15-112: Introduction to Programming and Computer Science, Fall 2020

Homework 3 Programming: Functions, Loops, and Lists

Due: Tuesday, September 15, 2020 by 22:00

This programming homework is designed to get you more practice with functions, loops, and lists. Your submission will be made through the web interface of Gradescope. In this homework, you will be writing nine functions. Write all functions in the same file and call that file YourAndrewIDhw3.py. You should not have any test code in this file besides the function definitions required in this handout plus any other helper functions you may want to write. You should not have any main code that is executed. Your functions should be named according to the specifications given in the questions below. Again, if you want to write helper functions within the same file to help you organize your code, you are more than welcome to do so and you can name them whatever you want. You should submit this python file on Gradescope

1 Style

This is the first homework where you will be graded on style. In this course, I discourage (read forbid) you to read other student's code. However, as a software developer, you will need to read code much more often than writing. It pays to adopt good programming style so that your code is readable and, in turn, maintanable. Although there are several style guides out there, a majority of them have common conventions. You can read the official Python style guide at `https://www.python.org/dev/peps/pep-0008/`.

We will use the following rubric[1] for grading style points. You can lose a maximum of 5 points for style errors.

- Ownership
  - You must include your name and andrewId in a comment at the top of every file you submit.
  - This is good practice for later in life, when you will want to document all code that you contribute to projects.
  - 2-point error: not writing your name/andrewId in a submitted file
- Comments

[1] Adopted with minor modifications from `https://www.cs.cmu.edu/~112/notes/notes-style.html`

  - You should write concise, clear, and informative comments that supplement your code and improve understanding.
  - Comments should be included with any piece of code that is not self-documenting.
  - Comments should also be included at the start of every function (including helper functions).
  - Comments should not be written where they are not needed.
  - 5-point error: not writing any comments at all.
  - 2-point error: writing too many or too few comments, or writing bad comments.

- Helper Functions (Top-Down Design)
  - You should use top-down design to break large programs down into helper functions where appropriate.
  - This also means that no function should become too long (and therefore unclear).
  - 5-point error: not using any helper functions (where helper functions are needed).
  - 2-point error: using too many or too few helper functions.
  - 2-point error: writing a function that is more than 20 lines long. Exceptions: blank lines and comments do not count towards this line limit, and this rule does not apply to graphics functions and init()/run() functions in animations.

- Variable Names
  - Use meaningful variable and function names (whenever possible).
  - Variables and functions should be written in the camelCase format. In this format, the first letter is in lowercase, and all following words are uppercased (eg: tetrisPiece).
  - Variable names should not overwrite built-in function names; for example, str is a bad name for a string variable. Common built-in keywords to avoid include dict, dir, id, input, int, len, list, map, max, min, next, object, set, str, sum, and type.
  - 5-point error: not having any meaningful variable names (assuming variables are used).
  - 2-point error: using some non-meaningful variable names. Exceptions: i/j for index/iterator, c for character, s for string, and n/x/y for number.
  - 2-point error: not using camelCase formatting.
  - 2-point error: using a built-in function name as a variable.

- Unused Code
  - Your code should not include any dead code (code that will never be executed).
  - Additionally, all debugging code should be removed once your program is complete, even if it has been commented out.

  - 2-points error: having any dead or debugging code.

- Formatting
  - Your code formatting should make your code readable. This includes:
    * Not exceeding 79 characters in any one line (including comments!).
    * Indenting consistently. Use spaces, not tabs, with 4 spaces per indent level (most editors let you map tabs to spaces automatically).
    * Using consistent whitespace throughout your code.
    * Good whitespace: x=y+2, x = y+2, or x = y + 2
    * Bad whitespace: x= y+2, x = y +2, or x = y + 2
  - 2-point error: having bad formatting in any of the ways described above.

2 So many triangles

So there are different types of triangles. I knew for the longest time that there are right triangles, isosceles, scalene, and equilateral triangles but, until I started teaching my son Geometry over one summer, I did not know that there were obtuse angled triangles, acute angled triangles, and even acute angled isosceles triangles. The following is a list of triangles that I found and I am sure there are more

- Equilateral triangle.
- Right triangle.
- Obtuse angled scalene triangle.
- Obtuse angled isosceles triangle.
- Acute angled scalene triangle.
- Acute angled isosceles triangle.

The cool thing is that you can use the measure of the three sides to figure out if, infact, the three sides can represent a triangle or not , and if they can, what kind of triangle they represent.

Task 1 (4 points) *Your task is to write a function that will take three values as input parameters. These values represent the measure of three sides of a triangle and you can assume that these numbers would be integers. You function will determine whether these three sides represent a triangle and if so what kind of triangle. Based on this determination, the function should return the appropriate value based on the following list (Notice that the function never returns 2 - don't ask me why):*

- *Sides cannot be a trianlge - return 0*

- *Right triangle - return 1.*
- *Equilateral triangle - return 3.*
- *Obtuse angled scalene triangle - return 4.*
- *Obtuse angled isosceles triangle - return 5.*
- *Acute angled scalene triangle - return 6.*
- *Acute angled isosceles triangle - return 7.*

You should call this function whichTriangle.

