CS Scholars - Programming Hw3 - Written
Due Date: Friday 07/21 EOD

Name:

AndrewID:

For full credit on the assignment, complete either all Review + Core problems (#1-#10) or all Core + Spicy problems (#1-#4, #8-#13).

Bonus problems are related to the Advanced Track content, and are optional.

Core Problems - Written

#1 - String Code Tracing

Can attempt after Strings and Lists I lecture

Assuming that the following two lines of code have been run:

```
s1 = "coding is cool"
s2 = "CMU rocks!"
```

What will each of the following expressions evaluate to? Don't just run the code in the editor- try to figure out the answer on your own.

Expression	**Value**
`s1[7] + s2[6]`	
`s1[1] + s2[len(s2)-1]`	
`s1[7:11]`	
`s2[2:len(s2)-2]`	
`s1[::4]`	

#2 - List Code Tracing

Can attempt after Strings and Lists II lecture

Consider the following code:

```
1  lst = [ "hello", "world", "and", "all" ]
2  allLetters = ""
3  result = []
4  for word in lst:
5      tmp = ""
6      for i in range(len(word)):
7          if word[i] not in allLetters:
8              tmp += word[i]
9              allLetters += word[i]
10     result = result + [tmp]
11 print(result)
```

How many times will the loop on line 4 iterate when run on the given input?

In the first iteration of the loop on line 4, how many times will the loop on line 6 iterate when run on the given input?

What will this code print when run on the given input?

In general (non-code) terms, what does this code do?

#3 - Parsing 2D Lists

Can attempt after Strings and Lists II lecture

Consider the following snippet of code:

```
lst = [ [ 2, 3, 5, 7 ],
        [ 11, 13, 17, 19 ],
        [ 23, 29 ],
        [ 31, 37 ],
        [ 41, 43, 47 ] ]

a = lst[3]

b = lst[1][2]

c = []
for i in range(1, len(lst)-1):
    for j in range(1, len(lst[i])-1):
        c.append(lst[i][j])
```

What will each of the three created variables hold at the end of the code?

a	
b	
c	

#4 - Mystery Interactive Program

Can attempt after Time-Based Interaction lecture

Consider the code below. You can assume the supporting interaction code (like `runSimulation`) was also provided.

```
def init(data):
    data.x = 200
    data.y = 200
    data.score = 0

def redrawAll(canvas, data):
    canvas.create_oval(data.x - 25, data.y - 25, data.x + 25, data.y + 25,
                       fill="green")
    canvas.create_text(200, 50, text=str(data.score), font="Arial 32")

def timerFired(data):
    dx = random.choice([-25, 0, 25])
    dy = random.choice([-25, 0, 25])
    data.x += dx
    data.y += dy
    if (data.x < 25 or data.x > 375) or (data.y < 5 or data.y > 375):
        data.x = 200
        data.y = 200
        data.score += 1
```

What are the **component(s)** of this program's model, in plain language (not code)? Include all relevant details!

What are the **rule(s)** of this program, in plain language (not code)? Include all relevant details!

Programming Problems

Each of these problems should be solved in the starter file available on the course website. They should be submitted to the Gradescope assignment Hw3 - Programming to be autograded.

Make sure to 'Run file as Script' to check your work before submitting, then check the autograder feedback after you submit!

Review Problems - Programming

#5 - `getAllWords`

Can attempt after the Strings and Lists II lecture

Write the function `getAllWords(text)` which takes a string (a piece of text) and returns a sorted list of all the words in the text, all in lowercase and with no duplicates. You are guaranteed that the text has no punctuation and that all words are separated by spaces.

You'll want to solve this problem by using **built-in methods**. Try using the following approach:

- Use a method to make all the text lowercase
- Use a method to split up the text by spaces
- Using a loop, create a new list that contains each word just once (the `in` operator will be useful to determine if a word is already in the new list)
- Use a method to sort the list

#6 - Simple Gradebook Average

Can attempt after the Text-Based Interaction lecture

Download the files **simple_gradebook_1.txt** and **simple_gradebook_2.txt** from the course website and put them in the same folder as your **hw3.py** file. You'll need them to test your code!

Write a function `getClassAverage(filename)` which takes a filename, extracts gradebook data from the file, computes the average of the given exam scores, and returns that average as a floating point number. For example, given a file holding the text:

```
Eleanor:82
Chidi:95
Tahani:86
Jason:64
```

You would read the text, extract the numbers 82, 95, 86, and 64, and average them to get `81.75`.

Before you write code, investigate the two provided files to understand the **general pattern** you can expect from a gradebook!

#7 - Simple Grid Game

Can attempt after the Event-Based Interaction lecture

Using the interaction framework we discussed in class, build a simple grid game that has the following properties.

