ENGR 102 - Lab and Assignment #4

Activity #1: To do in lab (Team)

When performing numerical computations, one of the challenges you can run into is floating-point roundoff error. This occurs when the computer needs to represent a number that would require an infinite number of decimal digits, but rounds them off after some point. That small roundoff error can cause some significant issues.

This activity is meant to help you understand floating-point roundoff error a bit more, and learn about one way of dealing with it.

Part 1: Identifying floating-point problems.

We first want to illustrate some examples where floating-point roundoff can cause trouble, and see other cases where it turns out not to.
First, as a team, type in and run the following program:

```
a = 1/7
print(a)
b = a*7
print(b)
```

Notice that the value of a is rounded off. The value of b, if we have no roundoff, should be 1. Is it?
Now try the following program:

```
a = 1/7
print(a)
b = 7*a
print(b)
c = 2*a
d = 5*a
e = c+d
print(e)
```

In this case, the value of e, if we have no roundoff, should be 1. Is it?

Finally, try the following program:

```
Import math
x = math.sqrt(1/3)
print(x)
y = x*x*3
print(y)
z = x*3*x
print(z)
```

Again, the value of y and z, if we have no roundoff error, should be 1 in both cases. Is it?

Was that surprising? You should have seen from those examples that sometimes we will encounter issues due to roundoff error and sometimes we won't. We can't always predict when roundoff error will be obvious.

Your team should write a program to demonstrate generating roundoff error in computations. It should have the following structure:

- Generate at least 8 different examples of roundoff error from numerical computations. You can use small variations of the ones shown above for 2 of them. Try to come up with different types of ways that roundoff can come up in a computation.
- For each of the examples, you should output a line saying: “Is XXX the same as YYY? AAA” where XXX is one way of computing a value and YYY is another, and AAA is either True or False.
    - Note: You can output a Boolean value and it will output either “True” or “False”. For example, a line like `print(1==2)` would output `False`.
- For at least one case, try to see how large of a difference you can get between the two values (i.e. how big you can get the numerical error to become). This usually requires performing multiple operations: one where the numerical error is introduced, then subsequent operations that cause that error to grow.

Part 2: Tolerances for Comparisons.

In part 1, you should have seen that two different ways of computing values that “should” be identical might actually come out differently. In some cases, this is not a problem. For example, we usually don’t care if speed is incorrect in the 10th decimal place, since we usually can’t measure that precisely, anyway. But, floating point error can become a big problem if comparisons are made with floating-point values. Think about the cases from part 1. The Boolean value you output on each line was likely coming out “False” even if the values should have been the same.

A common way for dealing with floating-point error is to use **tolerances**. Tolerances let you compare two values that are close, but not identical to each other. Rather than checking whether a==b, we instead compute a-b, and see if that is within some small distance away from 0 (above or below). To do this, you take the absolute value of (a-b), and check whether it’s less than some small value, called the tolerance. If it is, the two things are considered to be equal. The tolerance is commonly abbreviated TOL or EPS (short for epsilon).

As a team, modify your program from part 1:

- Each case should contain a second line, stating: “Is XXX within tolerance of YYY? AAA”, where now you perform a tolerance-based comparison.
- You should set a single variable, named TOL, TOLERANCE, EPSILON, EPS, or something similar, and use that as the tolerance for your computation.
- Note that if you created a large numerical error in any of your cases, you might find it is outside of the tolerance zone.

Part 3: Tolerance sizes.

Keep in mind that the larger the tolerance, the more numerical error you can tolerate. A large enough tolerance can catch almost any numerical errors. However, this also increases the chances of mistakenly identifying two things that should not be the same as being identical.

Finally, add one more section to your program. You may reuse any of the examples from the earlier parts in this section, as you wish.
You want to create two pairs of numbers: a and b, and x and y, along with a tolerance value (you may use different names for your variables).

- Print a line to clearly separate this part of the program from the earlier parts.
- You should output all values: a, b, x, y, and the tolerance. You should output True or False as to whether a and b are within tolerance, and whether x and y are within tolerance.
- Numbers a and b should be two numbers that SHOULD be the same, but are different due to roundoff error. Feel free to reuse examples from part 1.

