以遞迴程式設計實作

Factorial, Fabonacci number, Determinant and Hanoi Tower

Factorial

Fibonacci number

Matrix determinant

Hanoi tower

Problem size

以自然數代表問題大小

6!的問題大小為6

4!的問題大小為4

遞迴定義

問題大小n為初始問題，直接代入答案

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

$$17! = 17 * 16!$$

遞迴定義

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

問題大小n不是初始問題，將問題表示為前一階的問題，
並且使用前一階的答案，建立問題大小n的答案表示式

問題大小n的答案表示式

$$n! = n * \boxed{(n-1)!}$$

問題大小n不是初始問題，在等號右邊使用前一階的答案，合成問題大小n的答案表示式

Fibonacci numbers

Primitive problem 問題大小n為初始問題，直接代入答案

$$F_0 = 0, \quad F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \quad if \quad n > 1$$

$$F_{17} = F_{16} + F_{15}$$

Fibonacci numbers

$$F_0 = 0, \quad F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \quad if \quad n > 1$$

問題大小n不是初始問題，將問題表示為前兩階的問題．
並且使用前一階的答案，建立問題大小n的答案表示式

問題大小n的答案表示式

$$F_n = F_{n-1} + F_{n-2}, \quad if \quad n > 1$$

問題大小n不是初始問題，在等號右邊使用前兩階的答案，合成問題大小n的答案表示式

Determinant

n=2

問題大小n為初始問題，直接代入答案

$$\begin{vmatrix} 2 & 1 \\ 1 & 1 \end{vmatrix}$$

問題大小n為初始問題，直接代入答案

```
A = np.matrix([[2,1],[1,1]])
ans = A[0,0]*A[1,1] - A[1,0]*A[0,1]
```

```
import numpy as np
from numpy.linalg import det
A = np.matrix([[2,1],[1,1]])
ans = A[0,0]*A[1,1] - A[1,0]*A[0,1]
print(ans)
print(det(A))
```

```
1
1.0
```

Determinant

n>2

$$\begin{vmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{vmatrix}$$

問題大小n不是初始問題，將問題表示為前一階的問題．
並且使用前一階的答案，建立問題大小n的答案表示式

$$\begin{vmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{vmatrix}$$

$$\begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix}$$

$$\begin{vmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{vmatrix}$$

$$\begin{vmatrix} 2 & 1 \\ 0 & 2 \end{vmatrix}$$

$$\begin{vmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{vmatrix}$$

$$\begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix}$$

$$i = 0,\ n=3$$

$$\begin{vmatrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{vmatrix}$$

A[1:n, 0:i]

A[1:n, i+1:n]

$$\begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix}$$

```
np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
```

$$i = 1, n=3$$

$$\left|\begin{matrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{matrix}\right|$$

A[1:n, 0:i]

A[1:n, i+1:n]

$$\left|\begin{matrix} 2 & 1 \\ 0 & 2 \end{matrix}\right|$$

```
np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
```

$$i = 2, n=3$$

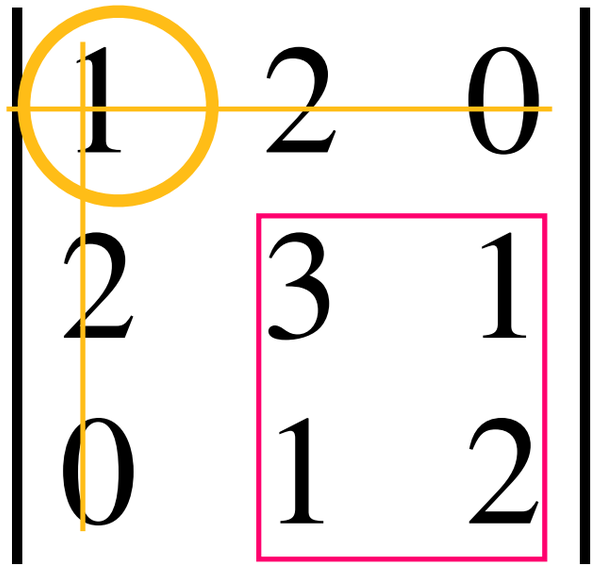
$$\left|\begin{matrix} 1 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 2 \end{matrix}\right|$$

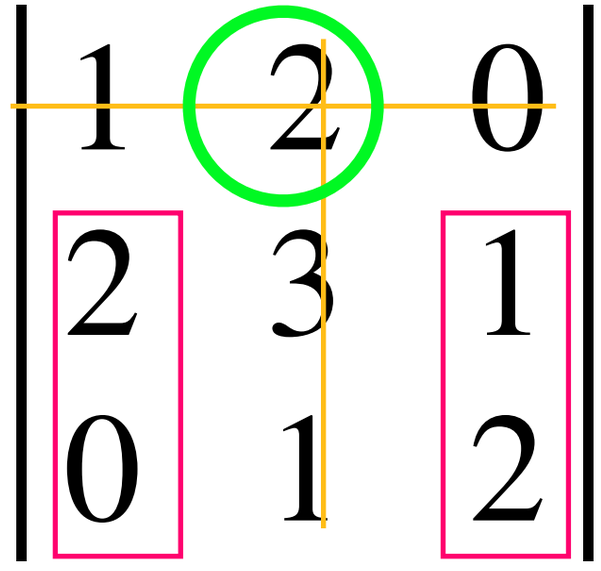
A[1:n, 0:i]

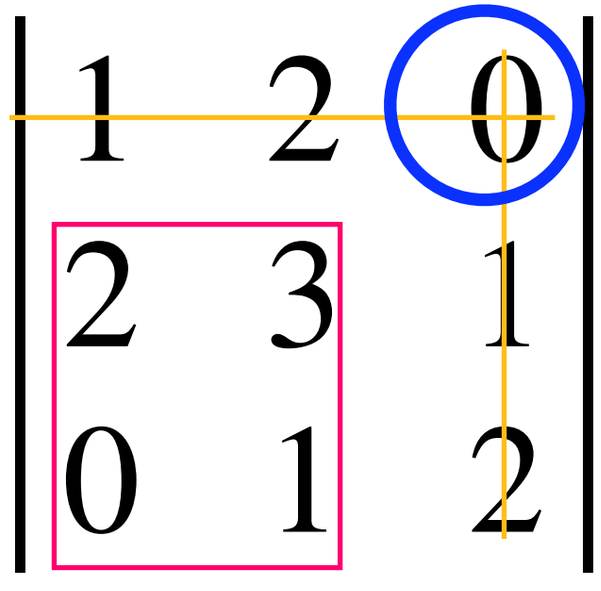
A[1:n, i+1:n]

$$\left|\begin{matrix} 2 & 3 \\ 0 & 1 \end{matrix}\right|$$

```
np.hstack((A[1:n,0:i],A[1:n,i+1:n]))
```







$$ans = 1 \times \begin{vmatrix} 3 & 1 \\ 1 & 2 \end{vmatrix} - 2 \times \begin{vmatrix} 2 & 1 \\ 0 & 2 \end{vmatrix} + 0 \times \begin{vmatrix} 2 & 3 \\ 0 & 1 \end{vmatrix}$$

問題大小n不是初始問題，在等號右邊使用前一階問題的答案，合成問題大小n的答案表示式

遞迴的程式設計

n!

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

```
def fac(n):
    if n == 0:
        return 1
    else:
        return n*fac(n-1)
```

問題大小n為初始問題，直接代入答案

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

問題大小n為初始問題，直接代入答案

```
def fac(n):
    if n == 0:
        return 1
    else:
        return n*fac(n-1)
```

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases} \qquad \forall n \in \mathbb{N}.$$

```
def fac(n):
    if n == 0:
        return 1
    else:
        return n*fac(n-1)
```

