Recursion

Correction

Exercise 1

See .py file.

Exercise 2

See .py file.

Exercise 3

See .py file.

Exercise 4

Identify the **Base Cases** and the **Recursive Case** of the following recursive functions:

1. one that counts down from a given number to 1.

    a. **Base Case**: When the number reaches 1.
    b. **Recursive Case**: If the number is greater than 1, call the function with the number decreased by 1.

2. one that finds the minimum value in an array of integers.

    a. **Base Case**: When the array has only one element.
    b. **Recursive Case**: Compare the first element of the array with the minimum of the rest of the array.

    Ex: [3,5,0,8,2]
    1. Search min between 3 and [5,0,8,2] → Call function again with [5,0,8,2]
    2. Search min between 5 and [0,8,2] → Call function again with [0,8,2]
    3. Search min between 0 and [8,2] → Call function again with [8,2]
    4. Search min between 8 and [2] → Call function again with [2]
    5. (*Base case !!*) → 2
    6. Then, step 4 is now Search min between 8 and 2 → 2
    7. Then, step 3 is now Search min between 2 and 0 → 0
    8. Then, step 2 is now Search min between 5 and 0 → 0
    9. Then, step 1 is now Search min between 3 and 0 → 0
    10. Final minimum is 0

3. one that checks if a list of numbers is sorted in ascending order.
    a. **Base Case**: When the list has one or zero elements.
    b. **Recursive Case**: Check if the first element is less than or equal to the second element, and then check if the rest of the list is sorted.

    Ex: [1,2,3,4,0]
    1. Is 1 lower than 2 → True and Call function again with [2,3,4,0]
    2. Is 2 lower than 3 → True and Call function again with [3,4,0]
    3. Is 3 lower than 4 → True and Call function again with [4,0]
    4. Is 4 lower than 0 → False and Call function again with [0]
    5. *(Base case !!)* → True
    6. Then, step 4 is now False and True
    7. Then, step 3 is now False and True
    8. Then, step 2 is now False and True
    9. Then, step 1 is now False and True
    10. Final result is False

4. one that calculates x raised to the power of n (i.e. x^n).
    a. **Base Case**: When n is 0 (return 1) and when n is 1 (return x).
    b. **Recursive Case**: Multiply x with the result of the function called with n-1.

    Ex: 5^5
    1. 5 x output of the function(4) → Call function again with 3
    2. 5 x output of the function(3) → Call function again with 2
    3. 5 x output of the function(2) → Call function again with 1
    4. 5 x output of the function(1) → Call function again with 0
    5. *(Base case !!)* → 5
    6. Then, step 4 is now 5 x 5
    7. Then, step 3 is now 5 x 25
    8. Then, step 2 is now 5 x 125
    9. Then, step 1 is now 5 x 625
    10. Final result is 3125

5. one that checks if a string is a palindrome.
    a. **Base Case**: When the string length is 0 or 1.
    b. **Recursive Case**: Check if the first and last characters of the string are the same, and if the substring excluding these characters is a palindrome.

    Ex: racecar
    1. Is r equal to r and aceca → True and Call function again with aceca
    2. Is a equal to a and cec → True and Call function again with cec
    3. Is c equal to c and e → True and Call function again with e
    4. *(Base case !!)* → True
    5. Then, step 3 is now True and True
    6. Then, step 2 is now True and True
    7. Then, step 1 is now True and True
    8. Final result is True

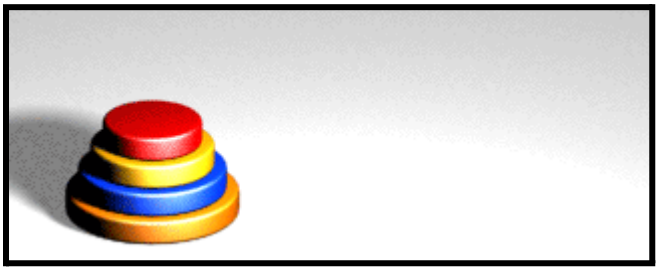
Exercise 5 – List-Ception

See .py file.

