knopp_daniel_assignment_04

October 25, 2023

1 Assignment # 4

Daniel Knopp

Implement the data structure and their main methods for: **DoubleLinkedList**, **Stack**, and **Queue**

[ ]:
```
# import necessary modules
import random
import time
```

1.1 Q1 (2 pts) Double LinkedList

Convert the singly LinkedList data structure to a double LinkedList. Ensure the methods are well implemented, most of them need to be modified.

In addition, implement the following methods

1. **sort**: Sort the list (ascend)
2. **shuffle**: unsort the list. Every time you call this function, the list will shuffle again.
3. **removeAt**(index): remove the *item* in the position *index*
4. **insertAt**(index,value): Add a *item* in the position *index*
5. **reverse**(): reverse de list

[ ]:
```
class Node:
  def __init__(self, data):
    self.data = data
    self.next = None
    self.prev = None

class DoubleLinkedList:
  def __init__(self):
    self.head   = None
    self.tail   = None
    self.__size = 0 # private variable, can not modified outside of this class

  # overwrite default len() method
  def __len__(self):
    return self.__size # return the private size attribute

  # override defalt print method
```

```
 def __str__(self):
   cur    = self.head                                # initialize current node␣
↪(start)
   output = ""                                       # initialize output string
   for i in range(self.__size):                      # loop until after last node
     if cur is self.head:                            # if on the first node
       output += "None<-"                            # print that points to None
     else:                                           # Else not first node
       output += f"{cur.prev.data}<-"                # print the data previous␣
↪points to
     if cur is self.tail:                            # if on the final node
       output += f"{cur.data}->None"                 # print that points to None
     else:                                           # Else not final node
       output += f"{cur.data}->{cur.next.data}|"     # print the data next points␣
↪to
     cur = cur.next                                  # update current node as the␣
↪next in the list
   return output                                     # Return text output

 # method to add item at the front of the linked list
 def push(self, data):
   newNode = Node(data)          # create a new node
   if self.__size == 0:          # if there is no head node yet (empty list)
     self.tail = newNode         # set the new node as the list tail node
   else:                         # else the list is not empty
     newNode.next   = self.head # set new node next to the current list head␣
↪node
     self.head.prev = newNode    # set the former list head node previous␣
↪attribute to the new node
   self.head    = newNode       # set the list head node as the new node
   self.__size += 1             # increase size of list by 1
   return self.__str__()        # print the printed list as output

 # Add items at the end of the linkedlist
 def append(self, data):
   if self.__size == 0:
     self.push(data)
   else:
     newNode        = Node(data) # create a new node
     newNode.prev   = self.tail   # set the previous attribute of new node to␣
↪current list tail node
     self.tail.next = newNode     # set the next attribute of the current list␣
↪tail node to new node
     self.tail      = newNode     # set list tail node to new node
     self.__size   += 1           # increase size of list by 1
   return self.__str__()          # print the printed list as output
```

```
 # method to sort nodes in ascending order (using selection sort)
 def sort(self):
   if self.__size <= 1: return self.__str__()       # If there is only 1 or␣
↪less nodes, no need to sort
   cur = self.head                                  # initialize the starting␣
↪node
   for i in range(self.__size-1):                   # for selection sort, need␣
↪to loop as many times as have nodes minus 1
     min_val  = cur.data                              # initialize the min␣
↪value to current node (will use min selection sort)
     min_node = cur                                   # initialize the node␣
↪associated with min value to current node
     for j in range(self.__size-i):                   # loop until after the␣
↪last node
       if cur.data < min_val:                           # if current node␣
↪value is lesser than the smallest know value for this for loop iteration
         min_val  = cur.data                              # store min value
         min_node = cur                                   # store node␣
↪associated with min value
       cur = cur.next                                   # move to next node
     if min_node == self.tail:                        # if the min node for␣
↪this for loop iteration is the list tail node
       min_node.prev.next = None                        # set the next␣
↪attribute of the previous node to None
       self.tail          = min_node.prev               # set the tail of the␣
↪list to the previous node
     elif min_node != self.head:                      # else if is not a first␣
↪node (is a middle node)
       min_node.prev.next = min_node.next               # set the next␣
↪attribute of the previous node to the node after the min node
       min_node.next.prev = min_node.prev               # set the previous␣
↪attribute of the next node to the node before the min node
     if i == 0:                                       # if this is the fist␣
↪iteration of the for loop, min node will go to the beginning
       self.head.prev = min_node                        # set the previous␣
↪attribute of the list head node to the min node
       min_node.next  = self.head                       # set the next␣
↪attribute to the current list head node
       min_node.prev  = None                            # set the previous␣
↪attribute to None
       self.head      = min_node                        # set the list head␣
↪node to be the min nodemin_node.prev = None
     else:                                            # else this is not the␣
↪first iteration, min node will go after previous min node
```

```
      min_node.prev           = prev_min_node          # set the previous␣
↪attribute to the previous min node
      min_node.next           = prev_min_node.next     # set the next␣
↪attribute to the next node after the previous min node
      prev_min_node.next.prev = min_node               # set the previous␣
↪attribute of the next node after the previous min node to the new min node
      prev_min_node.next      = min_node               # set the next␣
↪attribute of the previous min node to the new min node
    prev_min_node = min_node                           # store current min node␣
↪as the new previous min node
    cur = prev_min_node.next                           # for next for loop␣
↪iteration, set the current node to the node just after the last min node
  return self.__str__()                                # print the printed list␣
↪as output

# method to reverse the order of the nodes in the list
def reverse(self):
  if self.__size <= 1: return self.__str__() # If there is only 1 or less␣
↪nodes, no need to reverse
  cur = self.head               # start at the list head node
  for i in range(self.__size): # loop until after last node
    next_cur = cur.next         # store the next node (will overwrite next␣
↪attribute later)
    if cur is self.head:        # if this is the list head node
      new_tail = cur              # store current node as new list tail node
    elif cur is self.tail:      # if this is the list tail node
      new_head = cur              # store the current node as new list head␣
↪node
    prev = cur.prev             # store the previous node (will be␣
↪overwritten later)
    cur.prev = cur.next         # set current node previous attribute as␣
↪next node after current node
    cur.next = prev             # set current node next attribute as␣
↪previous node before current node
    cur = next_cur              # increment the current node to the original␣
↪next node after the current node
  self.tail = new_tail         # set the list tail node
  self.head = new_head         # set the list head node
  return self.__str__()        # return the printed list as output

# method to remove a node at a specific index within the list and print the␣
↪data of the removed node
def remove_at(self, idx):
  if self.__size == 0:                return  'Error, list is empty'
  if idx < 0 or idx > self.__size - 1: return f'Error, index ({idx}) not␣
↪possible with list size: {self.__size}'
```

```
    cur = self.head
    for i in range(idx+1):
      if i == idx:
        if i == 0:
          cur.next.prev = None
          self.head     = cur.next
        elif i == self.__size-1:
          cur.prev.next = None
          self.tail     = cur.prev
        else:
          cur.prev.next = cur.next
          cur.next.prev = cur.prev
      out = cur.data
      cur = cur.next
    self.__size -= 1
    return out

  # method to insert a node at a specific index within the list
  def insert_at(self, idx, data):
    if idx < 0 or idx > self.__size: return f'Error, index ({idx}) not possible␣
↪with list size: {self.__size}'
    new_node = Node(data)
    cur = self.head
    for i in range(idx+1):
      if i == idx:
        if i == 0:
          new_node.next = cur
          cur.prev      = new_node
          self.head     = new_node
        elif i == self.__size:
          new_node.prev  = self.tail
          self.tail.next = new_node
          self.tail      = new_node
        else:
          cur.prev.next = new_node
          new_node.prev = cur.prev
          new_node.next = cur
          cur.prev      = new_node
      cur = cur.next if i < self.__size else None
    self.__size += 1
    return self.__str__()

  # method to shuffle the list
  def shuffle(self):
    tmp_list = DoubleLinkedList()
    shuffled_idx = random.sample(range(self.__size), self.__size)
    # print(shuffled_idx)
```

```
    for idx in shuffled_idx:
      cur = self.head
      for i in range(idx+1):
        if i == idx:
          tmp_list.append(cur.data)
        cur = cur.next
      # print(tmp_list)
    cur     = self.head
    tmp_cur = tmp_list.head
    for i in range(self.__size):
      cur.data = tmp_cur.data
      cur.prev = tmp_cur.prev
      cur      = cur.next
      tmp_cur  = tmp_cur.next
    return self.__str__()
```

[ ]:
```
# Create a helper function to make printing doubly linked lists easier to␣
↪visualize
def print_doubly_linked_list(my_list):
    node_prev = []
    node_curr = []
    node_next = []
    for i, node in enumerate(str(my_list).split(sep='|')):
        prv, cur, nxt = node.replace('<', '').replace('>', '').split(sep="-")
        node_prev.append(prv)
        node_curr.append(cur)
        node_next.append(nxt)

    max_num_char = max([len(s) for s in node_prev] + [len(s) for s in␣
↪node_curr] + [len(s) for s in node_next] + [len('node.prev.data')])

    lines = [f'{"NodeID":<6} | {"node.prev.data":<{max_num_char}}  | {"node.
↪data":<{max_num_char}}  | {"node.next.data":<{max_num_char}}']
    lines.append('-'*(6 + 3 + 4 + 4 + max_num_char*3))

    for i in range(len(node_prev)):
        lines.append(f'Node {i + 1:02d}: {node_prev[i]:<{max_num_char}} <-␣
↪{node_curr[i]:<{max_num_char}} -> {node_next[i]:<{max_num_char}}')

    print('\n'.join(lines))
```