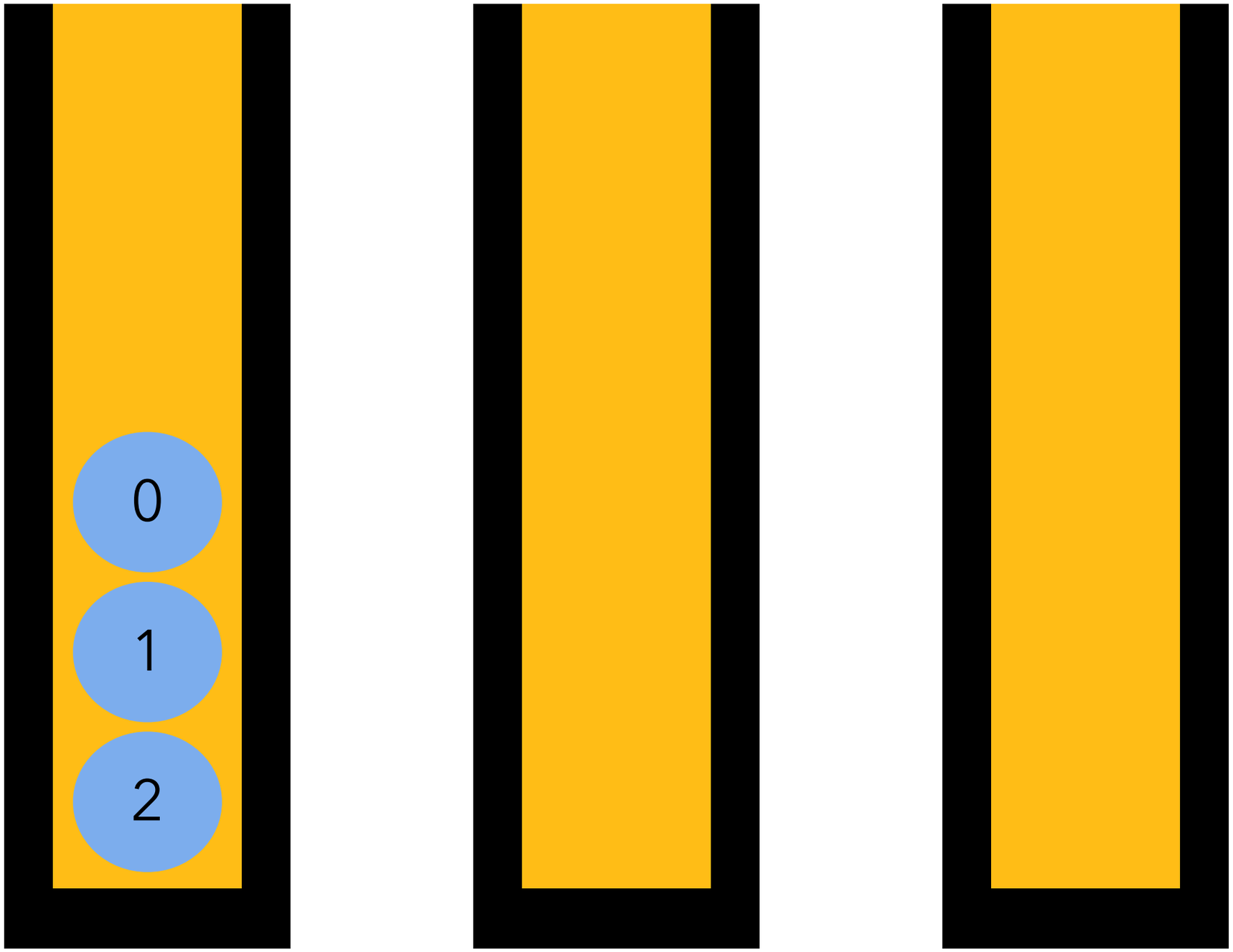
問題大小n不是初始問題，在等號右邊使用前一階的答案，合成問題大小n的答案表示式

Hanoi Tower

n = 3

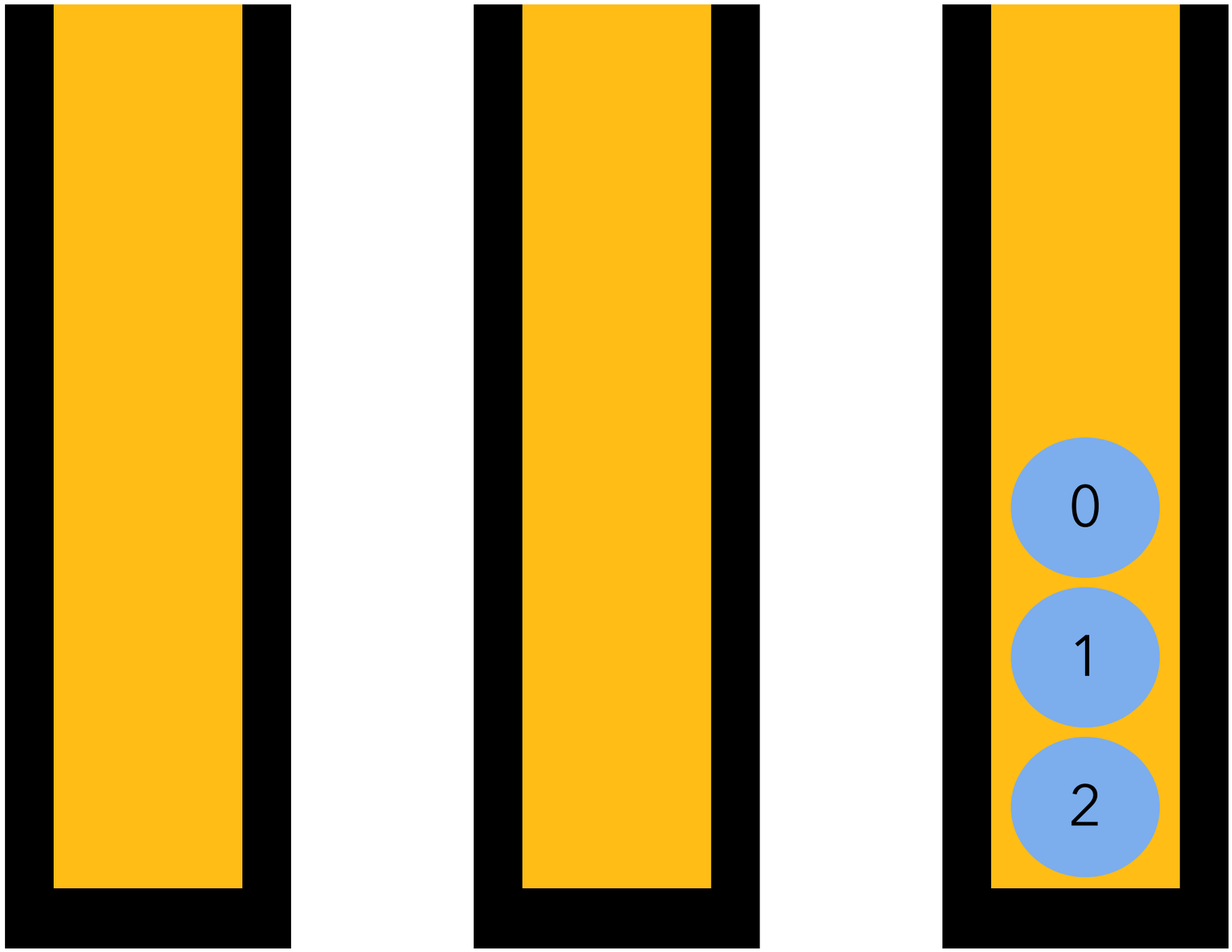
初始狀態

移動限制：
編號大的球
不能在編號
小的球上方



起始
堆疊

目標狀態



a b c

目標
堆疊

Manual design

手動設計

要設計的河內塔模組

河內塔模組的初始化方法

n=3的手動解決方案

```
from hanoi_tower import *

n = 3
a,b,c = init_hanoi_tower(n)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
print()
item = a.pop()
c.push(item)
item = a.pop()
b.push(item)
item = c.pop()
b.push(item)
item = a.pop()
c.push(item)
item = b.pop()
a.push(item)
item = b.pop()
c.push(item)
item = a.pop()
c.push(item)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
```

```
a: ['2', '1', '0']
b: []
c: []

a: []
b: []
c: ['2', '1', '0']
```

要設計的河內塔模組

```
from hanoi_tower import *

n = 3
a,b,c = init_hanoi_tower(n)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
print()
item = a.pop()
c.push(item)
item = a.pop()
b.push(item)
item = c.pop()
b.push(item)
item = a.pop()
c.push(item)
item = b.pop()
a.push(item)
item = b.pop()
c.push(item)
item = a.pop()
c.push(item)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
```

