HW 2

Problem 1> Write a Python function to check whether a number is perfect or not.

Video Solution

Explanation starts here!!!!

Our goal is to obtain

```
> According to Wikipedia : In number theory, a perfect number is a positive integer
that is equal to the sum of its proper positive divisors, that is, the sum of its
positive divisors excluding the number itself (also known as its aliquot sum).
Equivalently, a perfect number is a number that is half the sum of all of its
positive divisors (including itself).
> Example : The first perfect number is 6, because 1, 2, and 3 are its proper
positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number 6 is equal to half
the sum of all its positive divisors: ( 1 + 2 + 3 + 6 ) / 2 = 6. The next perfect
number is 28 = 1 + 2 + 4 + 7 + 14. This is followed by the perfect numbers 496
and 8128
```

Perfect number => A positive integer that equals itself when all positive factors excluding itself are added.

That is, first, you need to find the factors of the quantity of the received factor first. (Except yourself) Put those divisors into the list and if the sum of all the numbers in the list using sum is the same as the argument received, let's print This number is perfect!

Explanation ends here!!!!

<<<**Finally Answer**>>>

In [1]:
```python
def perfect_number(n):
  number_list = []
  for i in range(1,n):
    if n%i==0:
      number_list.append(i)

  if sum(number_list)==n:
    print('This number is perfect')
  else:
    print('This number is not perfect')
```

In [2]:
```python
perfect_number(28)
```

```
This number is perfect
```

Problem 2> Write a Python function that checks whether a passed string is palindrome or not

Video Solution

Explanation starts here!!!!

Our goal is to obtain

```
A palindrome is a word, phrase, or sequence that reads the same backward
as forward, e.g., madam or nurses run
따라서 어떤 string을 input으로 받은 뒤 그 string이 palindrome인지 아닌지를 판별해야한다.
```

One side starts from the front and the other side compares each element starting from the back. If all the elements are the same, I came up with a function that prints the phrase This string is palindrome. At this time, only the characters in the string need to be compared, so if there are spaces or special characters in the string at first, use replace to remove them. Since the length of the incoming character is odd or even, it returns the for loop only to the value rounded down by length/2. If it is rotated, this string is palindrome is returned because it is a palindrome.

Explanation ends here!!!!

<<<**Finally Answer**>>>

In [3]:
```python
def palindrome(string):
  import math
  string = string.lower()
  characters = "'!?,.~@ "
  for x in range(len(characters)):
    string = string.replace(characters[x],"")

  for i in range(math.floor(len(string)/2)):
    if string[i]!=string[len(string)-(i+1)]:
      return print('This string is not palindrome')
  return print('This string is palindrome')
```

In [4]:
```python
palindrome('madam')
```

```
This string is palindrome
```

In [5]:
```python
palindrome('nurses run')
```

```
This string is palindrome
```

In [6]:
```python
palindrome('@@a brro!!ORrbA')
```

```
This string is palindrome
```

In [7]:
```python
palindrome('ddcsscd')
```

```
This string is not palindrome
```

Problem 3> Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

Video Solution

Explanation starts here!!!!

Our goal is to obtain

```
> a=3.0
> cir0 = Circle(a)
> print( cir0.area(), cir0.perimeter() )
```

In the class, we implement two functions (Method) that calculates area and one that calculates perimeter (Method). If you use **init** as the python method name, this method becomes a constructor.

Explanation ends here!!!!

<<<**Finally Answer**>>>

In [8]:
```python
from math import pi

class Circle:
  def __init__(self,radius):
    self.radius = radius

  def area(self):
    area = pi * self.radius ** 2
    return area

  def perimeter(self):
    perimeter = 2 * pi * self.radius
    return perimeter
```

In [9]:
```python
a=4
cir0 = Circle(a)
print(cir0.area(), cir0.perimeter())
```

```
50.26548245743669 25.132741228718345
```

Problem 4> Write a Python class named more_Circle inherited from Circle in Problem 3. This more_Circle class has one more member (color) and additional method which print the information of the circle.

Video Solution

Explanation starts here!!!!

Our goal is to obtain

```
> a=3.0
> b="red"
> cir0 = more_Circle(a, b)
> cir0.get_information()
Radious 3.0, color red, perimeter 9.42477796076938, area = 28.274333882308138,
```

In order to inherit a class, enter the class name (name of the class to inherit) as follows. Inheritance is usually used to add functions or change existing functions without changing the existing class.

