CPE 101
Project 1

Objectives:

- To develop a basic Python program
- To perform keyboard input and screen output
- To use mathematical operators to perform simple calculations
- To use functions from existing libraries
- To develop and use your own functions
- To use conditional branching

Collaboration:

You may not collaborate in any way on your project.

Sources you may consider for help:

- Your instructor, via office hours, email, etc.
- Free tutoring Sunday-Thursday, 7-9 PM, 14-302
- Your TA

Project Specifications:

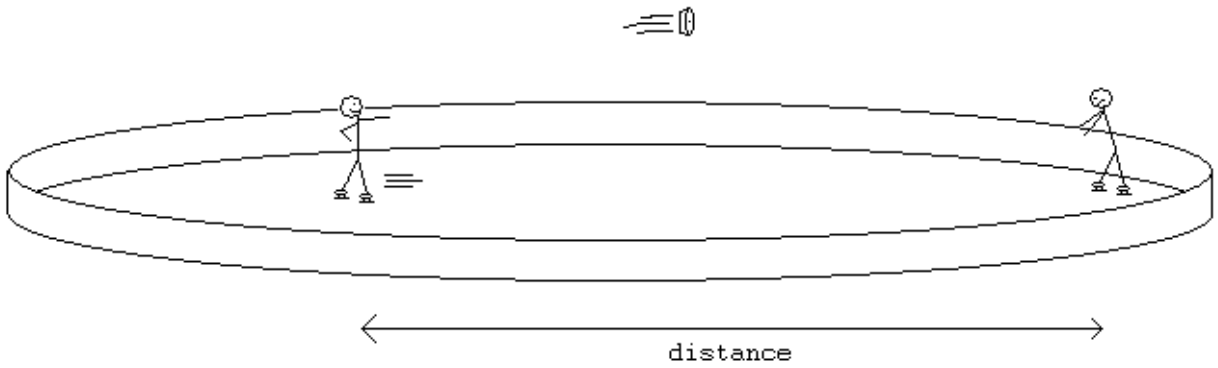
- You must have header comment at the top of all your source files.
- The header includes:

```
# Project 1
#
# Name: Your Name
# Instructor: S. Einakian
# Section: Your Section
```

- For every function, you must have purpose statement and signature.

Problem Description

You are on your winter break enjoying a relaxing skate at the local ice rink, when you notice that your professor is "skating" next to the railing holding on for dear life. You look down to see that conveniently next to you are a rotten tomato, a banana cream pie, a large rock, a light saber, and a lawn gnome. Taking advantage of the opportunity, you pick up one of the items and throw it at your professor. Assuming you are standing still on your skates when you throw the object and that you hit your professor directly in the head, what velocity will you be traveling backwards after the throw?
A beautiful diagram of the problem is shown bellow:



First we must figure out the velocity of the object in the horizontal direction of the professor. To make the problem easier the skater will always throw the object at a 45 degree angle. Then the horizontal velocity of the object will be:

$$\boldsymbol{veocityObject} = \sqrt{\frac{\boldsymbol{gravity} * \boldsymbol{distance}}{\mathbf{2}}}$$

To determine the resulting velocity of the skater in the opposite direction, we use the Conservation of Momentum. That is, the (mass * velocity) of the object is equal the (mass * velocity) of the skater in the opposite direction.

$$\mathbf{velocitySkater} * \mathbf{massSkater} \;=\; \mathbf{massObject} * \mathbf{velocityObject}$$

The resulting equation is:

$$\mathbf{velocitySkater} = \frac{\mathbf{massObject} * \mathbf{velocityObject}}{\mathbf{massSkater}}$$

Your task is to write a program simulating the above problem.

Program inputs:

- distance - the distance (in meters) between the skater and the professor
- object – a character representing the object the skater chooses to throw
- weight of skater - the weight (in pounds) of the skater

Program output:

- a comment based on weight of the object and the distance of the throw
- velocity - the backwards velocity (meters/second) of the skater after the throw
- *possibly* a comment based on the velocity of the skater

Sample Runs:

Note 1: These are sample runs - many other results are possible with many different inputs - be sure to read the specifications carefully for details.

Note 2: All user input is **bold** in the sample runs for clarity only - your program is not expected to behave this way.

Note 3: The output to the right of the ":" (if not user-input which is bold) is a combination of calculated values and hard-coded text.

Sample Run 1:

How much do you weigh (pounds)? **140**
How far away your professor is (meters)? **30**
Will you throw a rotten (t)omato, banana cream (p)ie, (r)ock, (l)ight saber, or lawn (g)nome? **t**
Nice throw! You're going to get an F!
Velocity of skater: 0.019 m/s
My grandmother skates faster than you!

Sample Run 2:

How much do you weigh (pounds)? **100**
How far away your professor is (meters)? **50**
Will you throw a rotten (t)omato, banana cream (p)ie, (r)ock, (l)ight saber, or lawn (g)nome? **g**
Nice throw! RIP professor.
Velocity of skater: 1.829 m/s
Look out for that railing!!!

Specification:

1. For this assignment you will define four functions to break the problem into smaller parts. Write your functions in a file named **funcs.py**. Your functions must be named exactly as specified below.