河內塔模組的初始化方法

n=2的手動解決方案

將問題size為2的河內塔從堆疊a搬移到堆疊b

將問題size為1的河內塔從堆疊a搬移到堆疊c

n=2的手動解決方案

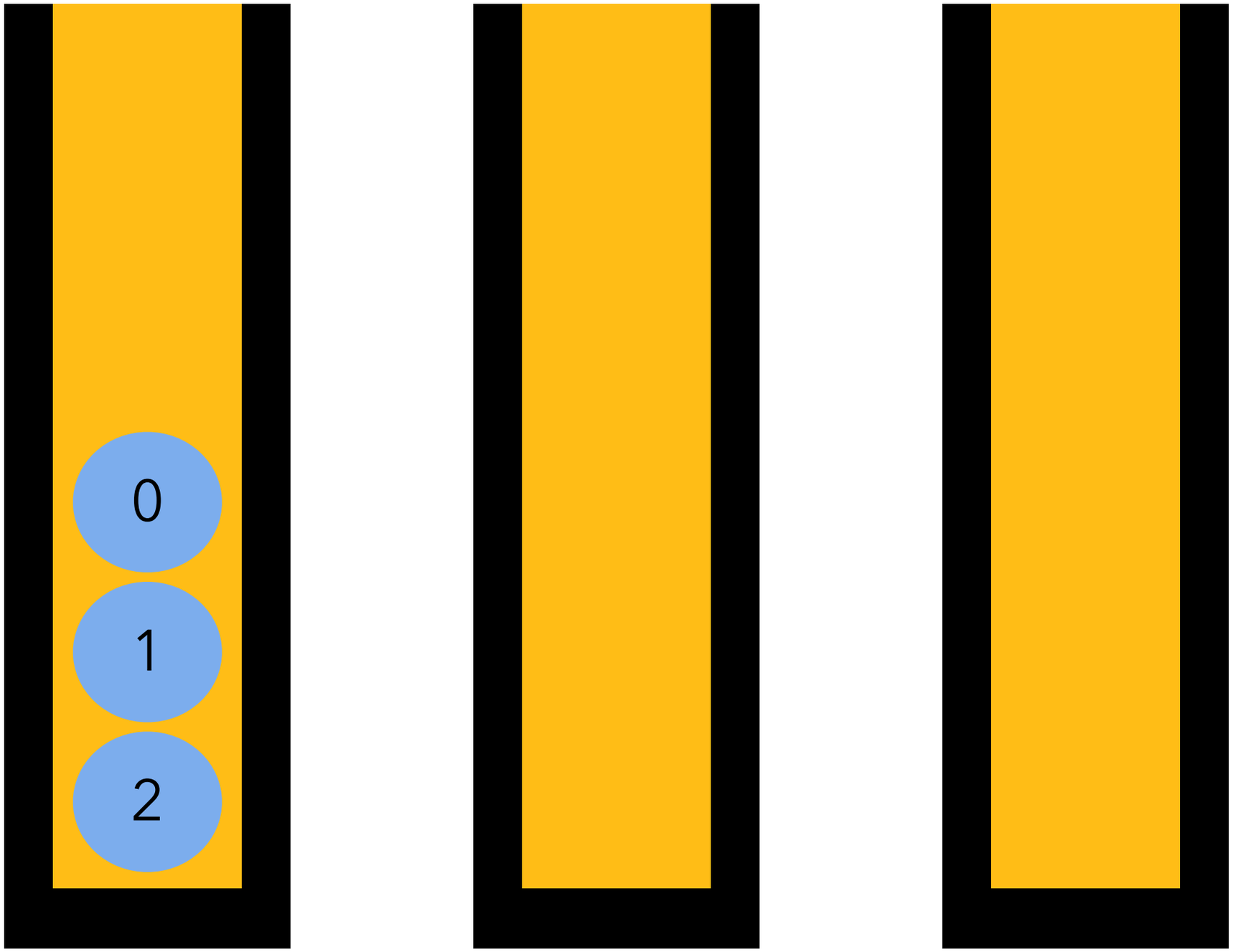
將問題size為2的河內塔從堆疊b搬移到堆疊c

move(3,a,b,c)

n = 3

初始狀態

移動限制：
編號大的球
不能在編號
小的球上方



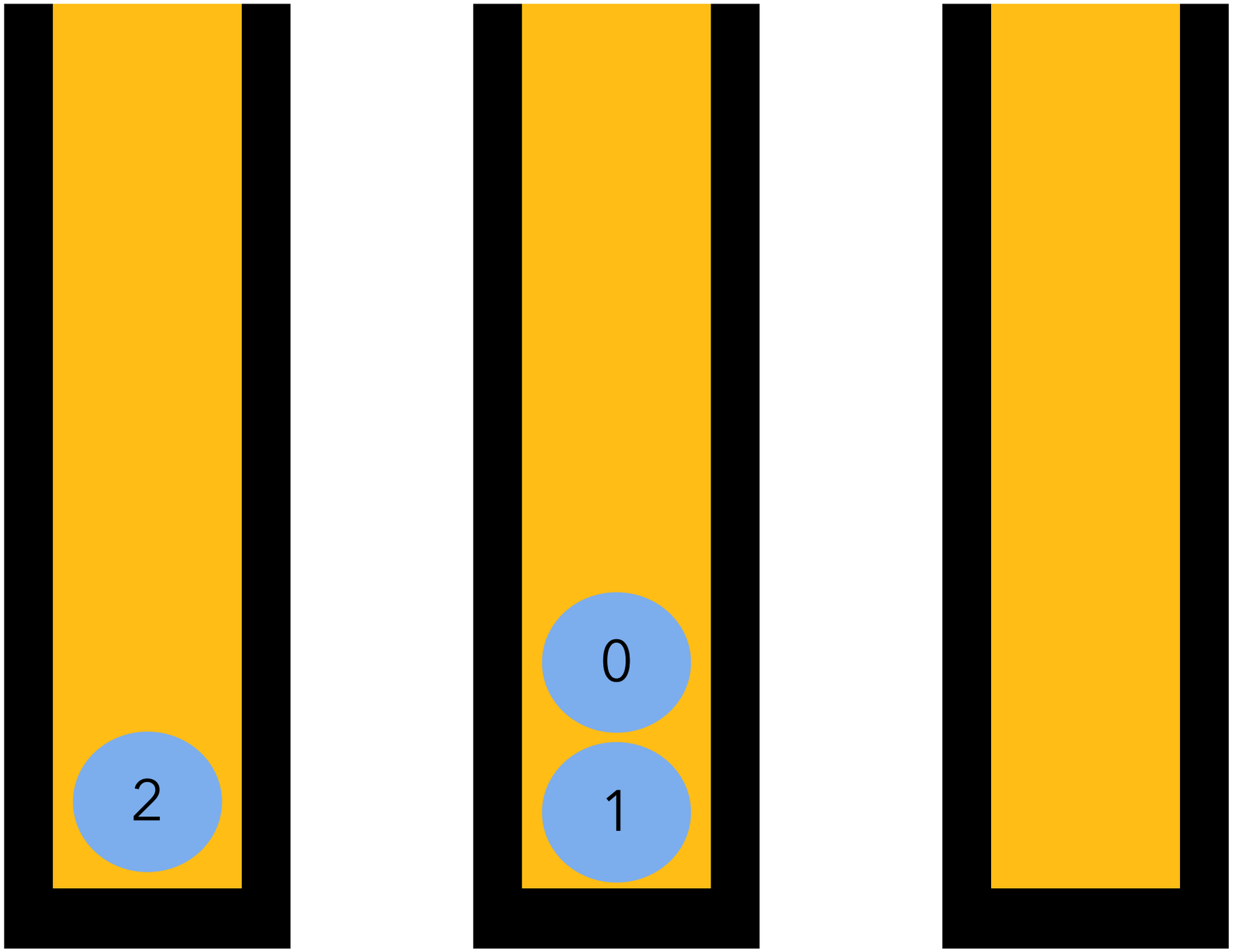
起始
堆疊

move(2,a,c,b)

n = 3

初始狀態

移動限制：
編號大的球
不能在編號
小的球上方



起始
堆疊

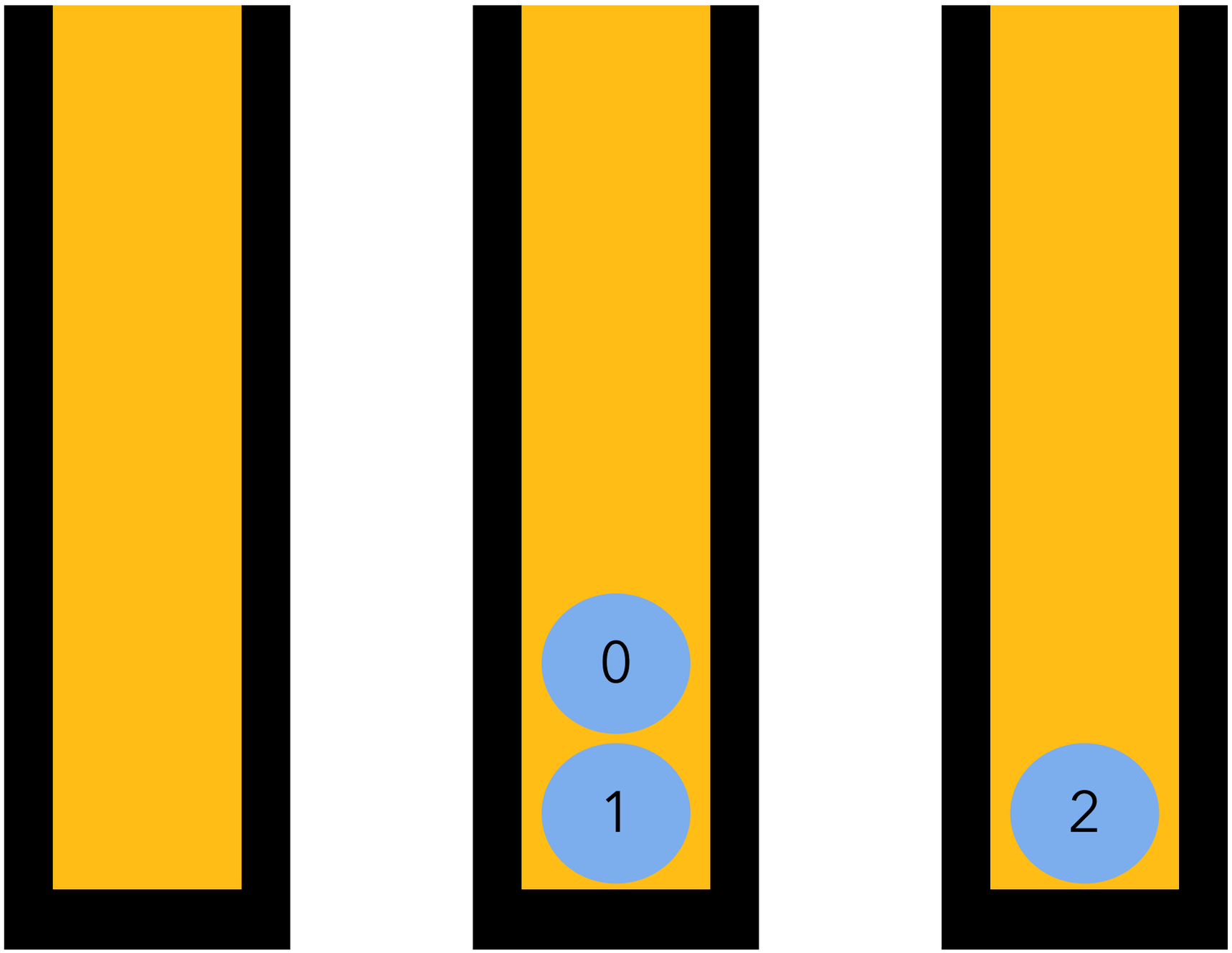
n = 3

初始狀態

移動限制：
編號大的球
不能在編號
小的球上方

起始
堆疊

item = a.pop()
c.push(item)