[ ]:
```
# Test the basic class methods

print('Testing basic class methods')
my_list = DoubleLinkedList()
print(f'my_list.push(2)  : {my_list.push(2)}')
print(f'my_list.append(3): {my_list.append(3)}')
```

```
print(f'my_list.push(1)  : {my_list.push(1)}')
print(f'my_list.append(4): {my_list.append(4)}')
print(f'len(my_list)     : {len(my_list)}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()

print('Test the reverse() method')
print(f'my_list.reverse(): {my_list.reverse()}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()

print('Test the sort() method')
print(f'my_list.sort(): {my_list.sort()}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()

print('Test the remove_at() method')
print(f'my_stack.remove_at(-1): {my_list.remove_at(-1)}')
print(f'my_stack.remove_at( 4): {my_list.remove_at( 4)}')
print(f'my_stack.remove_at( 2): {my_list.remove_at( 2)}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()

print('Test the insert_at() method')
print(f'my_list.insert_at(-1, 3): {my_list.insert_at(-1, 3)}')
print(f'my_list.insert_at( 4, 3): {my_list.insert_at( 4, 3)}')
print(f'my_list.insert_at( 2, 3): {my_list.insert_at( 2, 3)}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()

print('Test the shuffle() method')
print(f'my_stack.shuffle(): {my_list.shuffle()}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()
print(f'my_stack.shuffle(): {my_list.shuffle()}')
```

```
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()
print(f'my_stack.shuffle(): {my_list.shuffle()}')
print()
print('Visualize the list nodes:')
print_doubly_linked_list(my_list)
print()
```

```
Testing basic class methods
my_list.push(2)  : None<-2->None
my_list.append(3): None<-2->3|2<-3->None
my_list.push(1)  : None<-1->2|1<-2->3|2<-3->None
my_list.append(4): None<-1->2|1<-2->3|2<-3->4|3<-4->None
len(my_list)     : 4

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
------------------------------------------------------------
Node 01: None           <- 1              -> 2
Node 02: 1              <- 2              -> 3
Node 03: 2              <- 3              -> 4
Node 04: 3              <- 4              -> None

Test the reverse() method
my_list.reverse(): None<-4->3|4<-3->2|3<-2->1|2<-1->None

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
------------------------------------------------------------
Node 01: None           <- 4              -> 3
Node 02: 4              <- 3              -> 2
Node 03: 3              <- 2              -> 1
Node 04: 2              <- 1              -> None

Test the sort() method
my_list.sort(): None<-1->2|1<-2->3|2<-3->4|3<-4->None

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
------------------------------------------------------------
Node 01: None           <- 1              -> 2
Node 02: 1              <- 2              -> 3
Node 03: 2              <- 3              -> 4
Node 04: 3              <- 4              -> None

Test the remove_at() method
```

```
my_stack.remove_at(-1): Error, index (-1) not possible with list size: 4
my_stack.remove_at( 4): Error, index (4) not possible with list size: 4
my_stack.remove_at( 2): 3

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
-------------------------------------------------------------
Node 01: None           <- 1              -> 2
Node 02: 1              <- 2              -> 4
Node 03: 2              <- 4              -> None

Test the insert_at() method
my_list.insert_at(-1, 3): Error, index (-1) not possible with list size: 3
my_list.insert_at( 4, 3): Error, index (4) not possible with list size: 3
my_list.insert_at( 2, 3): None<-1->2|1<-2->3|2<-3->4|3<-4->None

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
-------------------------------------------------------------
Node 01: None           <- 1              -> 2
Node 02: 1              <- 2              -> 3
Node 03: 2              <- 3              -> 4
Node 04: 3              <- 4              -> None

Test the shuffle() method
my_stack.shuffle(): None<-3->4|3<-4->2|4<-2->1|2<-1->None

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
-------------------------------------------------------------
Node 01: None           <- 3              -> 4
Node 02: 3              <- 4              -> 2
Node 03: 4              <- 2              -> 1
Node 04: 2              <- 1              -> None

my_stack.shuffle(): None<-3->2|3<-2->1|2<-1->4|1<-4->None

Visualize the list nodes:
NodeID | node.prev.data  | node.data       | node.next.data
-------------------------------------------------------------
Node 01: None           <- 3              -> 2
Node 02: 3              <- 2              -> 1
Node 03: 2              <- 1              -> 4
Node 04: 1              <- 4              -> None

my_stack.shuffle(): None<-3->2|3<-2->1|2<-1->4|1<-4->None

Visualize the list nodes:
```

```
NodeID | node.prev.data  | node.data       | node.next.data
-------------------------------------------------------------
Node 01: None           <- 3              -> 2
Node 02: 3              <- 2              -> 1
Node 03: 2              <- 1              -> 4
Node 04: 1              <- 4              -> None
```

1.2 Q2 (2 pts) FILO : Stack

q2.1. Describe your approach, how to use linked list to implememnt the stack, singly linked list or double linked list, which one and why.

q2.2 The implementation Code

Implement a Stack Data Structure and the following basic methods:

- **push(object)**: Inserts an object at the top of the Stack.
- **pop()**: Removes and returns the object at the top of the Stack.
- **peek()**: Returns the object at the top of the Stack without removing it.

In addition to the basic methiods, implements : clear, display, count and toArray.

- **clear()**: Removes all objects from the Stack.
- **contains(Object)**: Determines whether an element is in the Stack.
- **display()**: Returns a string that represents the current object. (Inherited from Object)
- **ToArray()**: Copies the Stack to a new array.

Q2.1 Response Here

I will use a singly linked list because I will be using the linked list head attribute as the "top" of the stack and will only ever need to move in the direction towards the tail ("down" the stack); thus I do not need the extra information a "previous" attribute provides for each node in the list and can save some memory by using a singly linked list instead of doubly. I plan to use the push() method to add elements, such that the list head attribute will be continually updated with the newest added element. Thus the first element into the list will be the last one to come out.

[ ]:
```python
# Q2.2
## Elements to add in the Stack
class ElementStack:
  def __init__(self, value):
    self.data = value
    self.next = None

#FILO - First In, Last Out
class Stack:

  def __init__(self):
    self.head   = None
    self.__size = 0
```

```
# Overwrite default method for print()
def __str__(self):
  if self.__size == 0: return 'None'
  cur = self.head
  out = ""
  while cur != None:
      out += f"{cur.data}->"
      cur  = cur.next
  out += "None"
  return out

# Overwrite default method for len()
def __len__(self):
  return self.__size

# Inserts an object at the top of the Stack.
def push(self, value):
  new_node      = ElementStack(value)
  new_node.next = self.head
  self.head     = new_node
  self.__size  += 1
  return self.__str__()

# Removes and returns the object at the top of the Stack.
def pop(self):
  if self.__size == 0: return 'Error, stack is empty'
  out = self.head.data
  if self.__size == 1:
    self.head = None
  else:
    self.head = self.head.next
  self.__size -= 1
  return out

# Returns the object at the top of the Stack without removing it.
def peek(self):
  return self.head.data


# Removes all objects from the Stack.
def clear(self):
  self.head = None
  self.__size = 0
  return self.__str__()
```

```
  # Returns a string that represents the current object. (Inherited from Object)
  def display(self):
    return self.__str__()


  #Determines whether an element is in the Stack.
  def contains(self, value):
    if self.__size == 0: return False
    cur = self.head
    while cur != None:
      if cur.data == value:
        return True
      cur = cur.next
    return False

  # Copies the Stack to a new array.
  def toArray(self):
    out = []
    if self.__size != 0:
      cur = self.head
      while cur != None:
        out.append(cur.data)
        cur = cur.next
    return out
```

[ ]:
```
print('Test creating stack')
my_stack = Stack()
print(f'my_stack        : {my_stack}')
print(f'my_stack.push(5): {my_stack.push(5)}')
print(f'my_stack.push(4): {my_stack.push(4)}')
print(f'my_stack.push(3): {my_stack.push(3)}')
print(f'my_stack.push(2): {my_stack.push(2)}')
print(f'my_stack.push(1): {my_stack.push(1)}')
print(f'len(my_stack)   : {len(my_stack)}')
print()

print('Test pop() method')
print(f'my_stack      : {my_stack}')
print(f'my_stack.pop(): {my_stack.pop()}')
print(f'my_stack      : {my_stack}')
print()

print('Test peek')
print(f'my_stack       : {my_stack}')
print(f'my_stack.peek(): {my_stack.peek()}')
print(f'my_stack       : {my_stack}')
print()
```

```
print('Test contains()')
print(f'my_stack            : {my_stack}')
print(f'my_stack.contains(1): {my_stack.contains(1)}')
print(f'my_stack.contains(3): {my_stack.contains(3)}')
print()

print('Test toArray() method')
print(f'my_stack          : {my_stack}')
print(f'my_stack.toArray(): {my_stack.toArray()}')
print(f'my_stack          : {my_stack}')
print()

print('Test clear() method')
print(f'my_stack        : {my_stack}')
print(f'my_stack.clear(): {my_stack.clear()}')
print(f'my_stack        : {my_stack}')
print()
```

```
Test creating stack
my_stack        : None
my_stack.push(5): 5->None
my_stack.push(4): 4->5->None
my_stack.push(3): 3->4->5->None
my_stack.push(2): 2->3->4->5->None
my_stack.push(1): 1->2->3->4->5->None
len(my_stack)   : 5

Test pop() method
my_stack      : 1->2->3->4->5->None
my_stack.pop(): 1
my_stack      : 2->3->4->5->None

Test peek
my_stack       : 2->3->4->5->None
my_stack.peek(): 2
my_stack       : 2->3->4->5->None

Test contains()
my_stack            : 2->3->4->5->None
my_stack.contains(1): False
my_stack.contains(3): True

Test toArray() method
my_stack          : 2->3->4->5->None
my_stack.toArray(): [2, 3, 4, 5]
my_stack          : 2->3->4->5->None
```

```
Test clear() method
my_stack        : 2->3->4->5->None
my_stack.clear(): None
my_stack        : None
```

1.3 Q3 (2 pts): FIFO - Queue

q3.1. Describe your approach, how to use linked list to implememnt the queue, singly linked list or double linked list, which one and why.

q3.2 The implementation Code

Implement the Queue Data Structure with the methods: Enqueue, Dequeue, Peek.

