Final exam spring 2020

In all the assignments you can look at the ***__main__*** part of the given code as well as the text files ***expected_out.txt*** for clarification of how the program should behave.

<u>This is a home exam, so all data can be used but it is forbidden to use help from any other people, fellow students or not, or to ask specific questions on forums during the exam.</u>

Multiple choice 20%
These questions are in a separate quiz assignment on Canvas.

P1: array_operations 20%
You are given the class Array. Implement the operations ***contains*** and ***remove_all***.
Limitations from previous array assignments apply. In short the only operation you can use directly on the array is the brackets (***[integer_value]***) with a single integer value.

5% contains(self, val)
returns ***True*** if an item in the array has the value ***val***, otherwise ***False***

15% remove_all(self, other):
Removes all items from the array that have the same value as any of the values in the other array. The remaining items must not have gaps between them in the array and must retain their order.
The operation then returns an integer with the number of items removed.
The final 5% are rewarded for solutions that ***<u>only go once through the array</u>*** *and for each item in the array only at most once through the other array.*
*Time complexity: O(n*m) where n is the length of self and m is the length of other.*

P2: SLL_operations 20%
Implement the operation ***is_doubled*** on a ***singly-linked list*** using ***recursion***
The function returns ***True*** if for each item in the list starting with ***head2***, the list starting with ***head1*** has two items with the same value side by side and *in the same order* <u>and no other item</u>.
Example: ***1 1 2 2 3 3*** is ***1 2 3*** doubled but ***1 1 3 3 2 2*** and ***1 1 2 3 3 2*** are not

Half marks or less are given for solutions that do not use recursion.

P3: boolean_tree 20%

Implement the class ***BooleanTree*** that holds a tree structure following these rules:

- Each *leaf* in the tree represents either the value ***True*** or the value ***False***.
- Each *non-leaf node* represents either the operator ***AND*** or the operator ***OR***.
    - The *node*'s two children represent the *operands* to the operator.

Implement the operation ***build_tree*** that takes a single string as a parameter.
The string holds a statement with the operations ***AND(***bool1***,***bool2***)*** and ***OR(***bool1***,***bool2***)***.
Each parameter (bool1 or bool2) to the operations is either another instance of ***AND*** or ***OR*** or a boolean constant, either ***T*** or ***F***.
Examples:

- AND(T,T) - (*meaning: True and True*)
- OR(F,T) - (*meaning: False or True*)
- AND(OR(F,T),T)
- OR(F,AND(F,F))
- AND(OR(F,OR(T,F)),OR(AND(T,T),OR(F,F)))

The operation ***build_tree*** parses the statement string *and builds the respective tree*.

At the top of the ***__main__*** part of the test code you are given a method of dropping the delimiters (***'('***, ***','***, ***')'***) and getting a list of tokens.
Then the statement string **“OR(AND(T,F),F)”** becomes the list **['OR', 'AND', 'T', 'F', 'F']**
Use this if you feel that it helps, but you can also parse the string in any way you see fit.

Implement the operation ***get_root_value*** that returns a single *boolean* value, ***True*** or ***False***, based on the operators and leaf values in the tree. This operation should *traverse the tree* in order to find the correct return value. The return value should be the correct boolean value based on the statement string last parsed by the instance of ***BooleanTree***.

P4: wait_list 20%

Implement the class ***FunCourse*** which holds the registration for a fun summer course.

The registration follows these rules:

- If the number of participants does not exceed the maximum, any added student is added to the course.
- If the number of participants ***does*** exceed the maximum, any added student is added to the waitlist.
- If a student, who is registered in the course, is removed, the next student on the waitlist is added to the course itself.
- A student removed from the system is removed from whichever list they are on.
- *You do not need to worry about duplicate students or names*.

Implement the following operations:

- __init__(max_participants)
    - The parameter represents how many students can be registered in the course.
- add_student(student)
    - The parameter is an instance of the class ***Student***
    - Adds a student to the registration system.
- remove_student(id)
    - The parameter is the ***id*** variable from an instance of ***Student***
    - Removes a student from the registration system.
- get_participant_string()
    - Returns a string which is the string representation of each instance of Student in the course, followed by a newline character (***"\n"***).
    - ***The participants are ordered alphabetically by their name***.
- get_wait_list_string()
    - Returns a string which is the string representation of each instance of Student on the waitlist, followed by a newline character (***"\n"***).
    - ***The participants are in the same order as they were added to the waitlist***.

You can add operations to the ***Student*** class if needed.
Implementing the less than operator (**<**), by defining the function ***__lt__*** can make a class sortable. Then a list can be sorted with some_lis.sort().