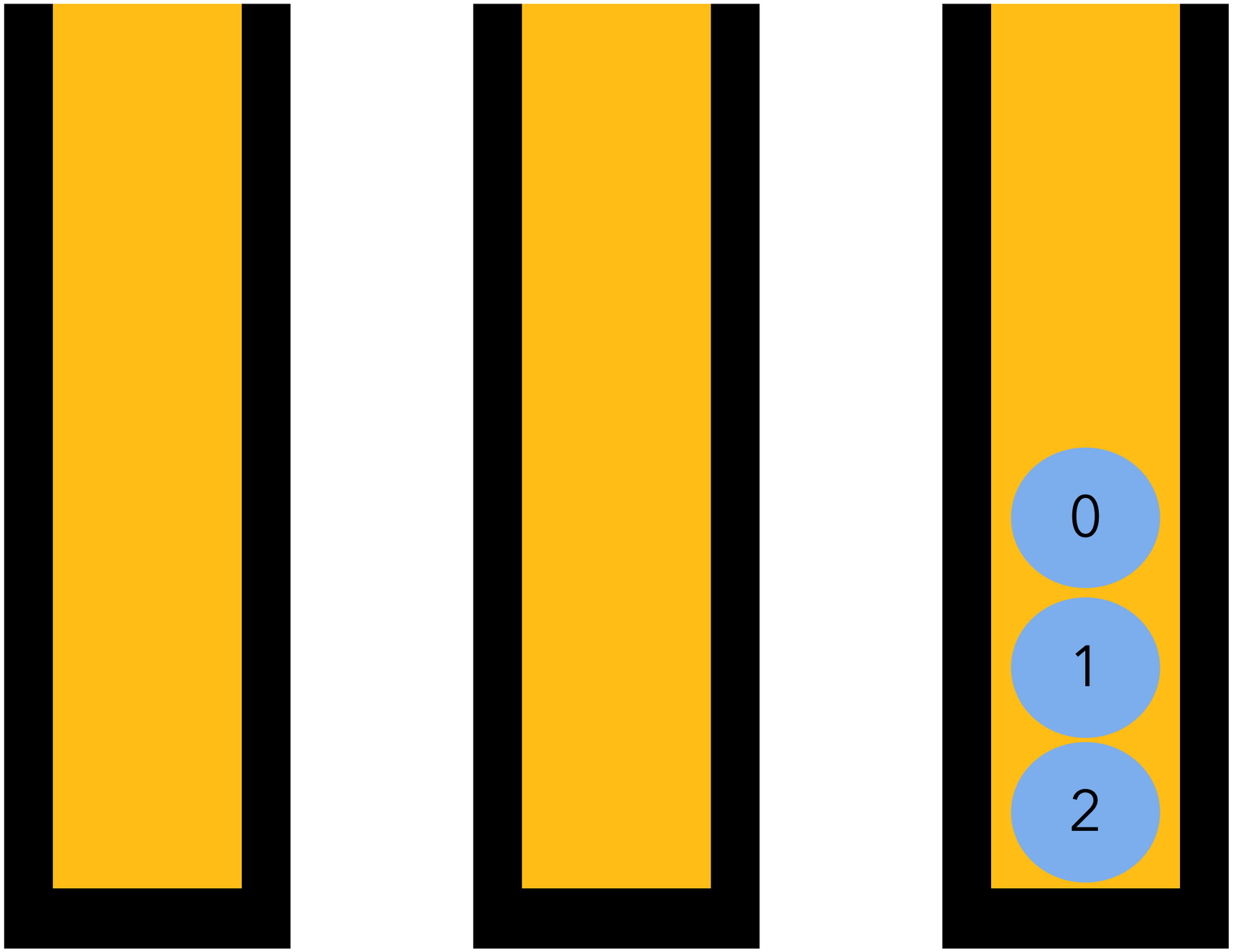


move(2,b,a,c)