3 Error Checking

We have seen in some lecture notes that if we want to read a float from the user, we use the function raw_input() to read a string from the user and then use the function float() to convert that string to a float number. We also learned that if the user enters an invalid number, our program will crash. Now we don't like programs that crash. So, we would like to check if a string can be a float before we attempt to convert it to a float. This way, if the string is not float, we can print an error message and exit gracefully instead of crashing the program. If would be nice to have a function called isFloat that takes an input of string and returns back True if the input is a valid float and False if it is not. We can use this function in our programs as given below:

```
inp = raw_input("Enter your QPA: ")
if( not isFloat(inp)):
   print "You entered a value that is not a valid QPA. Exiting gracefully :)"
   exit()
else:
   # we can safely decode the float now
   qpa = float(inp)
```

Task 2 (4 points) *Your task is to write the isFloat function that checks the string passed to it for validity of being a float number. If the input is a valid float number, it should return True, otherwise, it should return False. I will be calling the function as shown in the above example, so make sure the return values, function name, and input parameters are appropriate. You should NOT use a try/except structure for this function.*

4 Base Bafflement

Numbers can be written in many different ways. For example, we know that the decimal numbers we use everyday such as 12, 4 and 21 are represented inside the computers as binary numbers: 1100, 100 and 10101 respectively or hexadecimal numbers: 14, 4, 25. The reason all these numbers represent the same values is that they all follow the number base systems, but on different bases (decimal is base 10, octal is base 8 and binary is base 2).

We represent our decimal numbers using the digits from 0..9 where the position of each digit determines the power of 10 that this digit should be multiplied with. For instance the number 432 is broken down to:

$$432 = (4 \times 10^2) + (3 \times 10^1) + (2 \times 10^0)$$

We can generlize this decimal system to:

$$a_n a_{n-1} \ldots a_2 a_1 a_0 = (a_n \times 10^n) + (a_{n-1} \times 10^{n-1}) + \cdots + (a_2 \times 10^2) + (a_1 \times 10^1) + (a_0 \times 10^0)$$

In fact, this can be generalized to any number with base b:

$$a_n a_{n-1} \ldots a_2 a_1 a_0 = (a_n \times b^n) + (a_{n-1} \times b^{n-1}) + \cdots + (a_2 \times b^2) + (a_1 \times b^1) + (a_0 \times b^0)$$

In this task, you will help write a base-converter that takes in a number presented in b1 and returns the same value in a different base b2. In order to convert from one base to another, we need to do arithmetic in base other than which can get really confusing. So to make things easier, we will convert the number from b1 to decimal, first.

Task 3 (4 points) . *Having understood how base-systems work, you should now be able to write a python function toDecimal(n, b) that takes in any number n written in base b and returns the number in base 10.*

Now all we need to do is convert the result from decimal to b_2. We can do this by repeatedly dividing the number by b_2 and keeping track of the remainders. For example to convert 21 from decimal to hexadecimal (base-8):

$$21 = 8 \times 2 + 5$$

$$2 = 8 \times 0 + 2$$

The remainders in this case are 5 and 2. Notice that the base-8 number of 21 is 25 not 52! By dividing repeatedly, we get the new base's digits in reverse order.

Task 4 (2 points) . *Use the function you created in previous task to create another function convertBase(n, b_1, b_2) that takes n as a number presented in base b_1 and returns this number's representation in base b_2.*

5 Cryptarithm

a cryptarithm is a puzzle where we start with a simple arithmetic statement but then we replace all the digits with letters (where the same digit is replaced by the same letter each time). We will limit such puzzles to strings the form "A+B=C" (no spaces), where A, B, and C are positive integers. For example, "SEND+MORE=MONEY" is such a puzzle. The goal of the puzzle is to find an assignment of digits to the letters to make the math work out properly. For example, if we assign 0 to "O", 1 to "M", 2 to "Y", 5 to "E", 6 to "N", 7 to "D", 8 to "R", and 9 to "S" we get:

```
  S E N D         9 5 6 7
+ M O R E       + 1 0 8 5
---------       ---------
M O N E Y       1 0 6 5 2
```

And so we see that this assignment does in fact solve the problem! Now, we need a way to encode a possible solution. For that, we will use a single string where the index of the letter corresponds to the digit it represents. Thus, the string "$OMY--ENDRS$" represents the assignments listed above (the dashes are for unassigned digits).

