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 current script' on the starter file, which calls the function `testAll()` to run test cases on all programs.

#1 - `onlyPositive(lst)` - 10pts

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 - `addToEach(lst, s)` - 5pts

Can attempt after References and Memory lecture

Write the function `addToEach(lst, s)` which takes a list of strings and a string `s` and **destructively** modifies the list so that every element has `s` concatenated to it, returning None once done. For example, if `a = ["how", "are", "you"]`, calling the function `addToEach(a, "yah")` will evaluate to None, but will also change a to hold `["howyah", "areyah", "youyah"]`.

#3 - `recursiveLongestString(lst)` - 10pts

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 the built-in max function 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 where the list is empty, because an empty list can't have a longest string. What should it be instead?

#4 - generateBubbles(canvas, bubbleList) - 10pts

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:
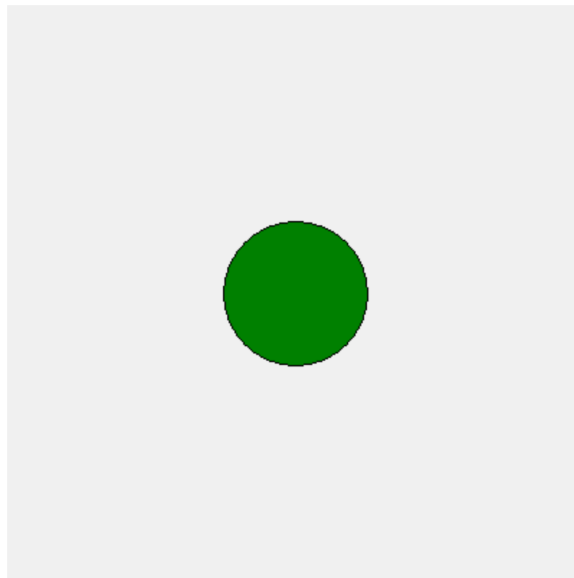
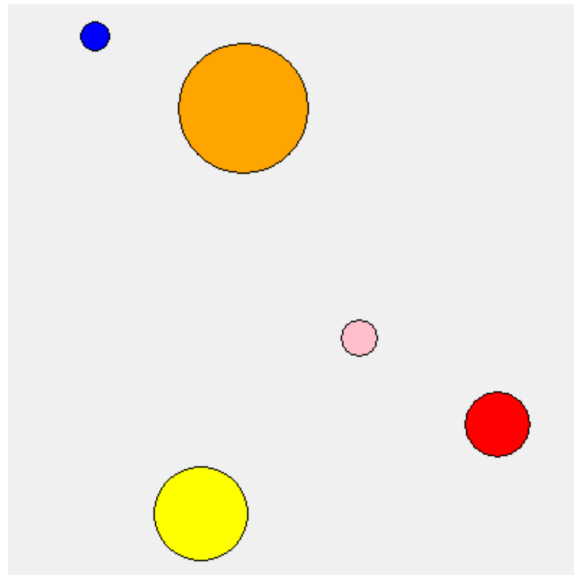And the second test, which has:

```
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 key-index into the dictionary with the four known keys to get the needed values.

#5 - `getBookByAuthor(bookInfo, author)` - 5pts

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 for a dictionary!

#6 - `makeIMDB(actorList, movieList)` - 10pts

Can attempt after Dictionaries lecture

Write the function `makeIMDB(actorList, movieList)` that takes two lists, a list of actor names and a list of movie names (both lists of strings), and returns a dictionary mapping names of actors to names of movies. You may assume that the two lists match up, i.e., each actor is at the same index as their movie.

You should use a **loop** to construct the dictionary. You'll need to loop over both the `actorList` and the `movieList` *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. Which type of loop will allow you to do this?

If a person occurs in `actorList` multiple times (in other words, if they were in multiple movies), you should map their name to the **first** movie they were paired with. For example, given the names `["Ni Ni", "Sofia Vergara", "Ni Ni"]` and the movies `["Suddenly Seventeen", "Hot Pursuit", "Love Will Tear Us Apart"]`, the function would return the dictionary `{"Ni Ni" : "Suddenly Seventeen", "Sofia Vergara" : "Hot Pursuit" }`.