CSC 120 EXAM 1 REVIEW GUIDE

WHERE TO STUDY:

- Weekly Videos
- ICAs
- This Study Guide

QUESTIONS?

- Ask Your UGTA
- Discord
- Office Hours

Note: These are not necessarily the questions that will be on your test. These questions were written collectively by the TAs (who haven't seen the test yet) and resemble what we think you need practice with for the test. Do not use this as your only resource for studying.

That being said, feel free to jump between questions, and do the ones you think will help you the most. Ask questions on Discord, and attend office hours for additional clarification.

For all problems, you may refer to the following ListNode definition:

```python
class ListNode:
    def __init__(self, val):
       self.val = val
       self.next = None

    def __str__(self):
       vals = []
       objs = set()
       curr = self

       while curr is not None:
          curr_str = str(curr.val)

          if curr in objs:
             vals.append("{} -> ... (to infinity and beyond)".format(curr_str))
             break
          else:
             vals.append(curr_str)
             objs.add(curr)

          curr = curr.next

       return " -> ".join(vals)
```

Review Problems

1. Write a function `next_to_last(head)` which, given the first node of a linked list, returns the value of the next-to-last element. If the linked list has fewer than 2 nodes, then return `None`.

   For Example:
   If the linked list is: `8 -> 9 - 12 -> 3`, the method should return `12`.
   If the linked list is: `2`, the method should return `None`.

2. Write a method for the LinkedList class called `duplicate(head)` that modifies the list such that every element is duplicated. Return the head of the list after it has been modified.

   For Example:
   If the linked list is: `1 -> 2 -> 3 -> 4`, the method should return `1 -> 1 -> 2 -> 2 -> 3 -> 3 -> 4 -> 4`.

3. Write a function, `file_to_dict(file_name)` that takes a text file name as a parameter, reads it, and returns the content of *every other line* inside a dictionary. Each line consists of a word, followed by a number, separated by a comma. The word should be the dictionary entry's key and the number should be its value. Repeated elements within the text file should also have the last entry as its final value. Do not use the csv library.

   Example Text File:

   ```
   Hi,   3
   Boo,     5
   Foo, 1
   Baz,   3
   Bar,      6
   Boo,    4
   Hi,  2
   ```

   Should return: `{'Hi': 2, 'Foo': 1, 'Bar': 6}`

4. Look at the function `palindrome_array(arr)` provided below, which is meant to check if a list is the same forward as it is backwards:

```
def palindrome_array(arr):
    if len(arr) == 0:
        return True

    else:
        rev_arr = arr

        for i in range(len(arr)):
            temp = rev_arr.pop()
            rev_arr.insert(0, temp)

        for i in range(len(arr)):
            if arr[i] != rev_arr[i]:
                return False

        return True
```

This function will return a boolean representing whether or not the input array's contents are the same forward and backwards. It is meant to accomplish this by making a reversed version of the array and then comparing it to the original one. However, when this code is run, it will always return `True`. Why is that? How could you modify the code to fix the mistake?

5. Write a function `find_median_of_five()` to find the median of five input integers. Prompt the user to enter **5 integers**, one per line. This will require you to call the input function 5 times. (For an added challenge, try doing this with a for-loop!) Store these integers in an array, sort the array, then return the median. *The median of a set of numbers is the value that's exactly in the middle of that set after it has been ordered.*

   For Example:
   If the user inputs numbers in the following order:

```
3
1
4
5
2
```

The program should return `3`.

6. Describe the contents of `data` after the following code has executed and explain why.

```
data = [None, [10, 20, 30], None, ["hello", None], 50]
data[0] = data[3]
data[2] = data[1]
data[3][1] = data[1][1]
```

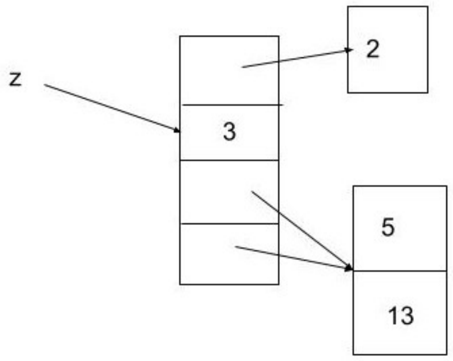
7. Write a function `find_max(head)` that returns the maximum value in a linked list. If the list is empty, return None.

For example, given a linked list constructed by the following code:

```
head = ListNode(1)
node2 = ListNode(2)
node3 = ListNode(3)

head.next = node2
node2.next = node3
```

The function should return `3`.

8. What code would produce this diagram?



9. Write a function `count_words(sentence)` that returns a dictionary with keys representing each word in a sentence, and values representing how many times it occurs in said sentence. The words should be case insensitive, and always displayed in lowercase. You can also assume that all words are separated by a space.

   For Example:
   `count_words("Float like a butterfly sting like a bee")`
   Should return the dictionary: `{'float': 1, 'like': 2, 'a': 2, 'butterfly': 1, 'sting': 1, 'bee': 1}`

10. Write a Python class named Vehicle that abstracts the concept of a vehicle. The class should be constructed with `__init__(self, make, model, year)`, which accepts `make`, `model`, and `year` as parameters and stores them as attributes. Additionally, define a method `display_info()` that prints out these attributes in the format "Year Make Model".