- **Enqueue** adds an element to the end of the Queue.
- **Dequeue** removes the oldest element from the start of the Queue.
- **Peek** returns the oldest element that is at the start of the Queue but does not remove it from the Queue.

In addition to the basic

In addition to the basic methods, implement : clear, display, count and toArray.

- **clear()**: Removes all objects from the Queue.
- **contains(Object)**: Determines whether an element is in the Queue.
- **display()**: Returns a string that represents the current Queue. (Inherited from Object)
- **ToArray()**: Copies the Queue to a new array.

Q3.1 Response Here

Similar to the stack, I only ever need to traverse the Queue in a single direction - so I will implement it as a singly linked list. This time, however, I will implement the enqueue() method as an append() style method so that the first element added to the queue will be at the head and will then be the first one taken out (I'll use the list head node as the "front" of the queue). I will also keep track of the list tail node to make appending simpler (so this will be a double ended singly linked list, technically).

[ ]:
```
#Q3.2
## Elements to add in the Queue
class ElementQueue:
  def __init__(self, value):
    self.data = value
    self.next = None

class Queue:
  def __init__(self):
    self.head   = None
    self.tail   = None
    self.__size = 0

  # Overwrite default method for print()
```

```
def __str__(self):
  if self.__size == 0: return 'None'
  cur = self.head
  out = ""
  while cur != None:
      out += f"{cur.data}->"
      cur  = cur.next
  out += "None"
  return out

# Overwrite default method for len()
def __len__(self):
  return self.__size

# Adds an element to the end of the Queue.
def enqueue(self, value):
  new_node = ElementQueue(value)
  if self.__size == 0:
    self.head = new_node
    self.tail = new_node
  else:
    self.tail.next = new_node
    self.tail      = new_node
  self.__size += 1
  return self.__str__()

# Removes the oldest element from the start of the Queue.
def dequeue(self):
  if self.__size == 0: return 'Error, queue is empty'
  out = self.head.data
  if self.__size == 1:
    self.head = None
  else:
    self.head = self.head.next
  self.__size -= 1
  return out

# Returns the oldest element that is at the start of the Queue but does not␣
↪remove it from the Queue.
def peek(self):
  return self.head.data


#Removes all objects from the Queue.
def clear(self):
  self.head = None
  self.tail = None
```

```
    return self.__str__()


  # Returns a string that represents the Queue. (Inherited from Object)
  def display(self):
    return self.__str__()


  #Determines whether an element is in the Queue.
  def contains(self,value):
    if self.__size == 0: return False
    cur = self.head
    while cur != None:
      if cur.data == value:
        return True
      cur = cur.next
    return False

  # Copies the Queue to a new array.
  def toArray(self):
    out = []
    if self.__size != 0:
      cur = self.head
      while cur != None:
        out.append(cur.data)
        cur = cur.next
    return out
```

[ ]:
```
print('Test creating queue')
my_queue = Queue()
print(f'my_queue           : {my_queue}')
print(f'my_queue.enqueue(5): {my_queue.enqueue(5)}')
print(f'my_queue.enqueue(4): {my_queue.enqueue(4)}')
print(f'my_queue.enqueue(3): {my_queue.enqueue(3)}')
print(f'my_queue.enqueue(2): {my_queue.enqueue(2)}')
print(f'my_queue.enqueue(1): {my_queue.enqueue(1)}')
print(f'len(my_queue)      : {len(my_queue)}')
print()

print('Test pop() method')
print(f'my_queue          : {my_queue}')
print(f'my_queue.dequeue(): {my_queue.dequeue()}')
print(f'my_queue          : {my_queue}')
print()

print('Test peek')
print(f'my_queue       : {my_queue}')
```

```
print(f'my_queue.peek(): {my_queue.peek()}')
print(f'my_queue       : {my_queue}')
print()

print('Test contains()')
print(f'my_queue             : {my_queue}')
print(f'my_queue.contains(5): {my_queue.contains(5)}')
print(f'my_queue.contains(3): {my_queue.contains(3)}')
print()

print('Test toArray() method')
print(f'my_queue          : {my_queue}')
print(f'my_queue.toArray(): {my_queue.toArray()}')
print(f'my_queue          : {my_queue}')
print()

print('Test clear() method')
print(f'my_queue        : {my_queue}')
print(f'my_queue.clear(): {my_queue.clear()}')
print(f'my_queue        : {my_queue}')
print()
```

```
Test creating queue
my_queue           : None
my_queue.enqueue(5): 5->None
my_queue.enqueue(4): 5->4->None
my_queue.enqueue(3): 5->4->3->None
my_queue.enqueue(2): 5->4->3->2->None
my_queue.enqueue(1): 5->4->3->2->1->None
len(my_queue)      : 5

Test pop() method
my_queue          : 5->4->3->2->1->None
my_queue.dequeue(): 5
my_queue          : 4->3->2->1->None

Test peek
my_queue       : 4->3->2->1->None
my_queue.peek(): 4
my_queue       : 4->3->2->1->None

Test contains()
my_queue            : 4->3->2->1->None
my_queue.contains(5): False
my_queue.contains(3): True

Test toArray() method
my_queue          : 4->3->2->1->None
```

```
my_queue.toArray(): [4, 3, 2, 1]
my_queue          : 4->3->2->1->None

Test clear() method
my_queue        : 4->3->2->1->None
my_queue.clear(): None
my_queue        : None
```

1.4 Q4 (1 pt) Using Stacks/Queue

Using your Data Structure for Stacks or Queue write a code to reverse a string. Spaces on leading/tailing will be ignored.

Input: "GeeksQuiz" Output:"ziuQskeeG" <- Edited by Dan Knopp, I think you forgot to reverse the string since you had Output: "GeeksQuiz" ...

[ ]:
```
# Use a stack structure for refersing a string
string = "GeeksQuiz"

# Initialize a stack
reverse_char_stack = Stack()

# Add each character to the stack
print('initial stack: ', end='')
print(reverse_char_stack)
for char in string: reverse_char_stack.push(char)
print('filled  stack: ', end='')
print(reverse_char_stack)

# Get each character back out
reversed_string = ''.join([reverse_char_stack.pop() for char in string])
print('emptied stack: ', end='')
print(reverse_char_stack)
print()

# Return output reversed string
print(f'original string: {string} | reversed string using stack:␣
 ↪{reversed_string}')
```

```
initial stack: None
filled  stack: z->i->u->Q->s->k->e->e->G->None
emptied stack: None

original string: GeeksQuiz | reversed string using stack: ziuQskeeG
```

1.5 Q5 (1 pt) Using Stacks/Queue

Write a code to reverse a sentence using stack or queue. Given a sentence, reverse it. Spaces on leading/tailing will be ignored.

Input: “Geeks Quiz” Output: “Quiz Geeks”

[ ]:
```
# Use a stack structure for refersing a string
string = "Geeks Quiz"

# Initialize a stack
reverse_word_stack = Stack()

# Add each character to the stack
print('initial stack: ', end='')
print(reverse_word_stack)
for word in string.split(sep=' '): reverse_word_stack.push(word)
print('filled  stack: ', end='')
print(reverse_word_stack)

# Get each character back out
reversed_string = ' '.join([reverse_word_stack.pop() for word in string.
 ↪split(sep=' ')])
print('emptied stack: ', end='')
print(reverse_word_stack)
print()

# Return output reversed string
print(f'original string: {string} | reversed string using stack:␣
 ↪{reversed_string}')
```

```
initial stack: None
filled  stack: Quiz->Geeks->None
emptied stack: None

original string: Geeks Quiz | reversed string using stack: Quiz Geeks
```

1.6 Q6 (1 pt.) Using your DoubleLinkedList

Provide a code to fill out a DoubleLinkList

- Q6.1 Fill out with a standard 52-card deck. (https://en.wikipedia.org/wiki/Standard_52-card_deck). You can represent each card as a string.
- Q6.2 disply the deck on screen
- Q6.3 Shuflle the deck and display the result
- Q6.4 take the firt 12 cards, add them to an array and show the deck (remains cards)