Explanation ends here!!!!

<<<**Finally Answer**>>>

In [10]:
```python
class more_Circle(Circle):
  def __init__(self, radius, color):
    super().__init__(radius)
    self.color = color

  def get_information(self):
    return print('Radious : {}, Color : {}, Perimeter : {}, Area : {}'.format(self.radius, s
elf.color,self.perimeter(),self.area()))
```

In [11]:
```python
a=3.0
b='red'
cir0 = more_Circle(a, b)
cir0.get_information()
```

```
Radious : 3.0, Color : red, Perimeter : 18.84955592153876, Area : 28.274333882308138
```

Problem 5> If you have a calculator class, Create a class named "MaxLimitCalculator", inherited from the class Calculator, whose member "x" starts from 0. You have "add" method but the result cannot exceed 100 working as follows:

Video Solution

Explanation starts here!!!!

Below is the caculator class.

```
class Calculator:
 def __init__(self):
  self.x = 0
 def add(self, val):
  self.x += val
```

And the result we want is this.

```
> cal = MaxLimitCalculator()
> print(cal.x)
0
> cal.add(50)
> cal.add(60)
> print(cal.x)
100
```

Method overriding is redefining a method of a parent class in a child class. The add() method of the parent class Calculator is ignored, and the add() method of the child class MaxLimitCalculator is executed.

Explanation ends here!!!!

<<<**Finally Answer**>>>

In [12]:
```python
class Calculator:
  def __init__(self):
    self.x = 0
  def add(self, val):
    self.x += val
```

In [13]:
```python
class MaxLimitCalculator(Calculator):
  def add(self, val):
    self.x += val
    if self.x > 100:
      self.x = 100
```

In [14]:
```python
cal = MaxLimitCalculator()
```

In [15]:
```python
print(cal.x)
```

```
0
```

In [16]:
```python
cal.add(50)
```

In [17]:
```python
print(cal.x)
```

```
50
```

In [18]:
```python
cal.add(70)
```

In [19]:
```python
print(cal.x)
```

```
100
```

Problem 6> (Algorithm: recursion)Using recursive function, write a Python function to get the sum of each digit in a non-negative integer.

