Unit Five

First of all, if you haven't done so, create an account at **Project Euler**. Once you have done that, go to the "Friends" area and become my friend. When you create a name, make sure it is your name. In the past, students have made up stupid names that I can't identify. I won't give you credit for any problem that I can't verify. My code is 598884_2b1218892c27ef616ec8791c9bd5e8e3. I'll be able to track your progress through the website.

For all of these problems, create functions as appropriate to break them into testable parts. The Euler Problems all have a few test cases. Make assertion statements with these cases for testing purposes. I'll track what problems you have done through the Euler website. You will still be required to submit your code for all of the problems you solve. In addition to the Euler Problems, I offer others. They are shown below. When and if more are created, I'll let you know.

For the remainder of the semester, you have to turn in 75 points every two weeks. Your points can come from any combination of Euler Problems or the others I have provided. I'm also open to you doing something else that you have interest in (say from Linear Algebra). Just let me know what you would like to do and we'll decide if it's appropriate for this class.

This table gives values for Euler Problems. Essentially, the more people that have solved an Euler problem, the easier it is (Not always true, but more or less so). I'm going to try to add several more problems that are beyond #50. We have had students do those in the past. Just let me know and we'll come up with a point value. Usually, Euler Problems are 5 or 10 points.

Euler Problem Point Values #1-50

ID	Description/Title	Points		ID	Description/Title	Points
1	Multiples of 3 and 5	3		26	Reciprocal cycles	5
2	Even Fibonacci numbers	3		27	Quadratic primes	5
3	Largest prime factor	3		28	Number spiral diagonals	4
4	Largest palindrome product	3		29	Distinct powers	4
5	Smallest multiple	3		30	Digit fifth powers	4
6	Sum square difference	3		31	Coin sums	5
7	10001st prime	3		32	Pandigital products	6
8	Largest product in a series	3		33	Digit cancelling fractions	6
9	Special Pythagorean triplet	3		34	Digit factorials	5
10	Summation of primes	3		35	Circular primes	5
11	Largest product in a grid	3		36	Double-base palindromes	5
12	Highly div triangular number	3		37	Truncatable primes	6
13	Large sum	3		38	Pandigital multiples	7
14	Longest Collatz sequence	3		39	Integer right triangles	6
15	Lattice paths	3		40	Champernowne's constant	5
16	Power digit sum	3		41	Pandigital prime	6
17	Number letter counts	3		42	Coded triangle numbers	6
18	Maximum path sum I	3		43	Sub-string divisibility	7
19	Counting Sundays	3		44	Pentagon numbers	8
20	Factorial digit sum	3		45	Triangular, pent, and hex	6
21	Amicable numbers	3		46	Goldbach's other conjecture	7
22	Names scores	3		47	Distinct primes factors	7
23	Non-abundant sums	4		48	Self powers	4
24	Lexicographic permutations	4		49	Prime permutations	7
25	1000-digit Fibonacci number	3		50	Consecutive prime sum	7

A final Note about the Euler Problems: "Solutions" are easily available online for all of these problems and most of the others. You are expected to do your own work. I'll probably be able to tell if you copied your solution from other places, so don't do that.

For each submission, you have to turn in code that represents a solution to a problem and some means to keep track of the points associated with every problem. There is no starting point. I've included an example (submissionExample.py that can be found in the Unit Five Data Folder) that gives you an idea of what you should do. It even gives you your first 6 points if you want to use it.

Note that guassianElimination is required. It is a course requirement that has to be accomplished before the end of the semester.

I've added everyone to a shared folder Unit Five Data. There are a variety of data files and sample programs there. Several Euler problems require data. Look in the folder first because I've taken care of a lot of the details. If you're having problems with any of the data files, let me know.

findSmithNumbers(n) (10 points)

Finds all Smith Numbers less than n. Return is a list with the Smith Numbers. Here is a Wikipedia Description of Smith Numbers. To do this, you will need these functions: getPrimes(n), findPrimeFactorization(n) and probably a few others of your choosing. Test by generating the Smith Numbers up to 1000 and comparing to the list on Wikipedia.

findRationalZeros (P) (10 points)

Finds all rational zeros of a polynomial. Input is a list with polynomial coefficients in descending order of degree—including all zero terms. To learn more about rational zeros, check the Wikepedia Description of the Rational Root Theorem. Return is a list with the rational zeros in ascending order. This function will be used later in a function that factors polynomials. Hopefully. Examples:

findRationalZeros ([1,5,6])==[-3,-2]
findRationalZeros ([1,0,0,8])==[-2]
findRationalZeros([10, -101, 184, 573, -954, -1512])==[-1.5, -1.4, 3.0, 4.0, 6.0]

simpleSyntheticDivision(P,z) (5 points)

simpleSyntheticDivision(P,z) **s**ynthetically substitutes a real single zero (z) into a Polynomial (P). If there is no remainder, return is a list of coefficients of the polynomial that is one degree less than P that divides evenly into P. Examples:

simpleSyntheticDivision([1,5,6],-3)==[1, 2]
simpleSyntheticDivision([1,0,0,8],-2)==[1, -2,4]
simpleSyntheticDivision([10, -101, 184, 573, -954, -1512],3)==[10, -71, -29, 486, 504]

simplifyRadical(n,r) (5 points)

simplifyRadical(n,r) takes in an integer value n and simplifies the 'rth' root of n. A simplified radical contains no factor of the radicand that is a root of the index of the radical. Some examples are:

$\sqrt{50} = 5\sqrt{2}$ $\quad\quad$ $\sqrt{50} \to \sqrt{5^2 \bullet 2} \to 5\sqrt{2}$

$\sqrt[3]{1053} = 3\sqrt[3]{39}$ $\quad\quad$ $\sqrt[3]{1053} \to \sqrt[3]{3^4 \bullet 13} \to 3\sqrt[3]{3 \bullet 13} \to 3\sqrt[3]{39}$

$\sqrt[4]{68040000} = 30\sqrt[4]{84}$ $\quad\quad$ $\sqrt[4]{68040000} \to \sqrt[4]{2^6 \bullet 3^5 \bullet 5^4 \bullet 7} \to 2 \bullet 3 \bullet 5\sqrt[4]{2^2 \bullet 3 \bullet 7} \to 30\sqrt[4]{84}$

Return is a tuple with two values; the first is the value outside the radical and the second the value within. So:

simplifyRadical(50,2)==(5,2)
simplifyRadical(1053,3)==(3,39)
simplifyRadical(68040000,4)==(30,84)

triangleInteriors() (20 points)

Attached is a file triangleInteriors.txt of approximately 10,000 sets of 4 ordered pairs in this format:
-7401,-9314,-8573,4856,8924,3773,6368,5345
Consider each line to be 4 ordered pairs like this:
A(-7401,-9314),B(-8573,4856),C(8924,3773),D(6368,5345)

Above is the first line of the file, but for our purposes it will be easier to use the simple example:
A(0,6),B(-5,-1),C(5,-1),D(0,0)
If you graph the four points it is easy to see that D is internal to triangle ABC while A,B and C are not internal to the other possible triangles.

Define the function I(D,ABC) as a Boolean function. I is True if D is internal to triangle ABC and False if D is on any segment of triangle ABC or exterior to triangle ABC.

Define N(I) as the number of True values for a collection of sets of Points A,B,C and D.

You are given: N(I(D,ABC))=787 for the file.

Find N(I(A,BCD)), N(I(B,ACD)) and N(I(C,ABD))

solveSudoku(board) (15 or 30 points)

Go here for a description of how to solve sudoku puzzles. Carefully read the techniques on how to solve the puzzles. The easiest puzzles are solvable by uncovering "naked singles." From my limited experience, I have found that by exposing all naked singles, updating the grid and repeating the process, the easiest puzzles can be solved. Fifteen iterations are plenty for this method. Any more than that is just wasted processing time. There are other techniques that experienced puzzle solvers use, but we will not do any of that. We will limit our methods to updating through naked singles and then using recursive backtracking. There are many websites/youtube videos available to help you with the recursive backtracking.

I've posted a file solveSudokuTestCases.py that has four 3D lists. All four have 6 members, each of which is a 9x9 2D list.

List A contains 6 unsolved Sudoku puzzles that can be solved exclusively by naked single methods.
List B contains the solutions to the puzzles of List A.
List C contains 6 unsolved Sudoku puzzles that can't be solved through the simple method.
List D contains the solutions to the puzzles of List C.

Your output should look something like this.

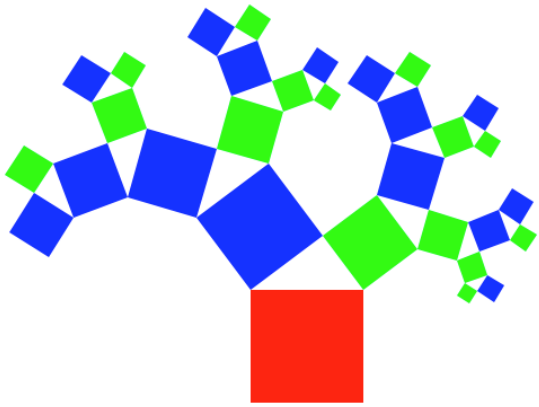
Unsolved Grid	Solved Grid
[0, 0, 3, 0, 2, 0, 6, 0, 0]	[4, 8, 3, 9, 2, 1, 6, 5, 7]
[9, 0, 0, 3, 0, 5, 0, 0, 1]	[9, 6, 7, 3, 4, 5, 8, 2, 1]
[0, 0, 1, 8, 0, 6, 4, 0, 0]	[2, 5, 1, 8, 7, 6, 4, 9, 3]
[0, 0, 8, 1, 0, 2, 9, 0, 0]	[5, 4, 8, 1, 3, 2, 9, 7, 6]
[7, 0, 0, 0, 0, 0, 0, 0, 8]	[7, 2, 9, 5, 6, 4, 1, 3, 8]
[0, 0, 6, 7, 0, 8, 2, 0, 0]	[1, 3, 6, 7, 9, 8, 2, 4, 5]
[0, 0, 2, 6, 0, 9, 5, 0, 0]	[3, 7, 2, 6, 8, 9, 5, 1, 4]
[8, 0, 0, 2, 0, 3, 0, 0, 9]	[8, 1, 4, 2, 5, 3, 7, 6, 9]
[0, 0, 5, 0, 1, 0, 3, 0, 0]	[6, 9, 5, 4, 1, 7, 3, 8, 2]

