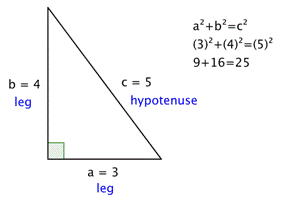
1. **(25 Points)** Write a complete, working Python 3 program called euidA.py (where “euid” is your EUID) that does the following:

* Write your EUID in comments at the top of the file.
* Consider the following diagram of a right triangle:



Given the length of leg a and leg b, the length of the hypotenuse of the right triangle can be calculated using .

* Define a namedtuple called RightTriangle with the attributes 'a', 'b', and 'c', for the legs and hypotenuse of a right triangle, respectively.
* Prompt for and read in the length of leg a of the right triangle as a floating-point number. If the user enters a non-positive number (i.e., 0 or a negative number) for leg a, print out an error message and terminate the program.
* Otherwise, if the user enters a positive number for leg a, generate a random number between 5 and 10, inclusively, and assign the result to b.
* Using the formula given above, calculate the length of the hypotenuse and assign the result to c.
* Create a new namedtuple called my\_triangle using the variables a, b, and c as applicable.
* Print 'my\_triangle: ' and the values for legs a and b and the hypotenuse c, formatted to show 2 decimal places. Note that you must use the attributes from the my\_triangle namedtuple when specifying the values in the print statement.
* You may assume the user enters all elements using the appropriate data type. Due to time constraints, no further comments are required.

Here is a sample output to help you write the code. The items in bold are entered by the user.

$ **python3 mat0299A.py**

Enter length of leg 'a' of right triangle: **-3**

Error: leg 'a' must be positive

$ **python3 mat0299A.py**

Enter length of leg 'a' of right triangle: **0**

Error: leg 'a' must be positive

$ **python3 mat0299A.py**

Enter length of leg 'a' of right triangle: **7.29**

my\_triangle: a=7.29 b=6.00 c=9.44

1. **(25 Points)** Write a complete, working Python 3 program called euidB.py (where “euid” is your EUID) that does the following:

* Write your EUID in comments at the top of the file.
* Create a price list dictionary called price\_list that have the following names of fruit and price per pound:
  + bananas 0.58
  + apples 1.69

where the names of fruit are strings and the price per pound are floating-point numbers.

* Prompt for and read in the name of the fruit to add to the price list.
* Prompt for and read in the price per pound of the fruit as a floating-point number.
* Then, add the fruit and price per pound just read in to the price list.
* Print the price list (both keys and values).
* Prompt for and read in the name of the fruit you wish to purchase.
* Prompt for and read in the number of pounds of the fruit you wish you buy as an integer.
* If the name of the fruit is found in the price list:
  + Compute the total cost of the fruit based on the number of pounds being purchased.
  + Print a message describing the purchase, being sure to include the number of pounds, name of the fruit, and total cost, formatted to 2 decimal places.
* If the name of the fruit was not found in the price list, simply print that the fruit was out of stock.
* You may assume the user enters all elements using the appropriate data type. Due to time constraints, no further comments are required.

Here is a sample output to help you write the code. The items in bold are entered by the user.

$ **python3 mat0299B.py**

Enter name of fruit to add: **pears**

Enter price per pound: **1.19**

{'bananas': 0.58, 'apples': 1.69, 'pears': 1.19}

Enter name of fruit you wish to purchase: **apples**

Enter number of pounds you wish to buy: **3**

3 pounds of apples totals $5.07

$ **python3 mat0299B.py**

Enter name of fruit to add: **pears**

Enter price per pound: **1.19**

{'bananas': 0.58, 'apples': 1.69, 'pears': 1.19}

Enter name of fruit you wish to purchase: **oranges**

Enter number of pounds you wish to buy: **4**

oranges out of stock