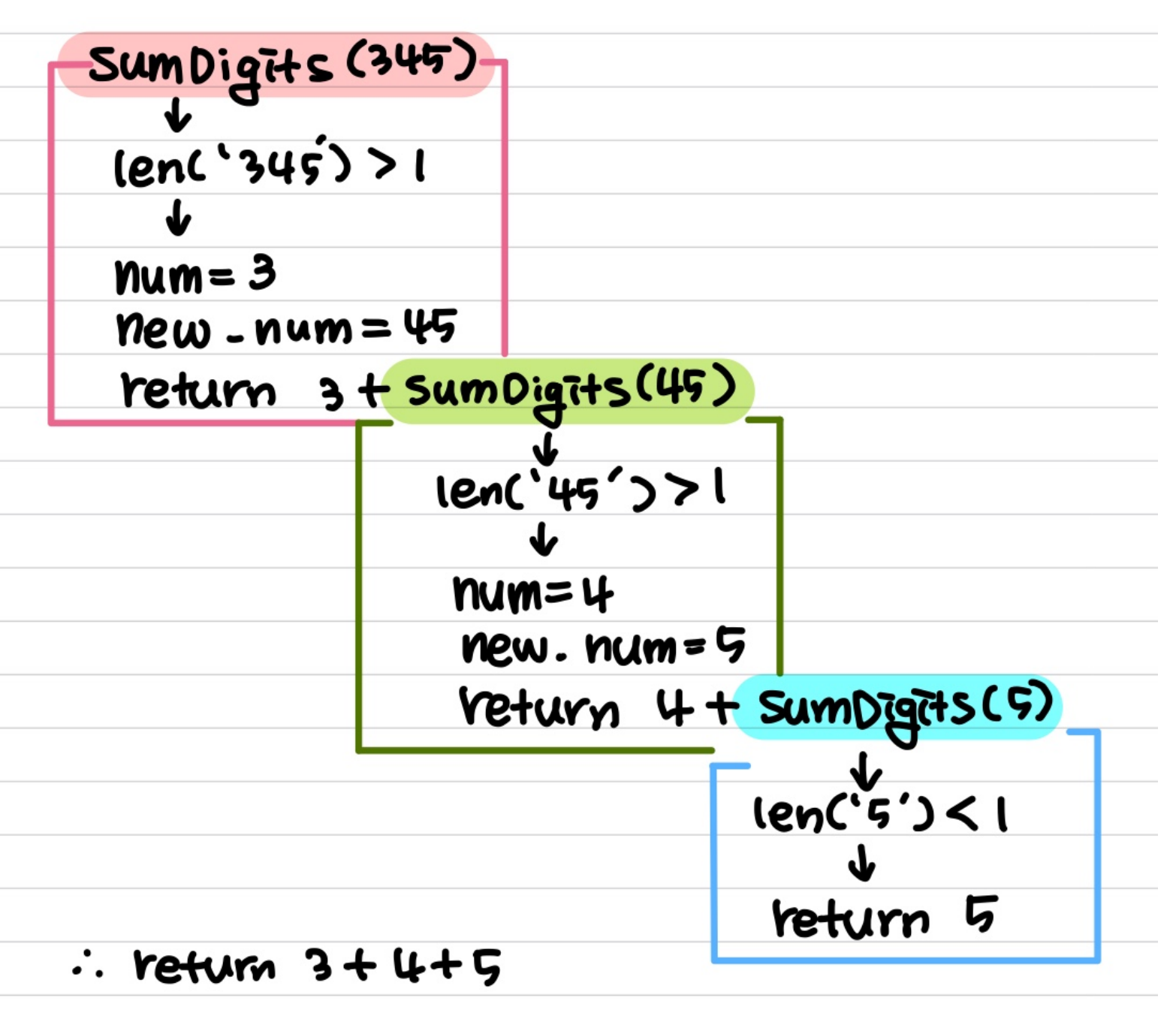
Video Solution

Explanation starts here!!!!

Our goal is to obtain

```
> # Test Data:
> sumDigits(345) #-> 12
> sumDigits(45) #-> 9
```

A recursive function is a function that calls itself.



Explanation ends here!!!!

<<<**Finally Answer**>>>

In [20]:
```python
def sumDigits(n):
  import math
  if len(str(n)) > 1:
    leng = len(str(n))
    num = math.floor(n/(10**(leng-1)))
    new_n = n - num*(10**(leng-1))
    return num + sumDigits(new_n)
  else:
    return n
```

In [21]:
```python
