CMU 15-112 Fall 2018: Fundamentals of Programming and Computer Science Homework 1 (Due Sunday 2-Sep, at 5pm)

- Reminder: all problems that are not explicitly marked COLLABORATIVE must be completed individually, as stated in the course syllabus.

- To start:
  1. Create a folder named 'week1'
  2. Download both hw1.py and cs112_f18_week1_linter.py to that folder
  3. Edit hw1.py using Pyzo
  4. When you are ready, submit hw1.py to Autolab. For this hw, you may submit up to 20 times (which is way more than you should require), but only your last submission counts.
- Do not use string indexing, loops, lists, or recursion this week.
- Do not hardcode the test cases in your solutions.

1. Syllabus Review [5pts, hand-graded]
   You should read the entire syllabus so that you're familiar with the class's policies. Do this, then return the answers to the five questions below inside the triple-string in syllabusAnswer. This question will be hand-graded by TAs.

   1. (Fill in the blank): Extensions on assignments will only be given for university-approved absences, medical emergencies, and _____.
   2. How many grace days do you have to use?
   3. When are you allowed to use a laptop in lecture?
   4. Should Piazza posts by students be public or private?
   5. Are you ever allowed to copy code from another student?

2. Code Description [5pts, hand-graded]
   Read over the code shown in the block below, then describe what it does in less than 10 words using common language. This question will be hand-graded by TAs.

```
def mysteryCode(s):
    a = ("a" <= s) and (s <= "z") and len(s) == 1
    b = (s == "a") or (s == "e") or (s == "i") or (s == "o") or
    return a and (not b)
```

   Copy Visualize Run

3. Reasoning Over Code [5pts]
   Given the function roc (shown below), find a value that will make roc return True. You should then modify the function rocAnswer() to return that value so that the autograder will accept your work.

```
def roc(x):
    if type(x) != int:
        return False
    elif x <= 0:
        return False
    elif x % 100 == x:
        a = x // 10
        b = x % 10
        if a != b:
            return False
        return True
    else:
        return x == 42
```

Note: yes, you could try every possible value in this function until you find one that works. That won't help you learn, though. Instead, try working through the problem on paper, to see if you can figure out what a correct answer should look like without going through all the possibilities.

4. Code Writing: isEvenPositiveInt [10pts]
   Write the function isEvenPositiveInt(n) which, given a value n, returns True if it is even, positive, and an integer, and False otherwise.

5. Code Writing: getTheCents(n) [10pts]
   Write the function getTheCents(n) which takes a value n (which represents a payment in US dollars) and returns the number of cents in the payment. For example, if n is 2.45, the function should return 45. If n is an int, the function should return 0, as it has 0 cents; if it isn't a number, it should return None, because a non-number payment make no cents (ha!). If the payment has partial cents (for example, 3.953), it should be rounded up to the nearest cent (in this example, 96 cents).

6. Code Writing: playGuessingGame() [10pts]
   Write the function playGuessingGame() which runs a short guessing game with the user. The program should ask the user two Yes-or-No questions, then guess what the user is thinking based on their answers. You must use the specific prompts shown in the example exchanges below to pass the test cases (where the bolded words are user input), so you should copy the prompts directly into your code to avoid typos.

```
Let's play a guessing game! Think of a type of pet.
Does it have fur?Yes
Can you teach it to play fetch?Yes
It's a dog!

Let's play a guessing game! Think of a type of pet.
Does it have fur?Yes
Can you teach it to play fetch?No
It's a cat!

Let's play a guessing game! Think of a type of pet.
Does it have fur?No
Can it swim?Yes
It's a fish!

Let's play a guessing game! Think of a type of pet.
Does it have fur?No
Can it swim?No
It's a bird!
```

   Hint 1: For this problem, instead of using parameters and return, you'll want to use input() and print().

   Hint 2: Make sure that you don't leave any trailing whitespace after the prompts or responses. This will mess up the autograder.

7. Code Writing: threeLinesArea(m1, b1, m2, b2, m3, b3) and helpers [20pts]
   Write the function threeLinesArea(m1, b1, m2, b2, m3, b3) that takes six int or float values representing the 3 lines:

```
y = m1*x + b1
y = m2*x + b2
y = m3*x + b3
```

   First find where each pair of lines intersects, then return the area of the triangle formed by connecting these three points of intersection. If no such triangle exists (if any two of the lines are parallel), return 0.

To do this, you must write three helper functions:

- lineIntersection(m1, b1, m2, b2) to find where two lines intersect (which you will call three times)
  - This function takes four int or float values representing two lines and returns the x value of the point of intersection of the two lines. If the lines are parallel, or identical, the function should return None.
- distance(x1, y1, x2, y2) to find the distance between two points (again called three times)
  - This function takes four int or float values representing two points and returns the distance between those points.
- triangleArea(s1, s2, s3) to find the area of a triangle given its side lengths (which you will call once).
  - This function takes three int or float values representing side lengths of a triangle, and returns the area of that triangle. To do this, you may wish to to use Heron's Formula.

You may write other helper functions if you think they would be useful, but you must at least write these three exactly as described below, and then you must use them appropriately in your solution. Once you have written and tested your helper functions, then move on to writing your threeLinesArea function, which of course should use your helper functions. That's the whole point of helper functions. They help!

