Part 1: Multiple choice questions (25%)
These are in a separate quiz on Canvas.

Part 2: DLL_Ordered
Implement the following three functions in the class DLL_Ordered.

- **(10%)** ***find_node_to_insert_at(value)***
  - Returns an instance of **DLL_Node**.
  - The returned node is in the doubly-linked list at the correct place to insert this ***value*** next to so the order of the list is ascending.
- **(10%)** ***insert_at_node(value, node)***
  - Inserts a new **DLL_Node** containing ***value*** next to the ***node***.
  - If the node was found using **find_node_to_insert_at** then the list should now be in ascending order.
    - *It does not matter whether the new node is inserted in front of or behind the referenced node, as long as the whole implementation is consistent*.
- **(5%)** ***insert_ordered(value)***
  - Inserts a new DLL_Node with the data ***value*** into the list so that the list is in ascending order.
    - Use the previous two functions, ***find_node_to_insert_at*** and ***insert_at_node*** to implement this functionality.
  - *You can assume this is the only operation used to insert nodes, so the list will always stay in ascending order*.

Part 3: List programming
Implement the following function into the class DLL_Ordered.

- **(25%)** ***get_range_in_SLL(min, max)***
  - Returns an instance of ***SLL_Node***, which is the first node (head) in a **singly-linked list**.
  - The list returned should include all the values from the DLL_Ordered list with values ranging from ***min*** to ***max***, both values included, in the same order and the same number of nodes as they are in the original list (see expected output).
  - ***Note*** *that the operation is implemented on a class containing a doubly-linked list, but should return the first node in a singly linked list*.

Part 4: SLL recursion

Implement the following functions

These functions must be completed fully recursive for full marks

- **(15%) find_index(head, value)**
  - Takes in as parameters the head of a SLL and a value
  - Returns the index at which the first matching value is found in the list
  - Returns None if the value doesn't exist in the list
  - Examples:
    - find_index(head: 5 6 3 4, Value: 3) => 2
    - find_index(head: 5 6 3 4, Value: 7) => None
    - find_index(head: 5 6 3 4 5, Value: 5) => 0

- **(10%) ordered_subset(head1, head2)**
  - Takes in as parameters the heads of two SLLs sets
  - Returns **True** if **head1** is an ordered subset of **head2**
    - Head1 is an ordered subset of head2 if all elements in head1 exist in head2 and exist in the same order in both sets
    - *Both heads are sets (they do not have duplicates)*
  - Returns **False** otherwise
  - Examples:
    - ordered_subset(head1: 1 3 5, head2: 6 1 4 3 2 5) => True
    - ordered_subset(head1: 5 3 1, head2: 6 1 4 3 2 5) => False