sumDigits(345)
```

Out[21]:
```
12
```

Problem 7> (Algorithm: recursion) Towers of Hanoi

Rules of the Game

Tower of Hanoi is a mathematical puzzle where we have three rods (say rod A, rod B, and rod C) and n disks. The objective of the puzzle is to move the entire stack from rod A to another rod C, obeying the following simple rules:

1) Only one disk can be moved at a time.

2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

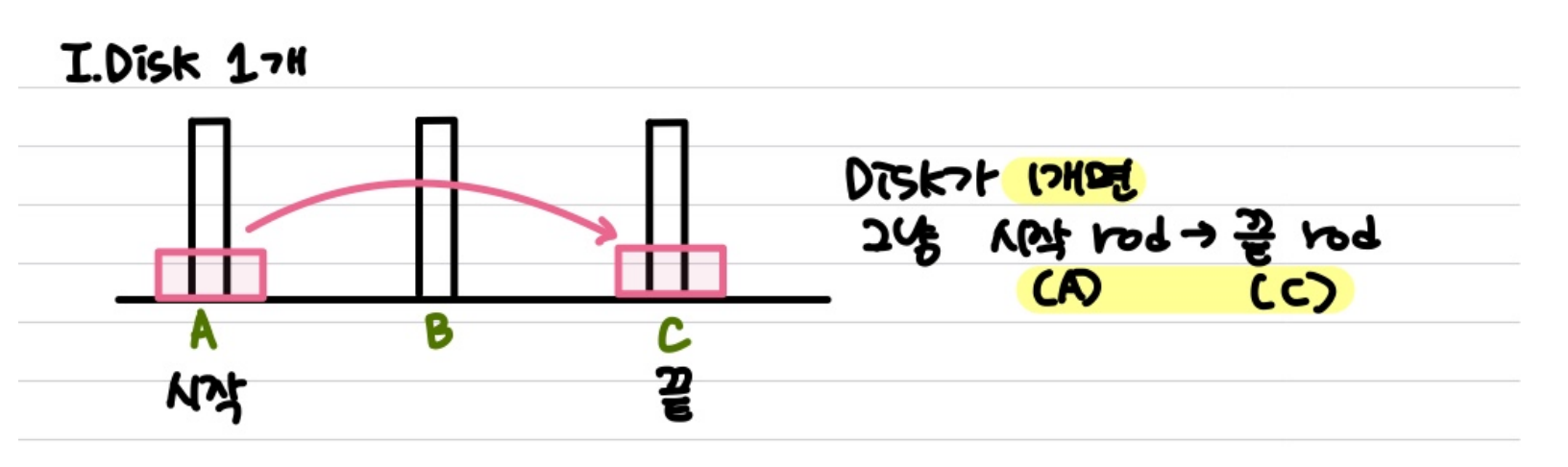
83) No disk may be placed on top of a smaller disk.

