Lab 8

Stacks, Queues and Linked Lists

Stacks

Task S01: Basic Stack Operations

You are given the “`Stack.py`” class which implements basic stack functionality such as `push`, `pop`, `top` etc. Go to the `TaskS01()` function and apply the following operations on the stack in given order:

```
push(6), push(7), push(9), pop(), top(), pop(), push(10), pop(),
pop(), push(20)
```

`TaskS01()` looks like following:

```
def TaskS01():
    S = Stack()                 # DO NOT MODIFY THIS LINE
    #===================
    # Insert code here:
    #-------------------
    pass
    #===================
    return S                    # DO NOT MODIFY THIS LINE
```

Run the autograder to check your taskS01:

```
> python grade
```

Queues

Task Q01: Basic Queue Operations

You are also given the “`CircularQueue.py`” class which implements basic queue functionality such as `enqueue`, `dequeue`, `first` etc. Go to the `TaskS01()` function and apply the following operations on the stack in given order:

```
enqueue(6), enqueue(7), enqueue(9), dequeue(), first(),
dequeue(), enqueue(10), dequeue(), dequeue(), enqueue(20)
```

`TaskQ01()` looks like following:

```
def TaskQ01():
    Q = CircularQueue()         # DO NOT MODIFY THIS LINE
    #===================
    # Insert code here:
    #-------------------
    pass
    #===================
    return Q                    # DO NOT MODIFY THIS LINE
```

Run the autograder to check your taskQ01:

```
> python grade
```

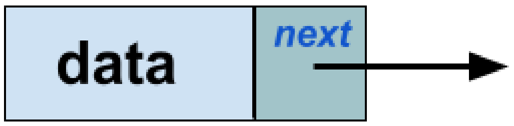
Playing with Linked Lists

Task L01: Create Node Class

First we will make a node. Look at the starter code, you will find the node class, this is the most basic element in a linked list.

```
class Node:
    def __init__(self):
        pass
```

A node has a value, which can be an object even, and a reference to the next node, as shown in the diagram below:



Let's create a node class first. Open up the "`lab05.py`" file and find the `Node` class. Add the following instance variables to the class as shown in the code below:

```
class Node:
    def __init__(self, data, next=None):
        self.data = value
        self.next = None
```

Once you've completed this, run the autograder to check task L01:

```
> python grade
```

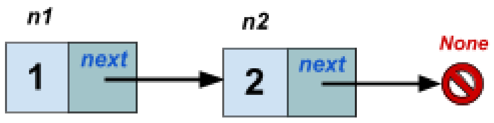
Task L02: Create Node Objects

In this task you'll create 3 node objects. Open up the "`lab05.py`" file and find `TaskL02()` function. Create nodes having data 2,3 and 4 in this order:

```
def TaskL02():
        #==================
        # Insert code here:
        #------------------
        n1 = __________
        n2 = __________
        n3 = __________
        #==================
        return (n1,n2,n3)      # DO NOT MODIFY
```

Task L03: Linking Two Nodes

The way we link two nodes is to point the next of one one node to the other node, as shown below:



Let's actually implement this ourselves. Go to `TaskL03()` function and the following piece of code:

```
def TaskL03():
   n1 = Node(1)                  # DO NOT MODIFY THIS LINE
   #==================
   # Insert code here:
   #------------------
   n2 = Node(2)
   n1.next = n2
   #==================
   return n1                     # DO NOT MODIFY THIS LINE
```

This time `run.py` file is not provided due to the complexity of input parameters. However you can test these tasks directly in the python interpreter. Launch python and type the following:

```
$ python
Python 3.9.6 (default, Sep 26 2022, 11:37:49)
....
>>> from lab05 import *
>>> n1 = TaskL03()
>>> n1.data
1
>>> n2 = n1.next
>>> n2.data
2
```

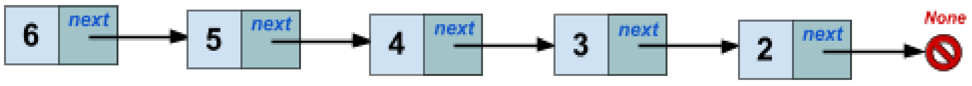
```
>>> n1.next.data
2
```

After verifying your function, run the autograder to check task L03:

```
> python grade
```

Task L04: Linking Multiple Nodes

In this task you will create a linked list of descending numbers 6,5,4,3,2 as shown below:



Go to the `TaskL04()` function. Add code to create a linked list shown in the figure above. Note that the first node is already created for you. You have to add the remaining four:

```
def TaskL04():
        n1 = Node(6)                  # DO NOT MODIFY THIS LINE
        #==================
        # Insert code here:
        #------------------
        pass
        #==================
        return n1                     # DO NOT MODIFY THIS LINE
```