For Example:

```python
my_car = Vehicle("Toyota", "Corolla", 2021)
my_car.display_info()    # Should Print "2021 Toyota Corolla"
```

11. Consider the following code:

```python
class Fruit:
    def __init__(self, name, color):
        self.name = name
        self.color = color

apple = Fruit("Apple", "red")
orange = apple
orange.name = "Orange"

print(apple.name)
```

What will be printed and why? Explain what this demonstrates about how Python handles object assignment and references.

12. Using the provided ListNode class, write a function `add_to_list(head, val)` that takes two arguments: a reference to the head of a linked list and a new value to add to the end of the list. The function should create a new ListNode with the given value and append it to the end of the list. If the list is initially empty (head is None), the new node should become the new head of the list.

    For Example:

```python
node = ListNode(1)
node = add_to_list(node, 2)
node = add_to_list(node, 3)

print(node)  # Output: 1 -> 2 -> 3
```

13. Write a function `val_at_pos(head, pos)` that returns the value of the ListNode at the given position. Assume that the head of the list begins indexing at 0, and that you will only be given indices with an existing node.

For Example:
If the linked list looks like: `1 -> 2 -> 3 -> 4`
Running `val_at_pos(head, 3)` should return `4`.

14. Write a function `create_linked_list(node_vals)` that takes an input array called `node_vals` and returns the head of a linked list containing those values.

For Example:
Running `create_linked_list([3, 5, 2, 1])` should return `3 -> 5 -> 2 -> 1`.
Running `create_linked_list([])` should return `None`

15. Can a Python array hold a linked list? How about a linked list holding a Python array? Explain both of your answers. Draw the reference diagrams for both of these scenarios to further illustrate your point.

16. Draw a reference diagram for the CSStudent class instance created in `main()`:

```python
class CSMajor:
    def __init__(self, name, history, grades, gpa):
        self.name = name
        self.grades = grades
        self.history = history
        self.gpa = gpa
        self.advanced_standing = len(self.grades) >= 5 and gpa > 3.0

def main():
    course_0 = ListNode(101)
    course_1 = ListNode(110)
    course_2 = ListNode(144)
    course_3 = ListNode(120)
    course_4 = ListNode(244)
    course_5 = ListNode(210)

    course_0.next = course_1
    course_1.next = course_2
    course_2.next = course_3
    course_3.next = course_4
    course_4.next = course_5

    student_grades = [4.0, 3.0, 2.0, 4.0, 3.0, 4.0]

    student = CSMajor("Jane Doe", course_1, student_grades, 3.33)

if __name__ == "__main__":
    main()
```

Solutions

1.

```python
def next_to_last(head):
    if head is None or head.next is None:
        return None

    cur = head
    while cur.next.next is not None:
        cur = cur.next

    return cur.val
```

2.

```python
def duplicate(self):
    cur = self

    while cur is not None:
        new_node = ListNode(cur.val)
        new_node.next = cur.next
        cur.next = new_node
        cur = new_node.next

    return self
```

3.

```python
def file_to_dict(file_name):
    file = open(file_name,'r')
    retval = {}
    counter = 0

    for line in file:
        if counter % 2 == 0:
            line_arr = line.split(",")
            retval[line_arr[0].strip()] = int(line_arr[1].strip())
        counter +=1

    file.close()
    return retval
```

4. This code does, in fact, make a reversed array and compare it to the original array when checking if the array is a 'palindrome array'. However, when the reversed array is created, it is assigned to the array as `arr`, which makes it an alias of `arr`. This means that any operation done to `rev_arr` will also be done to `arr`. Since `arr` points to `rev_arr`, each item in `arr` will be the same as the corresponding element in `rev_arr`, which causes this function to always return true. To fix this, instead of creating `rev_arr` with `rev_arr = arr`, you can initialize it as an empty array, and add the elements of `arr` one at a time.

```python
def palindrome_array(arr):
    if len(arr) == 0:
        return True

    else:
        # Creates a reversed copy of the array
        rev_arr = arr[::-1]

        for i in range(len(arr)):
            temp = rev_arr.pop()
            rev_arr.insert(0, temp)

        for i in range(len(arr)):
            if arr[i] != rev_arr[i]:
                return False

    return True
```

5.

```
def find_median_of_five():
    # Step 1: Read five integers from the user
    numbers = []

    for _ in range(5):
        num = int(input())
        numbers.append(num)

    # Step 2: Sort the array of integers
    numbers.sort()

    # Step 3: Find and print the median
    median = numbers[2]  # Since the list has 5 elements, the median is the third
element after sorting

    return median
```

6.

- `data[0] = data[3]` sets the first element of data to reference the array `["hello", None]`.
- `data[2] = data[1]` sets the third element of data to reference the array `[10, 20, 30]`.
- `data[3][1] = data[1][1]` changes the second element of the array at `data[3]` (which is also referenced by `data[0]`) to 20 (the second element of the array at `data[1]`).