n = 3

初始狀態

移動限制：
編號大的球
不能在編號
小的球上方

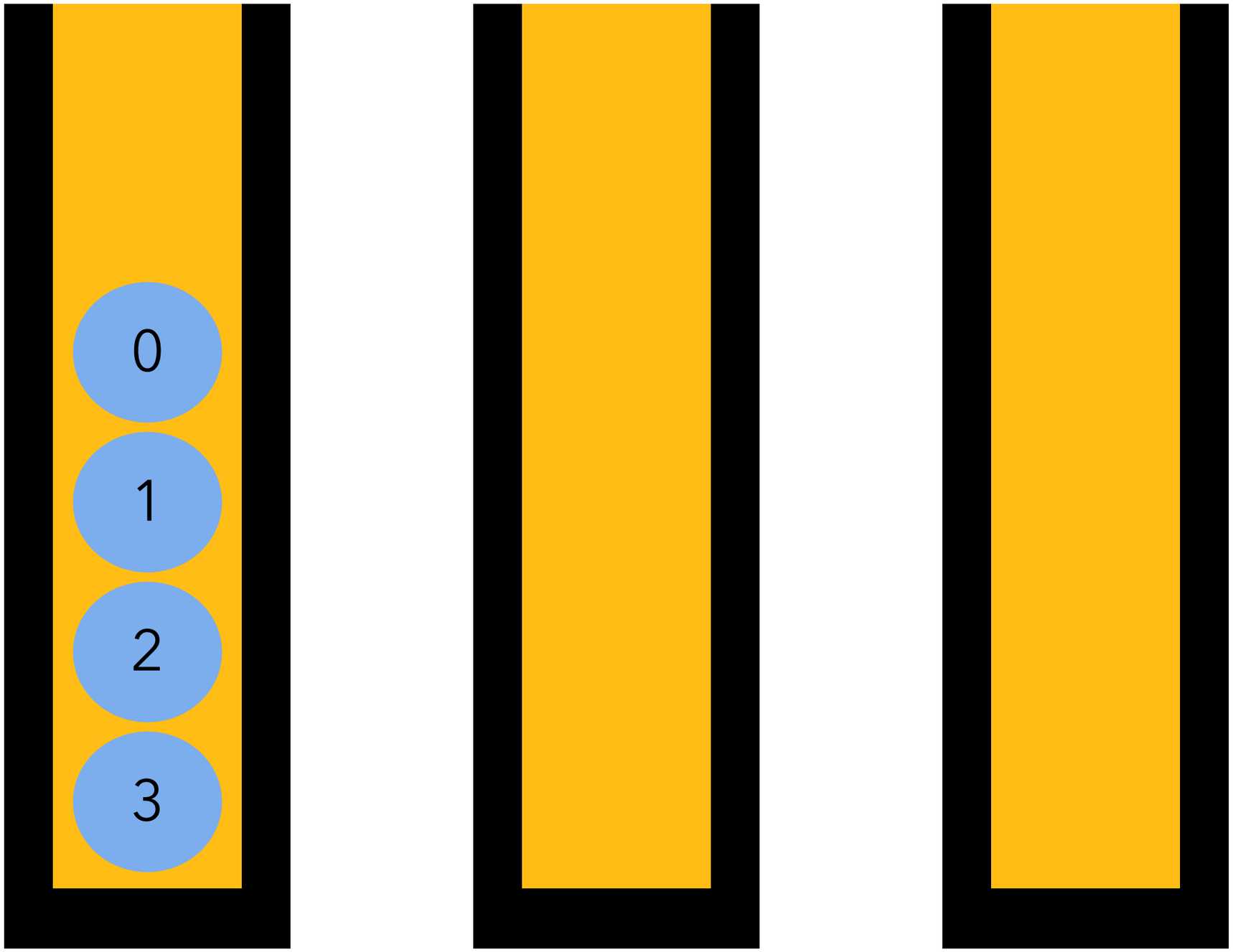


起始
堆疊

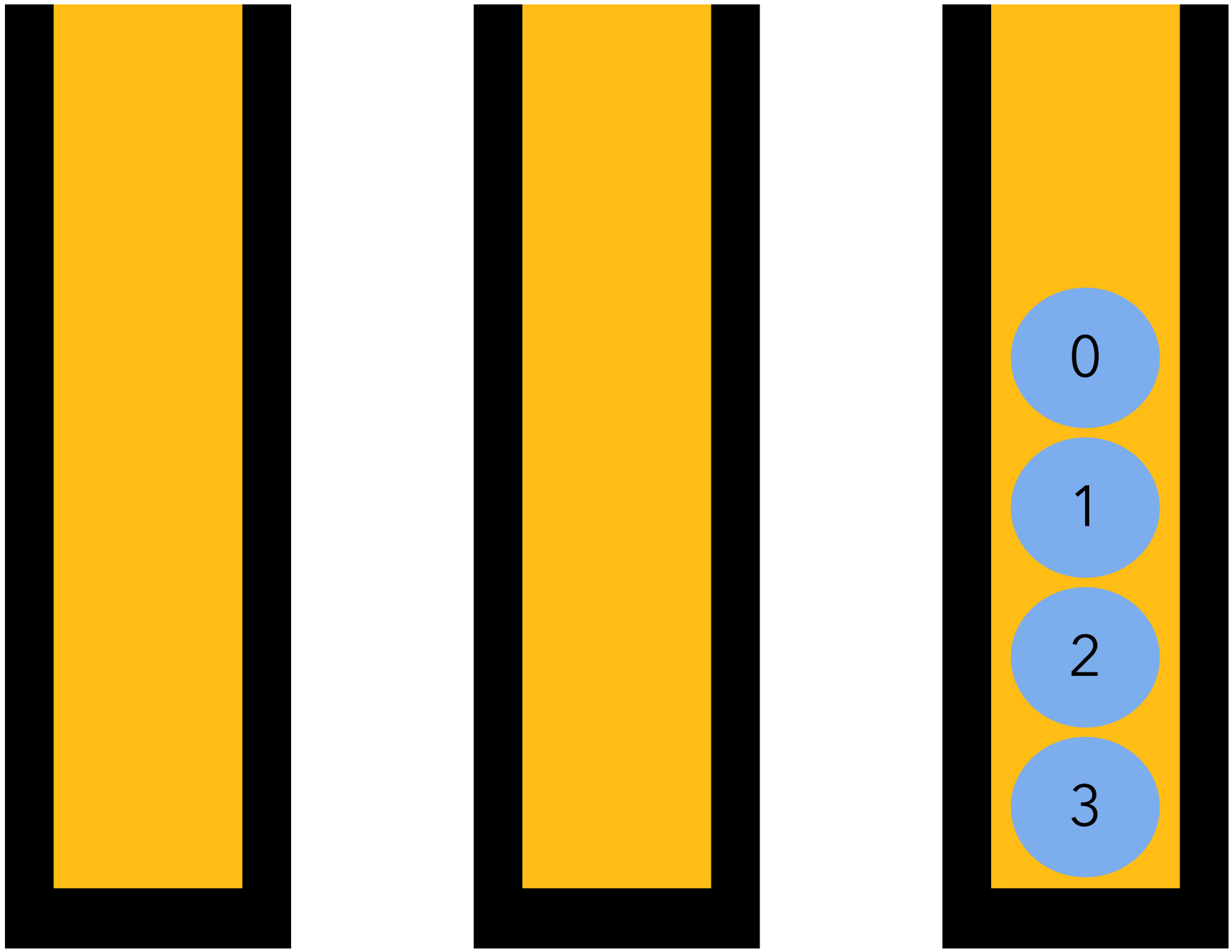
$n = 4$

初始狀態

移動限制：
編號大的球
不能在編號
小的球上方



目標狀態



```
from hanoi_tower import *

n = 4
a,b,c = init_hanoi_tower(n)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
print()
item = a.pop()
b.push(item)
item = a.pop()
c.push(item)
item = b.pop()
c.push(item)
item = a.pop()
b.push(item)
item = c.pop()
a.push(item)
item = c.pop()
b.push(item)
item = a.pop()
b.push(item)
item = a.pop()
c.push(item)
```

solve_hanoi_tower(3,b,a,c)
將問題size為3的河內塔從堆疊b搬移到堆疊c

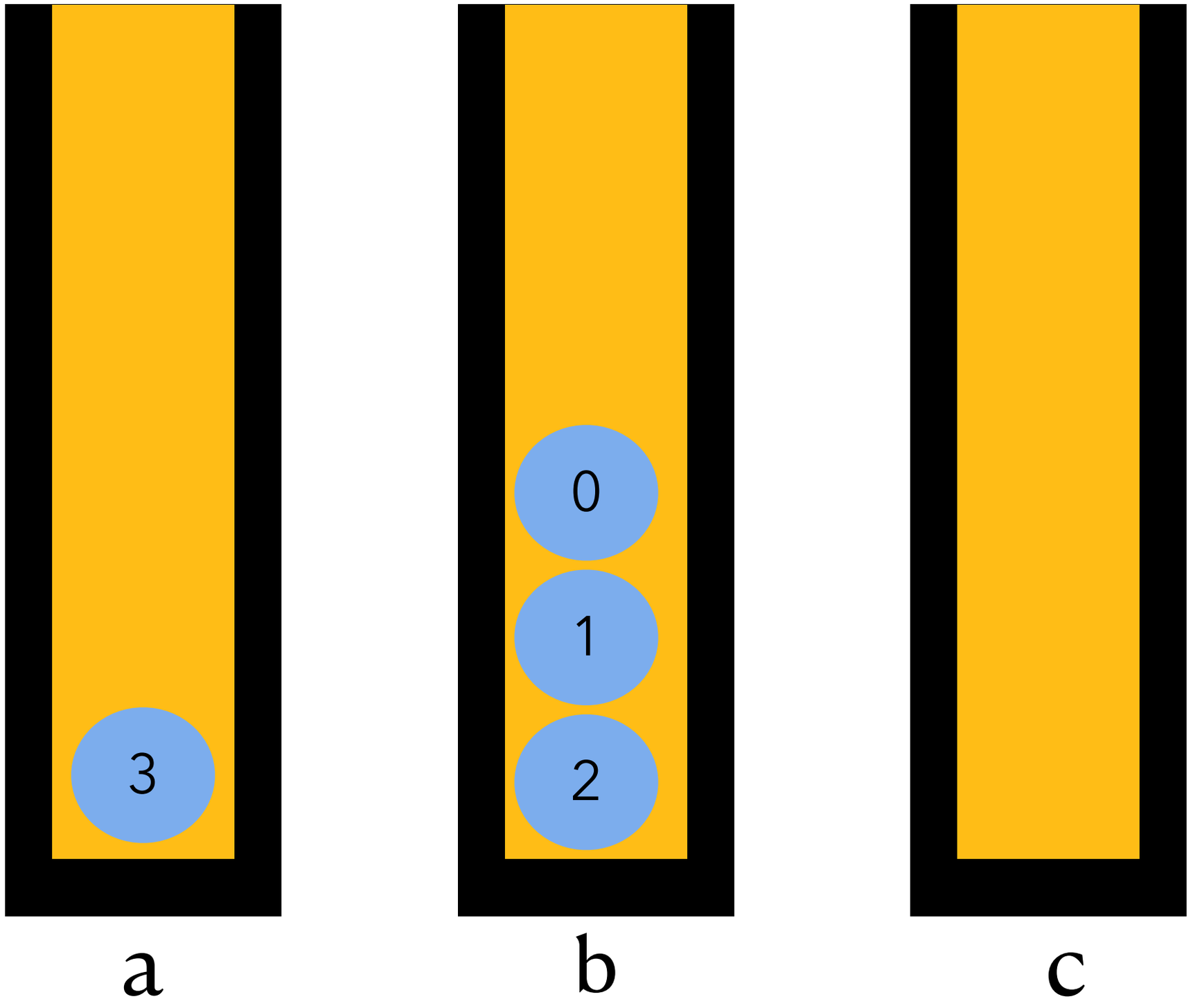
solve_hanoi_tower(3,a,c,b)
將問題size為3的河內塔從堆疊a搬移到堆疊b

```
item = b.pop()
c.push(item)
item = b.pop()
a.push(item)
item = c.pop()
a.push(item)
item = b.pop()
c.push(item)
item = a.pop()
b.push(item)
item = a.pop()
c.push(item)
item = b.pop()
c.push(item)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
```

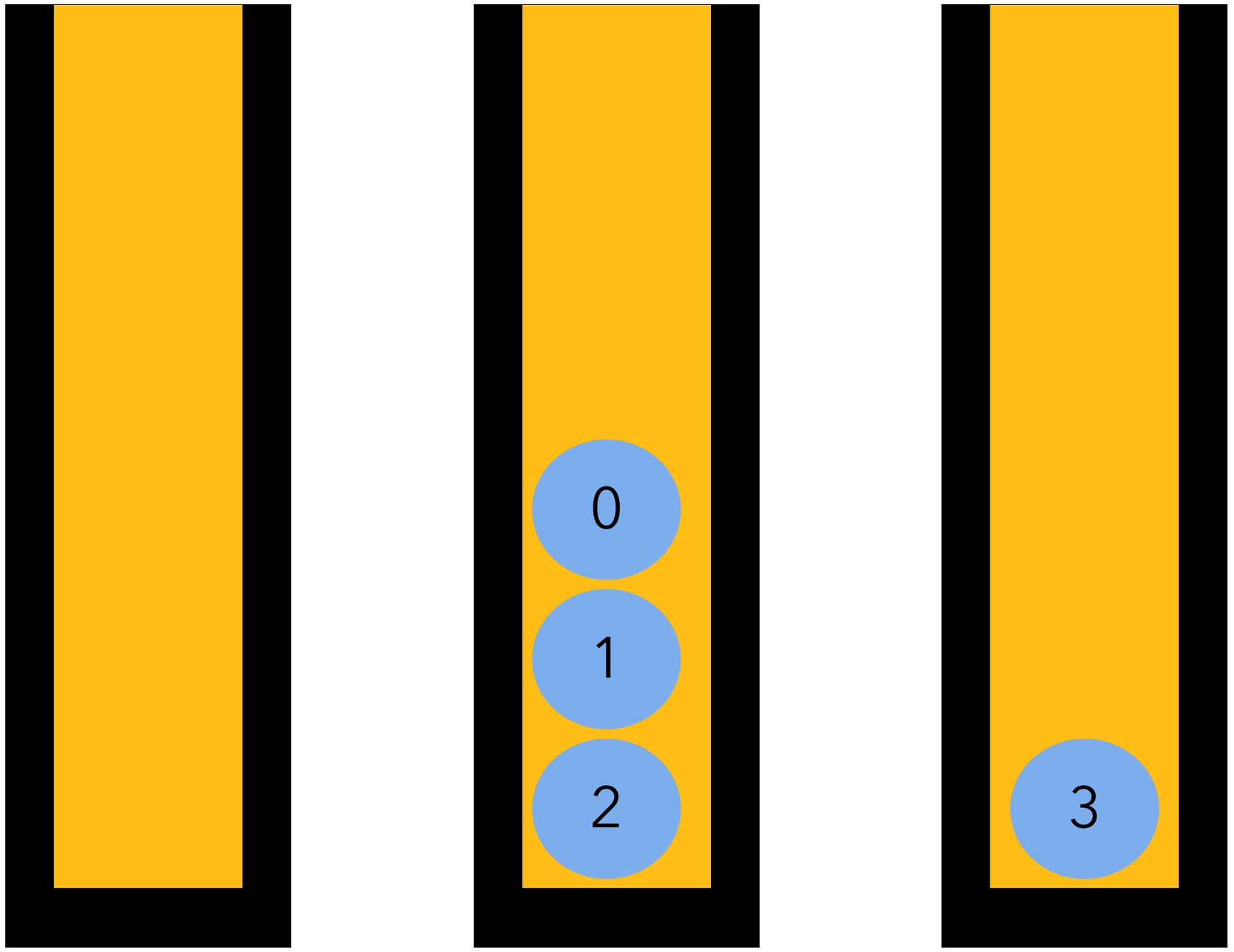
```
a: ['3', '2', '1', '0']
b: []
c: []

a: []
b: []
c: ['3', '2', '1', '0']
```

