class StudentTicket(Ticket):
    
    """
    SEE SUPER.
    Ticket -> StudentTicket
    """
    student_number - student number
    """
    Student_number: int
    """
    def __init__(self, movie: str, price: float, student_number: int) -> None:
        """
        SEE SUPER (+ and student number student_number.)
        EXTENDS
        """
        super().__init__(movie, price)
        self.student_number = student_number
    
    def get_price(self) -> float:
        """
        SEE SUPER.
        OVERIDES (or EXTENDS if they use super().get_price())
        """
        return self._price * 0.9
    
    def __str__(self) -> str:
        """
        SEE SUPER.
        OVERIDES
        """
        return "{} (STUDENT#{}: {} - ORIGINAL: {})".format(self.movie,
                                                            self.student_number,
                                                            self.get_price(),
                                                            self._price)

class AdultTicket(Ticket):
    pass
GRADING SCHEME

[Letter]: Marks

Leave a letter to indicate which criteria they lost a mark on. For example "E -0.5" means they forgot to write EXTENDS/OVERRIDE

- **T**: 1 mark for inheriting from Ticket for AdultTicket and StudentTicket
- **E/O**: 2 marks for correctly saying EXTENDS/OVERRIDES
  - 0.5 for __init__
  - 0.5 for get_price
  - 0.5 for __str__
  - 0.5 for AdultTicket (either they wrote pass, or something that's correct in it if they implemented something)
- **I**: 2 marks for StudentTicket's __init__
  - 1 mark for the correct __init__ header (taking in self, movie, price, student_number)
    - If they use 's' or something instead of self, don't remove a mark for it here (but count it as a mistake; see below)
  - 1 mark for correct __init__ implementation
- **P**: 1 mark for correct get_price implementation
- **S**: 1 mark for correct __str__ implementation
- **A**: 1 mark for correct AdultTicket
  - Writing pass, or implementing things correctly if they implement it
- **D**: 2 marks for documentation
  - Take off -0.5 for each method/class that's missing documentation; just having SEE SUPER is enough.
  - Except AdultTicket. If they just write pass, they don't need documentation. Don't remove marks for writing documentation there.
  - This goes for any missing type/return type annotations
  - -0.5 for missing class attribute annotations (e.g. student_number: int)

For any other mistakes they make, remove 0.5 for each type of mistake. For example:

- -0.5 if they use s (or something) instead of self
  - If they do this consistently (throughout everything), -1 instead.
- -0.5 if they call super().<whatever> or Ticket.<whatever> incorrectly (i.e. they pass self in to super().<whatever> or they don't pass in self for Ticket.<whatever>)
Question 2 Part A Solution

def empty_container(c: Container) -> list:
    """
    Remove all the items in c and return a list of those items.
    >>> s = Stack()
    >>> s.add(1)
    >>> s.add(2)
    >>> empty_container(s)
    [2, 1]
    >>> s.is_empty()
    True
    >>> q = Queue()
    >>> q.add(1)
    >>> q.add(2)
    >>> empty_container(q)
    [1, 2]
    >>> q.is_empty()
    True
    """
    lst = []
    while not c.is_empty():
        lst.append(c.remove())
    return lst

GRADING SCHEME (20% = 0.4; if they leave it blank/cannot answer)

[Letter]: Marks

- E: 0.5 marks for having "while not c.is_empty()" or equivalent
- R: 0.5 marks for having "c.remove()" somewhere
- L: 1 mark for returning a list of the removed elements from c
  - If they just forget to return, -0.5

-0.5 for any other minor coding errors (e.g. missing brackets after remove/is_empty).

If they try to check whether c is a Stack or Queue using type(), isinstance() or some comparison (c == Stack), or if they try to access c._content, then they can only get 1 mark at most.
Question 2 Part B Solution

def is_queue(c: Container) -> bool:
    """
    Return True if c is a Queue and False if c is a Stack.
    c should still be empty at the end of the function.
    