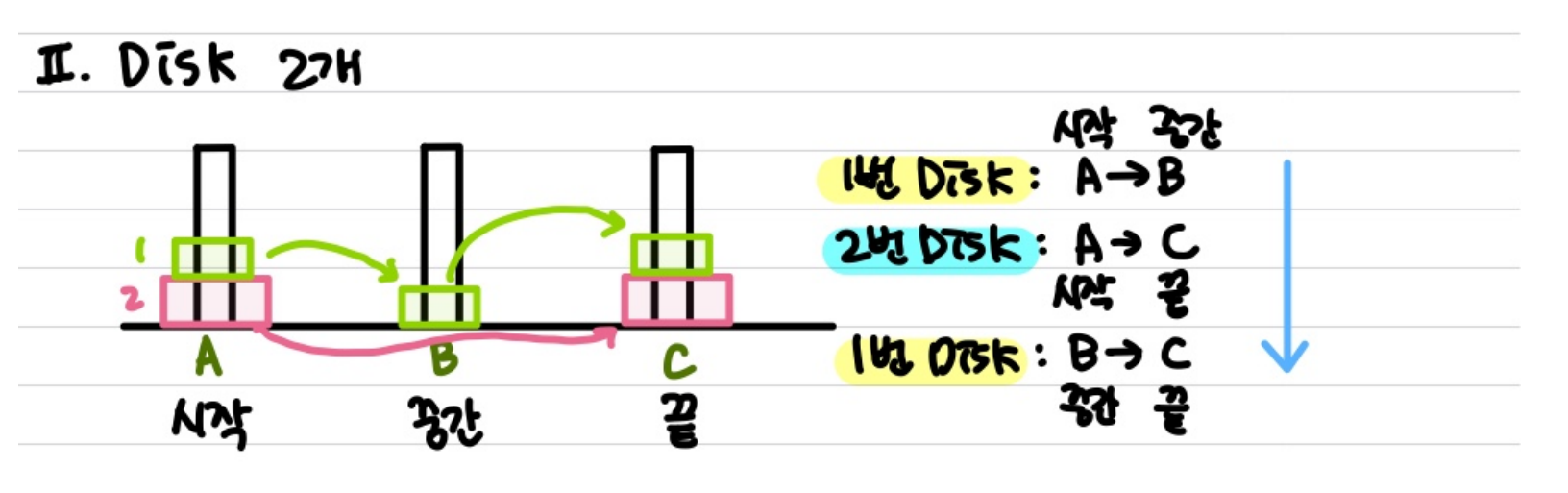
Video Solution

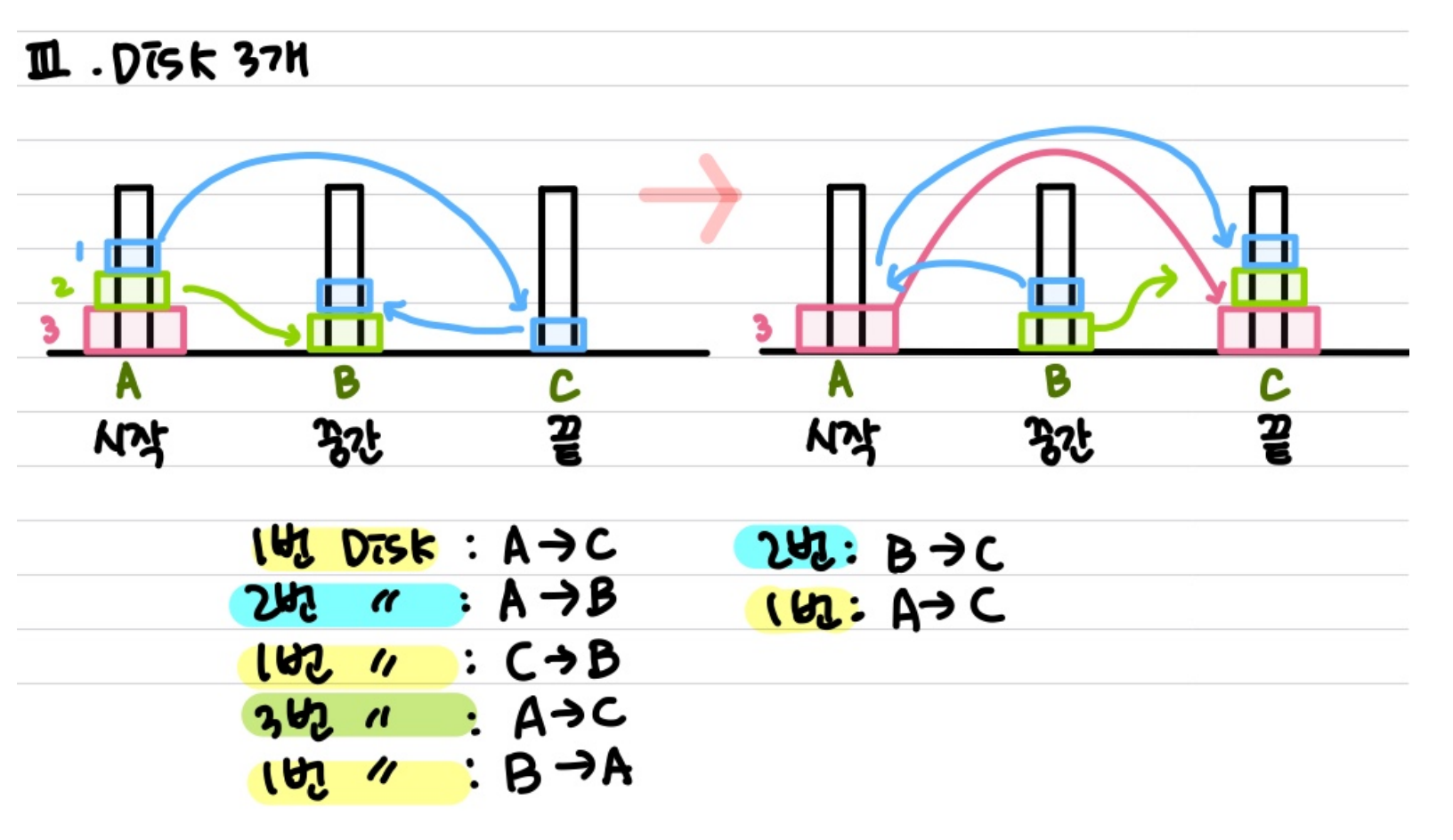
Explanation starts here!!!!

