CSc 120: Intro to Computer Programming II
Spring 22 (Lewis)

Test 1

Fri 25 Feb 2022

Solutions

Name: _____ NetID: _____

NOTE NOTE NOTE: In any problem involving linked lists, you may assume that the nodes are all `ListNode` objects, like you have used in the homework. In particular:

- The constructor requires that you pass it a single value.
- `val`, `next` are public fields.

NOTE 2: In all problems on this exam, recursion is allowed (although not expected!), and helper functions are allowed (unless explicitly banned).

Question	Points	Score
Short Answer, page 1	15	
Short Answer, page 2	17	
Misc Functions	8	
Reference Diagrams	20	
Similar Algorithms	20	
Class `ListOfValues`	20	
Total:	100	

NetID: _____

1. (a) (5 points) Suppose that you want to represent an empty array. Explain the difference between the `a` and `b` in the example below, and identify which one represents "an empty array." (Only one of them does!)

```
a = []
b = None
```

Solution: `a` is an empty array: its type is array, and its length is 0

`b` is not an object at all; it is `None`, which we use in Python to represent nothingness or "no object"

(b) (5 points) What is the difference between the `==` operator and the `is` operator in Python?

Solution: `==` : compares the values

`is` : checks to see if the two references point to the same object

(c) (5 points) Suppose that you are writing a program where you will store data in a sequential order, but you will often be adding and removing elements in the middle of that order.

Explain why using an array might be a very poor choice.

Solution: Every time that we add or remove an element, we have to shift lots of elements over, which is expensive.

2. (a) (2 points) Will the following code snippet print `True` or `False`?
(Please simply give `True` or `False` - there isn't enough room here to explain exactly how it works.)

```
def add_one(data, val):
    data.append(val)
    return data

before = [1,2,3]
after  = add_one(before, 4)

print(after is before)
```

Solution: True

(b) (5 points) Write a snippet of code (not a complete function) which will print out all of the multiples of 3, from 0 to 100 (inclusive).

Solution: for i in range(0,100,3): print(i)

(c) (10 points) What is printed by the snippet of code below?
(We won't be grading you on drawing a reference diagram. But I bet that if you draw one, you will get more accurate results!)

```
y = 1
x = y
z = [x,y]
x = 2
y = 3
a = [z, [x,y]]
a[0][1] = 4
print(x)
print(y)
print(z)
print(a)
```

Solution:

```
2
3
[1,4]
[[1,4], [2,3]]
```

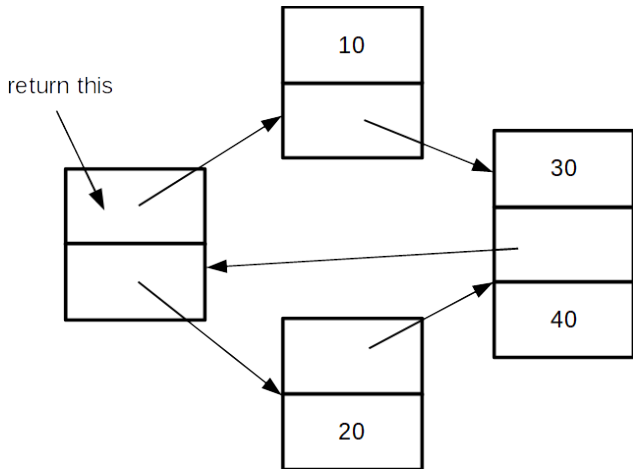
3. (8 points) Write a function, `ll_has_val(head,val)`, which scans a linked list to see if it contains a certain value. Return `True` if it exists, and `False` if not.

Solution:

```
def ll_has_val(head, val):
    cur = head
    while cur is not None:
        if cur.val == val:
            return True
    return False
```

4. (20 points) Write a function which builds and returns the shape defined by the reference diagram below. Return the object indicated.

   NOTE: This diagram is entirely made up of arrays - there are no linked lists here.



Solution:

```
def test_shape1():
    retval = [ [10,None], [None,20] ]
    last   = [30,retval,40]
    retval[0][1] = last
    retval[1][0] = last
    return retval
```

Grading Note: If the student builds the correct shape but ends up returning something equivalent to `retval[0]` instead of `retval`, we will only apply a **small** deduction.

5. Implement two functions, each of which perform the same task: it removes the third element, and returns the updated data structure. (If there are fewer than three elements, then it does nothing and returns the unchanged data structure.)

   The trick is: the first one has you do this on an array, and the second one has you doing that to a linked list.

   In the array problem, make sure to return a **new array** - do not modify your input.

   In the linked list problem, modify the list you are given - you **must not** create any new nodes, or move any values around between nodes.

   (a) (8 points) `def remove_third__array(old_arr):`

   Solution: Instructor's Note: Technically, according to the text above, if the array length is less than 3, then we should not even slice to duplicate the array. So the solution I gave below is technically incorrect. But we will allow both this version, and also the more correct one, which returns the original array if it was short.

```
# NOTE: it's permissible to check the length - but it's actually not
#       necessary, because of how slicing handles beyond-length
#       indices.
return old_arr[:2] + old_arr[3:]
```

   (b) (12 points) `def remove_third__linked_list(old_list):`

   Solution:

```
if old_list           is None or \
   old_list.next      is None or \
   old_list.next.next is None:
    return old_list

old_list.next.next = old_list.next.next.next
return old_list
```

6. (20 points) Write a class that stores a **linked list** of values. When the user calls the `append(val)` method, add a new node to the end of the list; when they call `removeHead()`, remove the head node from the list, and return its value. (You may assume, in `removeHead()`, that the list is not empty.)

   I have given you the first few lines of the class to get started.

```
from list_node import ListNode

class ListOfValues:
    def __init__(self):
        self._head = None

    def append(self, val):
        new_node = ListNode(val)
        # you write the rest of append(), plus the other two methods.
```

Solution:

```
        if self._head is None:
            self._head = new_node
            return

        cur = self._head
        while cur.next is not None:
            cur = cur.next
        cur.next = new_node

    def removeHead(self):
        head_val = self._head.val
        self._head = self._head.next
        return head_val
```