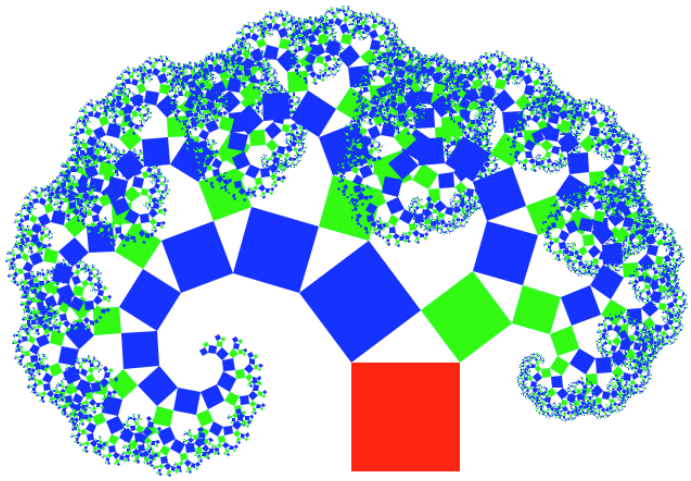
For an extra 5 points, solve Euler Problem 96.

Fractals (up to 20 points)

Go here for videos on graphics. Use the graphics package to create a recursive fractal drawing. A 'Pythagorean Tree' is a popular drawing but is certainly not the only fractal that can be drawn.

Below you can see a Pythagorean tree of level 4 and another of level 13.





squareRootByContinuedFractions(r,c) (20 points)

All square roots can be written as continued fractions. The general form of this representation is:

$$\sqrt{N} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ddots}}}$$

Specifically, the square root of two would look like this: $$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \ddots}}}$$

For $\sqrt{2}$, after each of the first four iterations, the convergents (results converge to a value) of the square root of two are:
1 + 1/2 = 3/2
1 + 1/(2 + 1/2) = 7/5
1 + 1/(2 + 1/(2 + 1/2)) = 17/12
1 + 1/(2 + 1/(2 + 1/(2 + 1/2))) = 41/29

After the next four iterations, the convergents are 99/70, 239/169, 577/408 and 1393/985.
So the eighth convergent has numerator 1393 and denominator 985.

squareRootByContinuedFractions(r,c) returns a tuple of the numerator and denominator of convergent 'c' of the square root of 'r.'

The simplest case is the square root of two. For example:

squareRootByContinuedFractions(2,1)==(3,2)
squareRootByContinuedFractions(2,2)==(7,5)
squareRootByContinuedFractions(2,8)==(1393,985)
squareRootByContinuedFractions(2,50)== (166161328781867496 07, 11749380235262596085)

For other roots, the continuing fractions are periodic but the same digit doesn't repeat as with the root of two. For example:

$$\sqrt{3} = 1 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{2 + \ddots}}}}$$

$$\sqrt{7} = 2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \ddots}}}}$$

Note for the square root of three, the digits on the bottom of each fraction alternate between 1 & 2. This alternate pattern repeats with a period of two. For the square root of seven, the pattern starts with a_0=2 and then the continued fraction begins. The compact way of representing continuing fractions of periodic square roots is like this.

$\sqrt{2}$ =[1,(2)], period=1
$\sqrt{3}$ =[1,(1,2)] period=2
$\sqrt{5}$ =[2,(4)], period=1
$\sqrt{6}$ =[2,(2,4)] period=2
$\sqrt{7}$ =[2,(1,1,1,4)] period=4
$\sqrt{8}$ =[2,(1,)] period=2
$\sqrt{10}$ =[3,(6)] period=1

The first number in each is a_0. The parentheses contain a_1 and all subsequent values, which are periodic. So for the non-trivial cases, you will have to determine the periodic values along with the length of the period. There is a very good description of the process for doing this in the problem statement of Euler Problem # 64. You will need to find a way to determine these values for any root before you can find numerator and denominator of the convergent. So:

squareRootByContinuedFractions(3,2)==(5,3)
squareRootByContinuedFractions(7,10)== (1307, 494)
squareRootByContinuedFractions(11,20)== (31539640338297, 9509559365899)
squareRootByContinuedFractions(23,50)== (2841025846841982334099, 592394839775260572964)

By the way, Euler problems 57, 64 and 65 all deal with this topic. If you solve them, you can get more points. Five points each for #57 & #65. Ten Points for #64.

gaussianElimination(M) (30 points) Required by end of semester

gaussianElimination(M) takes in an m x (m+1) 2D matrix (M) representing a m variable system of equations. Return is an m element 1D list representing the solutions to the system. Gaussian Elimination is the process of using row operations to reduce a matrix to row echelon form. For example, the system of equations shown below can be solved by guassian elimination as demonstrated.

