CS1110 Spring 2015

Assignment 1: Due Friday Feb 6 at 6pm

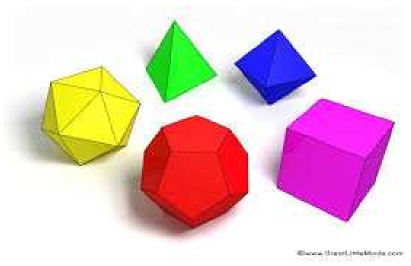
You must work either on your own or with one partner. If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group.

You may discuss background issues and general solution strategies with others, but the programs you submit must be the work of just you (and your partner). We assume that you are thoroughly familiar with the discussion of academic integrity that is on the course website. Any doubts that you have about “crossing the line” should be discussed with a member of the teaching staff before the deadline.

Objectives. Mastering the assignment statement and conditional execution. Practicing with strings and string slicing. Using `print`, `str`, `int`, `len`, `import` and `raw_input`. You will get experience bridging the gap from math formula to Python code. You will get practice processing strings according to given rules. Finally, your ability to manipulate files and directories and the Komodo editor will be enhanced. The assignment is based on Lectures 1,2, and 3 and Labs 1 and 2.

1 Music of the Spheres

There are five Platonic solids:



The *tetrahedron* has 4 faces, the *cube* has 6 faces the *octahedron* has 8 faces, the *dodecahedron* has 12 faces, and the *icosahedron* has 20 faces. All the edges of a Platonic solid have the same length. Platonic solids have many interesting properties. Here are a couple that relate to the cube:

1. If the edge of a cube has length E, then the radius of the largest sphere that can be put inside is given by $r = E/2$.

2. If the edge of a cube has length E, then the radius of the smallest sphere that encloses the cube is given by $R = (\sqrt{3}/2)E$.

We say that r is the *inradius* of a cube with edge E and that R is its *outradius*. Here is table of inradii and outradii for each the Platonic solid:

Solid	r	R
Tetrahedron with edge E	$\frac{\sqrt{6}}{12}E$	$\frac{\sqrt{6}}{4}E$
Cube with edge E	$\frac{1}{2}E$	$\frac{\sqrt{3}}{2}E$
Octahedron with edge E	$\frac{1}{\sqrt{6}}E$	$\frac{1}{\sqrt{2}}E$
Dodecahedron with edge E	$\frac{\sqrt{250+110\sqrt{5}}}{20}E$	$\frac{\sqrt{15}+\sqrt{3}}{4}E$
Icosahedron with edge E	$\frac{3\sqrt{3}+\sqrt{15}}{12}E$	$\frac{\sqrt{10+2\sqrt{5}}}{4}E$

Thus, the radius of the largest sphere that can be placed inside an octahedron with edge E is given by $r = E/\sqrt{6}$.

The facts in the table can be used to investigate an interesting idea that the astronomer/astrologer Kepler had about planetary orbits. Consider the following "Platonic Nesting" procedure:

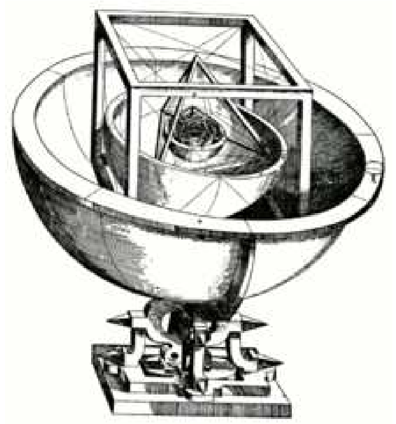
- Start with a sphere S_1 with radius 1. ($\rho_1 = 1$)
- Let O be the smallest octahedron that encloses S_1.
- Let S_2 be the smallest sphere that encloses O. Call its radius ρ_2.
- Let I be the smallest icosahedron that encloses S_2.
- Let S_3 be the smallest sphere that encloses I. Call its radius ρ_3.
- Let D be the smallest dodecahedron that encloses S_3.
- Let S_4 be the smallest sphere that encloses D. Call its radius ρ_4.
- Let T be the smallest tetrahedron that encloses S_4.
- Let S_5 be the smallest sphere that encloses T. Call its radius ρ_5.
- Let C be the smallest cube that encloses S_5.
- Let S_6 be the smallest sphere that encloses C. Call its radius ρ_6.