    Precondition: c.is_empty() == True
    """
    c.add(1)
    c.add(2)
    is_queue = c.remove() == 1
    # Alternatively: return empty_container(c) == [1, 2]
    c.remove()
    return is_queue

GRADING SCHEME (20% = 0.6; if they leave it blank/cannot answer)

[Letter]: Marks

- **A**: 1 mark for adding some elements to c
  - If they didn’t add their own elements (i.e. they tried to empty out c and add those items in), they lose this mark but can still get the other 2 if their logic is correct (i.e. they add it back in and remove it again, to see if the items are in the same order).
- **B**: 1 mark for getting the correct bool
- **C**: 1 mark for having c be empty upon returning

-0.5 for any other minor coding errors.

If they try to check whether c is a Stack or Queue using type(), isinstance() or some comparison (c == Stack), or if they try to access c._content, then they can only get 1 mark at most. EXCEPT if they just wrote return type(c) == Queue. That's a 0.
Question 2 Part C Solution

def add_to_container(c: Container, lst: list, is_queue: bool) -> None:
    
    Adds the elements of lst into c so that, when items are removed from c, they are removed in the order of lst. is_queue is a boolean that represents whether c is a Queue (is_queue is True), or a Stack (in which case, is_queue is False).

    It doesn't matter if lst is modified or not (i.e. if you change lst, you don't have to put it back in order).

    Precondition: c.is_empty() == True

    >>> s = Stack()
    >>> add_to_container(s, [1, 2, 3], False)
    >>> s.remove()
    1
    >>> q = Queue()
    >>> add_to_container(q, [1, 2, 3], True)
    >>> q.remove()
    1
    
    """
    if not is_queue:
        lst.reverse()
    for item in lst:
        c.add(item)
    """

GRADING SCHEME (20% = 0.6; if they leave it blank/cannot answer)

[Letter]: Marks

- **R**: 1 mark for realising they need to have Stacks be the revere of Queue
  - If they get both Q and S, they'll get this mark. This is mostly for students that realised they needed to reverse something but might have messed up which one they reversed.
  - If they reverse but reversed incorrectly, they get this mark but lose 0.5 for a coding error.
- **Q**: 1 mark for correctly adding to c if c is a Queue
- **S**: 1 mark for correctly adding to c if c is a Stack

-0.5 for any other minor coding errors (e.g. if they don't reverse correctly)

If they try to check whether c is a Stack or Queue using type(), isinstance() or some comparison (c == Stack), or if they try to access c._content, then they can only get 1 mark at most.
Question 3 Part A Solution

GRADING SCHEME (20% = 0.4; if they leave it blank/cannot answer)

[Letter]: Marks

- S: 0.5 for having the correct size (for their diagram, even if it's wrong)
- B: 0.5 for having the correct back (points to a new node)
- L: 1 for having the correct LinkedList

The format doesn't matter, as long as it's understandable.

If they misinterpreted the question and used that interpretation in Part B: They can get at most 3 marks in Part B using roughly the same scheme as B (i.e. updating back (or if they added before everything, updating front), updating size, setting pointers correctly)

In this case, also put a ☆ in Part B to know how to cap the grade.
Question 3 Part B Solution

def add_after_each(self, new_val: Any) -> None:
    
    Modify this LinkedList so that it adds a new linked list node with the
    value new_val after every linked list node in lnk. Update the attributes
    of lnk as needed.

    >>> lnk = LinkedList()
    >>> lnk.add_after_each("A")
    >>> print(lnk)
    |

    >>> lnk.prepend('H')
    >>> lnk.prepend('A')
    >>> print(lnk)
    H -> A -> |

    >>> lnk.add_after_each("A")
    >>> print(lnk)
    H -> A -> A -> A -> |
    
    cur_node = self.front
    while cur_node is not None:
        new_node = LinkedListNode(new_val)
        new_node.next_ = cur_node.next_
        cur_node.next_ = new_node
        self.size += 1
        cur_node = new_node.next_

        if self.back is not None:
            self.back = self.back.next_

GRADING SCHEME (20% = 1; if they leave it blank/cannot answer)

[Letter]: Marks

- **B**: 1 mark for updating back correctly
- **S**: 1 mark for updating size correctly
- **L**: 1 mark for making new LinkedListNodes at each step
- **N**: 1 mark for setting the new node's next pointer correctly
- **C**: 1 mark for setting each original node's next pointer correctly

If they add a new node in the empty case: -1

If they loop incorrectly (i.e. going to new_node instead of the node that was originally after cur_node): -1

-0.5 for any other minor coding errors (if they forget the _ in next, don't deduct anything.)