$$\begin{aligned} x+y&=10\\ 2x-y&=2 \end{aligned} \qquad \begin{bmatrix} 1 & 1 & 10\\ 2 & -1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 10\\ 0 & 1 & 6 \end{bmatrix}$$ Substitute y=6 into equation one for the solution $x=4, y=6$

The second matrix is formed by multiplying equation one by (-2) and adding it to equation two, followed by division by (-3). This is done to get a zero below the diagonal and to place '1' in all locations of the diagonal. Once the diagonal has all ones with zeros below, the last equation is solved and the result can be 'backsubstituted' to form a complete solution.

Here is another example.

$$\begin{aligned} a+b-2c+d+3e-f&=4\\ 2a-b+c+2d+e-3f&=20\\ a+3b-3c-d+2e+f&=-15\\ 5a+2b-c-d+2e+1&=-3\\ -3a-b+2c+3d+e+3f&=16\\ 4a+3b+c-6d-3e-2f&=-27 \end{aligned} \qquad \begin{bmatrix} 1 & 1 & -2 & 1 & 3 & -1 & 4\\ 2 & -1 & 1 & 2 & 1 & -3 & 20\\ 1 & 3 & -3 & -1 & 2 & 1 & -15\\ 5 & 2 & -1 & -1 & 2 & 1 & -3\\ -3 & -1 & 2 & 3 & 1 & 3 & 16\\ 4 & 3 & 1 & -6 & -3 & -2 & -27 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & -2 & 1 & 3 & -1 & 4\\ 0 & 1 & -5/3 & 0 & 5/3 & 1/3 & -4\\ 0 & 0 & 1 & -6/7 & -13/7 & 4/7 & -33/7\\ 0 & 0 & 0 & 1 & 2/9 & 11/6 & 113/18\\ 0 & 0 & 0 & 0 & 1 & 87/38 & -11/38\\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

Note how the last row has all zeros except for the last two columns. That represents a solution to the equation $f=-1$. If that result is substituted into the row above, $e+87/38(-1)=-11/38$ and solved for 'e', the result is $e=2$. Now both of these values are substituted into the row above and so forth until all values are found. For this system, the solutions are $a=1, b=-2, c=3, d=4, e=2, f=-1$.

Make sure to watch out for division by zero (in the process of reducing to row echelon form, zeros can appear as coefficients throughout). Also, make sure to NOT alter the contents of the original system of equations.

So: gaussianElimination([[1,1,10],[2,-1,2]])==[4,6]

gaussianElimination([[1, 1, -2, 1, 3, -1, 4], [2, -1, 1, 2, 1, -3, 20], [1, 3, -3, -1, 2, 1, -15],
[5, 2, -1, -1, 2, 1, -3], [-3, -1, 2, 3, 1, 3, 16], [4, 3, 1, -6, -3, -2, -27]])==[1,-2,3,4,2,-1]

gaussianElimination([[2,3,-1,1,2,-1,2,1], [1,-1,1,-1,1,1,1,2], [2,2,-2,1,-2,1,-1,1],[1,0,-2,1,-1,-1,1,1],
[1,-2,1,-1,1,1,1,2], [1,2,2,2,1,-1,2,1], [1,-1,-2,-1,-1,1,1,2]])== [1.25, -0.0, 0.5, -1.0, -0.75, -0.5, 0.5]

pointsOnAPlane() (15 points)

Attached is the file pointsOnAPlane.txt. In it you will find approximately 10,000 sets of four ordered triples in this format:

8909,-1638,2630,-2739,-8331,1864,-6582,-9666,-1242,-8544,-5322,-2306 (Actual first line of file)
Consider the values to be four points A,B,C and D that look like this:
A(8909,-1638,2630),B(-2739,-8331,1864),C(-6582,-9666,-1242),D(-8544,-5322,-2306)

Exactly one set of these sets of triples is coplanar. Find the set of ordered triples and identify an equation of the plane that contains the points. Your equation must contain integer value coefficients that are relatively prime to one another.

findPolynomialFunction(D): (10 points) (up to 30 points if graphed)

Given a data set (D) in the format D=[(x1,y1),(x2,y2),…(xn,yn)] find a polynomial function that exactly models the data. This can be done fairly easily (for most data sets) once you have completed gaussianElimination. There is a slight problem when a data set contains a y-intercept. You'll have to figure out how to deal with that. You can assume that the data set represents a function. Return is a list representing the coefficients of the minimum degree polynomial passing through all of the points in the data set (rounded to 2 decimal places).

