LISTS

Part 1: remove_vowels(str)

Write a function remove_vowels(str) that uses a list comprehension and returns a copy of the string with no vowels. E.g. remove_vowels('hello world') should return the string 'hll wrld'.

Note: you must use a list comprehension

Hint: your remove_vowels(str) function can be written in one line.
Hint: list('hello') returns ['h', 'e', 'l', 'l', 'o']
Hint: 'e' in 'hello' is True

Part 2: Password checker
Write a function check_pwd(pwd) that takes a string as a parameter and returns True or False if the password include at least one upper case character, at least one lower case character, and the password must **end** with at least one digit.

Note: you must use list comprehensions

Hint: 'a'.islower() is True, 'A'.isupper() is True, '1'.isdigit() is True
Hint: any(seq) is True if any element in the sequence is True, else False, E.g. any([False, True]) is True
Hint: My check_pwd(pwd) function has only one statement

Part 3: insertion sort
Write a function insertion_sort(l) that returns a copy of the argument sorted using a list and the insertion sort algorithm discussed in class. You **MAY NOT** simply use Python's sort utilities.
The idea is to start with an empty list, and then iterate through each of the elements in the list to be sorted, inserting each item in the proper spot in the new list.
For example:
insertion_sort([1, 5, 3, 3])

result before insertion	item to insert	result after insertion
[ ]	1	[1]
[1]	5	[1, 5]
[1, 5]	3	[1, 3, 5]
[1, 3, 5]	3	[1, 3, 3, 5]

Your resulting list should be sorted in ascending order.
Googling "insertion sort" will turn up an algorithm that moves each element to the left until it finds the proper spot. For this assignment, you must ***start with an empty list***

and then insert each element into the list at the proper spot. The Google solution does properly sort the list but but won't help you to learn about manipulating lists.

You should ***not*** shuffle the current value down to the left until you find the right spot for this assignment.

Special Offer! If you like a challenge, solve the homework on your own. Need a hint for insertion sort? See Canvas for hints, but only after you try to solve the problem without the hints.

FYI: My solution has 11 lines, including blanks and comments.

Hint: for/else is a great match for this problem

Part 4: Binary Trees

Write a BTree class to implement Binary Trees as described in the lecture. Your class should include at least the following methods:

- __init__(self, value) - Create a new BTree with a single node with the specified value, no left child, and no right child.
- find(self, value) - Return True if value is in the BTree or False
- insert(self, value) - insert value as a new terminal node in the appropriate spot in self if value is not already in the BTree. Return None
- traverse(self) - Traverse all of the nodes in BTree from smallest to largest and return a list of values from each node.

For example:

```
bt = BTree(27)
bt.insert(1)
bt.insert(15)
bt.insert(5)
bt.traverse() == [1, 5, 15, 27]
```

Be sure to include automated tests to demonstrate that your code works properly.

Hint: See the lecture for pseudo code for insert() and traverse()