Project 1 - Description
(Stacks and Queues)

Project Overview:
For this programming project, 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.

Core Tasks:
1. Write the Node class.
2. Write the queue class.
3. Write the stack class

Program Requirements:
Task 1: Write the Node class.

- **Class Name:** Node
- **Methods:**
  - __init__(<Node> self, <Generic> data) → <Nonetype> None
    - **Complexity:** O(1)
    - **Valid Input:** Any. The data parameter defaults to None if no data specified.
    - **Error Handling:** N/A
    - **Instance variables:**
      - <Generic> data
        - **Note:** This variable contains the data for a given node.
      - <Node> next
        - **Note:** This variable contains a pointer to the next node.
      - Any other instance variables you want.
      - Any other methods you want.

Task 2: Write the queue class.

For task 2, you will need to write a Queue class. To receive any credit, your class must follow the specifications below **exactly**:
- **Class Name:** Queue
- **Methods:**
\[\text{\texttt{\_\_init\_(\Queue self, \Int capacity) \to \Nonetype None}}\]

- **Complexity**: \(O(1)\)
- **Valid Input**: An integer from \((0, \infty]\).
- **Error Handling**: Raises a \texttt{QueueCapacityTypeError} if the Queue capacity is of the wrong type. Raises a \texttt{QueueCapacityBoundError} if the capacity is negative or 0.

**Instance variables:**
- \texttt{\langle\text{Node}\rangle head}
  - \textit{Note}: This variable contains the head of our linked list.
- \texttt{\langle\text{Node}\rangle tail}
  - \textit{Note}: This variable contains the tail of our linked list.
- \texttt{\langle\Int\rangle capacity}
  - \textit{Note}: This variable contains the total number of items that can be fit into the queue.
- \texttt{\langle\Int\rangle currentSize}
  - \textit{Note}: This variable contains the current number of items in the queue.
- Any other instance variables you want.

\[\text{\texttt{enqueue(\Queue self, \Generic item) \to \Bool returnValue}}\]

- **Complexity**: \(O(1)\)
- **Valid Input**: Any Python object.
- **Error Handling**: Raises a \texttt{QueueIsFull} exception if the enqueue method is called when the queue is full.
- **Note**: This is the enqueue method. It will add an item to the queue. It will then return True/False depending on if the enqueue was successful.

\[\text{\texttt{dequeue(\Queue self) \to \Generic/bool returnValue}}\]

- **Complexity**: \(O(1)\)
- **Error Handling**: Raises a \texttt{QueueIsEmpty} exception if the dequeue method is called on an empty queue.
- **Note**: This is the dequeue method. It will remove an item from the queue and return it or False if the queue is empty.

\[\text{\texttt{front(\Queue self) \to \Generic/bool returnValue}}\]

- **Complexity**: \(O(1)\)
- **Note**: This method lets the user peek at the item at the front of the queue without deleting it. It will either return the item at the front of the queue or False if the queue is empty.
isEmpty(<Queue> self) → <bool> returnValue
- Complexity: O(1)
- Note: This method will return True/False depending on if the queue is empty or not.

isFull(<Queue> self) → <boolean> returnValue
- Complexity: O(1)
- Note: This method will return True/False depending on if the queue is full or not.

**Task 2:** Write the stack class

For task 2, you will need to write a *Stack* class. To receive any credit, your class must follow the specifications below **exactly**:

- **Class Name:** Stack
- **Methods:**
  - __init__(<Stack> self, <Int> capacity) → <Nonetype> None
    - Complexity: O(1)
    - **Valid Input:** An integer from (0, ∞].
    - **Error Handling:** Raises a StackCapacityTypeError if the Queue capacity is of the wrong type. Raises a StackCapacityBoundError if the capacity is negative or 0.
    - **Instance variables:**
      - <Node> head
        - **Note:** This variable contains the head of our linked list.
      - <Int> capacity
        - **Note:** This variable contains the total number of items that can be fit into the stack.
      - <Int> currentSize
        - **Note:** This variable contains the current number of items in the stack.
      - Any other instance variables you want.
  - push(<Stack> self, <Generic> item) → <Bool> returnValue
    - Complexity: O(1)
    - **Valid Input:** Any Python object.
    - **Error Handling:** Raises a StackIsFull exception if the push method is called when the queue is full.
- **Note**: This is the push method. It will add an item to the stack. It will then return True/False depending on if the push was successful.
  - `pop(<Stack> self) → <Generic/bool> returnValue`