[ ]:
```
# Initialize my ceck of cards
deck = DoubleLinkedList()
for suit in ['Spades', 'Hearts', 'Diamonds', 'Clubs']:
```

```
    for card in ['Ace', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack',␣
 ↪'Queen', 'King']:
        deck.append('_'.join([suit, card]))

# Print the unshuffled order
print('Start with an Unshuffled Deck:')
print()
print_doubly_linked_list(deck)
print()

# Shuffle the deck and print the new order
deck.shuffle()
print('Shuffle the Deck:')
print()
print_doubly_linked_list(deck)
print()

# Draw the first 12 cards and add them to your hand
hand = [deck.remove_at(0) for i in range(12)]

# Print your hand and the remaining cards in the deck
print('Draw 12 Cards from The Deck and Place into My Hand:')
print()
print('Hand: ', end='')
print(hand)
print()
print('Remaining Cards in the Deck:')
print()
print_doubly_linked_list(deck)
print()
```

```
Start with an Unshuffled Deck:

NodeID | node.prev.data  | node.data       | node.next.data
------------------------------------------------------------
Node 01: None           <- Spades_Ace     -> Spades_2
Node 02: Spades_Ace     <- Spades_2       -> Spades_3
Node 03: Spades_2       <- Spades_3       -> Spades_4
Node 04: Spades_3       <- Spades_4       -> Spades_5
Node 05: Spades_4       <- Spades_5       -> Spades_6
Node 06: Spades_5       <- Spades_6       -> Spades_7
Node 07: Spades_6       <- Spades_7       -> Spades_8
Node 08: Spades_7       <- Spades_8       -> Spades_9
Node 09: Spades_8       <- Spades_9       -> Spades_10
Node 10: Spades_9       <- Spades_10      -> Spades_Jack
Node 11: Spades_10      <- Spades_Jack    -> Spades_Queen
Node 12: Spades_Jack    <- Spades_Queen   -> Spades_King
Node 13: Spades_Queen   <- Spades_King    -> Hearts_Ace
```

```
Node 14: Spades_King    <- Hearts_Ace     -> Hearts_2
Node 15: Hearts_Ace     <- Hearts_2       -> Hearts_3
Node 16: Hearts_2       <- Hearts_3       -> Hearts_4
Node 17: Hearts_3       <- Hearts_4       -> Hearts_5
Node 18: Hearts_4       <- Hearts_5       -> Hearts_6
Node 19: Hearts_5       <- Hearts_6       -> Hearts_7
Node 20: Hearts_6       <- Hearts_7       -> Hearts_8
Node 21: Hearts_7       <- Hearts_8       -> Hearts_9
Node 22: Hearts_8       <- Hearts_9       -> Hearts_10
Node 23: Hearts_9       <- Hearts_10      -> Hearts_Jack
Node 24: Hearts_10      <- Hearts_Jack    -> Hearts_Queen
Node 25: Hearts_Jack    <- Hearts_Queen   -> Hearts_King
Node 26: Hearts_Queen   <- Hearts_King    -> Diamonds_Ace
Node 27: Hearts_King    <- Diamonds_Ace   -> Diamonds_2
Node 28: Diamonds_Ace   <- Diamonds_2     -> Diamonds_3
Node 29: Diamonds_2     <- Diamonds_3     -> Diamonds_4
Node 30: Diamonds_3     <- Diamonds_4     -> Diamonds_5
Node 31: Diamonds_4     <- Diamonds_5     -> Diamonds_6
Node 32: Diamonds_5     <- Diamonds_6     -> Diamonds_7
Node 33: Diamonds_6     <- Diamonds_7     -> Diamonds_8
Node 34: Diamonds_7     <- Diamonds_8     -> Diamonds_9
Node 35: Diamonds_8     <- Diamonds_9     -> Diamonds_10
Node 36: Diamonds_9     <- Diamonds_10    -> Diamonds_Jack
Node 37: Diamonds_10    <- Diamonds_Jack  -> Diamonds_Queen
Node 38: Diamonds_Jack  <- Diamonds_Queen -> Diamonds_King
Node 39: Diamonds_Queen <- Diamonds_King  -> Clubs_Ace
Node 40: Diamonds_King  <- Clubs_Ace      -> Clubs_2
Node 41: Clubs_Ace      <- Clubs_2        -> Clubs_3
Node 42: Clubs_2        <- Clubs_3        -> Clubs_4
Node 43: Clubs_3        <- Clubs_4        -> Clubs_5
Node 44: Clubs_4        <- Clubs_5        -> Clubs_6
Node 45: Clubs_5        <- Clubs_6        -> Clubs_7
Node 46: Clubs_6        <- Clubs_7        -> Clubs_8
Node 47: Clubs_7        <- Clubs_8        -> Clubs_9
Node 48: Clubs_8        <- Clubs_9        -> Clubs_10
Node 49: Clubs_9        <- Clubs_10       -> Clubs_Jack
Node 50: Clubs_10       <- Clubs_Jack     -> Clubs_Queen
Node 51: Clubs_Jack     <- Clubs_Queen    -> Clubs_King
Node 52: Clubs_Queen    <- Clubs_King     -> None

Shuffle the Deck:

NodeID | node.prev.data  | node.data       | node.next.data
-----------------------------------------------------------
Node 01: None           <- Diamonds_4     -> Hearts_9
Node 02: Diamonds_4     <- Hearts_9       -> Spades_3
Node 03: Hearts_9       <- Spades_3       -> Clubs_Jack
Node 04: Spades_3       <- Clubs_Jack     -> Hearts_Ace
```

```
Node 05: Clubs_Jack     <- Hearts_Ace     -> Diamonds_7
Node 06: Hearts_Ace     <- Diamonds_7     -> Diamonds_Ace
Node 07: Diamonds_7     <- Diamonds_Ace   -> Diamonds_10
Node 08: Diamonds_Ace   <- Diamonds_10    -> Hearts_Queen
Node 09: Diamonds_10    <- Hearts_Queen   -> Spades_7
Node 10: Hearts_Queen   <- Spades_7       -> Hearts_5
Node 11: Spades_7       <- Hearts_5       -> Clubs_5
Node 12: Hearts_5       <- Clubs_5        -> Clubs_10
Node 13: Clubs_5        <- Clubs_10       -> Clubs_2
Node 14: Clubs_10       <- Clubs_2        -> Clubs_7
Node 15: Clubs_2        <- Clubs_7        -> Diamonds_Jack
Node 16: Clubs_7        <- Diamonds_Jack  -> Hearts_7
Node 17: Diamonds_Jack  <- Hearts_7       -> Spades_Jack
Node 18: Hearts_7       <- Spades_Jack    -> Diamonds_5
Node 19: Spades_Jack    <- Diamonds_5     -> Clubs_6
Node 20: Diamonds_5     <- Clubs_6        -> Clubs_9
Node 21: Clubs_6        <- Clubs_9        -> Diamonds_8
Node 22: Clubs_9        <- Diamonds_8     -> Spades_10
Node 23: Diamonds_8     <- Spades_10      -> Clubs_8
Node 24: Spades_10      <- Clubs_8        -> Spades_2
Node 25: Clubs_8        <- Spades_2       -> Diamonds_3
Node 26: Spades_2       <- Diamonds_3     -> Clubs_Ace
Node 27: Diamonds_3     <- Clubs_Ace      -> Hearts_Jack
Node 28: Clubs_Ace      <- Hearts_Jack    -> Diamonds_6
Node 29: Hearts_Jack    <- Diamonds_6     -> Hearts_10
Node 30: Diamonds_6     <- Hearts_10      -> Clubs_King
Node 31: Hearts_10      <- Clubs_King     -> Diamonds_2
Node 32: Clubs_King     <- Diamonds_2     -> Hearts_4
Node 33: Diamonds_2     <- Hearts_4       -> Spades_Queen
Node 34: Hearts_4       <- Spades_Queen   -> Clubs_Queen
Node 35: Spades_Queen   <- Clubs_Queen    -> Hearts_6
Node 36: Clubs_Queen    <- Hearts_6       -> Spades_9
Node 37: Hearts_6       <- Spades_9       -> Hearts_King
Node 38: Spades_9       <- Hearts_King    -> Spades_King
Node 39: Hearts_King    <- Spades_King    -> Spades_4
Node 40: Spades_King    <- Spades_4       -> Clubs_4
Node 41: Spades_4       <- Clubs_4        -> Diamonds_King
Node 42: Clubs_4        <- Diamonds_King  -> Spades_6
Node 43: Diamonds_King  <- Spades_6       -> Spades_8
Node 44: Spades_6       <- Spades_8       -> Clubs_3
Node 45: Spades_8       <- Clubs_3        -> Diamonds_9
Node 46: Clubs_3        <- Diamonds_9     -> Hearts_8
Node 47: Diamonds_9     <- Hearts_8       -> Spades_Ace
Node 48: Hearts_8       <- Spades_Ace     -> Hearts_3
Node 49: Spades_Ace     <- Hearts_3       -> Diamonds_Queen
Node 50: Hearts_3       <- Diamonds_Queen -> Spades_5
Node 51: Diamonds_Queen <- Spades_5       -> Hearts_2
Node 52: Spades_5       <- Hearts_2       -> None
```

```
Draw 12 Cards from The Deck and Place into My Hand:

Hand: ['Diamonds_4', 'Hearts_9', 'Spades_3', 'Clubs_Jack', 'Hearts_Ace',
'Diamonds_7', 'Diamonds_Ace', 'Diamonds_10', 'Hearts_Queen', 'Spades_7',
'Hearts_5', 'Clubs_5']

Remaining Cards in the Deck:

NodeID | node.prev.data  | node.data       | node.next.data
-------------------------------------------------------------
Node 01: None           <- Clubs_10       -> Clubs_2
Node 02: Clubs_10       <- Clubs_2        -> Clubs_7
Node 03: Clubs_2        <- Clubs_7        -> Diamonds_Jack
Node 04: Clubs_7        <- Diamonds_Jack  -> Hearts_7
Node 05: Diamonds_Jack  <- Hearts_7       -> Spades_Jack
Node 06: Hearts_7       <- Spades_Jack    -> Diamonds_5
Node 07: Spades_Jack    <- Diamonds_5     -> Clubs_6
Node 08: Diamonds_5     <- Clubs_6        -> Clubs_9
Node 09: Clubs_6        <- Clubs_9        -> Diamonds_8
Node 10: Clubs_9        <- Diamonds_8     -> Spades_10
Node 11: Diamonds_8     <- Spades_10      -> Clubs_8
Node 12: Spades_10      <- Clubs_8        -> Spades_2
Node 13: Clubs_8        <- Spades_2       -> Diamonds_3
Node 14: Spades_2       <- Diamonds_3     -> Clubs_Ace
Node 15: Diamonds_3     <- Clubs_Ace      -> Hearts_Jack
Node 16: Clubs_Ace      <- Hearts_Jack    -> Diamonds_6
Node 17: Hearts_Jack    <- Diamonds_6     -> Hearts_10
Node 18: Diamonds_6     <- Hearts_10      -> Clubs_King
Node 19: Hearts_10      <- Clubs_King     -> Diamonds_2
Node 20: Clubs_King     <- Diamonds_2     -> Hearts_4
Node 21: Diamonds_2     <- Hearts_4       -> Spades_Queen
Node 22: Hearts_4       <- Spades_Queen   -> Clubs_Queen
Node 23: Spades_Queen   <- Clubs_Queen    -> Hearts_6
Node 24: Clubs_Queen    <- Hearts_6       -> Spades_9
Node 25: Hearts_6       <- Spades_9       -> Hearts_King
Node 26: Spades_9       <- Hearts_King    -> Spades_King
Node 27: Hearts_King    <- Spades_King    -> Spades_4
Node 28: Spades_King    <- Spades_4       -> Clubs_4
Node 29: Spades_4       <- Clubs_4        -> Diamonds_King
Node 30: Clubs_4        <- Diamonds_King  -> Spades_6
Node 31: Diamonds_King  <- Spades_6       -> Spades_8
Node 32: Spades_6       <- Spades_8       -> Clubs_3
Node 33: Spades_8       <- Clubs_3        -> Diamonds_9
Node 34: Clubs_3        <- Diamonds_9     -> Hearts_8
Node 35: Diamonds_9     <- Hearts_8       -> Spades_Ace
Node 36: Hearts_8       <- Spades_Ace     -> Hearts_3
Node 37: Spades_Ace     <- Hearts_3       -> Diamonds_Queen
```

```
Node 38: Hearts_3       <- Diamonds_Queen -> Spades_5
Node 39: Diamonds_Queen <- Spades_5       -> Hearts_2
Node 40: Spades_5       <- Hearts_2       -> None
```

1.7 Q7 (1 pt) Implement a Queue using two Stacks

This is a tipical interview question for a SWE/DS position.