- An 8x8 grid is shown on the 400px x 400px screen
- A single randomly-chosen cell on the grid is 'selected' (colored yellow); the rest of the cells are left blank
- If the user presses the Up, Down, Left, or Right arrow keys, the selected cell moves one cell in the appropriate direction
- If the selected cell would move off the grid when the user presses an arrow key, the cell does not move
- If the user clicks on the selected cell, it teleports to a random location on the grid. (If the user clicks anywhere else in the grid, nothing happens).

Hint: you can represent a random cell as a random row and a random column.

Another Hint: to check whether the user has clicked on the selected cell, check whether the clicked x and y position falls within the expected left-right and top-bottom bounds of the selected cell.

Logistical note: the starter code for this problem is in the hw3.py file; you can find it by scrolling below the main set of problems.

Core Problems - Programming

#8 - `getSecretMessage`

Can attempt after the Strings and Lists I lecture

You can hide a secret message in a piece of text by setting a specific character as a key. Place the key before every letter in the message, then fill in extra (non-key) letters between key-letter pairs to hide the message in noise.

For example, to hide the message "computer" with the key "q", you would start with "computer", turn it into "q**c**q**o**q**m**q**p**q**u**q**t**q**e**q**r**", and then add extra letters as noise, perhaps resulting in "orup**qc**rzyp**qo**m**qm**hcy**qp**whh**qu**t**qt**xt**qe**ye**qr**pa". To get the original message back out, copy every letter that occurs directly after the key, ignoring the rest.

Write a function `getSecretMessage(s, key)` that takes a piece of text holding a secret message and the key to that message and returns the secret message itself. For example, if we called the function on the long string above and "q", it would return "computer". You are guaranteed that the key does not occur in the secret message.

Hint: loop over every character in the string. If the character you're on is the key, add the **next** character in the string to a result string.

#9 - `sumAnglesAsDegrees`

Can attempt after the Strings and Lists II lecture

When analyzing data, you need to convert the data from one format to another before processing it. For example, you might have a dataset where angles were measured in radians, yet you want to find the sum of the angles in degrees.

Write the function `sumAnglesAsDegrees(angles)` which takes a list of angles in radians (floats) and returns the sum of those angles **in degrees** (an integer). To do this, you will need to change each angle from radians to degrees before adding it to the sum. You can do this with the library function `math.degrees`. Make sure to **round** the final result to get an integer answer.

For example, `sumAnglesAsDegrees([math.pi/6, math.pi/4, math.pi])` should convert the radians to approximately `30.0`, `45.0`, and `180.0`, then return `255`.

Note: you are not allowed to use the built-in function `sum` for this problem. Use a loop instead!

#10 - Guessing Game

Can attempt after the Text-Based Interaction lecture

In the function `guessingGame()`, implement an interactive guessing game that the computer plays with the user. The computer should generate a random number between 1-10 and repeatedly ask the user to guess what it is until the user gets it right.

You can design your guessing game however you like, but it must meet the following constraints:

- The computer must actually pick a random number in the range [1, 10] each time the function is called (don't hardcode it!)
- If the user guesses right, the computer tells the user how many total guesses they made and ends the function
- If the user guesses wrong, the computer tells them whether the number they guessed was too high or too low and makes them guess again
- The program **must not crash** if the user enters an illegal input (a number outside of 1-10, or something that isn't an integer)

For example, here's a possible run of `guessingGame`, with user inputs bolded:

```
Welcome to the guessing game!
I'm thinking of a random number between 1 and 10
Make a guess: five
That's not an integer number! Try again.
Make a guess: 5
Not quite, it's larger than that.
Make a guess: 100
Not quite, it's smaller than that.
Make a guess: 8
You got it! Well done.
You made 4 total guesses.
```

Spicy Problems - Programming

#11 - `getCharacterLines`

Can attempt after the Strings and Lists II 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

#12 - Spicy Gradebook Average

Can attempt after the Text-Based Interaction lecture

Download the files **spicy_gradebook_1.txt** and **spicy_gradebook_2.txt** from the course website and put them in the same folder as your **hw3.py** file. You'll need them to test your code!

Write a function `getStudentAverage(filename, name)` which takes a filename and a student name, extracts gradebook data from the file, finds the specified student, computes the average of their exam scores, and returns that average as a floating point number. For example, given a file holding the following text and the name `"Tahani"`:

```
Eleanor
     Exam1: 54
     Exam2: 73
     Exam3: 95 [note: excellent improvement!]
Chidi
     Exam1: 91
     Exam2: 93
     Exam3: 97
Tahani
     Exam1: 70
     Exam2: 85
     Exam3: 91
Jason
     Exam1: 72
     Exam2: 64 [note: ask to talk after class]
     Exam3: 86
```

You would read the text, extract the numbers 70, 85, and 91, and average them to get `82.0`.