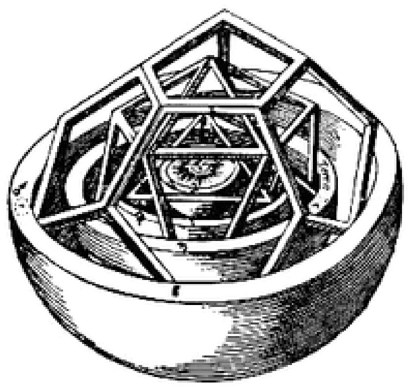
Kepler's "music of the spheres conjecture" is that

$$\frac{\rho_2}{\rho_1} = \frac{\text{Venus Orbit Radius}}{\text{Mercury Orbit Radius}} \qquad \frac{\rho_3}{\rho_2} = \frac{\text{Earth Orbit Radius}}{\text{Venus Orbit Radius}} \qquad \frac{\rho_4}{\rho_3} = \frac{\text{Mars Orbit Radius}}{\text{Earth Orbit Radius}}$$

$$\frac{\rho_5}{\rho_4} = \frac{\text{Jupiter Orbit Radius}}{\text{Mars Orbit Radius}} \qquad \frac{\rho_6}{\rho_5} = \frac{\text{Saturn Orbit Radius}}{\text{Jupiter Orbit Radius}}$$

In other words, the spheres that arise in the Platonic nesting could be used to predict the "spacing" of the planets. Below are schematics from the 1500s that depict the nesting. On the left you can see the Saturn, Jupiter, and Mars spheres, i.e., S_6, S_5, and S_4. The right schematic is on a much smaller scale, but the Mars, Earth, Venus, and Mercury spheres are in there, i.e., S_4, S_3 S_2, and S_1.





Music of the Spheres

In this problem you write a Python script `Kepler.py` that checks out Kepler's conjecture. The entries in the inradius/outradius table can be used to compute the sphere radii ρ_2, ρ_3, ρ_4, ρ_5, and ρ_6. For example,

Q: What is the smallest cube that encloses a sphere of radius ρ_5?

A: Using the cube inradius formula we conclude that the cube will have edge $E = 2\rho_5$.

For your information, the six planets orbits in question have these radii:

Planet	Orbit Radius
Mercury	35.98
Venus	67.24
Earth	92.90
Mars	141.60
Jupiter	483.80
Saturn	890.70

A template script is provided on the website that includes this information.

You now have all the facts and ideas necessary to write `Kepler.py`. When it is run, the script is to display the Kepler ratios and the actual orbit-radii ratios in a table like this:

```
Planet-Pair       Kepler   Actual
---------------------------------
Venus:Mercury      1.73     1.87
Earth:Venus        x.xx     x.xx
Mars:Earth         x.xx     x.xx
Jupiter:Mars       x.xx     x.xx
Saturn:Jupiter     x.xx     x.xx
```

We have reported the results for the Venus-Mercury ratios as a courtesy so that you can check that you are on the right track as you develop your program.

Strong advice: don't try to do everything at once. Start by computing the Venus-Mercury ratios and display its line in the output table. Then proceed to handle the Earth-Venus ratios etc. Never move on to the next "ratio computation" until everything is working.

The second and third columns in the table should be nicely aligned and show the ratios to two decimals. Choose good variable names and write succinct, informative comments that enable the reader to follow along as you encode the nesting process. In future assignments we will take points off for poor style and we will do it here if your effort is minimal. However, in this assignment you will just be given feedback if your commenting is substandard. Finally, you are NOT allowed to use loops or arrays should you already know about these Python constructions. Submit your finished rendition of `Kepler.py` to CMS.

2 You Are Late!

Suppose you show up 90 minutes late for a 2:45 appointment . The receptionist says "You missed your 1:15 appointment!" In this problem you write a script `YouAreLate.py` that solicits an "arrival time" using `raw_input` and then prints out the receptionist's admonition for your 90-minute mistake. Here are some sample "dialogs":

```
Enter the time in the form hh:mm :  03:46
You missed your 02:16 appointment!

Enter the time in the form hh:mm :  03:10
You missed your 01:40 appointment!

Enter the time in the form hh:mm :  01:05
You missed your 11:35 appointment!

Enter the time in the form hh:mm :  01:50
You missed your 12:20 appointment!
```

User responses shown in red.