Task 5 (5 points) *Write the function solvesCryptarithm(puzzle, solution) that takes two strings, a puzzle (such as "SEND+MORE=MONEY") and a proposed solution (such as "OMY--ENDRS"). Your function should return True if substituting the digits from the solution back into the puzzle results in a mathematically correct addition problem, and False otherwise. You do not have to check whether a letter occurs more than once in the proposed solution, but you do have to verify that all the letters in the puzzle occur somewhere in the solution (of course). You may not use the eval() function.*

6 Hopscotch

Task 6 (4 points) *Write a function called findExit, that takes list of integers as input parameters. Each number in this list represents the number of hops you need to make. You start from index 0 and make as many hops as the number at index 0. Each time you land on an index, make the number of hops at that index. Determine whether you can reach the last index or not. Return True if you reach the last index, and False otherwise. In an empty list,there is no last index, so you can never reach it. For example:*

- *findExit([2,0,1,0]) returns True*
- *findExit([1,1,0,1]) returns False*
- *findExit([1,2,0,3,1]) returns False*

7 Nearest Word

Task 7 (5 points) *Write a function nearestWords(wordlist, word) that takes a sorted wordlist and a single word (all words in this problem will only contain lowercase letters). If the word is in the wordlist, then that word is returned. Otherwise, the function returns a list of all the words (in order) in the wordlist that can be obtained by making a single small edit on the given word, either by adding a letter, deleting a letter, or changing a letter. If no such words exist, the function returns None.*

8 Transpose Matrix

We can use 1-D lists to represent 2-D data. Matrices are an example of 2-D data that we will represent using 1-D lists. Consider the following matrix:

$$\begin{pmatrix} 12 & 3 & -2 & 5 \\ 1 & 24 & 7 & 15 \\ 3 & 9 & 5 & -13 \end{pmatrix}$$

The above matrix can be represented by a 1-D list: [12,3,-2,5,1,24,7,15,3,9,5,-13]. Knowing that the number of columns (numColumns) and rows (numRows) in the 2-D data, given a row (r) and a column (c) we can find the index into the list as:

$index = r * numColumns + c$

Similarly, if we are given an index - Let's say i and we want to find the row and column, we can use the following equations:

$row = index/numColumns$

and

$column = index\%numColumns$

Task 8 (6 points) *Write a function transpose, that takes a list as input parameter. This list represents a matrix where the number of rows and columns are also passed to your function as arguments (in the order: matrix, rows , columns). Your function should find the transpose of the matrix and return that as a 1-D list.*

9 Oh So Many Units

Most questions of physics and chemistry require conversion between units to get the correct units needed for an equation. Einstein's famous equation $E = mc^2$ allows you to find Energy in Joules when you multiply mass in Kilograms by the square of the speed of light in meters per second. For this equation to work correctly, the units of each quantity have to be exact. If I know the mass of an object in pounds (pounds measure force but oh well), I would have to convert pounds to kilograms first and then apply the equation. In this task, we (that means you) will be performing conversions with length, mass, time and volume units.

Task 9 (6 points) *Write a function called convertUnits that takes 4 input arguments. These inputs are fromQuantity, fromUnit, toUnit, and category. "fromQuantity" is an amount that represents a quantity in "fromUnit" units. Your function should convert "fromQuantity" in "fromUnit" units, to a number in "toUnit" units. You should follow the tables below and only use the numbers in the table or else you will not get the points.*

For example, the function call convertUnits(1000.0,"mm","m","length") is asking you to convert 1000.0 millimeters to meters using length category. The return value of the above call should be 1.0.

The function call convertnits(156.5,"lb","kg","mass") converts 156.6 pounds to kilograms and returns 70.987148.

Your function should be able to handle the following units - we will only ask you to convert between units that belong to the same category (for example, we will not ask you to convert between seconds and meters):

Unit	*Description*	*Category*	*Conversion*
g	*grams*	*mass*	*1 g* = 10^{-6} *ton*
g	*grams*	*mass*	*1 g* = 0.0022 *lb*
ton	*tons*	*mass*	*1 ton* = 10^{6} *g*
lb	*pounds*	*mass*	*1 lb* = 453.592 *g*
s	*seconds*	*time*	*1 s* = 0.0003 *hr*
s	*seconds*	*time*	*1 s* = 0.0167 *min*
hr	*hours*	*time*	*1 hr* = 3600.0 *s*
min	*minutes*	*time*	*1 min* = 60.0 *s*
m	*meters*	*length*	*1 m* = 1.0936 *yard*
m	*meters*	*length*	*1 m* = 3.2802 *feet*
m	*meters*	*length*	*1 m* = 39.3701 *inch*
yard	*yards*	*length*	*1 yard* = 0.9144 *m*
foot	*feet*	*length*	*1 foot* = 0.3048 *m*
inch	*inches*	*length*	*1 inch* = 0.0254 *inch*
C	*Celsius*	*Temperature*	*Kelvin = Celsius + 273.15*
K	*Kelvin*	*Temperature*	*Celsius = Kelvin - 273.15*
C	*Celsius*	*Temperature*	*Fahrenheit = (9/5)*Celsius + 32*
F	*Fahrenheit*	*Temperature*	*Celsius = (5/9)*(Fahrenheit - 32)*

Your function should also be able to handle the following prefixes before each unit.

Prefix	*Symbol in function*	*Meaning*
Terra	T	10^{12}
Giga	G	10^{9}
Mega	M	10^{6}
Kilo	k	10^{3}
Deci	d	10^{-1}
Centi	c	10^{-2}
Milli	m	10^{-3}
Micro	u	10^{-6}
Nano	n	10^{-9}
Pico	p	10^{-12}