Week 11 Practice Problems

#1. Create a CashRegister class with the following functionality.

- create an object by passing in the initial cash that the register starts with in terms of the number of loonies, toonies, fives, tens, and twenties
- print the contents of the register
- calculate and return the value of the contents of the register
- add some cash (specify the number of loonies, toonies, etc.)
- remove some cash: enter the dollar amount to remove, calculate the number of loonies, toonies, etc. that you need to remove, and decrement these values from the register.

#2. How did you implement the storage of the data inside CashRegister in Q1? Reimplement the class (call it CashRegister2) with a different representation inside. Note that all the code that you wrote to use CashRegister from outside the class should not have to change. Compare the implementation of CashRegister with CashRegister2, think about the pros and cons of each implementation.

#3. Extend the CashRegister class from Q1 andQ2 by adding a method called `make_purchase` with the parameters: the cost of the item purchased and the number of 20s, 10s, 5s, toonies, and loonies given by the purchaser. The method should:

- add the cash to itself
- calculate the change that is due
- calculate what combination of 20s, 10s, 5s, toonies, and loonies will be used to make up the change
- remove the cash from itself

Some of this functionality should re-use methods you wrote already wrote. You may want to revisit and change the code you already wrote to make it more convenient to implement `make_purchase`.

#4. The Design Problem for Week 10 localized a robot on a 1D map where the user randomly chose a starting location and provided "sensor" feedback for the robot. This question asks you to change the code so that there is no need for user input. Another class will choose the position and provide the sensor readings rather than the user.

Create a new object called `World_Model` with the following functionality.

- The constructor (i.e., `__init__`) should take in the map and randomly generate a location for the robot.
- Has a method called `robot_move` which changes the known position of the robot to be one cell to the right (assume that the world wraps around so going past the last position returns the robot to the first position).
- Has a method called `read_sensor` which returns the color of the cell of the current position of the robot.

You should change your existing code as follows.

Main code:

- Create a `World_Model` object and pass it into the constructor of the Robot.
- At the end, then robot thinks it is localized, compare the position to the true position and declare success or failure.

You should make no other changes to your main code!

`Robot` class:

- Modify the constructor to take in a `World_Model` and store it as a variable of the `Robot` object.
- Replace all questions to the user by the appropriate call to a method of `World_Model`.
- When a robot moves, it needs to tell the `World_Model` object that it moved.

#5. A doubly linked list is like a linked list except each node points both to the next node *and* to the previous node. That is, the node class might look like the following:

```
class Node:
    def __init__(self, cargo=None, prev=None, next=None):
        self.cargo = cargo
        self.prev = prev
        self.next = next
```

It is natural for DoublyLinkedList class to not only keep a reference to the head of the list but also to the last element of the list (the "tail").

Reimplement the LinkedList class from Week 11 as a DoublyLinkedList. That is, implement all the methods from the LinkedList class to work with a doubly linked list.

Note that some of the functions, like `print_backward_nicely`, will be simpler because now you can "back-up" via `prev`. However, `remove_item` will probably be

more complicated because you have to deal with both the `next` and `prev` references and the head and tail. Add an `add_last` method too. Remember that whenever you change a list, you need to update both the `next` and `prev` references.