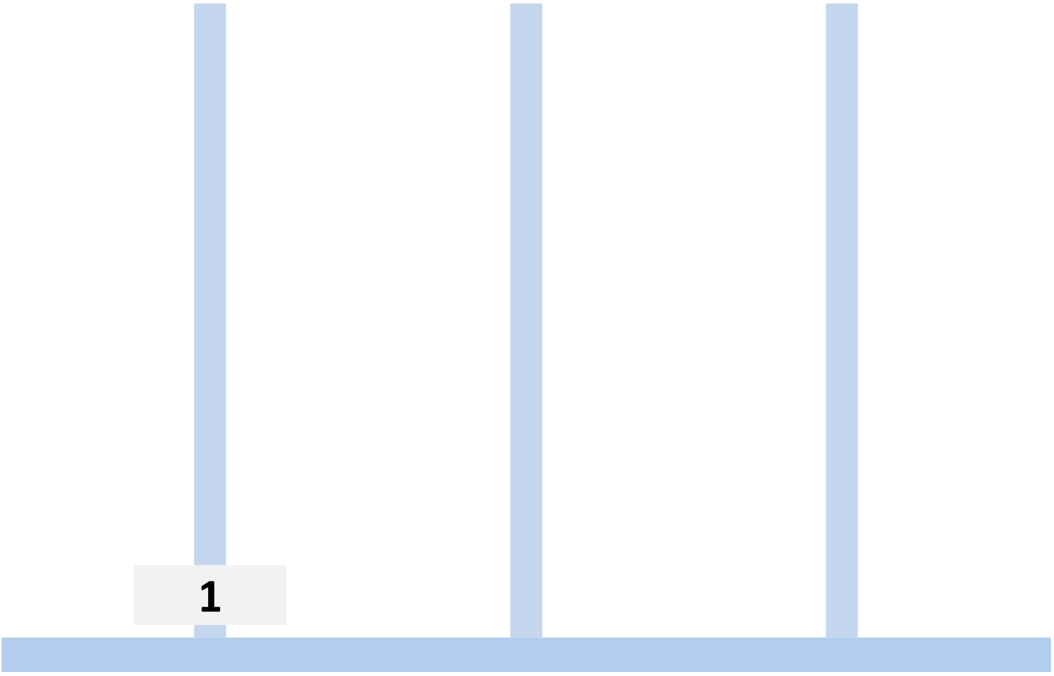
Lab 4-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - … 그럼, N = 3은?
    - N = 2일 때!



Lab 4-2. 하노이의 탑 2

- 예시 : N = 5 하노이 탑 문제
  - 이 논리는 N=1 일 때는 적용할 수 없다 (N > 0)
  - 하지만 자명하게 해결할 수 있다



Lab 4-2. 하노이의 탑 2

- `def play_hanoi(n):`
  - *n*: 원판의 개수
  - Expected outputs
    - 세 개의 기둥을 각각 0, 1, 2로 표현할 때,
    - 0에서 1로 n개의 원판 전체를 옮기는 과정을 기술한다.

    - 각각의 움직임(move)은 길이 2짜리 리스트 `[pole_from, pole_to]` 로 기술
    - 움직임들의 리스트로 정답 기술

```
assert play_hanoi(1) == [[0, 1]]
assert play_hanoi(2) == [[0, 2], [0, 1], [2, 1]]
```

Lab 4-2. 하노이의 탑 2

- 재귀적 접근 방법
  - (Base case) `n == 1`, return What?
  - (Induction) `play_hanoi(n-1)`로 `play_hanoi(n)` 만들기
    - **#1: (n-1)개 원판을 0에서 2로 옮긴다**
    - #2: 가장 큰 원판을 0에서 1로 옮긴다
    - **#3: #1에서 옮겼던 (n-1)개 원판을 2에서 1로 옮긴다**
    - return [#1] + [#2] + [#3]

  - [#1], [#3]은 `play_hanoi(n-1)`만으로는 해결할 수 없어 보인다
    - 판의 초기 위치와 목표 위치 정보가 추가로 주어져야 한다

Lab 4-2. 하노이의 탑 2

- *re*-`def play_hanoi(n, p_from=0, p_to=1):`
  - (Base case) `n == 1`, return **What?**
  - (Induction) `play_hanoi(n-1)`로 `play_hanoi(n)` 만들기
    - #1: (n-1)개 원판을 ***p_from***에서 ***[나머지 기둥]***로 옮긴다
    - #2: 가장 큰 원판을 ***p_from***에서 ***p_to***로 옮긴다
    - #3: #1에서 옮겼던 (n-1)개 원판을 ***[나머지 기둥]***에서 ***p_to***로 옮긴다
    - return [#1] + [#2] + [#3]

  - ***[나머지 기둥]***은 어떻게 계산하는가?