- Numbers x and y should be two numbers that SHOULD NOT be the same, but are close to each other.
- You should have a tolerance value that is of a size so that a and b are NOT within tolerance of each other, but x and y ARE within tolerance of each other.
    - Obviously, if you set a tolerance value large enough, you could find that all were within tolerance of each other.
    - And, if you set a small enough tolerance (like 0), you would find that neither is within tolerance.
    - The point of this exercise is to demonstrate to you that you can have cases where tolerances will fail on both sides: they might be too small to catch the cases they should, but too large to exclude cases that shouldn't be the same.

Summary:

Turn in a single program for this activity: it should incorporate all of the code from parts 1-3. There should be 8 examples at the beginning, each with two numbers that are compared for equality and then for being within tolerance of each other. Then, there should be the two examples of numbers that should be equal but are beyond tolerance, and two that should be different but are within tolerance.

As you write programs, please think about your comparisons and decide whether you need to use tolerances or not. If you are making a comparison where you are checking for exact equality and you might have some floating-point error, you probably will want to use tolerances, while if you are just checking which of two things is larger, a tolerance comparison is likely unnecessary. Tolerances are particularly helpful when checking things like whether a denominator is (nearly) 0, and thus a division is likely to create error.

Side note: Tolerances have problems of their own besides the difficulty of setting a "correct" tolerance value. Notably, with tolerances, you can might find that a==b, and b==c, but a != c. Dealing with issues of roundoff error is a long-standing and ongoing area of research.

Save a copy of your .py file. Lab4_Activity1_[Last_name]_[Team #].pdf

Activity #2: To do in lab (Individual)

The purpose of this program is to give you practice with performing Boolean logic.

You may discuss in general terms with your teammates how you can approach these problems.

Part a: Evaluating Booleans

Write a program that reads in true/false values for the variables a, b, and c and evaluates the following Boolean expressions. Your program should thus read in 3 True/False values, and should output 8 values.

1) a and b and c
2) a or b or c
3) (not (a and not b) or (not c and b)) and (not b) or (not a and b and not c) or (a and not b)
4) (not ((b or not c) and (not a or not c))) or (not (c or not (b and c))) or (a and not c) and (not a or (a and b and c) or (a and ((b and not c) or (not b))))

Part b: Writing Boolean expressions

Extend the program above to write Boolean expressions that meet the following criteria:

5) Is true if just one of a or b is true, but not if both are true or both or false

(a) (Note: this is also called an "exclusive or" between a and b.)

6) Is true if an odd number (i.e. exactly 1 or 3) of the variables a, b, and c is true, and is false otherwise.

Part c: Simplifying Booleans

Extend the program from above, but try to create simpler versions of complex Boolean expressions that still give the same results

7) For the expression #3, try to provide a simpler version
8) Likewise for the expression #4

Hint: There are 2 possible values for each of A, B, and C. Any expression that evaluates the same for all possible combinations of True/False of those values is an equivalent Boolean expression!

Each individual should write their own separate program. When all 4 team members have completed their programs, the whole team should look at everyone else's program, to see the alternatives that individuals developed for #5-6.

Save a copy of your .py file. Lab4_Activity2_[Last_name]_[Team #].pdf

Activity #3: To do in lab (Team)

Write a Python program to convert temperatures to and from Celsius, Fahrenheit, Rankine, Kelvin, and Reaumur. Your program should ask the user for a numerical value and the unit the temperature is measured. The program should then ask the user what temperature scale to report the temperature.

Expected Input:
Enter numerical value for temperature: 60
Enter the temperature scale of measured temperature (C-Celsius, K-Kelvin, F-Fahrenheit, R-Rankine, or X-Reumer): F

Expected Output:
60 Celsius is 140 in Fahrenheit

Please do the following in order:

1) Using a word processor, write down a set of human instructions – these should be high quality, descriptive, and use proper variable naming. Once you have completed your instructions, ask a peer TA to review your instructions. Once they are satisfied with the results, they will provide a keyword that you must include as a comment in your program.
2) Once you have an approved set of instructions, create at least three test cases and document these with your instructions.
3) Convert your instructions into code (in the word processor file).
4) Copy your code into an IDE and test your program.