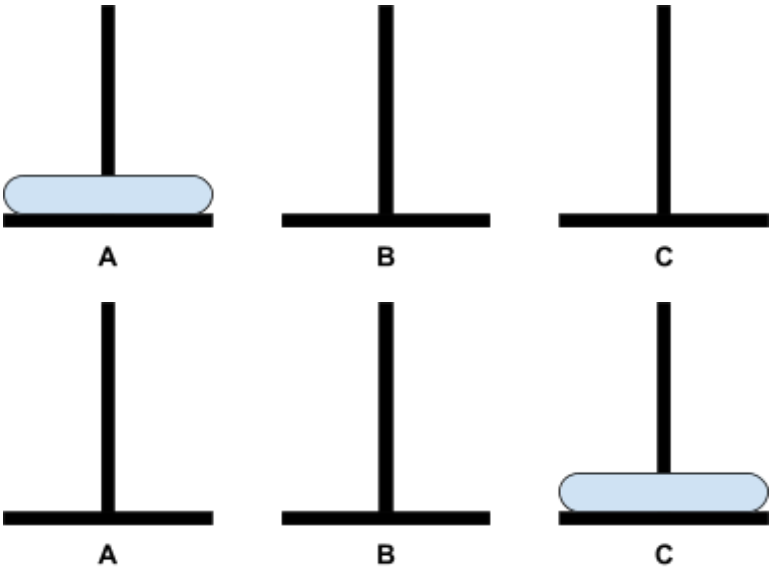
Exercise 6 – Greatest Common Divisor

See .py file.