So, the final `data` array is:

```
[['hello', 20], [10, 20, 30], [10, 20, 30], ['hello', 20], 50]
```

7.

```python
def find_max(head):

    if head is None:
        return None

    max_value = head.val
    current = head.next

    while current:
        if current.val > max_value:
            max_value = current.val

        current = current.next

    return max_value
```

8.

```python
x = [5,13]
z = [[2], 3, x, x]
```

9.

```python
def count_words(sentence):
    words = sentence.split()
    word_dict = {}

    for word in words:
       if word.lower() not in word_dict:
          word_dict[word.lower()] = 1
       else:
          word_dict[word.lower()] += 1

    return word_dict
```

10.

```python
class Vehicle:
    def __init__(self, make, model, year):
       self.make = make
       self.model = model
       self.year = year

    def display_info(self):
       print(f"{self.year} {self.make} {self.model}")
```

11. The output will be `"Orange"`. This is because, in Python, variables that hold objects do not store the objects themselves, but rather references to those objects. Thus, when `orange = apple` is executed, both `orange` and `apple` refer to the same `Fruit` object. Changing `orange.name` to `"Orange"` also changes `apple.name` since both variables point to the same object. This demonstrates that in Python, assignments of objects do not create a copy of that object, but rather point the new variable at an existing one.

12.

```python
def add_to_list(head, val):
    new_node = ListNode(val)

    if not head:
       return new_node

    current = head

    while current.next:
       current = current.next

    current.next = new_node
    return head
```

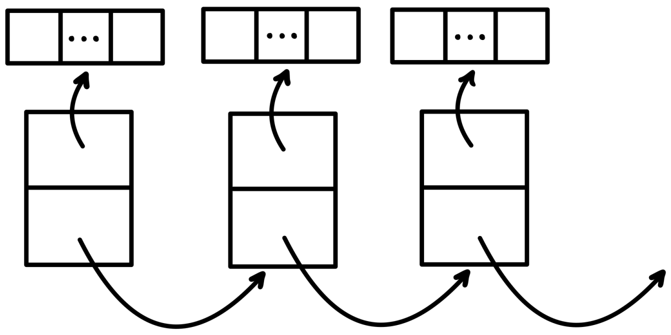
13.

```python
def val_at_pos(head, pos):
    curr = head
    curr_pos = 0

    while curr_pos != pos:
       curr = curr.next
       curr_pos += 1
    return curr.val
```

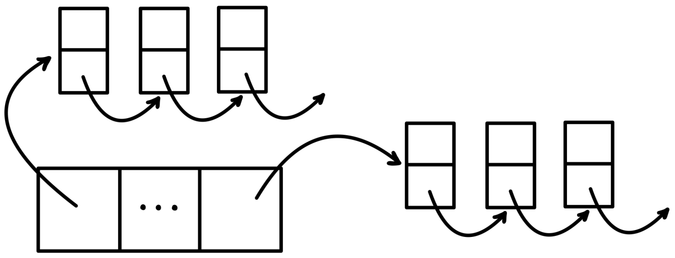
14.

```python
def create_linked_list(node_vals):
    if not node_vals:
        return None

    head = ListNode(node_vals[0])
    curr = head

    for val in node_vals[1:]:
        curr.next = ListNode(val)
        curr = curr.next

    return head
```

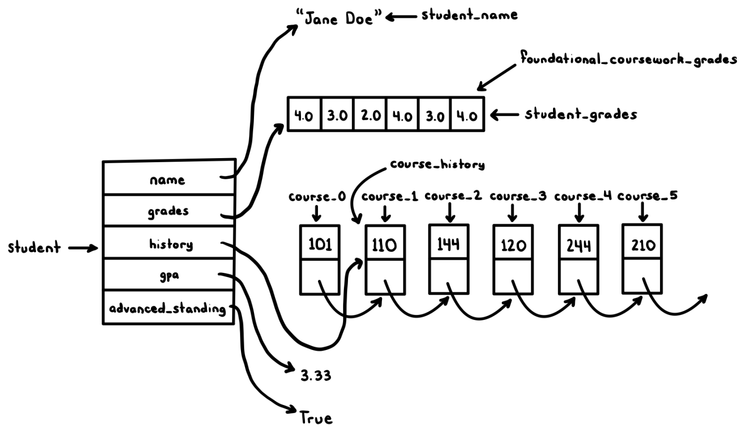
15. Both of these scenarios are possible because both data structures store references to the objects, rather than the objects themselves. In a linked list holding a Python array, each node of the linked list contains a reference to the Python array object in its `.val` attribute. Conversely, in a scenario where a Python array contains a linked list, each element of the array holds a reference to the some ListNode objects.
Python array in a LL:



LL in a python array:



16. Class reference diagram:


"Jane Doe"
student_name
foundational_coursework_grades
4.0
3.0
2.0
4.0
3.0
4.0
student_grades
course_history
course_0
course_1
course_2
course_3
course_4
course_5
student
name
grades
history
gpa
advanced_standing
101
110
144
120
244
210
3.33
True