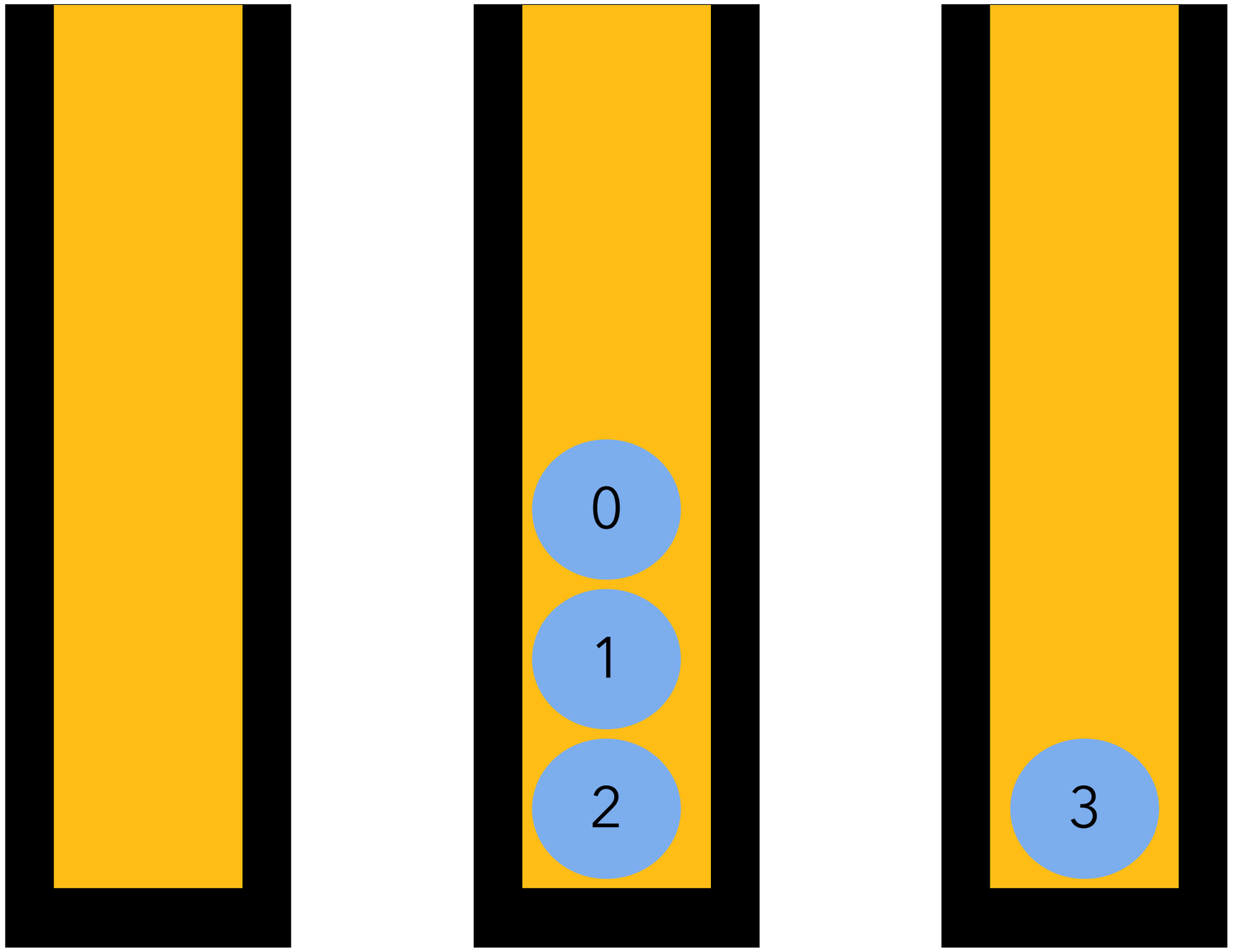
solve_hanoi_tower(3,a,c,b)
將問題size為3的河內塔從堆疊a搬移到堆疊b