Exercise 7 – Tower Of Hanoï

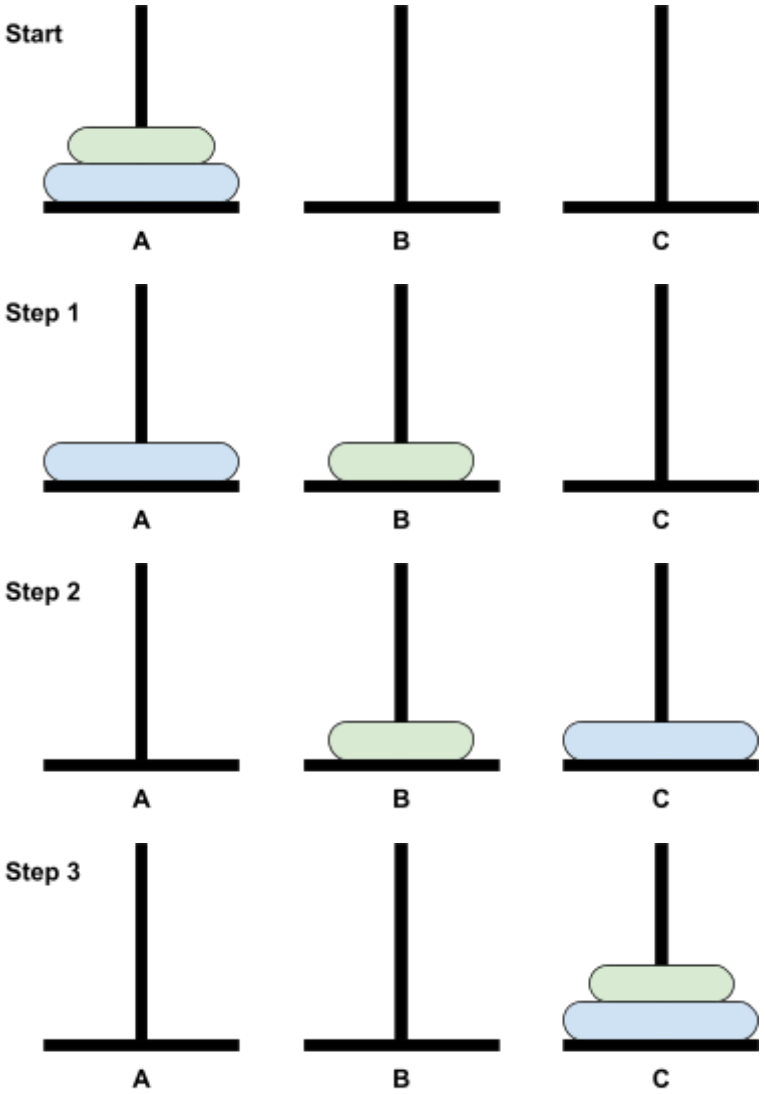


Case with 1 Disk



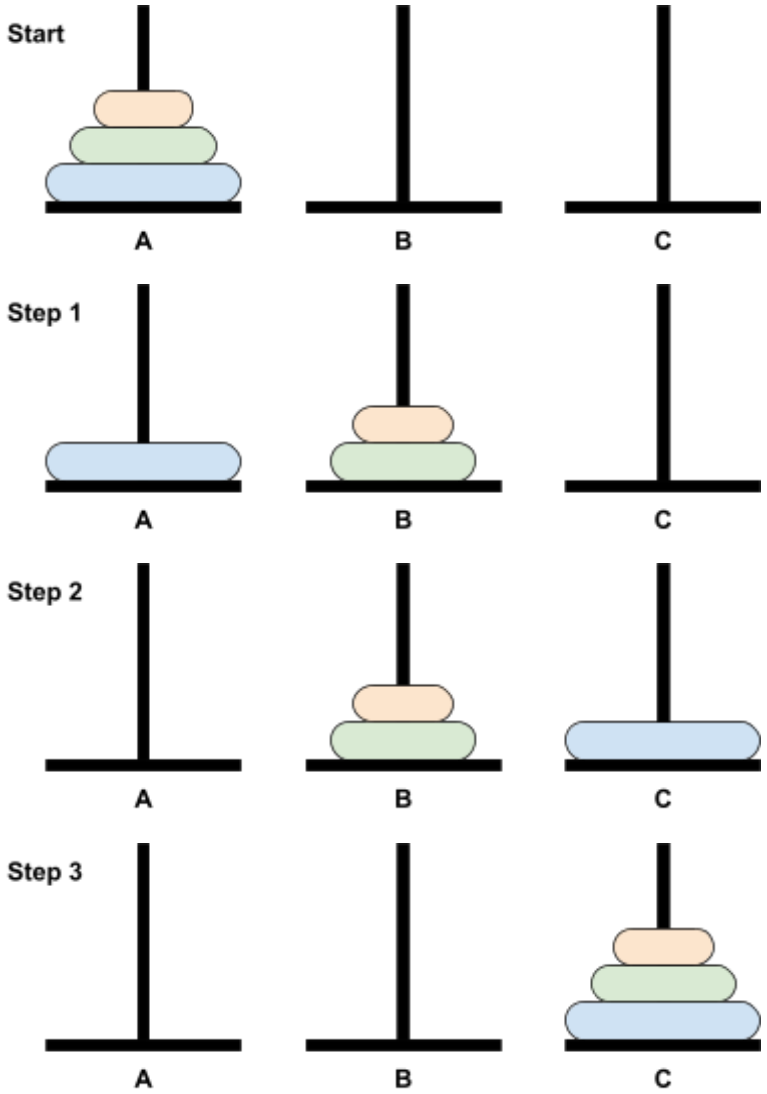
1. Move **one** disk from A to C (source to target).

Case with 2 Disks



1. Move **one** disk from A to B (source to auxiliary).
2. Move **one** disk from A to C (source to target).
3. Move **one** disk from B to C (auxiliary to target).

Case with 3 disks



1. Move **two** disks from A to B (source to auxiliary).
2. Move **one** disks from A to C (source to target).
3. Move **two** disks from B to C (auxiliary to target).

Step 2 is impossible to do because it's forbidden to move two disks at the same time. However, **it is possible to do step 2 by applying the Case with 2 Disks with:**

- **A as the source** because you want to move the 2 disks from A
- **B as the target** because you want to move the 2 disks to B
- **C as the auxiliary** because you want to use C to store the intermediate disks

Step 3 is also impossible to do because it's forbidden to move two disks at the same time. However, **it is possible to do step 3 by applying the Case with 2 Disks with:**

- **A as the auxiliary** because you want to use A to store the intermediate disks
- **B as the source** because you want to move the 2 disks from B
- **C as the target** because you want to move the 2 disks to C

Case with n Disks

All the successive cases with more than 3 disks will follow the same principle, thus a case with n disks can be achieved using the following steps:

1. STEP 1 : Move **n-1** disks from A to B (source to auxiliary).
2. STEP 2 : Move **one** disks from A to C (source to target).
3. STEP 3 : Move **n-1** disks from B to C (auxiliary to target).

Thus, a recursive function can be used to go back to a case with only two disks … two times, at steps 1 and 3 !

In the following code, the first call of the function does the step 1 (at line 8) with:

- **A as the source**
- **B as the target**
- **C as the auxiliary**

Then, the function moves one disk from the current source to the current target (the step 2).

Finally, the function is called once more to do the step 3 at line 13 with:

- **A as the auxiliary**
- **B as the source**
- **C as the target**

```
1  def tower_of_hanoi(n, source, auxiliary, target):
2
3      if n == 1:
4          disk = source[1].pop()
5          target[1].append(disk)
6          return
7
8      tower_of_hanoi(n-1, source, target, auxiliary)
9
10     disk = source[1].pop()
11     target[1].append(disk)
12
13     tower_of_hanoi(n-1, auxiliary, source, target)
```