National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2024/2025

Tutorial 7
List Processing

Release date: 14th October 2024
Due: 20th October 2024, 23:59

General Restrictions

- No importing packages unless explicitly allowed to do so.
- Use only `list` as your compound data structure. No `tuple`, `set`, `dict`, etc.

Questions

1. Ben Bitdiddle is required to implement a function `at_least_n` which takes in a list of integers and an integer n, modifies, and returns the **original** list with all the integers smaller than n removed.

   Sample Execution:

   ```
   >>> lst = list(range(0, 10, 1))
   >>> lst2 = at_least_n(lst, 5)
   >>> lst2
   [5, 6, 7, 8, 9]
   >>> lst is lst2
   True
   ```

   (a) He implemented his function like this

   ```
   def at_least_n(lst, n):
       for i in range(0, len(lst), 1):
           if lst[i] < n:
               lst.remove(lst[i])
       return lst
   ```

   Is this implementation correct? Why?

   (b) Obviously the above implementation is wrong. But Ben decides to try again!

   ```
   def at_least_n(lst, n):
       for i in lst:
           if i < n:
               lst.remove(i)
       return lst
   ```

   Is this implementation correct? What is the moral of the story?

(c) Obviously the above implementation is also wrong. Help him out by implementing your own!

(d) Now implement the function such that it returns a **new** list instead.

Sample Execution:

```
>>> lst = list(range(0, 10, 1))
>>> lst2 = at_least_n(lst, 5)
>>> lst2
[5, 6, 7, 8, 9]
>>> lst is lst2
False
```

2. A Bingo card can be represented in a one-dimensional list, where the values are arranged from left to right, top to bottom. A deck is a list-of-lists, where the elements are Bingo cards. A player can only make edits to a card if it is on top of the deck (i.e. the first element in the deck). To shift the cards, you are given a function `shift_card_deck(deck) -> None`. `shift_card_deck` moves the top card of the given `deck` to the bottom.

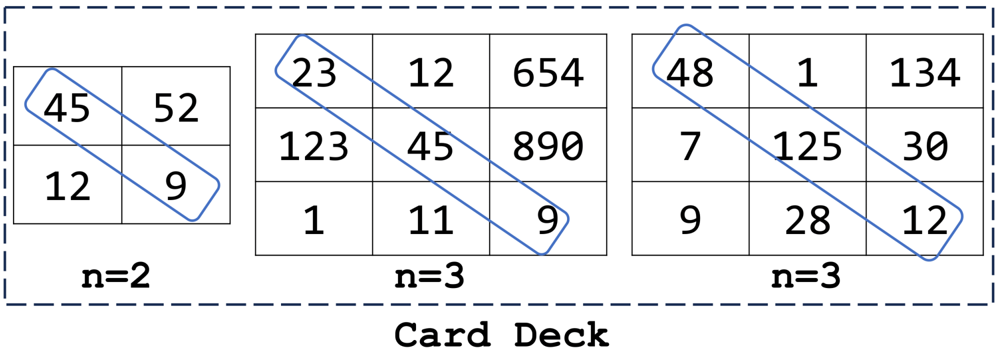

Figure 1: Bingo card deck with three cards of varying sizes with diagonal positions marked.

(a) Implement a function `cross_number` that takes in a deck and number as input, and modifies the **original** deck with all occurences of that number crossed out from all cards in the deck.

Sample Execution:

```
>>> deck
[[48, 1, 134, 7, 125, 30, 9, 28, 12], [45, 52, 12, 9],
[23, 12, 654, 123, 45, 890, 1, 11, 9]]

>>> cross_number(deck,45)
>>> deck
[[48, 1, 134, 7, 125, 30, 9, 28, 12], ['X', 52, 12, 9],
[23, 12, 654, 123, 'X', 890, 1, 11, 9]]

>>> cross_number(deck,23)
>>> deck
[[48, 1, 134, 7, 125, 30, 9, 28, 12], ['X', 52, 12, 9],
['X', 12, 654, 123, 'X', 890, 1, 11, 9]]
```

(b) Now we want to check if you managed to get BINGO! Implement a function `check_bingo` that takes in a deck as input. If there is at least one BINGO card (a card with its primary diagonals marked), the function will return `"Bingo!"` else it returns `"Not Bingo!"`. As mentioned, you may only check the card if it is top of the deck.

Sample Execution:

```
>>> deck
[[48, 1, 134, 7, 125, 30, 'X', 28, 12], ['X', 52, 12, 'X'],
['X', 12, 654, 123, 'X', 890, 1, 11, 'X']]
>>> check_bingo(deck)
'Bingo!'
>>> deck1      # another instance of a deck
[[48, 1, 134, 7, 125, 30, 'X', 28, 12], ['X', 52, 12, 9],
['X', 12, 654, 123, 'X', 890, 1, 11, 9]]
>>> check_bingo(deck)
'Not Bingo!'
```