Function Name: `applesAndOranges`

Inputs:

1. *(double)* The number of apples
2. *(double)* The number of oranges
3. *(double)* The number of good apples
4. *(double)* The number of good oranges

Outputs:

1. *(double)* The probability of randomly picking a bad apple
2. *(double)* The probability of randomly picking a bad orange

Function Description:

This function should calculate the probability of randomly picking a bad apple or bad orange from a bag. The number of apples and oranges will be given in the first two inputs and the number of good apples and good oranges will be given as the third and fourth inputs. The output value should be expressed as a percentage, not a decimal. So if there was a 25% chance of drawing a bad apple, this would be expressed as the integer 25 rather than the decimal 0.25. Round all percentages to the nearest hundredth of a percent.

Notes:

- The number of good apples/oranges will always be less than or equal to the number of apples/oranges.

Function Name: `lawOfCosines`

Inputs:

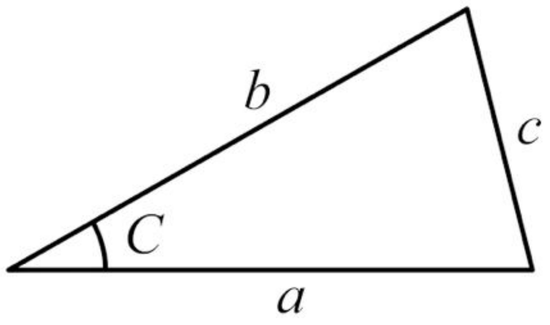
1. *(double)* Length of side a
2. *(double)* Length of side b
3. (*double)* The angle between sides a and b (C in the diagram) in degrees

Outputs:

1. *(double)* Length of side c

Function Description:

An SAS triangle (Side, Angle, Side) is a triangle where we know the length of two sides and the angle between them. It is one of the the combinations of three pieces of information we can use to solve for any other part of the triangle. The Law of Cosines is normally used to calculate the unknown side:



$$c^2 = a^2 + b^2 - 2abcosC$$

Solve for the length of the third side of a triangle (*c in the picture)* given the opposite angle and two other side lengths, as shown in the picture above. Round your answer to the hundredths decimal place.

Notes:

- MATLAB has a built in function for the evaluation of cosine in degrees: `cosd().`
- Angle C will always be less than 180 degrees.

Function Name: `gravity`

Inputs:

1. *(double)* Mass of object 1
2. *(double)* Mass of object 2
3. *(double)* Distance between centers of the objects

Outputs:

1. *(double)* Acceleration of object 1

Function Description:

You may be familiar with Newton's law of universal gravitation that states that every particle attracts another by use of a force that is directly proportional to the product of their masses and inversely proportional to the square of the distances between them, otherwise represented by the following equation:

$$F = G\frac{m_1 m_2}{r^2}$$

Where:

F is the force between masses
G is the gravitational constant = 6.67e-11
m_1 is the first mass
m_2 is the second mass
r is the distance between the centers

Using this, and Newton's second law:

$$F = ma$$

Calculate the magnitude of the acceleration of object 1. You should round your answer to the nearest hundredths decimal place.

Notes:

- You can assume that all the parameters are in the correct units (do not unit convert)

Function Name: `clockHands`

Inputs:

1. *(double)* The current position of the hour hand
2. *(double)* The current position of the minute hand
3. *(double)* A positive or negative number of minutes

Outputs:

1. *(double)* The position of the hour hand after the specified time
2. *(double)* The position of the minute hand after the specified time

Function Description:

It is not always immediately obvious where the hands of a clock will be after a certain amount of time. It is even harder to visualize where the hands *were* some amount of time in the past. Luckily, this is not a very difficult problem for a computer to solve, so you will use that to your advantage.

This function will take in the current position of the hour hand, as an integer between 0 and 11 (0 for noon/midnight), the current position of the minute hand, as an integer between 0 and 59 (0 for "on-the-hour") and a positive or negative number of minutes elapsed. Given this information, calculate the new position of the clock hands. You should assume that the hour hand does not move until the next hour has begun. For example, the hour hand stays on "2" from 2:00 until 2:59 and only at 3:00 does the hour hand move to "3".

Notes:

- The `mod()` and `floor()` functions will be useful.
- As you do this problem notice the behavior of `mod()` for negative inputs. This is a very important function in programming and will come up again in the class!

Hints:

- One way of solving this problem involves calculating the total number of minutes after noon/midnight before and after the given minutes have elapsed.
- Another way of solving it involves splitting the given number of minutes into a number of hours and a number of minutes.
- Pick whichever method makes more sense to you (or come up with your own method).