import

Lab 4-2. 하노이의 탑 2

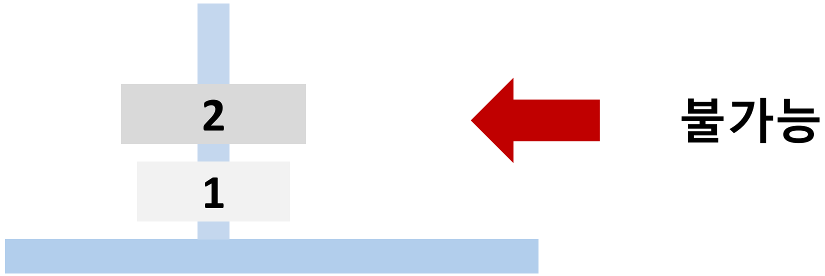
- import로 day2 하노이 탑 자동으로 풀기
  - import : 다른 파일의 객체를 사용할 수 있게 하는 명령

    from hanoi_play import play_hanoi

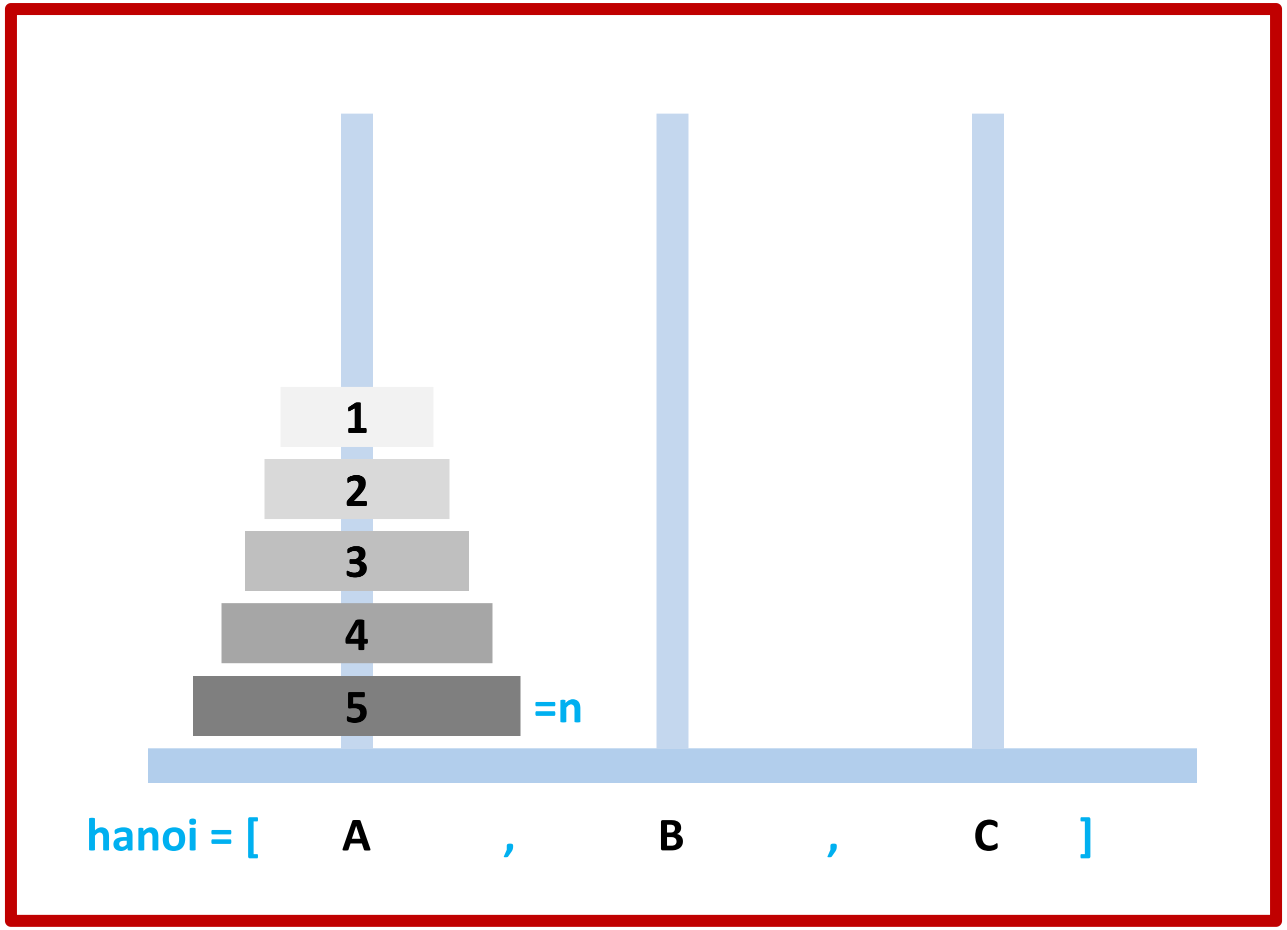
  - 이동 과정을 play_hanoi 함수로 계산해서 day2 에서 만든 hanoi 게임에 넣기
    - 자동으로 게임을 해결해주는 프로그램 완성!

Lab 2-4. Tower of Hanoi

- N 개의 크기가 다른 원판
- 기둥 3개
  - 한 번에 하나의 원판만 이동
  - 각 원판은 자기보다 큰 원판 위에만 위치할 수 있음



- 완료 조건
  - 모든 원판을 다른 기둥으로 옮기면 끝!



Lab 2-4. Tower of Hanoi

- 입출력 예시
  - 원판의 개수를 입력 받음
  - 매 시행 마다 from 기둥 (첫 번째), to 기둥 (두 번째) 입력 받음
  - 1 기둥으로 모두 옮기면 종료

```
$ python hanoi.py
원판의 수를 입력하세요. : 3↲
0 [3, 2, 1]
1 []
2 []

첫 번째 기둥 (0, 1, 2) : 0↲
두 번째 기둥 (0, 1, 2) :
```

Lab 4-2. 하노이의 탑 2

- 입출력 예시
  - 원판의 개수를 입력 받음
  - 사용자가 input 을 넣는 대신 hanoi_play 함수의 호출 결과를 넣어 자동 해결

```
$ python hanoi.py
원판의 수를 입력하세요. : 3↲
0 [3, 2, 1]
1 []
2 []
from 2 to 1
0 [3, 2]
... (후략)
```