You must to use your stack data structure implementation to solve the implementation of a new Queue.

- Q7.1 describe your approach (2-3 paragraphs)
- Q7.2 Write your code.

Q7.1 your response here

I will be creating a NewQueue class that has 2 internal stacks. My approach is to first store new items into the 1st stack, then strategically move the items from the 1st stack into a 2nd stack before outputting values. By using 2 stacks like this, the resulting 2nd stack will have the same order as if I had used only 1 queue and I can pop() items from the 2nd stack to get the order as if I had only used a single Queue. I can use all the existing methods in the Stack() class from above to my advantage within the NewQueue() methods below.

The way I will implement the double stack queue is that I will only transfer values from stack_1 to stack_2 when either the dequeue() method or the peek() method is called and if stack_2 is empty. This helps with efficiency because I don't need to transfer nodes between stacks (to reverse the order) until the user wants to get an item from the queue and the 2nd stack has been emptied - meaning I don't need to manage the order every time information is added, but only when information is extracted. For other methods like printing, constains, and toArray, I can utilize the already defined Stack() methods but then I must strategically merge the results for my NewQueue (see methods below for details).

[ ]:
```python
#Q7.2

class NewQueue:
  def __init__(self):
    self.stack_1 = Stack()
    self.stack_2 = Stack()

  # Overwrite the default len() method
  def __len__(self):
    return len(self.stack_1) + len(self.stack_2)

  # Overwrite the default print() method
  def __str__(self):
    if len(self.stack_1) == 0 and len(self.stack_2) == 0: return 'None'
```

```
    return '->'.join([i for i in self.stack_2.__str__().split(sep='->') if i !=␣
↪'None'] + [i for i in self.stack_1.__str__().split(sep='->') if i !=␣
↪'None'][::-1]) # gets numeric values from each stack __str__() methods and␣
↪put thems together with stack_2 values first, then reversed stack_1 values,␣
↪and joins the elements with '->' delimiter

  # Adds a new element to the queue (reordering magic happens in dequeue() when␣
↪removing items)
  def enqueue(self, data):
    self.stack_1.push(data)
    return self.__str__()

  # Returns the value of the next element in the queue and removes the item
  def dequeue(self):
    if len(self.stack_2) == 0:                      # if the 2nd stack is␣
↪currently empty, need to transfer them from stack 1 to stack 2
      if len(self.stack_1) == 0:                      # if there are nodes the␣
↪1st stack, can't do any transferring
        return 'Error, queue is empty'                  # return an error that␣
↪the queue is empty
      while len(self.stack_1) > 0:                  # loop until stack 1 is empty
        self.stack_2.push(self.stack_1.pop())           # transfer each node␣
↪currently in stack 1 to stack 2 (naturally reversing the order in the␣
↪process)
    return self.stack_2.pop()                       # pop the top node from␣
↪stack 2

  # Returns the value of the next element in the queue
  def peek(self):
    if len(self.stack_2) == 0:                      # if the 2nd stack is␣
↪currently empty, need to transfer them from stack 1 to stack 2
      if len(self.stack_1) == 0:                      # if there are nodes the␣
↪1st stack, can't do any transferring
        return 'Error, queue is empty'                  # return an error that␣
↪the queue is empty
      while len(self.stack_1) > 0:                  # loop until stack 1 is empty
        self.stack_2.push(self.stack_1.pop())           # transfer each node␣
↪currently in stack 1 to stack 2 (naturally reversing the order in the␣
↪process)
    return self.stack_2.peek()                      # peek top node from stack 2

  # Removes all objects from the Queue.
  def clear(self):
    self.stack_1.clear()
    self.stack_2.clear()
    return self.__str__()
```

```
  # Returns a string that represents the Queue. (Inherited from Object)
  def display(self):
    return self.__str__()

  #Determines whether an element is in the Queue.
  def contains(self, value):
    return self.stack_1.contains(value) or self.stack_2.contains(value)

  # Copies the Queue to a new array.
  def toArray(self):
    return self.stack_2.toArray() + self.stack_1.toArray()[::-1]
```

[ ]:
```
# Test the Queue
my_queue_from_stacks = NewQueue()

print('Test enqueue() method')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
for i in range(1, 6):
    print(f'my_queue_from_stacks.enqueue({i}): {my_queue_from_stacks.
 ↪enqueue(i)}')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print(f'len(my_queue_from_stacks): {len(my_queue_from_stacks)}')
print()

print('Test contains() method')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
for i in range(7):
    print(f'my_queue_from_stacks.contains({i}): {my_queue_from_stacks.
 ↪contains(i)}')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print()

print('Test peek() method')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print('my_queue_from_stacks.peek(): ', end='')
print(my_queue_from_stacks.peek())
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print()

print('Test toArray() method')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print('my_queue_from_stacks.toArray(): ', end='')
print(my_queue_from_stacks.toArray())
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print()
```

```python
print('Test dequeue() method')
for i in range(3):
    print(f'my_queue_from_stacks: {my_queue_from_stacks}')
    print(f'my_queue_from_stacks.dequeue(): {my_queue_from_stacks.dequeue()}')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print()

print('Test clear() method')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print(f'my_queue_from_stacks.clear(): {my_queue_from_stacks.clear()}')
print(f'my_queue_from_stacks: {my_queue_from_stacks}')
print(f'len(my_queue_from_stacks): {len(my_queue_from_stacks)}')
print()
```

```
Test enqueue() method
my_queue_from_stacks: None
my_queue_from_stacks.enqueue(1): 1
my_queue_from_stacks.enqueue(2): 1->2
my_queue_from_stacks.enqueue(3): 1->2->3
my_queue_from_stacks.enqueue(4): 1->2->3->4
my_queue_from_stacks.enqueue(5): 1->2->3->4->5
my_queue_from_stacks: 1->2->3->4->5
len(my_queue_from_stacks): 5

Test contains() method
my_queue_from_stacks: 1->2->3->4->5
my_queue_from_stacks.contains(0): False
my_queue_from_stacks.contains(1): True
my_queue_from_stacks.contains(2): True
my_queue_from_stacks.contains(3): True
my_queue_from_stacks.contains(4): True
my_queue_from_stacks.contains(5): True
my_queue_from_stacks.contains(6): False
my_queue_from_stacks: 1->2->3->4->5

Test peek() method
my_queue_from_stacks: 1->2->3->4->5
my_queue_from_stacks.peek(): 1
my_queue_from_stacks: 1->2->3->4->5

Test toArray() method
my_queue_from_stacks: 1->2->3->4->5
my_queue_from_stacks.toArray(): [1, 2, 3, 4, 5]
my_queue_from_stacks: 1->2->3->4->5

Test dequeue() method
my_queue_from_stacks: 1->2->3->4->5
my_queue_from_stacks.dequeue(): 1
```

```
my_queue_from_stacks: 2->3->4->5
my_queue_from_stacks.dequeue(): 2
my_queue_from_stacks: 3->4->5
my_queue_from_stacks.dequeue(): 3
my_queue_from_stacks: 4->5

Test clear() method
my_queue_from_stacks: 4->5
my_queue_from_stacks.clear(): None
my_queue_from_stacks: None
len(my_queue_from_stacks): 0
```

1.8 Q8 (1 pt) Extra Bonus... Music Library (Playlist - Infinity loop)

Problem: * Implement a Song class with attributes Title, Duration, Artist, Album

- Implement a circular linked list.
- Create a playlist (no less than 10 songs)
- Simulate a process to play the list (loop) of this playlist by hours long. Diplay the current song title, artist, album.
- The play list will stop when reach the limit time (2 hours of music time
- You **must to use a circular linked list** to solve your problem.
- You are no allow to alter the order (no shuffle).

Questions:

- Q81. Describe your solution (3-4 paragraphs)
- Q8.2 write your code for the data structure (song and playlist structure)
- Q8.3 write the code to play the Playlist for a 2 hours

Note: Modify the time scale to define one minute of simulation equal to one hour of playing music.

Q8.1 Response Here

For this problem I decided to reuse the DoubleLinkedList from question 1 as a super class and create a new subclass. I had to make adjustments to the above class so that the methods were general enough to accomodate the potential of a cirtular linked list (those edits were done in-place in question 1, and I made sure everything in question 1 also still worked as originally intended). Once I generalized the DoubleLinkedList enough, I then had to make some adjustments for the new Playlist class below. The primary adjustmeht is just to link up the head and tail nodes in the list so that they point to each other. To do this, I created a new method called make_circular() which only exists in the Playlist subclass. I then adjusted all the inherited classes such that they call this new method after each execution so that I will always have a circularly linked list even though I am reusing the methods from my DoublyLinkedList super class. I defined a Song class below becuase the question said to, but because I am creating a subclass of my super class in question 1, I am in reality using the Node class that was defined earlier. Since the DoublyLinkedList class in question 1 expects a single object for the data stored at each node, I decided that I would store all the data

for each song as a dictionary object. That way all my previously defined methods would still be compatible even though each node has multiple pieces of data in the Playlist class.