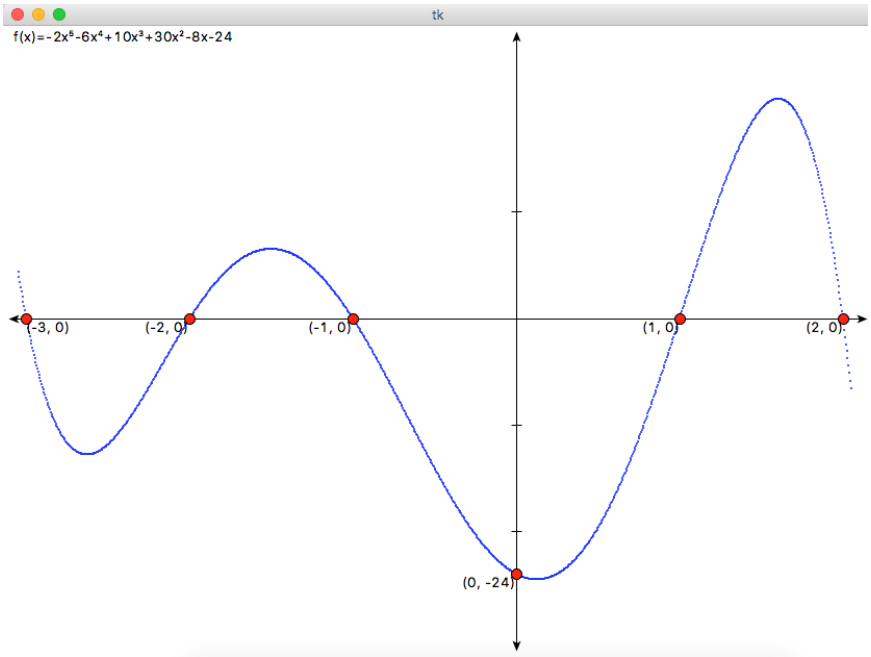
For example: (All of these examples will involve almostEqual)

findPolynomialFunction([(0,6),(-2,0),(-3,0)])==[1.0, 5.0, 6.0]

findPolynomialFunction([(-3,5280.5),(-2,675.4),(-1,27.2),(4,17732.2),(1,41.8),(2,470.6),(3,3500.4)])== [4.5, -3.7, 10.0, -1.0, 35.0, 12.0, -15.0]

findPolynomialFunction([(-3,0), (-2,0), (-1,0), (0,-24), (1,0), (2,0)])==[-2.0,-6.0,10.0,30.0,-8.0,-24.0]

The last example finds the polynomial f(x). A graph might look like this.



distanceBetweenLines(W,X,Y,Z): (5 points to 25 points)

Those of you who took my Precalculus class will remember that the distance between skew lines (non-parallel lines that do not intersect) can be found through vector techniques. It turns out that finding the distance is easy. Finding the two points (one on each line) where the two lines are at their closest is much more difficult. So you can get 5 points for finding the distance between two lines ($\overleftrightarrow{WX}$ & $\overleftrightarrow{YZ}$) or you can get up to 20 more points by finding both the distance and the points where that distance happens (among a few other things).

For example:

distanceBetweenLines((2,5,3),(5,19,21),(1,6,2),(3,11,16)) would return a distance of 0.926.

If all you return is this distance that's 5 points.

If you can find the two points where this distance happens you can get another 10 points:

For example:

distanceBetweenLines((2,5,3),(5,19,21),(1,6,2),(3,11,16)) would return a distance of 0.926

along with the two points (2.533, 7.488, 6.198) & (1.616, 7.540, 6.311)

For 5 more points account for the case where the two lines intersect. In this case you should find the point of intersection. Understandable discussions of this possibility can be found here OR here.

For example:

distanceBetweenLines((5,5,4),(10,10,6),(5,5,5),(10,10,3)) would return a distance of 0 and the single point (6.25, 6.25, 4.5)

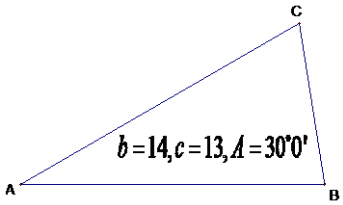
One more possibility (for 5 more points) is if the two lines are parallel.

In this case the distance should be returned as well as a statement that the lines are parallel.

For example:

distanceBetweenLines((0,0,0),(1,0,0),(0,0,3),(1,0,3)) should return a distance of 3 and a message that the lines are parallel.

solveTriangle(T): (up to 30 points)



Recall in trigonometry you can "solve" a triangle given three of the six parts.

For example, in Triangle ABC if you are given $b = 14, c = 13, A = 30^\circ 0'$, you can use the Law of Cosines to find $a = 7.05, B = 82^\circ 52', C = 67^\circ 8', Area = 45.5$

It's obvious that $a = 14, c = 13, B = 30^\circ 0'$ is the same triangle but with the given information assigned to different sides and angles.

Input for this function will be a list of 6 "parts" of a triangle, three of which will be zero. All triangles will be assumed to be Triangle ABC in this format: T=[a,b,c,A,B,C]. Side lengths are given as floats or ints and angles as tuples (degrees,minutes). For example, the first case above would have an input of [0,14,13,(30,0),(0,0),(0,0)] and the return should be [7.05,14,13,(30,0),(82,52),(67,8),45.5]. Side lengths and areas should be rounded to the nearest hundredth and angles should be returned as tuples of degrees and minutes. Be aware that Python does not print a trailing zero if rounding results in a zero in the hundredths digit.

