15-110 Hw3 - Written Portion

Name:

AndrewID:

Complete the following problems in the fillable PDF, or print out the PDF, write your answers by hand, and scan the results.

When you are finished, upload your hw3.pdf to **Hw3 - Written** on Gradescope, and upload your hw3.py file to **Hw3 - Programming** on Gradescope. Make sure to check the autograder feedback after you submit!

Written Problems

#1 - 2D Lists - 5pts

Can attempt after Lists and Methods lecture

Fill in the following table with the values in the 2D list returned by `mysteryFunction`. Write an X in the squares that are outside the bounds of the list.

```
def mysteryFunction(x, y):
    myList = []
    for i in range(x):
        lst = []
        j = 0
        while j < y:
            if j <= i:
                lst.append("o")
            else:
                lst.append("-")
            j = j + 1
        myList.append(lst)
    return myList
matrix = mysteryFunction(6, 5)
```


#2 - Recursion Tracing - 5pts

Can attempt after Recursion lecture

Trace the following code, then fill out the table below to indicate all the **recursive function calls** that are made, and which **value** is returned by each function call. You may not need all of the rows.

```
def gcd(x, y):
    if y == 0:
        return x
    return gcd(y, x % y)

print(gcd(20, 12))
```

Note: in the second column, make sure to indicate the actual returned **value**, not a set of arguments, a function call, or an expression.

Function Call	Returned Value
`gcd(20, 12)`	

#3 - Tracing Towers of Hanoi - 8pts

Can attempt after Recursion II & Search Algorithms lecture

Recall the algorithm we discussed in class to solve the Towers of Hanoi problem. Use that algorithm to fill out all the steps needed to move three discs from Peg A to Peg C in the table below. You might not need to use all the rows.

The three discs are called 1, 2, 3 (where 1 is the smallest and the disc on top). So the algorithm starts with the discs 1, 2, 3 on Peg A, and should end with 1, 2, 3 on Peg C. We've done the first step for you.

	Peg A	**Peg B**	**Peg C**
Start	1, 2, 3		
Step 1	2, 3		1
Step 2			
Step 3			
Step 4			
Step 5			
Step 6			
Step 7			
Step 8			
Step 9			
Step 10			
Step 11			

How many steps would it take to move 4 discs instead of 3?

#4 - Linear Search Debugging - 10pts

Can attempt after Recursion II & Search Algorithms lecture

The following three functions all attempt to implement the algorithm linear search, but with a twist: instead of identifying whether or not the target occurs in the list, each function returns the first index where the item occurs, or -1 if it never shows up. However, only one of the three is correct. Identify which of the three functions is correct, then explain what is wrong with the other two and how they can be fixed.

```
def linearSearchA(lst, target):
    for i in range(len(lst)):
        if lst[i] == target:
            return i
        return -1

def linearSearchB(lst, target):
    i = 0
    while i < len(lst):
        if lst[i] == target:
            return i
        i = i + 1
    return -1

# Initially called with index = 0
def linearSearchC(lst, target, index):
    if lst[0] == target:
        return index
    elif len(lst) == 0:
        return -1
    else:
        return linearSearchC(lst[1:], target, index+1)
```

Which implementation is correct?

- ☐ linearSearchA
- ☐ linearSearchB
- ☐ linearSearchC

Why are the other two incorrect, and how can they be fixed?

#5 - Dictionary Keys and Values - 7pts

Can attempt after Dictionaries lecture

Given the following set of code:

```
d = { "snow" : 7, "rain" : 4, "sun" : 10 }
d["fog"] = d["rain"] - 1
for k in d:
    d[k] = d[k] * 2
d.pop("sun")
```

What are the **keys** of the dictionary `d` after this code has run?

For each of the keys you listed above, what is its **value** associated with that key after the code has run?

What line of code would you write to add the **key-value pair** `("hail", 1)` to the code? Assume the line will be added at the *end* of the current code.

#6 - Good Use of Hashing? - 10pts

Can attempt after Designing Super-Fast Search lecture

Recall our discussion of what hash functions are and what they are used for. Below we've listed four different scenarios. Each scenario contains a data set, a hash function, and which values will need to be looked up in the hashtable. Select **all** the scenarios where you will generally be able to look up the given values in **constant time**.

- ☐ Given a set of integer phone numbers, hash a phone number based on the phone number itself. Use the hashtable to look up an individual phone number.

- ☐ Given a set of all the college essays sent to CMU (as strings), hash an essay based on the first character of the essay ("I want to go to CMU because.." hashes based on "I"). Use the hashtable to look up an individual essay.

- ☐ Given a set of string full names (like "Farnam Jahanian"), hash a name by adding together the numeric ASCII values of all the letters in the name. Use the hashtable to look up an individual name.

- ☐ Given a set of lists of high scores (so each list contains integers), hash a list based on the sum of its scores. Lists can be updated after hashing when new high scores are added. Use the hashtable to look up an individual high-score list.