To setup my song data, I just gave very generic names like 'artist01', 'album01', and 'song01' and to avoid confusion between similar song and album names, I made each song name a combination of 'artistID_albumID_songID'. Unfortunately this makes the output in the simulation a bit redundant, but it adds clarity when I visualize the contents of the Playlist using my helper function so I decided it was better to keep this format.

For the simulation, I run a while loop in realtime and keep a constant-time timestep of around 1s. The time scale of 1min realtime = 1hr simtime is handled by a simple conversion factor from elapsed realtime. Each timestep, I check if the previously running song has finished and if it has I play the next song in the list. Shown below are the details of how I track this and adjust for when a song ends in between timesteps (see comments in code below).

[ ]:
```
#Q8.2 your code Here

# I won't actually use the Song class because it is identical to the Node()␣
 ↪class from question 1, but I put it here because the question requests this␣
 ↪class be defined
class Song(Node):
    def __init__(self):
        super().__init__()

# Instead of redefining the doubly linked list from scratch, I'm using the␣
 ↪class from question 1 and only modifying what I need to to make it circular
class Playlist(DoubleLinkedList):

    def display(self, key):
        if len(self) == 0: return 'None'
        cur    = self.head
        output = []
        for i in range(len(self)):
            output.append(f"{cur.prev.data[key]}<-{cur.data[key]}->{cur.next.
 ↪data[key]}")
            cur = cur.next
        return '|'.join(output)

    def make_curcular(self):
        self.head.prev = self.tail
        self.tail.next = self.head

    def push(self, data):
        out = super().push(data)
        self.make_curcular()
        return out

    def append(self, data):
```

```
        out = super().append(data)
        self.make_curcular()
        return out

    def sort(self):
        out = super().sort()
        self.make_curcular()
        return out

    def reverse(self):
        out = super().reverse()
        self.make_curcular()
        return out

    def remove_at(self, idx):
        out = super().remove_at(idx)
        self.make_curcular()
        return out

    def insert_at(self, idx, data):
        out = super().insert_at(idx, data)
        self.make_curcular()
        return out

    def shuffle(self):
        out = super().shuffle()
        self.make_curcular()
        return out
```

[ ]:
```
# Create a helper function to make printing doubly linked lists easier to␣
↪visualize, modified for dictionary data
def print_doubly_linked_list_dict(my_list, key):
    node_prev = []
    node_curr = []
    node_next = []
    for i, node in enumerate(str(my_list.display(key)).split(sep='|')):
        prv, cur, nxt = node.replace('<', '').replace('>', '').split(sep="-")
        node_prev.append(prv)
        node_curr.append(cur)
        node_next.append(nxt)

    max_num_char = max([len(s) for s in node_prev] + [len(s) for s in␣
↪node_curr] + [len(s) for s in node_next] + [len(f'node.prev.data[{key}]')])

    lines = [f'{"NodeID":<6} | {f"node.prev.data[{key}]":<{max_num_char}}  |␣
↪{f"node.data[{key}]":<{max_num_char}}  | {f"node.next.data[{key}]":
↪<{max_num_char}}']
```

```
    lines.append('-'*(6 + 3 + 4 + 4 + max_num_char*3))

    for i in range(len(node_prev)):
        lines.append(f'Node {i + 1:02d}: {node_prev[i]:<{max_num_char}} <-␣
 ↪{node_curr[i]:<{max_num_char}} -> {node_next[i]:<{max_num_char}}')

    print('\n'.join(lines))
```

[ ]:
```
# Define generic names for artists, albums, and songs
artists = [f'artist{i:02d}' for i in range(1,4)]
albums  = [f'album{ i:02d}' for i in range(1,3)]
songs   = [f'song{  i:02d}' for i in range(1,7)]

# Create a list of dictionaries containing all playlist data
playlist_data = [{'title'   : '_'.join([artist, album, song]),
                  'album'   : '_'.join([artist, album      ]),
                  'artist'  : '_'.join([artist             ]),
                  'duration': round(random.random()*3 + 1, 2),} for artist in␣
 ↪artists for album in albums for song in songs]

# Create the playlist from the data
my_playlist = Playlist()
for data in playlist_data:
    my_playlist.append(data)

# Show playlist data (only title and duration to save space, but just need to␣
 ↪change the key below to show any data for each song in the playlist)
print('Show all song titles:')
print()
print_doubly_linked_list_dict(my_playlist, 'title')
print()
print('Show all song durations')
print()
print_doubly_linked_list_dict(my_playlist, 'duration')
```

```
Show all song titles:

NodeID | node.prev.data[title]    | node.data[title]         |
node.next.data[title]
-------------------------------------------------------------------------------
------
Node 01: artist03_album02_song06 <- artist01_album01_song01 ->
artist01_album01_song02
Node 02: artist01_album01_song01 <- artist01_album01_song02 ->
artist01_album01_song03
Node 03: artist01_album01_song02 <- artist01_album01_song03 ->
artist01_album01_song04
```

```
Node 04: artist01_album01_song03 <- artist01_album01_song04 ->
artist01_album01_song05
Node 05: artist01_album01_song04 <- artist01_album01_song05 ->
artist01_album01_song06
Node 06: artist01_album01_song05 <- artist01_album01_song06 ->
artist01_album02_song01
Node 07: artist01_album01_song06 <- artist01_album02_song01 ->
artist01_album02_song02
Node 08: artist01_album02_song01 <- artist01_album02_song02 ->
artist01_album02_song03
Node 09: artist01_album02_song02 <- artist01_album02_song03 ->
artist01_album02_song04
Node 10: artist01_album02_song03 <- artist01_album02_song04 ->
artist01_album02_song05
Node 11: artist01_album02_song04 <- artist01_album02_song05 ->
artist01_album02_song06
Node 12: artist01_album02_song05 <- artist01_album02_song06 ->
artist02_album01_song01
Node 13: artist01_album02_song06 <- artist02_album01_song01 ->
artist02_album01_song02
Node 14: artist02_album01_song01 <- artist02_album01_song02 ->
artist02_album01_song03
Node 15: artist02_album01_song02 <- artist02_album01_song03 ->
artist02_album01_song04
Node 16: artist02_album01_song03 <- artist02_album01_song04 ->
artist02_album01_song05
Node 17: artist02_album01_song04 <- artist02_album01_song05 ->
artist02_album01_song06
Node 18: artist02_album01_song05 <- artist02_album01_song06 ->
artist02_album02_song01
Node 19: artist02_album01_song06 <- artist02_album02_song01 ->
artist02_album02_song02
Node 20: artist02_album02_song01 <- artist02_album02_song02 ->
artist02_album02_song03
Node 21: artist02_album02_song02 <- artist02_album02_song03 ->
artist02_album02_song04
Node 22: artist02_album02_song03 <- artist02_album02_song04 ->
artist02_album02_song05
Node 23: artist02_album02_song04 <- artist02_album02_song05 ->
artist02_album02_song06
Node 24: artist02_album02_song05 <- artist02_album02_song06 ->
artist03_album01_song01
Node 25: artist02_album02_song06 <- artist03_album01_song01 ->
artist03_album01_song02
Node 26: artist03_album01_song01 <- artist03_album01_song02 ->
artist03_album01_song03
Node 27: artist03_album01_song02 <- artist03_album01_song03 ->
artist03_album01_song04
```

```
Node 28: artist03_album01_song03 <- artist03_album01_song04 ->
artist03_album01_song05
Node 29: artist03_album01_song04 <- artist03_album01_song05 ->
artist03_album01_song06
Node 30: artist03_album01_song05 <- artist03_album01_song06 ->
artist03_album02_song01
Node 31: artist03_album01_song06 <- artist03_album02_song01 ->
artist03_album02_song02
Node 32: artist03_album02_song01 <- artist03_album02_song02 ->
artist03_album02_song03
Node 33: artist03_album02_song02 <- artist03_album02_song03 ->
artist03_album02_song04
Node 34: artist03_album02_song03 <- artist03_album02_song04 ->
artist03_album02_song05
Node 35: artist03_album02_song04 <- artist03_album02_song05 ->
artist03_album02_song06
Node 36: artist03_album02_song05 <- artist03_album02_song06 ->
artist01_album01_song01

Show all song durations

NodeID | node.prev.data[duration]  | node.data[duration]       |
node.next.data[duration]
-------------------------------------------------------------------------------
---------
Node 01: 2.31                      <- 3.69                     -> 2.81
Node 02: 3.69                      <- 2.81                     -> 1.29
Node 03: 2.81                      <- 1.29                     -> 1.11
Node 04: 1.29                      <- 1.11                     -> 2.7
Node 05: 1.11                      <- 2.7                      -> 3.6
Node 06: 2.7                       <- 3.6                      -> 1.5
Node 07: 3.6                       <- 1.5                      -> 3.88
Node 08: 1.5                       <- 3.88                     -> 3.86
Node 09: 3.88                      <- 3.86                     -> 2.67
Node 10: 3.86                      <- 2.67                     -> 2.09
Node 11: 2.67                      <- 2.09                     -> 2.39
Node 12: 2.09                      <- 2.39                     -> 1.04
Node 13: 2.39                      <- 1.04                     -> 1.95
Node 14: 1.04                      <- 1.95                     -> 1.99
Node 15: 1.95                      <- 1.99                     -> 3.07
Node 16: 1.99                      <- 3.07                     -> 1.96
Node 17: 3.07                      <- 1.96                     -> 2.9
Node 18: 1.96                      <- 2.9                      -> 1.2
Node 19: 2.9                       <- 1.2                      -> 2.68
Node 20: 1.2                       <- 2.68                     -> 1.78
Node 21: 2.68                      <- 1.78                     -> 3.8
Node 22: 1.78                      <- 3.8                      -> 1.24
Node 23: 3.8                       <- 1.24                     -> 2.58
```

```
Node 24: 1.24                   <- 2.58                   -> 1.76
Node 25: 2.58                   <- 1.76                   -> 3.75
Node 26: 1.76                   <- 3.75                   -> 1.31
Node 27: 3.75                   <- 1.31                   -> 3.52
Node 28: 1.31                   <- 3.52                   -> 3.67
Node 29: 3.52                   <- 3.67                   -> 1.13
Node 30: 3.67                   <- 1.13                   -> 2.55
Node 31: 1.13                   <- 2.55                   -> 1.16
Node 32: 2.55                   <- 1.16                   -> 2.24
Node 33: 1.16                   <- 2.24                   -> 2.8
Node 34: 2.24                   <- 2.8                    -> 3.69
Node 35: 2.8                    <- 3.69                   -> 2.31
Node 36: 3.69                   <- 2.31                   -> 3.69
```