Your program must be able to decipher the "Case" and to apply the relevant laws to find all possible solutions to the given information. You should recall that given three of 6 parts of a triangle, there are potentially 0, 1 or 2 triangles that can be formed (or in the case of AAA, there is not a unique solution). If there are 2 solutions to a given input, return should be a list of lists.

Here is a sampling of inputs and outputs:

Input	Output
[0, 0, 0, (30, 0), (60, 0), (90, 0)]	Unique Triangle Not Possible
[3, 10, 0, (30, 0), (0, 0), (0, 0)]	No Such Triangle
[5, 10, 0, (0, 0), (30, 0), (0, 0)]	[5, 10, 14.01, (14, 29), (30, 0), (135, 31), 17.52]
[10, 5, 0, (90, 0), (0, 0), (0, 0)]	[10, 5, 8.66, (90, 0), (30, 0), (60, 0), 21.65]
[5, 7, 0, (3, 0), (0, 0), (0, 0)]	[[5, 7, 11.98, (3, 0), (4, 12), (172, 48), 2.19], [5, 7, 2.0, (3, 0), (175, 48), (1, 12), 0.37]]
[17.32050807568877, 10, 0, (120, 0), (0, 0), (0, 0)]	[17.32, 10, 10.0, (120, 0), (30, 0), (30, 0), 43.3]
[3, 10, 0, (30, 0), (0, 0), (0, 0)]	No Such Triangle
[5.773502691896258, 10, 0, (30, 0), (0, 0), (0, 0)]	[[5.77, 10, 11.55, (30, 0), (60, 0), (90, 0), 28.87], [5.77, 10, 5.77, (30, 0), (120, 0), (30, 0), 14.43]]
[10, 10, 0, (0, 0), (0, 0), (30, 0)]	[10, 10, 5.18, (75, 0), (75, 0), (30, 0), 25.0]
[15, 17, 0, (61.92751306414704, 0), (0, 0), (0, 0)]	[[15, 17, 8.0, (61, 56), (90, 0), (28, 4), 60.0], [15, 17, 8.0, (61, 56), (90, 0), (28, 4), 60.0]]
[10, 10, 0, (0, 0), (0, 0), (120, 0)]	[10, 10, 17.32, (30, 0), (30, 0), (120, 0), 43.3]
[10, 17.32050807568877, 0, (0, 0), (0, 0), (30, 0)]	[10, 17.32, 10.0, (30, 0), (120, 0), (30, 0), 43.3]
[17.32050807568877, 20, 10, (0, 0), (0, 0), (0, 0)]	[17.32, 20, 10, (60, 0), (90, 0), (30, 0), 86.6]
[7, 5, 0, (25, 50), (0, 0), (0, 0)]	[7, 5, 11.15, (25, 50), (18, 8), (136, 2), 12.15]
[10, 10, 20.01, (0, 0), (0, 0), (0, 0)]	No Such Triangle
[10, 10, 19.99, (0, 0), (0, 0), (0, 0)]	[10, 10, 19.99, (1, 49), (1, 49), (176, 23), 3.16]
[1, 10, 0, (30, 0), (0, 0), (0, 0)]	No Such Triangle
[5, 13, 12, (0, 0), (0, 0), (0, 0)]	[5, 13, 12, (22, 37), (90, 0), (67, 23), 30.0]
[0, 10, 6, (0, 0), (0, 0), (31, 10)]	[[11.59, 10, 6, (89, 14), (59, 36), (31, 10), 30.0], [5.52, 10, 6, (28, 26), (120, 24), (31, 10), 14.29]]
[0, 0, 30, (122, 50), (15, 0), (0, 0)]	[37.55, 11.57, 30, (122, 50), (15, 0), (42, 10), 145.78]
[0, 18, 40, (82, 30), (0, 0), (0, 0)]	[41.67, 18, 40, (82, 30), (25, 22), (72, 8), 356.92]
[18, 10, 9, (0, 0), (0, 0), (0, 0)]	[18, 10, 9, (142, 36), (19, 43), (17, 41), 27.33]
[0, 4, 5, (51, 0), (0, 0), (0, 0)]	[3.98, 4, 5, (51, 0), (51, 23), (77, 37), 7.77]
[6, 7, 12, (0, 0), (0, 0), (0, 0)]	[6, 7, 12, (20, 51), (24, 32), (134, 37), 14.95]
[0, 10, 6, (0, 0), (0, 0), (31, 10)]	[[11.59, 10, 6, (89, 14), (59, 36), (31, 10), 30.0], [5.52, 10, 6, (28, 26), (120, 24), (31, 10), 14.29]]
[10, 10, 0, (100, 0), (0, 0), (0, 0)]	No Such Triangle
[10, 11, 0, (100, 0), (0, 0), (0, 0)]	No Such Triangle
[10, 12, 0, (20, 0), (0, 0), (0, 0)]	[[10, 12, 20.4, (20, 0), (24, 14), (135, 46), 41.85], [10, 12, 2.16, (20, 0), (155, 46), (4, 14), 4.43]]
[7, 9, 0, (0, 0), (0, 0), (34, 0)]	[7, 9, 5.05, (50, 46), (95, 14), (34, 0), 17.61]
[8, 6, 0, (0, 0), (0, 0), (172, 0)]	[8, 6, 13.97, (4, 34), (3, 26), (172, 0), 3.34]
[5, 6, 8, (0, 0), (0, 0), (0, 0)]	[5, 6, 8, (38, 37), (48, 31), (92, 52), 14.98]
[12, 22, 16, (0, 0), (0, 0), (0, 0)]	[12, 22, 16, (32, 9), (102, 38), (45, 12), 93.67]
[6, 3, 12, (0, 0), (0, 0), (0, 0)]	No Such Triangle
[0, 1, 1.7320508075688772, (0, 0), (0, 0), (120, 0)]	[1.0, 1, 1.73, (30, 0), (30, 0), (120, 0), 0.43]
[0, 14, 13, (30, 0), (0, 0), (0, 0)]	[7.05, 14, 13, (30, 0), (82, 52), (67, 8), 45.5]
[0, 0, 6, (0, 0), (120, 6), (29, 35)]	[6.13, 10.51, 6, (30, 19), (120, 6), (29, 35), 15.92]