- **Complexity**: O(1)

- **Error Handling**: Raises a `StackIsEmpty` exception if the pop method is called on an empty stack.

- **Note**: This is the pop method. It will remove an item from the stack and return it or False if the queue is empty.
  - `peek(<Stack> self) → <Generic/bool> returnValue`

- **Complexity**: O(1)

- **Note**: This method lets the user peek at the ticket at the front of the stack without deleting it. It will either return the item or false if the stack is empty.
  - `isEmpty(<Stack> self) → <bool> returnValue`

- **Complexity**: O(1)

- **Note**: This method will return True/False depending on if the stack is empty or not.
  - `isFull(<Stack> self) → <bool> returnValue`

- **Complexity**: O(1)

- **Note**: This method will return True/False depending on if the stack is full or not.

**Remarks:**
1. All methods in the classes have a hard complexity upper bound of O(1). Your implementation must be in O(1).
2. For this assignment, you must make and use your own linked list as the base data structure. This linked list must be made with Node elements. Using lists, dictionaries or any other data structure will result in a 0 for the assignment.
3. **Warning**: You are not allowed to import anything into the source file with the classes. All programs must be written in basic Python 3.5+.
4. You will need to make your own main for testing your code using the methodology described in the lab. Testing your own code is a very common practice in the industry.
5. **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 objects. Keep this in mind while developing your data structures. You will need to use the methodology discussed in lab to figure out all corner cases and account for them. Your programs must be robust (i.e. does not crash when given bad input or told to peak/foreground when the stack/queue is empty etc.)
6. 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. 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.

Notes:
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/  Most recommended! Many thumbs up!
2. Python documentation: https://docs.python.org/3/
All projects in this class are designed with the understanding that you have either done CIS I/II/III or have completed courses with similar content. Knowledge of python is necessary. If you are unfamiliar with the language, I can provide additional resources to aid you.

Submission Requirements:
In order to receive any credit for the assignment the student must do the following:
1. Create a file named “<Duck-ID>_p1.py”. (i.e. my duck ID is jhall10 so my submission would be named jhall10_p1.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. Fill in your classes. (This file should not have a main function. It should just have the classes).
3. Submit your python file onto Canvas.
That’s it! Make sure that you test your code to make sure it works.

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

Correctness: (60% of total grade)
• You wrote the queue as a FIFO and the stack as a LIFO data structure.
• You wrote the class methods as specified.
• Your class methods meet the complexity requirements.
• You utilize a linked list as the base data structure.
• Your classes are robust and fault-tolerant.

Completeness (25% of total grade)
• Program contains 3 classes named: Node, Queue, and Stack
• Each class contains methods as defined above.
• The method signatures were implemented as specified.

Elegance: (15% of total grade)
• Your code is well organized with good use of whitespace.
• Your code is well documented.
• You write your code in a readable manner.
• You use descriptive variable/function/class names.
- See the programming guide for the full list of expectations.

**Late Policy:**
Make sure to read the info below carefully:
Your homework is always due on a Wednesday at 11:59 pm. On Thursday, I will take 10% off of the total points available. On Friday, I will take 20% off. No homework will be accepted after Friday. I do not start grading the homework until the following Monday. Keep this in mind when submitting your assignment: It may be better for you to seek help and submit the homework a day late than to submit on time and fail.

If you encounter an unfortunate event or are working with a disability: Please email or speak to me. I can be emailed at: jhall10@cs.uoregon.edu. Make sure to include the keyword CIS313 in your header. 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.