15-110 Hw3 - Programming Portion

Each of these problems should be solved in the starter file available on the course website. Submit your code to the Gradescope assignment Hw3 - Programming for autograding.

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

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

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]`.

#2 - `recursiveMax(lst)` - 5pts

Write a function `recursiveMax(lst)` that takes a list as input and returns the maximum value in the list. You may assume the list contains at least one element. 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, `recursiveMax([1, 2, 3])` returns `3`, and `recursiveMax([2, 4, 6, 9, 10, 2, 6])` returns `10`.

Hint: consider what properties the recursive result has 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...

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

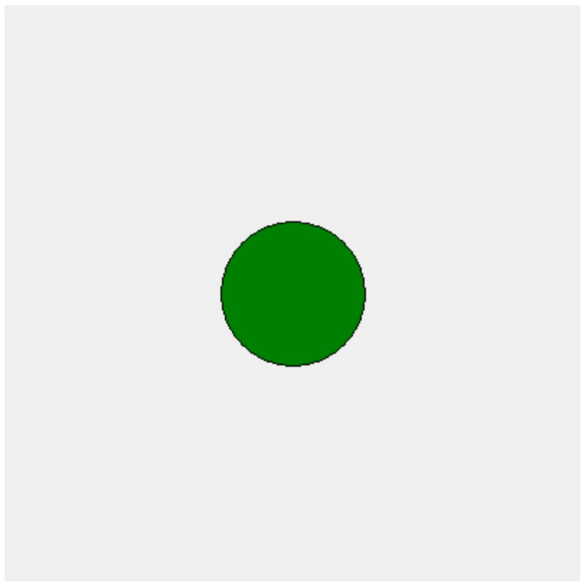
Write the tkinter function `generateBubbles(canvas, bubbleList)` which takes a tkinter canvas and a **list of dictionaries**, `bubbleList`, and draws bubbles as described in `bubbleList`.

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:

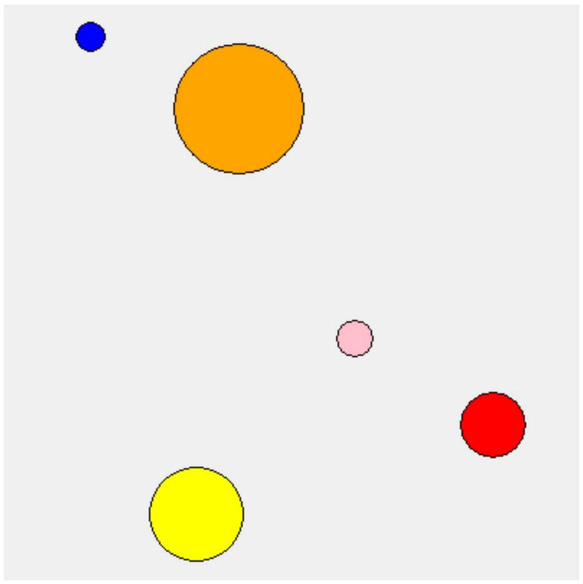


[continued on next page]

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.

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

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!

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

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" }`.