Djikstra's Algorithm(m) (20 points)

Djikstra's Algorithm is a method to find the best way to traverse a weighted network. You can read about it here and watch a video as to how it works here. There are many more websites and videos that you can find with a little research, but I found this video to be easy to understand. Your solution to this problem will require inputting a file that is organized as a square matrix where individual weights between nodes are given as numbers and unconnected nodes are given as a dash (-). Return is a 2D list with the shortest path to each vertex from each vertex. For the example on the YouTube video from above, the input and return will look like this:

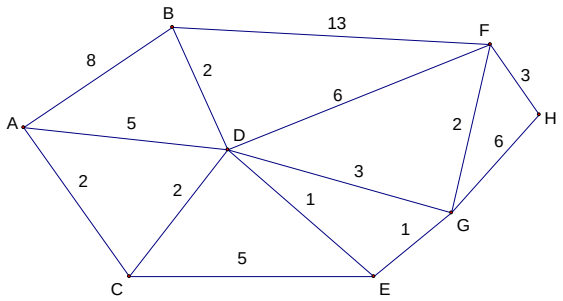
```
['-', 8, 2, 5, '-', '-', '-', '-']
[8, '-', '-', 2, '-', 13, '-', '-']
[2, '-', '-', 2, 5, '-', '-', '-']
[5, 2, 2, '-', 1, 6, 3, '-']
['-', '-', 5, 1, '-', '-', 1, '-']
['-', 13, '-', 6, '-', '-', 2, 3]
['-', '-', '-', 3, 1, 2, '-', 6]
['-', '-', '-', '-', '-', 3, 6, '-']
```

```
[0, 6, 2, 4, 5, 8, 6, 11]
[6, 0, 4, 2, 3, 6, 4, 9]
[2, 4, 0, 2, 3, 6, 4, 9]
[4, 2, 2, 0, 1, 4, 2, 7]
[5, 3, 3, 1, 0, 3, 1, 6]
[8, 6, 6, 4, 3, 0, 2, 3]
[6, 4, 4, 2, 1, 2, 0, 5]
[11, 9, 9, 7, 6, 3, 5, 0]
```

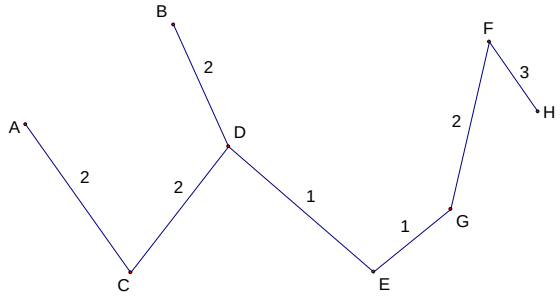
You'll notice that both the beginning and ending matrix are symmetric about the diagonal. With a little thought you should be able to understand why. To do this problem you might want to take advantage of the Python module heapq. Very conveniently, these functions insert values into a list (heap stack) and then remove only the smallest element when asked. heapq is also flexible in that you can store additional information (vertex location) along with the value on the heap stack. An example of how it is used is included in the starting point. Elements are pushed onto the stack and popped off of the stack. The great thing about heapq is that it finds the smallest element efficiently so there is no need to sort the stack array when new data is added. Euler problems 81, 82 and 83 can be solved using this algorithm. You can get 5 points each for those.