Note that helper functions help in several ways. First, they are logically simpler; they break down your logic into smaller chunks that are easier to reason over. Second, they are independently testable, so you can more easily isolate and fix bugs. And third, they are reusable, so you can use them as helper functions for other functions in the future. All good things!

8. Code Writing: getKthDigit(n, k) and setKthDigit(n, k, d=0) [20pts]
Write the function getKthDigit(n, k) that takes a possibly-negative int n and a non-negative int k, and returns the kth digit of n, starting from 0, counting from the right. So:

```
getKthDigit(789, 0) == 9
getKthDigit(789, 1) == 8
getKthDigit(789, 2) == 7
getKthDigit(789, 3) == 0
getKthDigit(-789, 0) == 9
```

Then write the function setKthDigit(n, k, d=0) that takes three integers -- n, k, and d -- where n is a possibly-negative int, k is a non-negative int, and d is a non-negative single digit (between 0 and 9 inclusive) with a default value of 0. This function returns the number n with the kth digit replaced with d. Counting starts at 0 and goes right-to-left, so the 0th digit is the rightmost digit. For example:

```
setKthDigit(468, 0, 1) == 461
setKthDigit(468, 1, 1) == 418
setKthDigit(468, 2, 1) == 168
setKthDigit(468, 3, 1) == 1468
setKthDigit(468, 1) == 408
```

9. COLLABORATIVE Code Writing: colorBlender(rgb1, rgb2, midpoints, n) [15pts]
NOTE: This problem is collaborative. That means you can work on it with other students! However, you must follow the collaboration rules as specified by the syllabus, and you must list all of your collaborators' andrewIDs in a comment above your solution.

This problem implements a color blender, inspired by this tool. In particular, we will use it with integer RGB values (it also does hex values and RGB% values, but we will not use those modes). Note that RGB values contain 3 integers, each between 0 and 255, representing the amount of red, green, and blue respectively in the given color, where 255 is "entirely on" and 0 is "entirely off".

For example, consider this case. Here, we are combining crimson (rgb(220, 20, 60)) and mint (rgb(189, 252, 201)), using 3 midpoints, to produce this palette (using our own numbering convention for the colors, starting from 0, as the tool does not number them):

```
color0: rgb(220,  20,  60)
color1: rgb(212,  78,  95)
color2: rgb(205, 136, 131)
color3: rgb(197, 194, 166)
color4: rgb(189, 252, 201)
```

There are 5 colors in the palette because the first color is crimson, the last color is mint, and the middle 3 colors are equally spaced between them.

So we could ask: if we start with crimson and go to mint, with 3 midpoints, what is color #1? The answer then would be rgb(212, 78, 95).

One last step: we need to represent these RGB values as a single integer. To do that, we'll use the first 3 digits for red, the next 3 for green, the last 3 for blue, all in base 10 (decimal, as you are accustomed to). Hence, we'll represent crimson as the integer 220020060, and mint as the integer 189252201.

With all that in mind, write the function colorBlender(rgb1, rgb2, midpoints, n), which takes two integers representing colors encoded as just described, a non-negative integer number of midpoints, and a non-negative integer n, and returns the nth color in the palette that the tool creates between those two colors with that many midpoints. If n is out of range (too small or too large), return None.

For example, following the case above: colorBlender(220020060, 189252201, 3, 1) returns 212078095

Hint: RGB values must be ints, not floats. When calculating midpoint colors, you can mostly use the built-in round function. However, the built-in round function has one major flaw: it varies in whether it chooses to round .5 up or down (ugh!). You can fix this by doing an extra check for whether a number is <number>.5 and choosing to always round up in that case.

10. Bonus: bonusFindIntRootsOfCubic(a,b,c,d) [3 pts]
    Note: we usually offer bonus problems for students who want an extra challenge. These problems are optional, and should only be attempted after the rest of the assignment has been finished.

    Write the function bonusFindIntRootsOfCubic(a,b,c,d) that takes the int or float coefficients a, b, c, d of a cubic equation of this form:

    y = ax3 + bx2 + cx + d

    You are guaranteed the function has 3 real roots, and in fact that the roots are all integers. Your function should return these 3 roots in increasing order. How does a function return multiple values? Like so:

```
return root1, root2, root3
```

To get started, you'll want to read about Cardano's cubic formula here. Then, from that page, use this formula:

x = {q + [q2 + (r-p2)3]1/2}1/3 + {q - [q2 + (r-p2)3]1/2}1/3 + p where:

$$p = -b/(3a), \quad q = p3 + (bc-3ad)/(6a2), \quad r = c/(3a)$$

This isn't quite as simple as it seems, because your solution for x will not only be approximate (and not exactly an int, so you'll have to do something about that), but it may not even be real! Though the solution is real, the intermediate steps may include some complex values, and in these cases the solution will include a (possibly-negligibly-small) imaginary value. So you'll have to convert from complex to real (try c.real if c is complex), and then convert from real to int.

Great, now you have one root. What about the others? Well, we can divide the one root out and that will leave us with a quadratic equation, which of course is easily solved. A brief, clear explanation of this step is provided here. Don't forget to convert these to int values, too!

So now you have all three int roots. Great job! All that's left is to sort them. Now, if this were later in the course, you could put them in a list and call a built-in function that will sort for you.; But it's not, so you can't. Instead, figure out how to sort these values using the limited built-in functions and arithmetic available this week. Then just return these 3 values and you're done.