Lab 1 - Description
(Stacks and Queues)

Lab Overview:
For this lab, we will be learning how to build and use linear data structures like stacks and queues. Stacks and queues are very common data structures in the CS world. In this lab, we will focus on implementing these data structures using the linked list data structure as a base. The linked list will then be expanded upon to build our stack and queue.

Core Tasks:
1. Write the queue class.
2. Write the stack class
3. Test your classes with the provided main function.

Program Requirements:
Task 1: Write the queue class.

For task 1, you will need to write a Queue class. The class is detailed as follows:

- __init__(self, size) => None
  - This is the constructor for the class. You will need to store the following here:
    - The head pointer
    - The tail pointer
    - A variable maxSize
    - A variable currentSize
    - Any other instance variables you want.

- enqueue(self, ticket) => True/False
  - This is the enqueue method. It will take a ticket and enqueue it at the tail of the queue. It will then return True/False depending on if the enqueue was successful (Hint: if the queue is full).

- dequeue(self) => MealTicket/False
  - This is the dequeue method. It will remove the ticket at the front of the queue and return it or False if the queue is empty.

- front(self) => MealTicket/False
  - This method lets the user peak at the ticket at the front of the queue without deleting it.

- isEmpty(self) => True/False
  - This method will return True/False depending on if the queue is empty or not.

- isFull(self) => True/False
  - This method will return True/False depending on if the queue is full or not,
Note: See Figure 1 at the bottom of this section for a sample test run of the Queue class. Your program must perform exactly as shown in the figure. Note 2: All methods in the queue class have a hard complexity upper bound of O(1). Your implementation must be in O(1). Note 3: For this assignment you must make and use your own linked list as the base data structure. Using lists, dictionaries or any other data structure will result in a 0 for the assignment.

Task 2: Write the stack class

For task 2, you will need to write a Stack class. The class is detailed as follows:

- `__init__(self, size) => None`
  - This is the constructor for the class. You will need to store the following here:
    - The head pointer
    - A variable maxSize
    - A variable currentSize
    - Any other instance variables you want.

- `push(self, ticket) => True/False`
  - This is the push method. It will take a ticket and push it at the top of the stack. It will then return True/False depending on if the push was successful (Hint: if the stack is full).

- `pop(self) => MealTicket/False`
  - This is the pop method. It will remove the ticket at the top of the stack and return it or False if the stack is empty.

- `peek(self) => MealTicket/False`
  - This method lets the user peak at the ticket at the top of the stack without deleting it.

- `isEmpty(self) => True/False`
  - This method will return True/False depending on if the stack is empty or not.

- `isFull(self) => True/False`
  - This method will return True/False depending on if the stack is full or not,

Note: See Figure 2 at the bottom of this section for a sample test run of the Stack class. Your program must perform exactly as shown in the figure. Note 2: All methods in the stack class have a hard complexity upper bound of O(1). Your implementation must be in O(1). Note 3: For this assignment you must make and use your own linked list as the base data structure. Using lists, dictionaries or any other data structure will result in a 0 for the assignment.

I/O Specifications:

For this assignment 2 files are provided: lab1-main.py and mealticket.py. You are free to use the main provided or to make your own. Warning: I will be testing your code with a more robust main that will check all of the corner cases and uses a wider variety of tickets. Keep this in mind while developing your data structures. The description above gives you a solid idea on what needs to be done but does not describe all corner cases. Your programs must be robust (i.e. do not crash when given bad input or told to peak/front when the stack/queue is empty etc.)
Figure 1: Queue testing output
Remarks:
All programs written in this class will be done using the newest version of Python available in the lab (Python 3.7/3.8). This is because Python 3.5+ is platform independent (i.e. you can code on a PC and run it on a Mac). The computers in KLA 26 all have python on them so make sure your code runs on the computers in the lab as it will be tested on an identical system. Feel free to install python on your own computer and bring it to the lab.

If you are new to python, here are some super helpful resources to get you up to speed:
1. Cheat sheets: [https://www.pythoncheatsheet.org/](https://www.pythoncheatsheet.org/) ➜ Most recommended! Many thumbs up!
2. Python documentation: [https://docs.python.org/3/](https://docs.python.org/3/)
All programming assignments are to be done **individually**. Your code will be looked at with professional software for cheating. **Warning:** This includes using online sources. (e.g. Do not go online and copy code from stack overflow. People have tried this before. You will fail.) Be extra careful with your code. Do not ever show your work to anyone other than the TA (me) or the professor. They will most likely copy your work and your will both fail.

**Submission Requirements:**
In order to receive any credit for the assignment the student **must** do the following:
1. Name your program “\(<\text{Duck-ID}\>\_\text{lab0.py}\)”. (i.e. my duck ID is jhall10 so my submission would be named jhall10\_lab0.py. **Note:** your duck-ID is the same as your email id and the username to log on to CIS computers **not** your 951… number that is your UO PID.
2. Submit your python file onto Canvas.
That’s it! Make sure that you test your code on the lab computers to make sure it works.

**Grading:**
Your work will be graded along three primary metrics: Correctness, Completenss, and Elegance.

**Correctness: (60% of total grade)**
- You wrote the class methods as specified.
- Your class methods have a complexity of O(1).
- There are no logic bugs in your code.
- You utilize a linked list.
- You wrote the queue as a FIFO and the stack as a LIFO data structure.

**Completeness (29% of total grade)**
- Your class methods do not have any syntax errors.
- Your program is robust.
- You completed both classes with all of their methods.

**Elegance: (11% of total grade)**
- Your program is well organized.
- You make good use of whitespace, comments, and the file doc string.
- You write your code in a readable manner.
- You use descriptive variable/function/class names.

**Late Policy:**
The late policy is as follows:
Your homework is always due on a Wednesday. On Thursday, I will take 10% off. On Friday, I will take 20% off. No homework will be accepted after Friday.
If you encounter an unfortunate event or are working with a disability: Please email or speak to me. I am super flexible and am always on your side. I will give extensions as needed and am willing to work with you to make sure you get the most out of this course.