    1) **poundsToKG (pounds)**
    2) **getMassObject (object)**
    3) **getVelocityObject (distance)**
    4) **getVelocitySkater (massSkater, massObject, velObject)**

poundsToKG (pounds): this function takes as an input parameter a weight in pounds. It returns the equivalent mass in KG. *You must use the constant 0.453592 in your function to pass my test cases.*

$$kilograms = pounds * 0.453592$$

getMassObject (object): this function takes as an input parameter the character representing which object to throw. It returns the mass of the object (kg).

Object	Character	Mass
tomato	't'	0.1
banana cream pie	'p'	1.0
rock	'r'	3.0
lawn gnome	'g'	5.3
light saber	'l'	9.07

If the input parameter isn't one of the 5 expected characters the function should return 0.0.

getVelocityObject (distance): this function takes as an input parameter the distance of the professor (m). It returns the velocity of the object (m/s). Use 9.8 for gravity.

$$\boldsymbol{veocityObject} = \sqrt{\frac{\boldsymbol{gravity} * \boldsymbol{distance}}{\mathbf{2}}}$$

getVelocitySkater (massSkater, massObject, velObject): this function takes as input parameters the mass of the skater (kg), the mass of the object (kg), and the velocity of the object (m/s). It returns the velocity of the skater (m/s). You may assume the mass of the skater will be greater than zero.

$$\mathbf{velocitySkater} = \frac{\mathbf{massObject} * \mathbf{velocityObject}}{\mathbf{massSkater}}$$

2. Test your functions using unit testing (as discussed in lecture and lab) in a file named **funcs_tests.py**. You must write ***at least* two test cases** for each function. You also must test *every path* through each function. This means that some functions may require more than two test cases! I strongly recommend writing your functions and testing them before continuing to write the rest of the program.

Note: that your functions will be tested thoroughly using test cases that you have not seen. Test your functions so that you are satisfied you have implemented them correctly. *Remember to name them exactly as specified above or they will not pass my test cases!*

3. Write the "main" part of your program in a file named **skater.py**. Write your code in a function named **main**. You will need to include this code:

```
def main():
# your code goes here

if __name__ == '__main__':
    main()
```

4. Your program prompts must match the specified prompts ***exactly*** (spelling, capitalization, et cetera).
5. Your program outputs must match the specified outputs ***exactly*** (spelling, capitalization, number of decimal places, et cetera).

6. The velocity output should be shown to three decimal places.
7. Your program will be tested with input you have not seen and must work correctly for any valid distance (>= 0), weight (> 0), and chosen object.
8. Use the sqrt function in the math library to do the square root calculation. You will need to import math.
9. After taking all input, print "Nice throw!" and the following comments based on the mass of the thrown object and the distance thrown:
    a. mass <= 0.1 - "You're going to get an F!"
    b. mass > 0.1 and <= 1.0 - "Make sure your professor is OK."
    c. mass > 1.0:
        i. if the distance thrown is less than 20 meters - "How far away is the hospital?"
        ii. if the distance is greater than or equal to 20 meters - "RIP professor."
10. After displaying the velocity of the skater, possibly display a comment based on the velocity:
    a. velocity < 0.2 - "My grandmother skates faster than you!"
    b. velocity >= 0.2 and < 1.0 - Don't display a comment
    c. velocity >= 1.0 - "Look out for that railing!!!"

Testing:

1. Test your functions in funcs.py using Python unit testing as described in the Specification section.
2. To test your finished program I will provide you with two sample input files containing input for a few "test cases". I will also provide you with the instructor solution executable. Output files from your program should match the output from the instructor's solution *exactly*.
3. Here are some instructions on how to use diff and how to create the output files.

diff analyzes two files and prints the lines that are different. Essentially, it outputs a set of instructions for *how to change one file to make it identical to the second file.* For example, the following command compares out1 and myout1 files then returns nothing if both files are same. Otherwise, shows the differences between files.

$ diff out1 myout1

4. You may copy the sample input files and instructor executable from the PolyLearn

5. The sample input files are for **some** of the test cases. It is strongly advised that you try other cases as well. I will use other test cases when grading your program.

Run the given sample

skaterInst is an executable file that you can run to check how your output looks like. For running this file:

1) Login to the server
2) Create a directory Project1
3) Copy files to your directory
4) Type the following command for testing in1 input file:

   ./skaterInst < in1 > myout1
5) (Note: If there is error on permission to access to the files. Use the following command to fix the permission error:
   chmod +x skaterInst)
6) Now compare your output with given out1 file by the following command:
   diff out1 myout1

Grading – Read this carefully

Your program grade will be based on the number of test cases passed, program style, adherence to the specification, and perfect matching of program output to the instructor solution output. Your program will be tested with test cases that you have not seen.

Be sure to test your program thoroughly!!! To pass a test case, your output must match mine EXACTLY.

Use diff to make sure your output matches mine in the test cases given and in test cases that you make up. If there are any differences whatsoever, you will not pass the test case.

Submission:

Submit the following files to the PolyLearn

- skater.py
- funcs.py
- funcs_tests.py

Project 1 Grading Rubric:

• skater.py submitted and all function calls are written	10 Points
• runs without error	10 Points
• funcs.py submitted and all functions are written	10 Points
• runs without error	10 Points
• funcs_tests.py submitted with all test cases	10 Points
• Comments	10 Points
• Test with in1 no difference with solution	20 Points
• Test with in2 no difference with solution	20 Points
	========
	100 Points