Range Extraction

A format for expressing an ordered list of integers is to use a comma separated list of either

- individual integers
- or a range of integers denoted by the starting integer separated from the end integer in the range by a dash, '-'. The range includes all integers in the interval including both endpoints. It is not considered a range unless it spans at least 3 numbers. For example "12,13,15-17"

Complete the solution so that it takes a list of integers in increasing order and returns a correctly formatted string in the range format.

The way I would write this code and try to make it O(n)

The list has two checks, so it has to check two things

One : has to decide if its an individual integer or a range of integers, so the pointer would start at index i

if index i is numerically adjacent to the next number, then it will then check again if the number next is also adjacent, until then

a 2nd check : checks if the number exceeds it to be enough for a range flag, meaning it will just add a - between the starting and ending number

This code assumes that the number given is already sorted in increasing order

```
def solution(lst):
    if not lst:
        return "" error handling

    # Remove duplicates and sort the list
    lst = sorted(set(lst))    this is to just put it into a set
```

```
result = [] allocates new list
range_start = lst[0] tracks start of a range
previous = lst[0] tracks the previous number

for i in range(1, len(lst)):
    current = lst[i]
    if current != previous + 1:
        # Check if the current range qualifies as a range
        if previous == range_start:
            result.append(str(range_start))
        elif previous == range_start + 1:
            result.append(str(range_start))
            result.append(str(previous))
        else:
            result.append(f"{range_start}-{previous}")
        # Start a new range
        range_start = current

    previous = current

# Handle the final range or single number
if previous == range_start:
    result.append(str(range_start))
elif previous == range_start + 1:
    result.append(str(range_start))
    result.append(str(previous))
else:
    result.append(f"{range_start}-{previous}")

return ",".join(result)
```

```
""" noob solution, explained for noobs :) """

def printable(arr):
    return (','.join(str(x) for x in arr) if len(arr) < 3  # one
            else f'{arr[0]}-{arr[-1]}')                    # mor

def solution(args):
    chunk, ret = [], []                                    # ins

    for i in args:                                         # for
        if not len(chunk) or i == chunk[-1] + 1:           #
            chunk.append(i)                                #
        else:                                              #
            ret.append(printable(chunk))                   #
            chunk = [i]                                    #

    ret.append(printable(chunk))                           # do

    return ','.join(ret)                                   # ret
```

two variables are created

one to track “chunk” which is tracking consecutive numbers

ret is the variable “return” to return the processed numbers

in def solution(args):

it iterates through every item in args

if chunk is empty, it will append whatever number is shown (as its normally the first number)

or it will compare the last added integer whether it was consecutive or not, and then add it to the last chunk

otherwise, it'll see that the list is neither empty or the item is not consecutive and will just appended onto the ret variable

and chunk will restart to the current index

the printable function exists so that numbers in the current chunk can be added to the return variable and be formatted appropriately

its important that you need to add the last number to be processed

and then a return function