課題1: 再帰のプログラムを作ってみよう

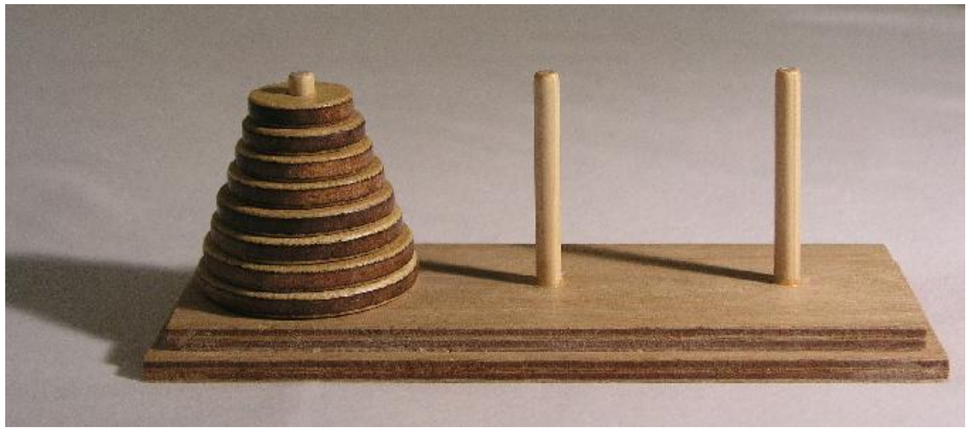
1. 1+1+ … + 1（n回足す）を再帰で計算する関数 one(n) を作ってみよう
   - ヒント：one(n) = one(n-1) + 1

2. x*x* … *x（n回掛ける）を再帰で計算する関数 power(x, n) を作ってみよう
   - ヒント：power(x, n) = power(x, n-1) * x

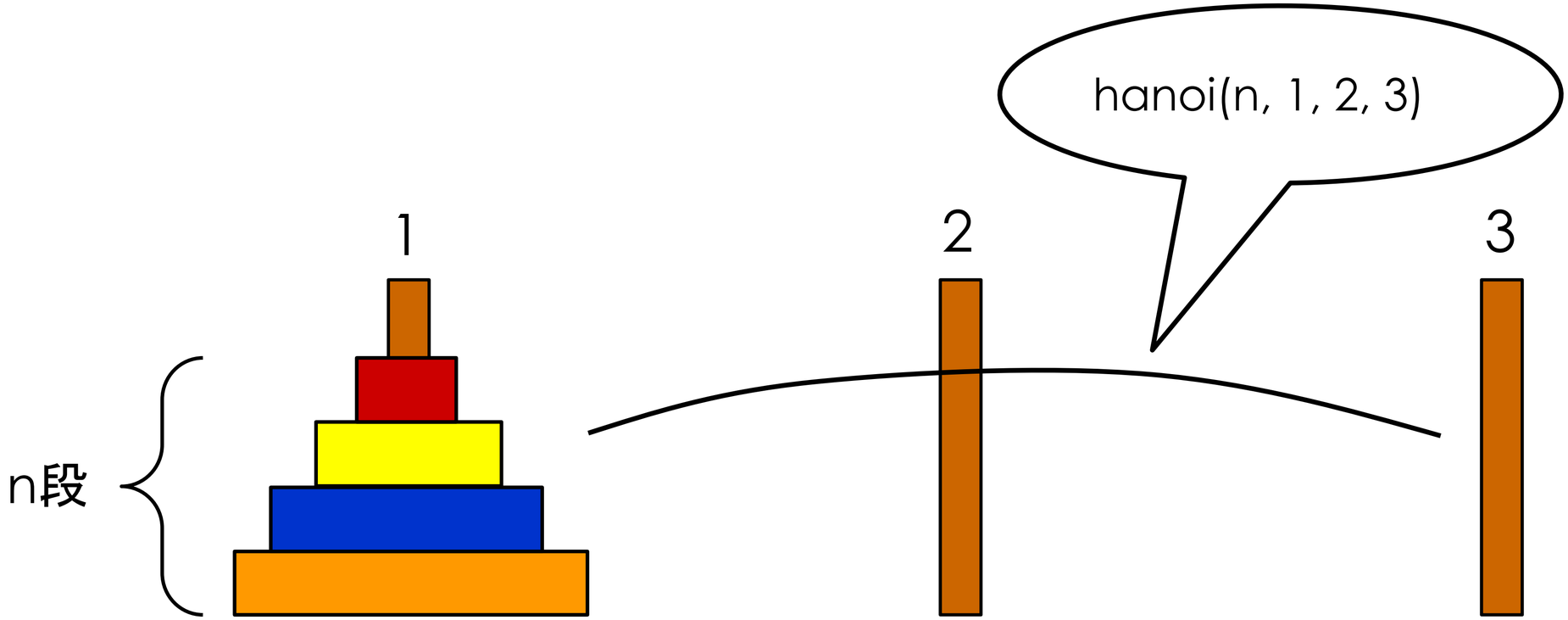
3. 二項係数 ${}_nC_k$ を再帰で計算する関数 comb(n, k) を作ってみよう
   - ヒント：comb(n, k) = comb(n-1, k) + comb(n-1, k-1)

（発展）課題1-4: ハノイの塔

- 以下のルールに従ってすべての円盤を右端の杭に移動させられれば完成。
  - 3本の杭と、中央に穴の開いた大きさの異なる複数の円盤から構成される。
  - 最初はすべての円盤が左端の杭に小さいものが上になるように順に積み重ねられている。
  - 円盤を一回に一枚ずつどれかの杭に移動させることができるが、小さな円盤の上に大きな円盤を乗せることはできない

