15-110 Check4 - 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 Check4 - 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.

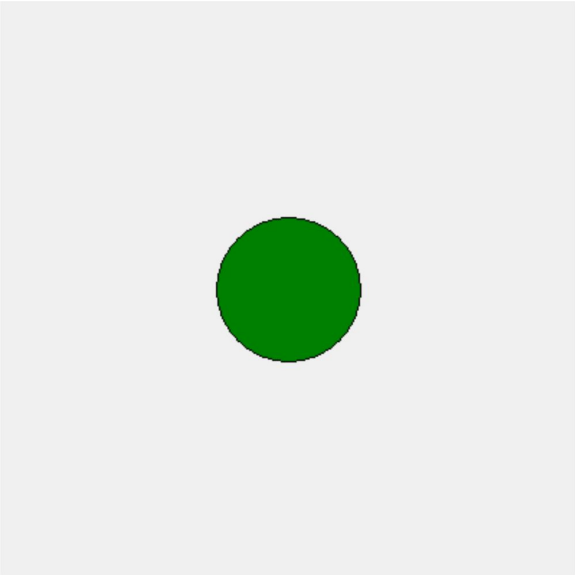
#1 - `generateBubbles(canvas, bubbleList)` - 20pts

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 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"} ]`

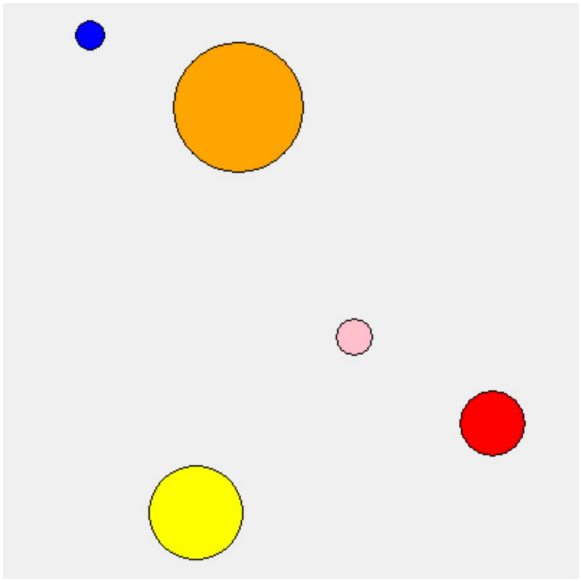
We'll get:



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 deal with the dictionary normally.

#2 - `makeFirstPhonebook(nameList, numberList)` - 25pts

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.

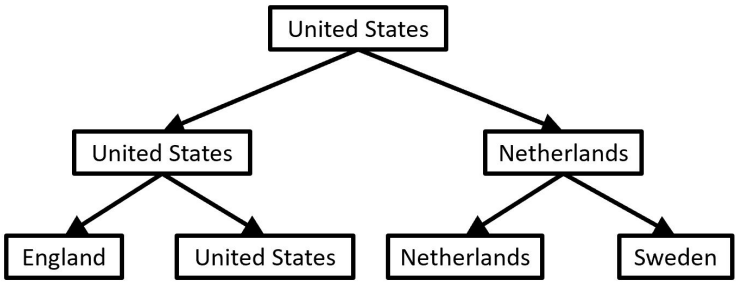
If a person occurs in `nameList` multiple times (in other words, if they have multiple phone numbers), you should map their name to the **first** phone number they were paired with. For example, given the list of names `["Kelly", "Dave", "Kelly"]` and the list of numbers `["0000", "1234", "9876"]`, the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

Hint: you need to loop over both the `nameList` and the `numberList` *at the same time* to access the key and value together. To do this, make sure to use a for-range loop, not a for-each loop!

#3 - `getInitialTeams(bracket)` - 25pts

We can represent a tournament bracket from a sports competition as a **binary tree**. To do this, store the winning team as the root node. Its children are the winning team again, as well as the second-place team. In general, every node represents the winner of a match, and its two children are the two teams that competed in that match.

For example, the following bracket represents the last two rounds of the Women's World Cup in 2019.



In our binary tree dictionary format, this would look like:

```
t1 = { "value" : "United Stated",
       "left" : { "value" : "United Stated",
           "left"  : { "value" : "England", "left" : None, "right" : None },
           "right" : { "value" : "United States", "left" : None, "right" : None } },
       "right" : { "value" : "Netherlands",
           "left"  : { "value" : "Netherlands", "left" : None, "right" : None },
           "right" : { "value" : "Sweden", "left" : None, "right" : None } }
    }
```

Write the function `getInitialTeams(bracket)` which takes a tournament bracket and returns a list of all the teams that participated in that tournament. You will need to implement this function **recursively**, since `bracket` is a binary tree. We recommend that you start by looking at the `sumNodes` and `listNodes` examples from the slides.

Hint: how can we get all of the teams to show up in the list exactly once? Every team occurs at the very beginning of the tournament, in the first set of matches. In the tree, this is represented by the **leaves,** so you should ignore values on non-leaf nodes.

Another hint: think about the **type** of data value you need to return. It should be the same in both the case case and the recursive case.