Before you write code, investigate the two provided files to understand the **general pattern** you can expect from a gradebook.

#13 - Spicy Grid Game

Can attempt after the Event-Based Interaction lecture

Start by going to #7 (Review: Simple Grid Game) and programming the game described there. Once you've done that, add the following features to the game:

- Instead of not allowing the user to move off the board, when the user would move off the board, wrap the selected cell around to the other side of the board (for example, if the cell would move above the top row, wrap it around to the bottom row).
- When the selected cell is clicked and teleports to a new location, color the space it was in before green. This should happen every time the square is clicked (so if the selected cell is clicked in four locations, all four of those locations should be colored green).
- Ensure that when the selected cell teleports, it does **not** teleport to a location that has already been visited (is colored green).
- Ensure that if the selected cell would move on top of a visited cell (green cell) on responding to an arrow key press, it does not move instead.
- When every cell on the grid is colored green, the game displays a message congratulating the user on the screen. The game should no longer be playable at this point.

Hint: you can implement wraparound with if statements, but it may be easier to implement by using the mod operator.

Another Hint: to make sure that you teleport the selected cell to an open location, just repeatedly generate new positions until you find one that wasn't already visited.

Logistical note: the starter code for this problem is in the hw3.py file; you can find it by scrolling below the main set of problems.

Bonus Problems

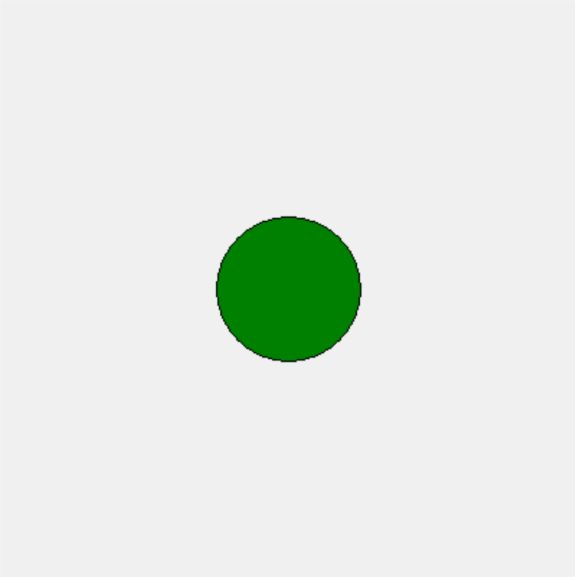
Advanced Programming 1 - Dictionaries

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:

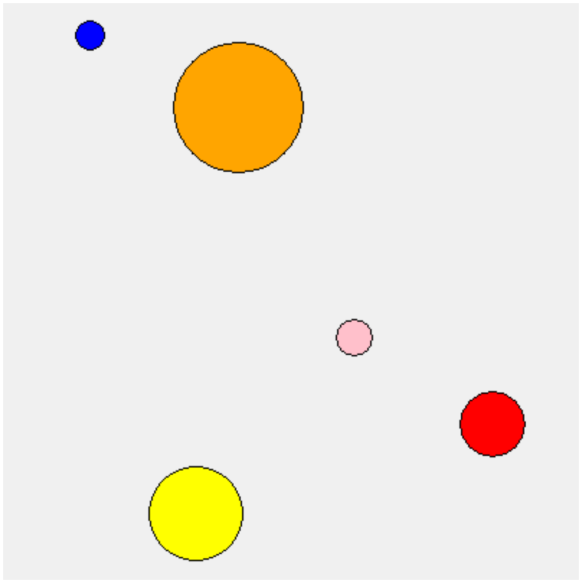


[continued on next page]

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:



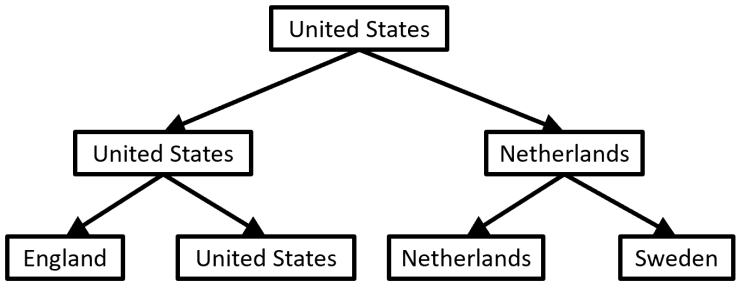
The third test randomly generates 10 bubbles using the provided `makeNBubbles(n)` function. Try changing the size of `n` to generate more or less bubbles, and see how it looks! Your bubbles will be different every time.