To check your logic, launch python and type the following:

```
$ python
Python 3.9.6 (default, Sep 26 2022, 11:37:49)
....
>>> from lab05 import *
>>> n1 = TaskL04()
>>> n1.data
6
>>> n2 = n1.next
>>> n2.data
5
>>> n1.next.next.data
4
>>> n2.next.data
4
>>> n1.next.next.next.next.data
2
>>> print(n1.next.next.next.next.next)
None
```

After verifying your function, run the autograder to check task L04:

```
> python grade
```

Task L05: Adding Numbers in a Linked List

Let's call the first node of the linked list the "`head`". Given the head, we want to hop the nodes one by one until we reach the end of the list, that is `None`. The `TaskL05(head)` function takes one parameter that is the head of a linked list and returns the sum of all numbers stored in the list. Open up this function and complete its logic:

```
def TaskL05(head):
    sum = 0
    temp = __________
    while temp != __________:
        sum += __________
        temp = __________
    return sum
```

Run the autograder to check task L05:

```
> python grade
```

Task L06: Inserting an Item at the Head

The function `TaskL06(head,data)` takes two parameters. First one is the head of the linked list and second one is the data that is to be inserted before the head. Note that the data first needs to be put inside a `Node`. Insert the item before the head, reassign the head to new node and return the new head:

```
def TaskL06(head, data):
    #===================
    # Insert code here:
    #-------------------
    pass
    #-------------------
    return head                      # DO NOT MODIFY THIS LINE
```

Task L07: Removing the Head

The function `TaskL07(head)` takes a linked list as its parameter. It removes the first node and returns the new head pointer:

```
def TaskL07(head):
    #===================
    # Insert code here:
    #-------------------
    pass
    #-------------------
    return head                      # DO NOT MODIFY THIS LINE
```

Task L08: Get Tail Node

We can’t directly access the last item of the list. We have to begin from the head and jump from node to node. Remember the last node points to None. The function `TaskL08(head)` takes a linked list as its parameter and returns the very last node or the tail node of this list:

```
def TaskL08(head):
    tail = None                # DO NOT MODIFY THIS LINE
    #===================
    # Insert code here:
    #-------------------
    pass
    #-------------------
    return tail                    # DO NOT MODIFY THIS LINE
```

Run the autograder to check your tasks so far:

```
> python grade
```

Task L09: Add Tail Node

Now you know how to get the tail node, write code inside `TaskL09(head,node)` function to add a given node at the very end of a linked list:

```
def TaskL09(head, node):
    #===================
    # Insert code here:
    #-------------------
    pass
    #-------------------
```

Task L10: Remove Tail Node

While iterating we can’t check if the current node is the last node because we need to set the previous node’s next pointer to `None`. But we can’t go back to the previous node. Instead we have to look at one node in advance whether the next node is the last node. In this case we'll check if the next node’s next pointer is `None`. Go to `TaskL10(head)` and write code to remove the tail node from the list:

```
def TaskL10(head):
    #===================
    # Insert code here:
    #-------------------
    pass
    #-------------------
```

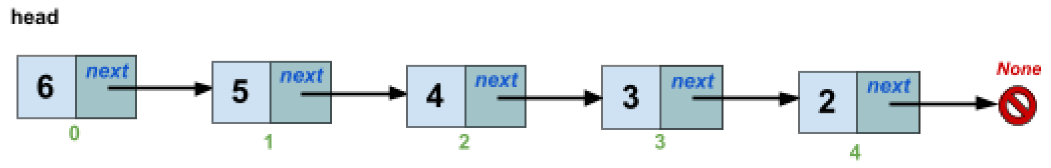
This function does not return anything.

Run the autograder to check your tasks so far:

```
> python grade
```

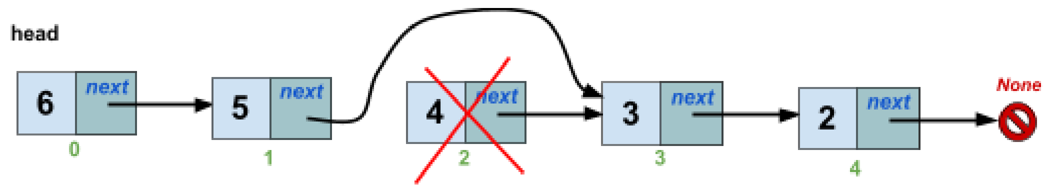
Task L11: Remove item from position ‘k’

Consider the following linked list in which we want to remove an item by giving its position. We’ll call the first item to be item number 0, second item to be item number 1 and so on, as shown below:



Note: These numberings are only logical. You are not storing these numbers anywhere. So item number 3 means you have to make 3 jumps from the head in order to reach it.

The function `TaskL11(head,k)` takes a linked list and an integer position of the item to remove. For example `TaskL11(head,2)` would remove the node containing value 4:



Special case: What if we call the function with position 0, i.e `TaskL11(head,0)`? This will remove the head. This is the reason this function always returns the head *(potentially changed)*:

```
def TaskL11(head,k):
    #==================
    # Insert code here:
    #------------------
    pass
    #------------------
    return head                  # DO NOT MODIFY THIS LINE
```