- ☐ None of the situations described above can be searched in constant time.

Programming Problems

For each of these problems (unless otherwise specified), write the needed code directly in the Python file, in the corresponding function definition.

All programming problems may also be checked by running 'Run File As Script' on the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `onlyPositive(lst)` - 5pts

Can attempt after Lists and Methods lecture

Write a function `onlyPositive(lst)` that takes as input a **2D list** and returns a new 1D list that contains only the positive elements of the original list, in the order they originally occurred. You may assume the list only has numbers in it.

Example: `onlyPositive([[1, 2, 3], [4, 5, 6]])` returns `[1, 2, 3, 4, 5, 6]`, `onlyPositive([[0, 1, 2], [-2, -1, 0], [10, 9, -9]]` returns `[1, 2, 10, 9]`, and `onlyPositive([[-4, -3], [-2, -1]])` returns `[ ]`.

#2 - `getCharacterLines(script, character)` - 10pts

Can attempt after Lists and Methods lecture

Assume you're provided a string script that has been formatted in a specific way. Each line of the script begins with a character's name, followed by a colon, followed by their line of dialogue. Lines are separated by newlines, which are represented in Python by the string `'\n'`. For example:

```
'''Buttercup: You mock my pain.
Man in Black: Life is pain, Highness.
Man in Black: Anyone who says differently is selling something.'''
```

Write the function `getCharacterLines(script, character)`, which takes a script and a character name (both strings) and returns a list of the lines spoken by that character. The lines should be stripped of the leading character name and any leading/trailing whitespace. So if we use the following script:

```
'''Burr: Can I buy you a drink?
Hamilton: That would be nice.
Burr: While we're talking, let me offer you some free advice: talk less.
Hamilton: What?
Burr: Smile more.
Hamilton: Ha.
Burr: Don't let them know what you're against or what you're for.
Hamilton: You can't be serious.
Burr: You want to get ahead?
Hamilton: Yes.
Burr: Fools who run their mouths oft wind up dead.'''
```

Then:

```
getCharacterLines(script, "Hamilton") ==
[ "That would be nice.", "What?", "Ha.", "You can't be serious.", "Yes." ]
```

Hint: you'll want to use string and list **methods** and **operations** to make this problem more approachable. Specifically:

- `split` can help you separate the lines of text
- `index` can help you locate where a line of text switches from name to dialogue
- slicing can help you separate the name from the dialogue
- `strip` can remove excess whitespace from the front and end of the string

#3 - `addToAll(lst, x)` - 5pts

Can attempt after References and Memory lecture

Write the function `addToAll(lst, x)` which takes a list of numbers and a number and **destructively** modifies the list so that every element has `x` added to it, then returns `None`. For example, if `lst = [1, 2, 3]`, calling the function `addToAll(lst, 2)` will evaluate to `None`, but will also change `lst` to hold `[3, 4, 5]`.

#4 - `recursiveCount(lst, item)` - 5pts

Can attempt after Recursion lecture

Write a function `recursiveCount(lst, item)` that takes a list and a value as input and returns a count of the number of times that item occurs in the list. This function must use **recursion** in a meaningful way; a solution that uses a loop or built-in count functions will receive no points.

For example, `recursiveCount([2, 4, 6, 8, 10], 6)` returns `1`, `recursiveCount([4, 4, 8, 4], 4)` returns `3`, and `recursiveCount([1, 2, 3, 4], 5)` returns `0`.

#5 - `recursiveLongestString(lst)` - 5pts

Can attempt after Recursion lecture

Write a function `recursiveLongestString(lst)` that takes a list of strings as input and returns the longest string in the list. You may assume the list contains at least one element and there will not be a tie. This function must use **recursion** in a meaningful way; a solution that uses a loop or built-in max functions will receive no points.

For example, `recursiveLongestString(["a", "bb", "ccc"])` returns `"ccc"`, and `recursiveLongestString(["hi", "its", "fantastic", "here"])` returns `"fantastic"`.

Hint: what properties does the recursive result have if the function works as expected?
Another hint: consider what the **base case** for this algorithm should be. It isn't the usual list base case...

#6 - `generateBubbles(canvas, bubbleList)` - 5pts

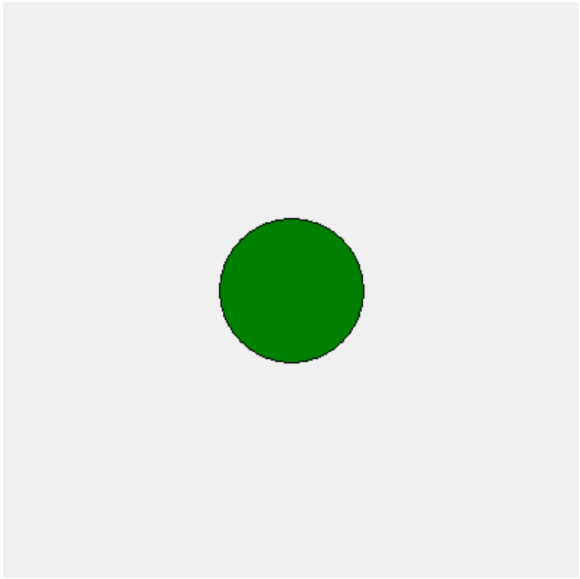
Can attempt after Dictionaries lecture

Write the tkinter function `generateBubbles(canvas, bubbleList)` which takes a tkinter canvas and a **list of dictionaries**, `bubbleList`, and draws bubbles as described in `bubbleList`.