Save a pdf of your word processor document and your .py file. Lab4_Activity3_[Last_name]_[Team #].pdf, Lab4_Activity3_[Last_name]_[Team #].py

Activity #4: To do in lab (Team)

Convert your program in Activity 3 into a function. The function should eliminate the user input statements and use the function arguments as shown below:

Temp_convert(number,input_units,output_units)

The function should return the converted temperature (float) and the output units (string). Test your function using your test cases developed in Activity 3.

Save a copy of your .py file. Temp_convert.py

Activity #5: To do in lab (Team)

Create an algorithm that generates random (x,y) pairs between 0 and 1. Assuming an unit circle centered at the origin, (0,0), the program should output the (x,y) pair and whether it is inside or outside the circle.

Possible Outputs:
(0.2,0.1) is inside the unit circle
(0.9,0.9 is outside the unit circle

You should do the following:

1) Using a word processor, write down a set of human instructions – these should be high quality, descriptive, and use proper variable naming. Once you have completed your instructions, ask a peer TA to review your instructions.
2) Convert the instructions to a flow chart similar to what we have done in class. https://www.programiz.com/article/flowchart-programming
3) Convert your instructions into code (in the word processor file).
4) Copy your code into an IDE and test your program.

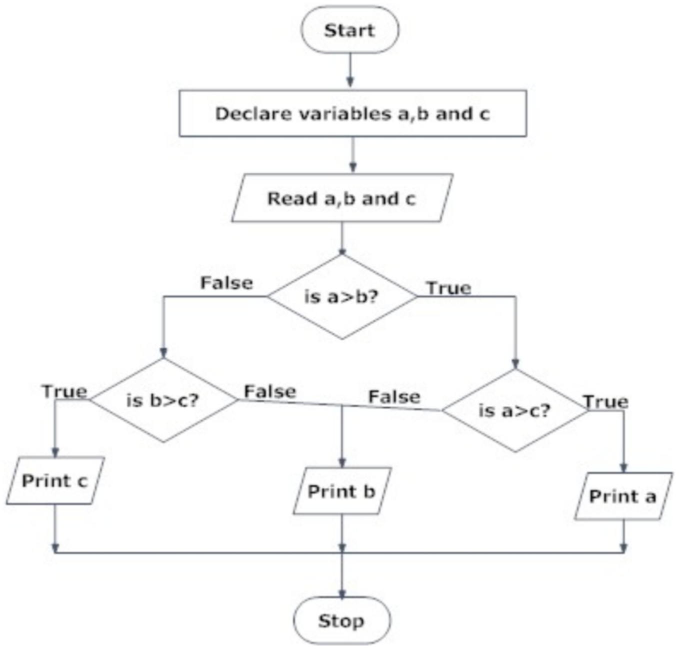
Save a pdf of your word processor document and your .py file. Lab4_Activity5_[Last_name]_[Team #].pdf, Lab4_Activity5_[Last_name]_[Team #].py

Activity #6: To do in lab (Team)

Create an algorithm that follows the logic of the flow chart shown below. The start and stop terms do not have any Python syntax and are used to provide a start and stop of the program. Consider these in comments in your code (i.e. # Start and # Stop). Note that several of the boxes are actual compound statements that will most likely require you to utilize multiple lines of code.

Add comments to the end of your code that provide results/output to the inputs below:

i) a = 5, b = 8, c = 12
ii) a = 5, b = 5, c = 12
iii) a = 5, b = 5, c = 5
iv) a = 8, b = 5, c = 8



Save a copy of your .py file. Lab4_Activity6_[Last_name]_[Team #].XXX

Activity #7: To do in lab (Individual)

Write a Python program that accepts a word from the user and reverse it.
Write a Python program that accepts a word from the user and checks to see if it is a palindrome.

Save a copy of your two .py files. Lab4_Activity7_[Last_name]_[Team #]_a.py,
Lab4_Activity7_[Last_name]_[Team #]_b.py

Completion

Please name your files as directed. You will submit all files in the appropriate submission box. DO NOT COMPRESS.