[ ]:
```python
#Q8.3 Your Code Here

# Initialize variables
timestep         = 1
duration         = 2 * 3600
elapsed_realtime = 0
elapsed_simtime  = 0
current_song     = my_playlist.head
song_endtime     = elapsed_simtime + current_song.data['duration']*60
start_time       = time.time()

# Loop until reach the end of the simulation
while elapsed_simtime < duration:

    # Compute the elapsed realtime and convert it to simtime by factor␣
 ↪specified in above question prompt
    elapsed_realtime = time.time() - start_time
    elapsed_simtime  = elapsed_realtime * 60

    # If the previous song has finished playing
    if elapsed_simtime > song_endtime:

        # Compute how long it has been since the song finished (songs typically␣
 ↪won't finish exactly on a simulation timestep)
        offset = elapsed_simtime - song_endtime

        # Switch to the next song
        current_song = current_song.next

        # Compute the new song endtime
        song_endtime = elapsed_simtime - offset + current_song.
 ↪data['duration']*60
```

```
    # Display information for the user
    print(f"RealTime = {round(elapsed_realtime):>3}s, SimTime =␣
↪{round(elapsed_simtime/60,1):>6.2f}min ({round(elapsed_simtime/3600,2):>4.
↪2f}hr)  |  Now Playing: {current_song.data['title']} from {current_song.
↪data['album']} by {current_song.data['artist']} (dur: {current_song.
↪data['duration']}m)")

    # Sleep the remaining time in the current timestep
    time.sleep(timestep - elapsed_realtime % timestep)
```

```
RealTime =   0s, SimTime =   0.00min (0.00hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =   1s, SimTime =   1.00min (0.02hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =   2s, SimTime =   2.00min (0.03hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =   3s, SimTime =   3.00min (0.05hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =   4s, SimTime =   4.00min (0.07hr)  |  Now Playing:
artist01_album01_song02 from artist01_album01 by artist01 (dur: 2.81m)
RealTime =   5s, SimTime =   5.00min (0.08hr)  |  Now Playing:
artist01_album01_song02 from artist01_album01 by artist01 (dur: 2.81m)
RealTime =   6s, SimTime =   6.00min (0.10hr)  |  Now Playing:
artist01_album01_song02 from artist01_album01 by artist01 (dur: 2.81m)
RealTime =   7s, SimTime =   7.00min (0.12hr)  |  Now Playing:
artist01_album01_song03 from artist01_album01 by artist01 (dur: 1.29m)
RealTime =   8s, SimTime =   8.00min (0.13hr)  |  Now Playing:
artist01_album01_song04 from artist01_album01 by artist01 (dur: 1.11m)
RealTime =   9s, SimTime =   9.00min (0.15hr)  |  Now Playing:
artist01_album01_song05 from artist01_album01 by artist01 (dur: 2.7m)
RealTime =  10s, SimTime =  10.00min (0.17hr)  |  Now Playing:
artist01_album01_song05 from artist01_album01 by artist01 (dur: 2.7m)
RealTime =  11s, SimTime =  11.00min (0.18hr)  |  Now Playing:
artist01_album01_song05 from artist01_album01 by artist01 (dur: 2.7m)
RealTime =  12s, SimTime =  12.00min (0.20hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime =  13s, SimTime =  13.00min (0.22hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime =  14s, SimTime =  14.00min (0.23hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime =  15s, SimTime =  15.00min (0.25hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime =  16s, SimTime =  16.00min (0.27hr)  |  Now Playing:
artist01_album02_song01 from artist01_album02 by artist01 (dur: 1.5m)
RealTime =  17s, SimTime =  17.00min (0.28hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime =  18s, SimTime =  18.00min (0.30hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
```

```
RealTime =  19s, SimTime =  19.00min (0.32hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime =  20s, SimTime =  20.00min (0.33hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime =  21s, SimTime =  21.00min (0.35hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime =  22s, SimTime =  22.00min (0.37hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime =  23s, SimTime =  23.00min (0.38hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime =  24s, SimTime =  24.00min (0.40hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime =  25s, SimTime =  25.00min (0.42hr)  |  Now Playing:
artist01_album02_song04 from artist01_album02 by artist01 (dur: 2.67m)
RealTime =  26s, SimTime =  26.00min (0.43hr)  |  Now Playing:
artist01_album02_song04 from artist01_album02 by artist01 (dur: 2.67m)
RealTime =  27s, SimTime =  27.00min (0.45hr)  |  Now Playing:
artist01_album02_song04 from artist01_album02 by artist01 (dur: 2.67m)
RealTime =  28s, SimTime =  28.00min (0.47hr)  |  Now Playing:
artist01_album02_song05 from artist01_album02 by artist01 (dur: 2.09m)
RealTime =  29s, SimTime =  29.00min (0.48hr)  |  Now Playing:
artist01_album02_song05 from artist01_album02 by artist01 (dur: 2.09m)
RealTime =  30s, SimTime =  30.00min (0.50hr)  |  Now Playing:
artist01_album02_song06 from artist01_album02 by artist01 (dur: 2.39m)
RealTime =  31s, SimTime =  31.00min (0.52hr)  |  Now Playing:
artist01_album02_song06 from artist01_album02 by artist01 (dur: 2.39m)
RealTime =  32s, SimTime =  32.00min (0.53hr)  |  Now Playing:
artist02_album01_song01 from artist02_album01 by artist02 (dur: 1.04m)
RealTime =  33s, SimTime =  33.00min (0.55hr)  |  Now Playing:
artist02_album01_song02 from artist02_album01 by artist02 (dur: 1.95m)
RealTime =  34s, SimTime =  34.00min (0.57hr)  |  Now Playing:
artist02_album01_song02 from artist02_album01 by artist02 (dur: 1.95m)
RealTime =  35s, SimTime =  35.00min (0.58hr)  |  Now Playing:
artist02_album01_song03 from artist02_album01 by artist02 (dur: 1.99m)
RealTime =  36s, SimTime =  36.00min (0.60hr)  |  Now Playing:
artist02_album01_song03 from artist02_album01 by artist02 (dur: 1.99m)
RealTime =  37s, SimTime =  37.00min (0.62hr)  |  Now Playing:
artist02_album01_song04 from artist02_album01 by artist02 (dur: 3.07m)
RealTime =  38s, SimTime =  38.00min (0.63hr)  |  Now Playing:
artist02_album01_song04 from artist02_album01 by artist02 (dur: 3.07m)
RealTime =  39s, SimTime =  39.00min (0.65hr)  |  Now Playing:
artist02_album01_song04 from artist02_album01 by artist02 (dur: 3.07m)
RealTime =  40s, SimTime =  40.00min (0.67hr)  |  Now Playing:
artist02_album01_song05 from artist02_album01 by artist02 (dur: 1.96m)
RealTime =  41s, SimTime =  41.00min (0.68hr)  |  Now Playing:
artist02_album01_song05 from artist02_album01 by artist02 (dur: 1.96m)
RealTime =  42s, SimTime =  42.00min (0.70hr)  |  Now Playing:
artist02_album01_song06 from artist02_album01 by artist02 (dur: 2.9m)
```

```
RealTime =  43s, SimTime =  43.00min (0.72hr)  |  Now Playing:
artist02_album01_song06 from artist02_album01 by artist02 (dur: 2.9m)
RealTime =  44s, SimTime =  44.00min (0.73hr)  |  Now Playing:
artist02_album01_song06 from artist02_album01 by artist02 (dur: 2.9m)
RealTime =  45s, SimTime =  45.00min (0.75hr)  |  Now Playing:
artist02_album02_song01 from artist02_album02 by artist02 (dur: 1.2m)
RealTime =  46s, SimTime =  46.00min (0.77hr)  |  Now Playing:
artist02_album02_song02 from artist02_album02 by artist02 (dur: 2.68m)
RealTime =  47s, SimTime =  47.00min (0.78hr)  |  Now Playing:
artist02_album02_song02 from artist02_album02 by artist02 (dur: 2.68m)
RealTime =  48s, SimTime =  48.00min (0.80hr)  |  Now Playing:
artist02_album02_song02 from artist02_album02 by artist02 (dur: 2.68m)
RealTime =  49s, SimTime =  49.00min (0.82hr)  |  Now Playing:
artist02_album02_song03 from artist02_album02 by artist02 (dur: 1.78m)
RealTime =  50s, SimTime =  50.00min (0.83hr)  |  Now Playing:
artist02_album02_song03 from artist02_album02 by artist02 (dur: 1.78m)
RealTime =  51s, SimTime =  51.00min (0.85hr)  |  Now Playing:
artist02_album02_song04 from artist02_album02 by artist02 (dur: 3.8m)
RealTime =  52s, SimTime =  52.00min (0.87hr)  |  Now Playing:
artist02_album02_song04 from artist02_album02 by artist02 (dur: 3.8m)
RealTime =  53s, SimTime =  53.00min (0.88hr)  |  Now Playing:
artist02_album02_song04 from artist02_album02 by artist02 (dur: 3.8m)
RealTime =  54s, SimTime =  54.00min (0.90hr)  |  Now Playing:
artist02_album02_song05 from artist02_album02 by artist02 (dur: 1.24m)
RealTime =  55s, SimTime =  55.00min (0.92hr)  |  Now Playing:
artist02_album02_song05 from artist02_album02 by artist02 (dur: 1.24m)
RealTime =  56s, SimTime =  56.00min (0.93hr)  |  Now Playing:
artist02_album02_song06 from artist02_album02 by artist02 (dur: 2.58m)
RealTime =  57s, SimTime =  57.00min (0.95hr)  |  Now Playing:
artist02_album02_song06 from artist02_album02 by artist02 (dur: 2.58m)
RealTime =  58s, SimTime =  58.00min (0.97hr)  |  Now Playing:
artist03_album01_song01 from artist03_album01 by artist03 (dur: 1.76m)
RealTime =  59s, SimTime =  59.00min (0.98hr)  |  Now Playing:
artist03_album01_song01 from artist03_album01 by artist03 (dur: 1.76m)
RealTime =  60s, SimTime =  60.00min (1.00hr)  |  Now Playing:
artist03_album01_song02 from artist03_album01 by artist03 (dur: 3.75m)
RealTime =  61s, SimTime =  61.00min (1.02hr)  |  Now Playing:
artist03_album01_song02 from artist03_album01 by artist03 (dur: 3.75m)
RealTime =  62s, SimTime =  62.00min (1.03hr)  |  Now Playing:
artist03_album01_song02 from artist03_album01 by artist03 (dur: 3.75m)
RealTime =  63s, SimTime =  63.00min (1.05hr)  |  Now Playing:
artist03_album01_song02 from artist03_album01 by artist03 (dur: 3.75m)
RealTime =  64s, SimTime =  64.00min (1.07hr)  |  Now Playing:
artist03_album01_song03 from artist03_album01 by artist03 (dur: 1.31m)
RealTime =  65s, SimTime =  65.00min (1.08hr)  |  Now Playing:
artist03_album01_song04 from artist03_album01 by artist03 (dur: 3.52m)
RealTime =  66s, SimTime =  66.00min (1.10hr)  |  Now Playing:
artist03_album01_song04 from artist03_album01 by artist03 (dur: 3.52m)
```

```
RealTime =  67s, SimTime =  67.00min (1.12hr)  |  Now Playing:
artist03_album01_song04 from artist03_album01 by artist03 (dur: 3.52m)
RealTime =  68s, SimTime =  68.00min (1.13hr)  |  Now Playing:
artist03_album01_song04 from artist03_album01 by artist03 (dur: 3.52m)
RealTime =  69s, SimTime =  69.00min (1.15hr)  |  Now Playing:
artist03_album01_song05 from artist03_album01 by artist03 (dur: 3.67m)
RealTime =  70s, SimTime =  70.00min (1.17hr)  |  Now Playing:
artist03_album01_song05 from artist03_album01 by artist03 (dur: 3.67m)
RealTime =  71s, SimTime =  71.00min (1.18hr)  |  Now Playing:
artist03_album01_song05 from artist03_album01 by artist03 (dur: 3.67m)
RealTime =  72s, SimTime =  72.00min (1.20hr)  |  Now Playing:
artist03_album01_song06 from artist03_album01 by artist03 (dur: 1.13m)
RealTime =  73s, SimTime =  73.00min (1.22hr)  |  Now Playing:
artist03_album02_song01 from artist03_album02 by artist03 (dur: 2.55m)
RealTime =  74s, SimTime =  74.00min (1.23hr)  |  Now Playing:
artist03_album02_song01 from artist03_album02 by artist03 (dur: 2.55m)
RealTime =  75s, SimTime =  75.00min (1.25hr)  |  Now Playing:
artist03_album02_song01 from artist03_album02 by artist03 (dur: 2.55m)
RealTime =  76s, SimTime =  76.00min (1.27hr)  |  Now Playing:
artist03_album02_song02 from artist03_album02 by artist03 (dur: 1.16m)
RealTime =  77s, SimTime =  77.00min (1.28hr)  |  Now Playing:
artist03_album02_song03 from artist03_album02 by artist03 (dur: 2.24m)
RealTime =  78s, SimTime =  78.00min (1.30hr)  |  Now Playing:
artist03_album02_song03 from artist03_album02 by artist03 (dur: 2.24m)
RealTime =  79s, SimTime =  79.00min (1.32hr)  |  Now Playing:
artist03_album02_song04 from artist03_album02 by artist03 (dur: 2.8m)
RealTime =  80s, SimTime =  80.00min (1.33hr)  |  Now Playing:
artist03_album02_song04 from artist03_album02 by artist03 (dur: 2.8m)
RealTime =  81s, SimTime =  81.00min (1.35hr)  |  Now Playing:
artist03_album02_song04 from artist03_album02 by artist03 (dur: 2.8m)
RealTime =  82s, SimTime =  82.00min (1.37hr)  |  Now Playing:
artist03_album02_song05 from artist03_album02 by artist03 (dur: 3.69m)
RealTime =  83s, SimTime =  83.00min (1.38hr)  |  Now Playing:
artist03_album02_song05 from artist03_album02 by artist03 (dur: 3.69m)
RealTime =  84s, SimTime =  84.00min (1.40hr)  |  Now Playing:
artist03_album02_song05 from artist03_album02 by artist03 (dur: 3.69m)
RealTime =  85s, SimTime =  85.00min (1.42hr)  |  Now Playing:
artist03_album02_song05 from artist03_album02 by artist03 (dur: 3.69m)
RealTime =  86s, SimTime =  86.00min (1.43hr)  |  Now Playing:
artist03_album02_song06 from artist03_album02 by artist03 (dur: 2.31m)
RealTime =  87s, SimTime =  87.00min (1.45hr)  |  Now Playing:
artist03_album02_song06 from artist03_album02 by artist03 (dur: 2.31m)
RealTime =  88s, SimTime =  88.00min (1.47hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =  89s, SimTime =  89.00min (1.48hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =  90s, SimTime =  90.00min (1.50hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
```

```
RealTime =  91s, SimTime =  91.00min (1.52hr)  |  Now Playing:
artist01_album01_song01 from artist01_album01 by artist01 (dur: 3.69m)
RealTime =  92s, SimTime =  92.00min (1.53hr)  |  Now Playing:
artist01_album01_song02 from artist01_album01 by artist01 (dur: 2.81m)
RealTime =  93s, SimTime =  93.00min (1.55hr)  |  Now Playing:
artist01_album01_song02 from artist01_album01 by artist01 (dur: 2.81m)
RealTime =  94s, SimTime =  94.00min (1.57hr)  |  Now Playing:
artist01_album01_song02 from artist01_album01 by artist01 (dur: 2.81m)
RealTime =  95s, SimTime =  95.00min (1.58hr)  |  Now Playing:
artist01_album01_song03 from artist01_album01 by artist01 (dur: 1.29m)
RealTime =  96s, SimTime =  96.00min (1.60hr)  |  Now Playing:
artist01_album01_song04 from artist01_album01 by artist01 (dur: 1.11m)
RealTime =  97s, SimTime =  97.00min (1.62hr)  |  Now Playing:
artist01_album01_song05 from artist01_album01 by artist01 (dur: 2.7m)
RealTime =  98s, SimTime =  98.00min (1.63hr)  |  Now Playing:
artist01_album01_song05 from artist01_album01 by artist01 (dur: 2.7m)
RealTime =  99s, SimTime =  99.00min (1.65hr)  |  Now Playing:
artist01_album01_song05 from artist01_album01 by artist01 (dur: 2.7m)
RealTime = 100s, SimTime = 100.00min (1.67hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime = 101s, SimTime = 101.00min (1.68hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime = 102s, SimTime = 102.00min (1.70hr)  |  Now Playing:
artist01_album01_song06 from artist01_album01 by artist01 (dur: 3.6m)
RealTime = 103s, SimTime = 103.00min (1.72hr)  |  Now Playing:
artist01_album02_song01 from artist01_album02 by artist01 (dur: 1.5m)
RealTime = 104s, SimTime = 104.00min (1.73hr)  |  Now Playing:
artist01_album02_song01 from artist01_album02 by artist01 (dur: 1.5m)
RealTime = 105s, SimTime = 105.00min (1.75hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime = 106s, SimTime = 106.00min (1.77hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime = 107s, SimTime = 107.00min (1.78hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime = 108s, SimTime = 108.00min (1.80hr)  |  Now Playing:
artist01_album02_song02 from artist01_album02 by artist01 (dur: 3.88m)
RealTime = 109s, SimTime = 109.00min (1.82hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime = 110s, SimTime = 110.00min (1.83hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime = 111s, SimTime = 111.00min (1.85hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime = 112s, SimTime = 112.00min (1.87hr)  |  Now Playing:
artist01_album02_song03 from artist01_album02 by artist01 (dur: 3.86m)
RealTime = 113s, SimTime = 113.00min (1.88hr)  |  Now Playing:
artist01_album02_song04 from artist01_album02 by artist01 (dur: 2.67m)
RealTime = 114s, SimTime = 114.00min (1.90hr)  |  Now Playing:
artist01_album02_song04 from artist01_album02 by artist01 (dur: 2.67m)
```

```
RealTime = 115s, SimTime = 115.00min (1.92hr)  |  Now Playing:
artist01_album02_song05 from artist01_album02 by artist01 (dur: 2.09m)
RealTime = 116s, SimTime = 116.00min (1.93hr)  |  Now Playing:
artist01_album02_song05 from artist01_album02 by artist01 (dur: 2.09m)
RealTime = 117s, SimTime = 117.00min (1.95hr)  |  Now Playing:
artist01_album02_song06 from artist01_album02 by artist01 (dur: 2.39m)
RealTime = 118s, SimTime = 118.00min (1.97hr)  |  Now Playing:
artist01_album02_song06 from artist01_album02 by artist01 (dur: 2.39m)
RealTime = 119s, SimTime = 119.00min (1.98hr)  |  Now Playing:
artist01_album02_song06 from artist01_album02 by artist01 (dur: 2.39m)
RealTime = 120s, SimTime = 120.00min (2.00hr)  |  Now Playing:
artist02_album01_song01 from artist02_album01 by artist02 (dur: 1.04m)
```