Each dictionary in the bubble list contains exactly four keys: `"left"`, `"top"`, `"size"`, and `"color"`. The first three all map to integers (the left coordinate, top coordinate, and diameter size of the bubble), and the fourth maps to a string (its color). Use this information to draw the bubble (with `canvas.create_oval`) in the appropriate location, with the correct size and color.

For example, if we make run the function with the bubble list from the first test:
`bubbleList1 = [ {"left":150, "top":150, "size":100, "color":"green"} ]`

We'll get:

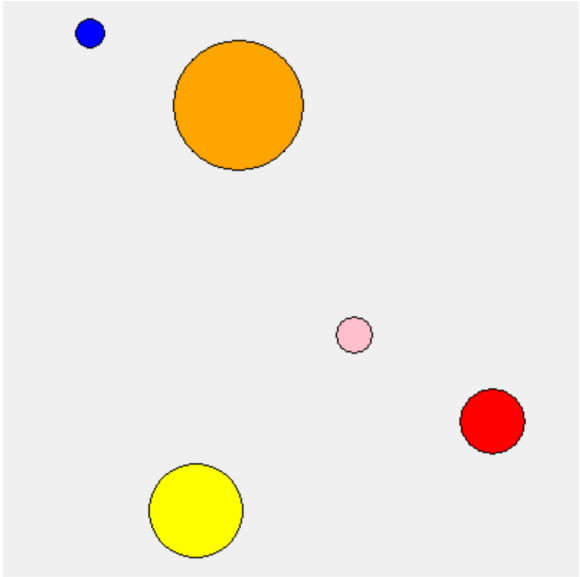


[continued on next page]

And the second test, which has:

```
bubbleList2 = [
      {'left': 317, 'top': 269, 'size': 45, 'color': 'red' },
      {'left': 118, 'top': 27, 'size': 90, 'color': 'orange'},
      {'left': 101, 'top': 321, 'size': 65, 'color': 'yellow'},
      {'left': 231, 'top': 219, 'size': 25, 'color': 'pink' },
      {'left': 50, 'top': 12, 'size': 20, 'color': 'blue' } ]
```

Should produce this:



The third test randomly generates 10 bubbles using the provided `makeNBubbles(n)` function. Try changing the size of `n` to generate more or less bubbles, and see how it looks! Your bubbles will be different every time.

Hint: a list of dictionaries might sound intimidating at first, but it's not so bad! Just loop over the list, access the dictionary using the loop control variable, then index into the dictionary to get the needed values.

#7 - `getBookByAuthor(bookInfo, author)` - 10pts

Can attempt after Dictionaries lecture

Dictionaries are very good at searching for keys, but not so good at searching for values. Write the function `getBookByAuthor(bookInfo, author)` which takes a dictionary mapping book titles (strings) to author names (also strings) and an author name (a string) and returns the book associated with that author, or `None` if the author does not appear in the dataset. You are guaranteed that no author will appear more than once in the dictionary.

For example, calling the function on `{ "The Hobbit" : "JRR Tolkein", "Harry Potter and the Sorcerer's Stone" : "JK Rowling", "A Game of Thrones" : "George RR Martin" }` and `"JK Rowling"` would return `"Harry Potter and the Sorcerer's Stone"`.

Hint: you basically want to implement **linear search** over a dictionary instead of a list. Make sure you use the right kind of loop!

#8 - `makeFirstPhonebook(nameList, numberList)` - 10pts

Can attempt after Dictionaries lecture

Write the function `makeFirstPhonebook(nameList, numberList)` that takes two lists, a list of names and a list of phone numbers (both strings), and returns a dictionary mapping names to phone numbers. You may assume that the two lists match up, i.e., each person is at the same index as their phone number.

You should use a **loop** to construct the dictionary. You'll need to loop over both the `nameList` and the `numberList` *at the same time* to access the key and value together. To do this, use the same loop control variable on both lists in each iteration.

If a person occurs in `nameList` multiple times (in other words, if they have multiple phone numbers), you should map their name to the **first** phone number they were paired with. For example, given the list of names `["Kelly", "Dave", "Kelly"]` and the list of numbers `["0000", "1234", "9876"]`, the function would return the dictionary `{ "Kelly" : "0000", "Dave" : "1234" }`.