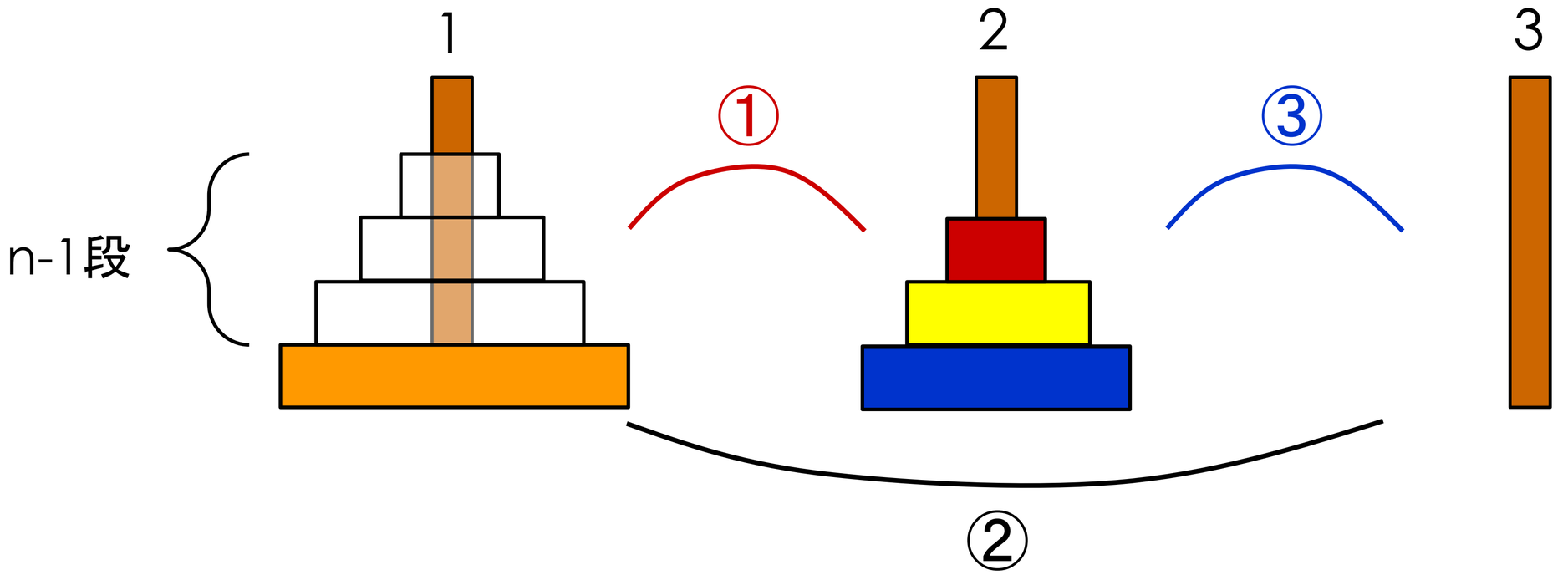


ハノイの塔も再帰で計算できる



n段のハノイの塔を杭 a から杭 c に移す手順を表示する
関数 hanoi(n, a, b, c)(a, b, c は変数)を作ってみよう。

問題を分割する



① n-1段のハノイの塔を杭 a から杭 b に移す
② 1段のハノイの塔を杭 a から杭 c に移す
③ n-1段のハノイの塔を杭 b から杭 c に移す

← ①②③ができれば
hanoi(n, a, b, c) ができる

関数で表現する

① n-1段のハノイの塔を杭 a から杭 b に移す
② 1段のハノイの塔を杭 a から杭 c に移す
③ n-1段のハノイの塔を杭 b から杭 c に移す

```
# n段のハノイの塔を a から c へ移動する
# 手順を表示する関数(未完成)
def hanoi(n, a, b, c):

    hanoi([A], [B], [C], [D]) # ①の部分
    hanoi([E], [F], [G], [H]) # ②の部分
    hanoi([I], [J], [K], [L]) # ③の部分
```

例外ケースを処理する

① n-1段のハノイの塔を杭 a から杭 b に移す
② 1段のハノイの塔を杭 a から杭 c に移す
③ n-1段のハノイの塔を杭 b から杭 c に移す

```
# n段のハノイの塔を a から c へ移動する
# 手順を表示する関数
def hanoi(n, a, b, c):
    if n == 1:
        print('move’, [M], 'to’, [N])
    else:
        hanoi([A], [B], [C], [D]) # ①の部分
        hanoi([E], [F], [G], [H]) # ②の部分
        hanoi([I], [J], [K], [L]) # ③の部分
```

n == 1 のときは
実際の移動手順を表示する

(発展)課題1-4: ハノイの塔

- 下のプログラムの引数 [A]~[N] を埋めてプログラムを完成させよう

```
# n段のハノイの塔を a から c へ移動する
# 手順を表示する関数
def hanoi(n, a, b, c):
    if n == 1:
        print('move’, [M], 'to’, [N])
    else:
        hanoi([A], [B], [C], [D])
        hanoi([E], [F], [G], [H])
        hanoi([I], [J], [K], [L])
```

実行例

```
hanoi(4, 1, 2, 3)
```

```
move 1 to 2
move 1 to 3
move 2 to 3
move 1 to 2
move 3 to 1
move 3 to 2

   （中略）

move 1 to 3
move 2 to 3
```

ここにハノイの塔を動かす手順が表示される