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.

Advanced Programming 2 - Trees

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 = { "contents" : "United States",
       "left" : { "contents" : "United States",
           "left"  : { "contents" : "England", "left" : None, "right" : None },
           "right" : { "contents" : "United States", "left" : None, "right" : None}},
       "right" : { "contents" : "Netherlands",
           "left"  : { "contents" : "Netherlands", "left" : None, "right" : None },
           "right" : { "contents" : "Sweden", "left" : None, "right" : None } }
     }
```

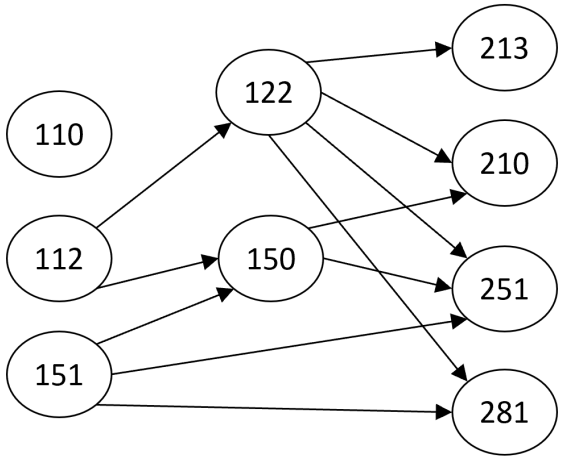
Write the function `getInitialTeams(bracket)` which takes a tournament bracket and returns a list of all the teams that participated in that tournament. For example, if the function is called on the tree above it might return `[ "England", "United States", "Netherlands", "Sweden" ]`. You will need to implement this function **recursively** to access all the nodes. We recommend that you start by looking at the `sumNodes` and `listValues` examples from the slides.

Hint 1: 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 not include values on non-leaf nodes.

Hint 2: make sure the **type** you return is the same in both base and recursive cases!

Advanced Programming 3 - Graphs

In college, you get to select your own schedule of classes. However, to take certain classes you must first pass an earlier class in the course sequence- a **prerequisite**. These prerequisites are notoriously complicated. However, we can make them a little easier to understand by representing the course dependency system as a **directed graph**, where the nodes are courses and an edge leads from course A to course B if A is a prerequisite of B. For example, the core Computer Science courses (almost) produce the following prereq graph:



Which would be represented in code as:

```
g = { "110" : [],
      "112" : ["122", "150"],
      "122" : ["213", "210", "251", "281"],
      "151" : ["150", "251", "281"],
      "150" : ["210", "251"],
      "213" : [],
      "210" : [],
      "251" : [],
      "281" : [] }
```

Write the function `getPrereqs(g, course)` that takes a directed graph (in our adjacency list dictionary format, without weights) and a string (a course name) and returns a list of all the immediate prerequisites of the given course. If we called `getPrereqs` on our graph above and `"210"`, for example, the function should return `["122", "150"]`.

Hint: you can't just return the neighbors of the course, because the edges are going in the opposite direction! Instead, iterate over all the nodes to find those that have the course as a neighbor. Construct a new list out of these nodes as the result.

Advanced Computer Science 1 - Efficiency

For the following function, describe an input that would result in **best-case efficiency,** then describe an input that would result in **worst-case efficiency**. This generic input must work at **any possible size**; don't answer 1, for example.

```
def isPrime(num):
    for factor in range(2, num):
        if num % factor == 0:
            return False
    return True
```

Best Case input:	
Worst Case input:	

Advanced Computer Science 2 - Efficiency

For each of the following functions, check the **Big-O function family** that function belongs to. You should determine the function family by considering how the number of steps the algorithm takes grows as the size of the input grows.

```
def countEven(L): # n = len(L)
  result = 0
  for i in range(len(L)):
    if L[i] % 2 == 0:
      result = result + 1
  return result
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L)
def sumFirstTwo(L):
  if len(L) < 2:
    return 0
  return L[0] + L[1]
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L1) = len(L2)
def binarySearchAll(L1, L2):
  count = 0
  for item in L1:
    # Hint: what's the complexity of
    # binary search?
    if binarySearch(L2, item) == True:
      count = count + 1
  return count
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$

```
# n = len(L); original call has i = 0
def recursiveSum(L, i):
  if i == len(L):
    return 0
  else:
    return L[i] + recursiveSum(L, i+1)
```

- ☐ O(1)
- ☐ O(logn)
- ☐ O(n)
- ☐ O(nlogn)
- ☐ $O(n^2)$