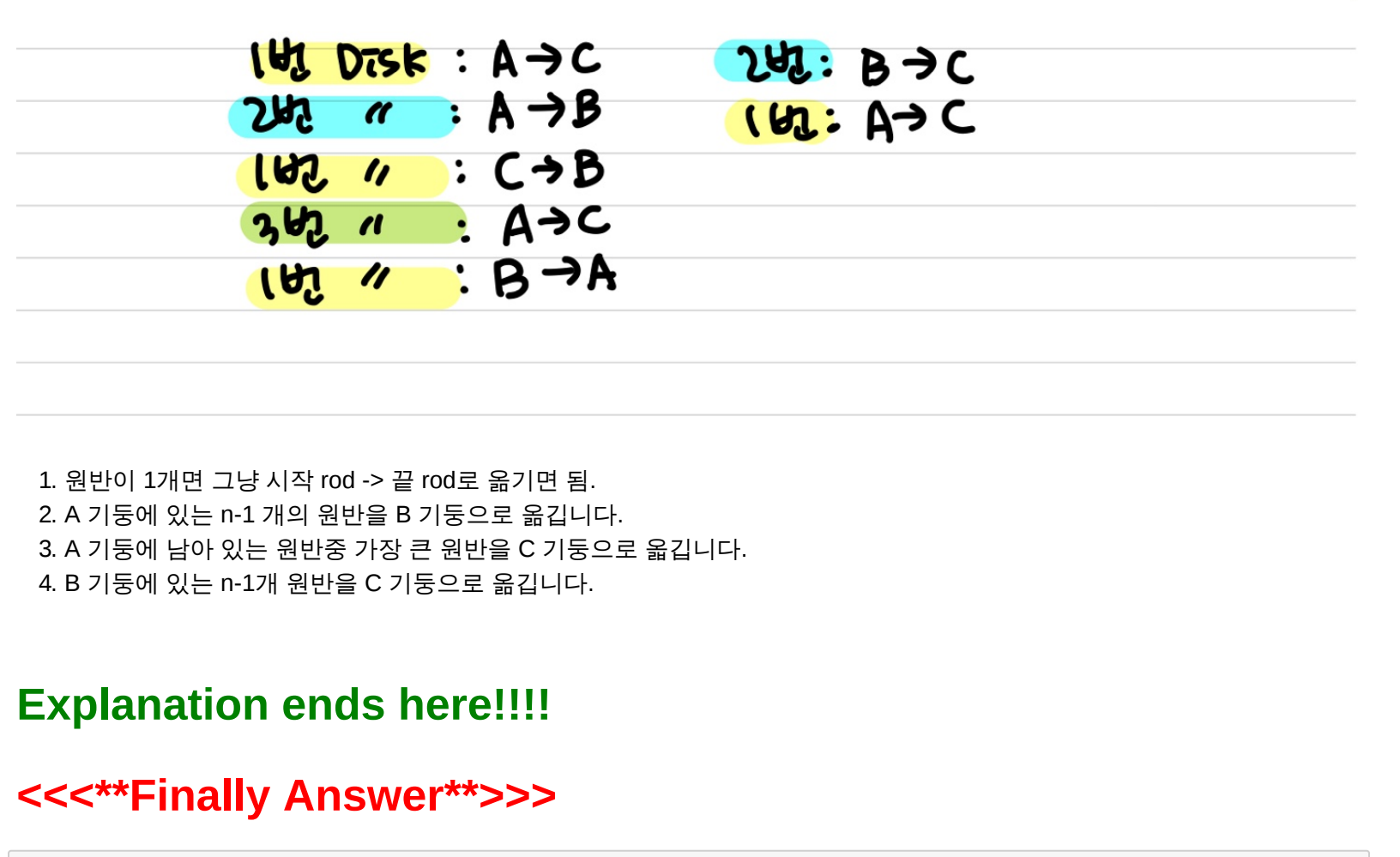
Write a recursive Python function which explains how to move n disks from rod A to rod C. For example, your function print the following:

```
> TowerOfHanoi(n, "A", "B", "C")
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```









1. 원반이 1개면 그냥 시작 rod -> 끝 rod로 옮기면 됨.
2. A 기둥에 있는 n-1 개의 원반을 B 기둥으로 옮깁니다.
3. A 기둥에 남아 있는 원반중 가장 큰 원반을 C 기둥으로 옮깁니다.
4. B 기둥에 있는 n-1개 원반을 C 기둥으로 옮깁니다.

Explanation ends here!!!!

<<<**Finally Answer**>>>

In [22]:
```python
def TowerOfHanoi(n, first_rod, mid_rod,final_rod ):
  if n == 1:
    print('Move disk {} from rod {} to rod {}'.format(n,first_rod, final_rod))
    return

  TowerOfHanoi(n - 1, first_rod, final_rod,mid_rod)
  print('Move disk {} from rod {} to rod {}'.format(n,first_rod,final_rod))
  TowerOfHanoi(n - 1, mid_rod,first_rod, final_rod)
```

In [23]:
```python
TowerOfHanoi(4,"A","B","C")
```

```
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```