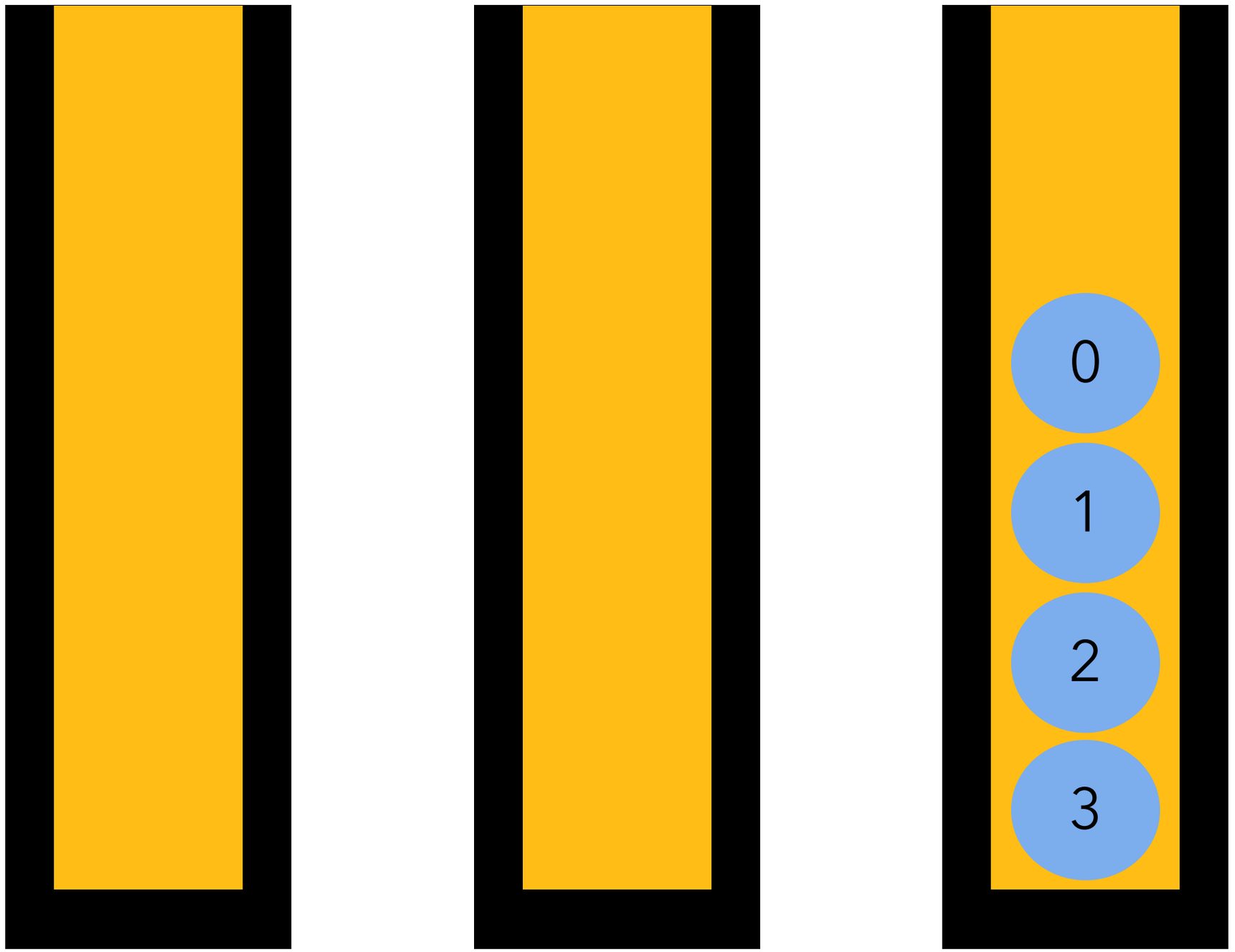


solve_hanoi_tower(1,a,b,c)
將問題size為1的河內塔從堆疊a搬移到堆疊c



solve_hanoi_tower(3,b,a,c)
將問題size為3的河內塔從堆疊b搬移到堆疊c

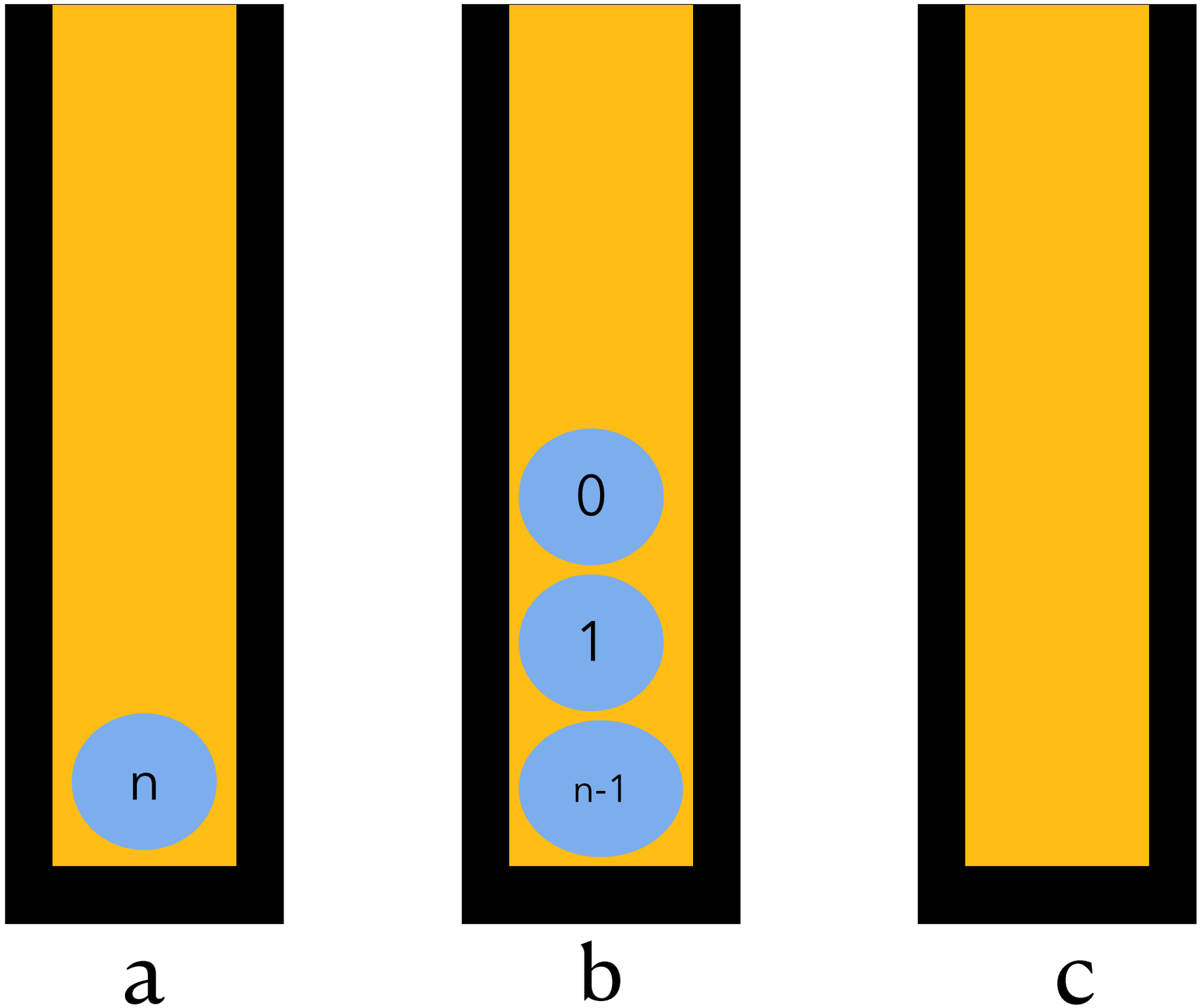



0
1
2
3
a
b
c

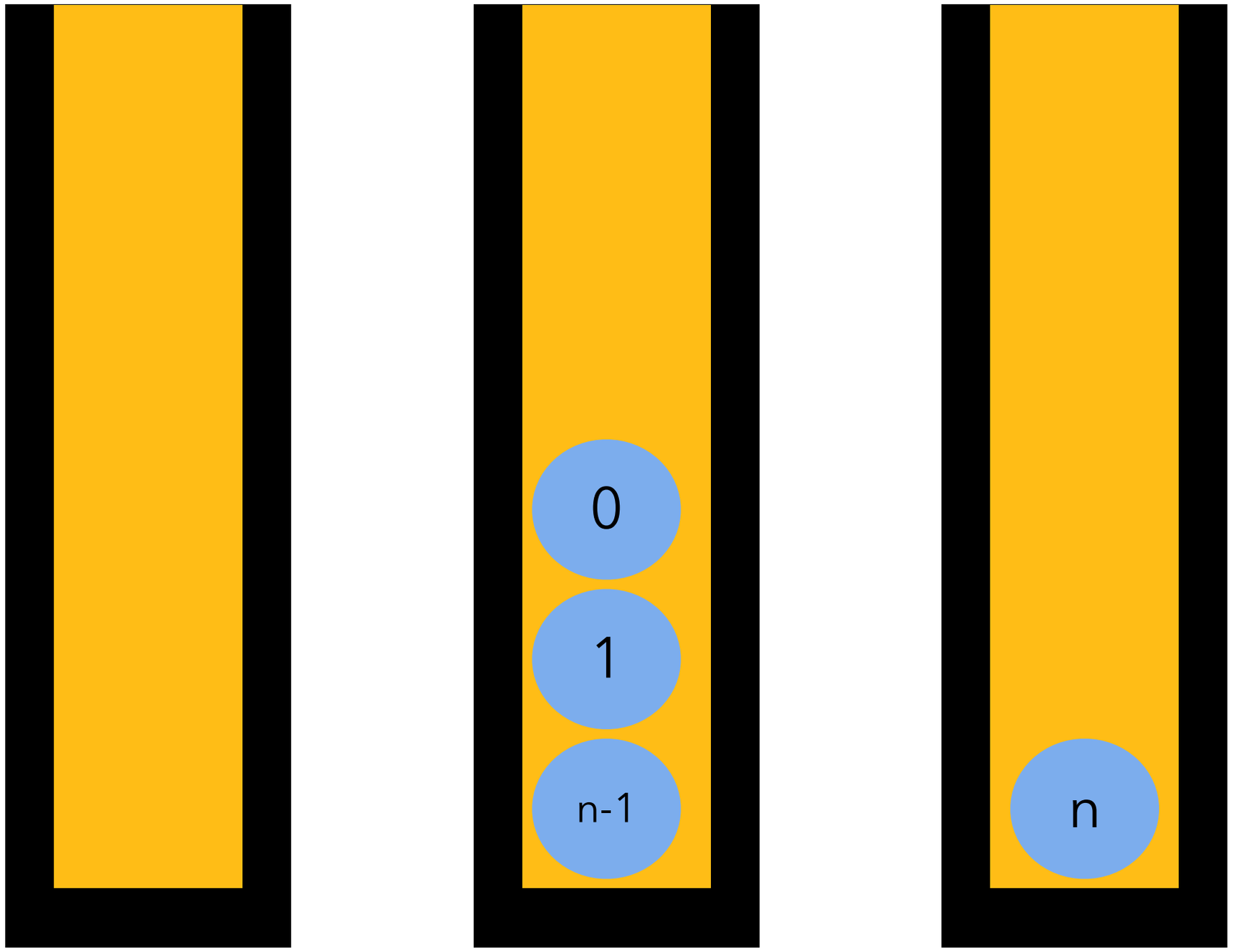
```
def move3(a,b,c):
    item = a.pop()
    c.push(item)
    item = a.pop()
    b.push(item)
    item = c.pop()
    b.push(item)
    item = a.pop()
    c.push(item)
    item = b.pop()
    a.push(item)
    item = b.pop()
    c.push(item)
    item = a.pop()
    c.push(item)
    return a,b,c
```

```
n = 4
a,b,c = init_hanoi_tower(n)
print(a.items,b.items,c.items)
a,c,b = move3(a,c,b)
print(a.items,b.items,c.items)
item = a.pop()
c.push(item)
b,a,c = move3(b,a,c)
print(a.items,b.items,c.items)
```