Input is a length-5 string where the first two characters are

`'01'`, `'02'`,...,`'11'`, or, `'12'`

and the last two characters are

`'00'`,`'01'`,...,`'58'`, or `'59'`

and the third character is a colon. Observe from the dialogs displayed above that the 90-minute-earlier time is to be conveyed in the same "hh:mm" style. Here are some examples:

Arrival Time String	90-minute-Earlier Time String
`'02:45'`	`'01:15'`
`'10:15'`	`'08:45'`
`'12:05'`	`'10:35'`
`'01:10'`	`'11:40'`

Two observations:

1. Regarding the "minute part" sometimes you subtract 30 and sometimes you add 30.

2. Regarding the "hour part" sometimes you subtract 1 and sometimes you subtract 2. And after you do that you may have to make an adjustment. Look at the last example in the table where the interviewee shows up at 1:10. If you subtract two hours from that you get -1:10 o'clock!

Here are some useful tips:

- Start the development of your implementation by setting up the acquisition of the input string. Use `raw_input`. Your program may assume that the input string is in the right format. `01:34` is OK. `1:34`, `123:4`, `one-thirty-four` are not OK. Your program is not expected to handle "bad input". (Later on in the course we will talk about graceful ways of handling this kind of user carelessness.)

- Get hold of the hour and minute portions of the input string by appropriate string slicing.

- Convert those substrings into integer values using `int`.

- Do the arithmetic as suggested above to compute the hour and minute values H and M of the (missed) appointment time.

- Make sure this part of the solution is working before proceeding further. Do this by printing out the values of H and M and confirming by hand that the calculations are correct. Make sure that your testing covers all the necessary cases.

- Once you get the integers H and M you have to produce the right 5-character string encoding for the 90-minute-earlier time. Remember that the `str` function can be used to convert an integer value like `15` to a string like `'15'`. You will have to do some concatenation if H or M are a single digit. For example, here is how to display the integer value 5 as a 2-character string: `'0'+ str(5)`.

- As illustrated above, you "embed" your length-5 string encoding of the appointment time in an "admonition" string. Thus, if the appointment time is 11:40, then your script prints

```
You missed your 11:40 appointment!.
```

Submit your implementation of `YouAreLate.py` to CMS.

3 Pluralizer

In this problem you write a script `Pluralizer.py` that prompts the user for string input using `raw_input` and then computes and displays its "plural" according to these rules:

1. If the last character in the input string is `'y'`, then change the `'y'` to `'i'` and add `'es'`. For example, `'diary'` becomes `'diaries'`.
2. If the last two characters are `'ro'` , `'to'` , or `'no'`, then add `'es'`. For example `'hero'` becomes `'heroes'`, `'potato'` becomes `'potatoes'`, and `volcano'` becomes `'volcanoes'`.
3. If the last character in the input string is `'s'`, `'h'`, or `'x'`, then add `'es'`. For example, `'mess'` becomes `'messes'`, `'fish'` becomes `'fishes'`, and `'sex'` becomes `'sexes'`.
4. If the last two characters are `'fe'`, then replace the `'fe'` with `'ves'`. For example, `'life'` becomes `'lives'`.
5. If none of the above situations apply, then just add `'s'`. For example, `'house'` becomes `'houses'`.

Put aside what you know about pluralization of English words because the above rules can get it wrong (and that's OK). For example, according to Rule 1, `'boy'` becomes `'boies'`! Moreover, the rules can be applied to *any* string, not just English words. Thus `'icu2'` becomes `'icu2s'`.

The heart of your implementation of `Pluralizer.py` is an `if-elif` construction that handles the five situations. Your script should produce dialogs like this:

```
Enter a string:  diary
The plural of ``diary'' is ``diaries''.

Enter a string:  potato
The plural of ``potato'' is ``potatoes''.

Enter a string:  mess
The plural of ``mess'' is ``messes''.

Enter a string:  wife
The plural of ``wife'' is ``wives''.

Enter a string:  2x1+=?
The plural of ``2x1+=?''  is ``2x1+=?s''
```

Submit your implementation of `Pluralizer.py` to CMS.