Prim's Algorithm(m) (20 points)

Prim's Algorithm also works on a weighted network but it reduces the network to its optimal form. The network from the last problem looks like this:



It can be optimized by removing connections between vertices that are unnecessary. Prim's Algorithm will not only remove the unneeded connections, it will do so in a way that ensures connectivity with minimum possible weight. After optimization, the network looks like this:



Input is a 2D list in the same format as for the Djikstra problem. Return is in the same format where '-' has been inserted wherever a connection between vertices has been removed . For the network above, input and return are:

```
['-', 8, 2, 5, '-', '-', '-', '-']
[8, '-', '-', 2, '-', 13, '-', '-']
[2, '-', '-', 2, 5, '-', '-', '-']
[5, 2, 2, '-', 1, 6, 3, '-']
['-', '-', 5, 1, '-', '-', 1, '-']
['-', 13, '-', 6, '-', '-', 2, 3]
['-', '-', '-', 3, 1, 2, '-', 6]
['-', '-', '-', '-', '-', 3, 6, '-']
```

```
['-', '-', 2, '-', '-', '-', '-', '-']
['-', '-', '-', 2, '-', '-', '-', '-']
[2, '-', '-', 2, '-', '-', '-', '-']
['-', 2, 2, '-', 1, '-', '-', '-']
['-', '-', '-', 1, '-', '-', 1, '-']
['-', '-', '-', '-', '-', '-', 2, 3]
['-', '-', '-', '-', 1, 2, '-', '-']
['-', '-', '-', '-', '-', 3, '-', '-']
```

polynomialDivision(p1,p2) (15 points)

Recall that polynomials can be long-divided in a manner similar to numbers. The several examples below show results for differing degrees of numerator and denominator.

$$\frac{x+1}{x+1}=1 \qquad \frac{x+1}{x+5}=1-\frac{4}{x+5} \qquad \frac{6x^2+7x-10}{2x^2+5}=3-\frac{7x-25}{2x^2+5} \qquad \frac{x^2+5x+6}{x+2}=x+3 \qquad \frac{x+2}{x^2+5x+6}=0+\frac{x+2}{x^2+5x+6}$$

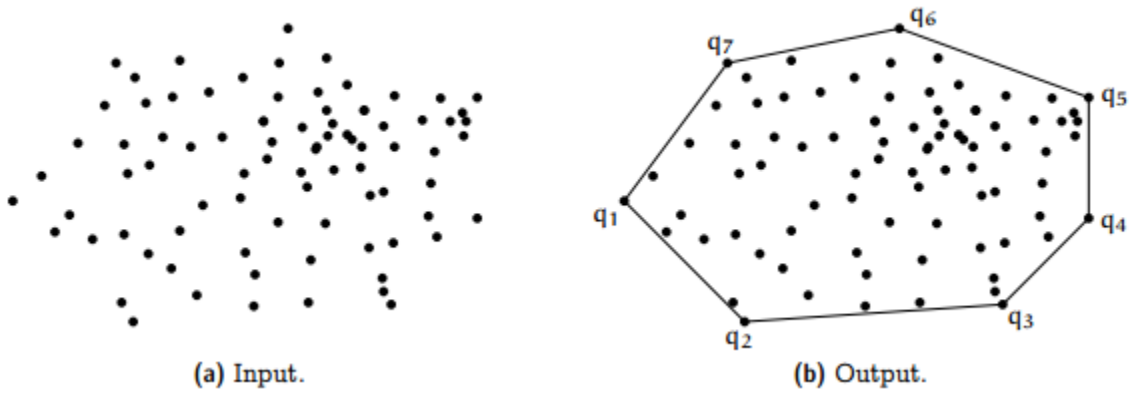
$$\frac{x^6}{(x+1)^3}=x^3-3x^2+6x-10+\frac{15x^2+24x+10}{(x+1)^3} \qquad \frac{2x^8-7x^7+6x^6-7x^5+15x^4-17x^3+38x^2-22x+12}{-x^3+2x+4}=-2x^5+7x^4-10x^3+13x^2-7x+3$$

polynomialDivision takes in two polynomial (in list form with all powers represented) and long-divides the first by the second. Return is a tuple with two lists—the first is the coefficients of the quotient and the second the numerator of the remainder. So from the examples above:

polynomialDivision([1,1],[1,1])== ([1], [0])
polynomialDivision([1,1],[1,5])== ([1.0], [-4.0])
polynomialDivision([6,7,-10],[2,0,5])== ([3.0], [7.0, -25.0])
polynomialDivision([1,5,6],[1,2])== ([1.0, 3.0], [0])
polynomialDivision([1,2],[1,5,6])== ([0], [1, 2])
polynomialDivision([1,0,0,0,0,0,0],[1,3,3,1])== ([1.0, -3.0, 6.0, -10.0], [15.0, 24.0, 10.0])
polynomialDivision([2,-7,6,-7,15,-17,38,-22,12],[-1,0,2,4])= ([-2.0, 7.0, -10.0, 13.0, -7.0, 3.0], [0])

convexHull(L) 20 points (L is a list of tuples that are ordered pairs)

Convex Hull is a classic geometry problem. Given a set of points, convexHull will find a subset of those points that form a convex polygon containing all of the points.



(a) Input. (b) Output.

One of the simplest algorithms to find the complex hull of a set of coplanar points is the gift-wrapping algorithm. Go here to see how it works. There are other ways to do this but the gift-wrapping method is fairly straight-forward. In the starting point you will find two test cases with solutions.