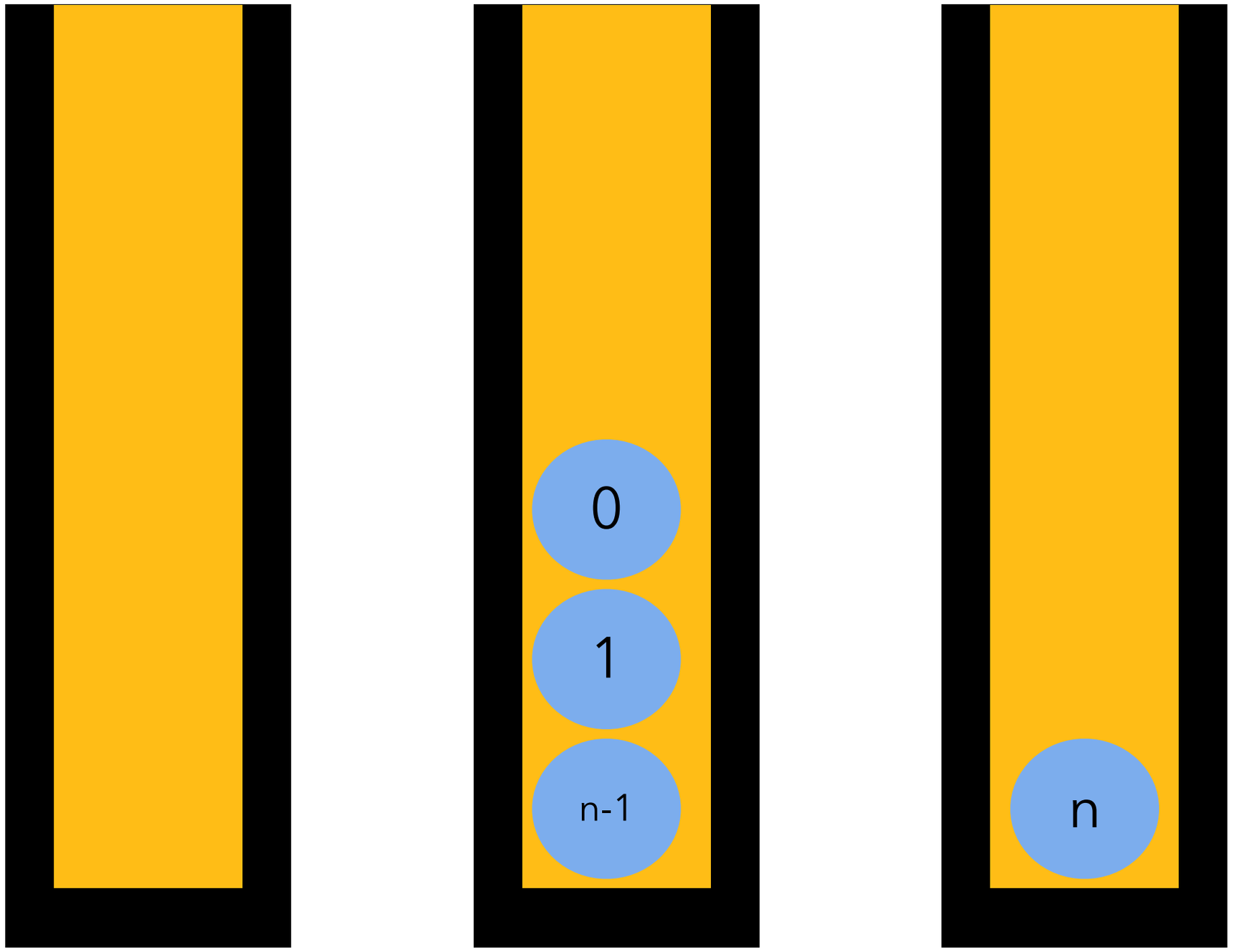
solve_hanoi_tower(n-1,a,c,b)
將問題size為n-1的河內塔從堆疊a搬移到堆疊b

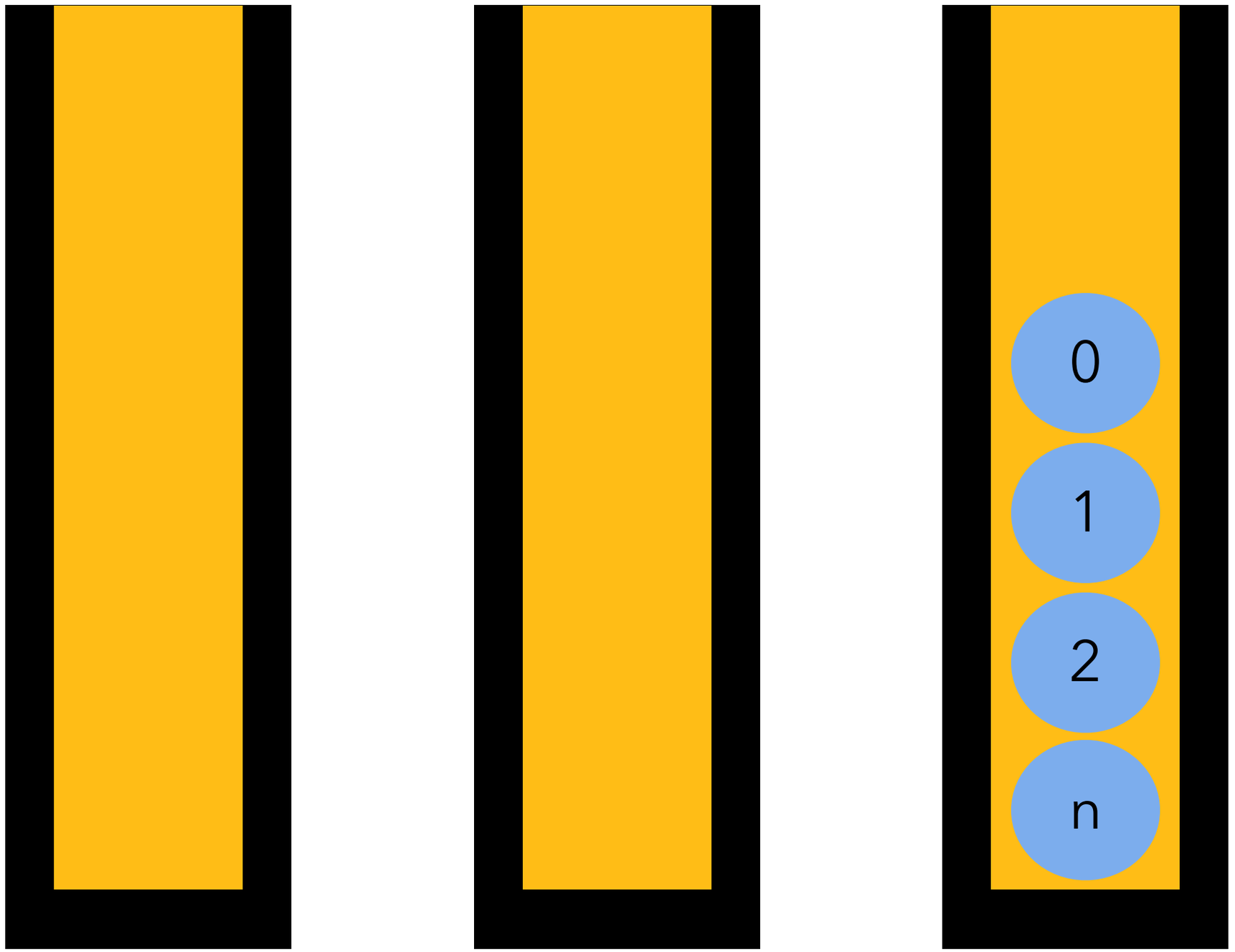


solve_hanoi_tower(1,a,b,c)
將問題size為1的河內塔從堆疊a搬移到堆疊c



solve_hanoi_tower(n-1,b,a,c)
將問題size為3的河內塔從堆疊b搬移到堆疊c




0
1
2
n
a
b
c

使用遞迴程式設計：

solve_hanoi_tower(n,a,b,c)

步驟一

修改stack2

```
class Stack:
    def __init__(self,name):
        self.items = []
        self.name = name
    def is_empty(self):
        return self.items == []
    def push(self, data):
        self.items.append(data)
```

Name代表堆疊名稱

```
    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            print("stack is empty")
```

步驟二

設計hanoi_tower模組中的初始化方法

hanoi_tower.py

```
from stack2 import Stack
def init_hanoi_tower(n):
    a = Stack("a")
    b = Stack("b")
    c = Stack("c")
    for i in range(n-1,-1,-1):
        item = str(i)
        a.push(item)
    return a,b,c
```

從模組stack2匯入類別Stack

定義hanoi_tower的初始化方法

宣告變數a的類別為Stack，堆疊名稱為"a"

依序將代表編號n-1,n-2,...,1,0的字串推入堆疊a

回傳堆疊a,b,c

hanoi_tower.py

```
from stack2 import Stack
def init_hanoi_tower(n):
    a = Stack("a")
    b = Stack("b")
    c = Stack("c")
    t = Stack("t")
    for i in range(n):
        item = str(i)
        t.push(item)
    for i in range(n):
        item = t.pop()
        a.push(item)
    return a,b,c
```

定義 hanoi_tower 的初始化方法

從模組stack2匯入類別Stack

宣告變數a的類別為Stack，堆疊名稱為"a"

依序將代表編號 n-1,n-2,...,1,0的字串推入堆疊a

回傳堆疊a,b,c

步驟三

設計hanoi_tower模組中的解決方法

hanoi_tower.py

```
def solve_hanoi_tower(n,a,b,c):
    if n == 1:
        [redacted]
        print(inst1)
        print(inst2)
        return a,b,c
    else:
        [redacted]

    return a, b, c
```

N代表問題大小，a代表啟始堆疊，c為目標堆疊

產生inst1與inst2

當問題size為1時，所需執行的搬移動作

當問題size大於1時，
以遞迴的概念
設計需執行的搬移動作
與子問題

步驟四：

設計demo_hanoi程式

demo_hanoi_2023.py

```
from hanoi_tower import *

n = 5
a,b,c = init_hanoi_tower(n)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
a,b,c = solve_hanoi_tower(n,a,b,c)
print("a:",a.items)
print("b:",b.items)
print("c:",c.items)
```