Function Name: `cellCat`

Inputs:

1. *(cell)* An 1xN cell array

Outputs:

1. *(char)* A concatenation of all of the strings in the cell array

Function Description:

If you remember back to when you did `cellSum()` on Homework 9, there was the restriction that no test cases would have nested cell arrays. Well, now that you are learning recursion, that restriction has been lifted! This function should concatenate all strings found in a cell array into one long string. The data in each cell may be any data type you have learned so far, **including** nested cell arrays.

Notes:

- There will only be 1xM character arrays (strings) in the cell array

Hints:

- Looking at your `cellSum()` code may help you get started on this problem because some of the logic is the same.

Function Name: `towersOfHaynoi`

Inputs:

1. *(double)* The number of hay bales to move
2. (*char*) The name of the starting platform
3. (*char*) The name of the destination platform
4. (*char*) The name of the helper platform

Outputs:

(none)

File Outputs:

1. A text file with the instructions to move the bales of hay from the starting platform to the destination platform

Function Description:

While visiting the local farm for the pumpkin patch, corn maze, and other fun Fall activities, the farmer asks for your help! He needs to move a stack of hay bales from one platform to another and doesn't know the best way to do it. The hay bales are all different sizes, and the starting state is the stack of hay on the starting platform with the smallest bale on top and the largest on the bottom and they are all in order by size (think like a tiered cake). The goal state is the same order of hay bales but all on the destination platform. There are, however, some restrictions on the task. First, the farmer can only lift one bale at a time, so he can only move one at a time. Second, a larger bale cannot be stacked on top of a smaller hay bale, or it might be unbalanced. Third, there is a third platform besides the starting and destination platforms that the farmer will leverage to make this task doable.

You decide to use your MATLAB skills to help the farmer out! You will write a function called `towersOfHaynoi` to give the farmer instructions. The function will take in the number of bales to move and the names of the three platforms and will write a text file with instructions on how to move the hay bales. The name of the text file will be

```
Instructions_<Name of start platform>to<Name of destination platform>.txt
```

The contents of the file will be formatted as follows: The first line will be of the form:

```
Instructions for moving <number> bales of hay from <start platform> to
                        <destination platform>:
```

The second line will be blank, and all subsequent lines will be the instructions, formatted as:

```
Move top bale from <start platform> to <destination platform>.
```

Note that the start and destination platforms in the first line are from the inputs (the overall start and destination platforms), and the start and destination platforms in the instructions indicate the other movements. For example, if the function is called as `towersOfHaynoi(2, 'A', 'B', 'C')`, your function should produce a file called `Instructions_AtoB.txt` with the following content:

```
Instructions for moving 2 bales of hay from A to B:

Move top bale from A to C.
Move top bale from A to B.
Move top bale from C to B.
```

There is a blank line in between the first line and the rest, but no blank line at the end.

Notes:

- Your solution must produce the optimal number of moves to move the stack of hay
- If you are not sure if you are getting the optimal number, your solution should have $2^n - 1$ moves, where n is the number of bales.
- Because of the exponential nature of how the number of instructions increases as the number of bales increases, know that if you try to test your function with large n, it may take a very long time--if you have 20 bales it will take over 1 million moves. If you have 50 bales it will take over 10^{15} moves. If your computer could calculate one move every microsecond, this would take about 35 years to compute! You will not be tested with any inputs nearly that large, and though you are free to try to test your code however you please, this is your warning.

Hints:

- To move a large stack of hay, remember that you cannot move a larger stack on top of a smaller one, which means that the biggest stack can only be moved to an empty platform.
- You will need a helper function for this problem.
- Try to avoid thinking about how you would construct a strategy for moving the bales and instead try to think of a recursive pattern.

Function Name: `subsetSum`

Inputs:

1. *(double)* A 1xN vector
2. *(double)* A single "target" number

Outputs:

1. *(logical)* Whether there is a subset of the vector that sums to the target number
2. *(double)* The subset, if any, that sums to the target value

Function Description:

Given a vector and a "target", determine if there is a subset of the vector that sums to the target number. The first output should be a logical of whether or not such a subset exists and the second output should be the subset. If no such subset exists, the first output should be false and the second output should be the empty vector (`[]`).

A target of 0 will always return true (selecting no elements sums to zero). Additionally, an empty vector as the first input will always return false (there are no elements to construct a sum out of).

Notes:

- You are guaranteed there will only be one unique subset that sums to the target.
- The only case where the first output is true and the second output is empty is if the target value is 0.

Hints:

- The last paragraph should give you a good idea of what base cases to use.
- To approach the base case, think about the following: you know that a single element in the input vector is either in the final subset or not. How can you use recursion to test if a particular element is in the final set?

EXTRA CREDIT

Function Name: `snowflake`

Inputs:

1. *(double)* The recursive depth

Outputs:

(none)

Plot Outputs:

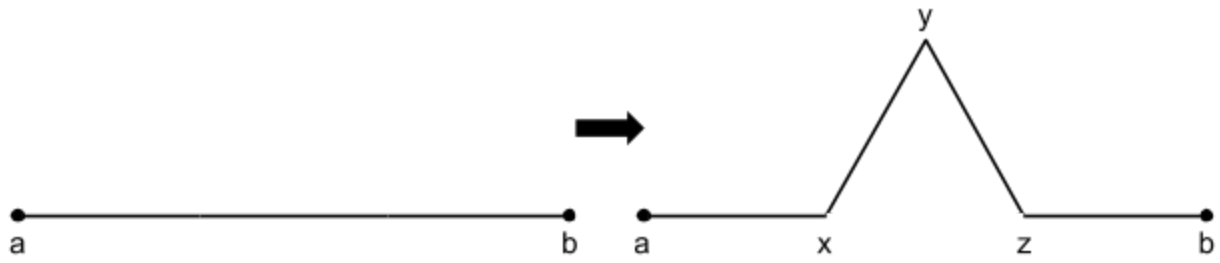
1. A plot of the Koch Snowflake

Function Description:

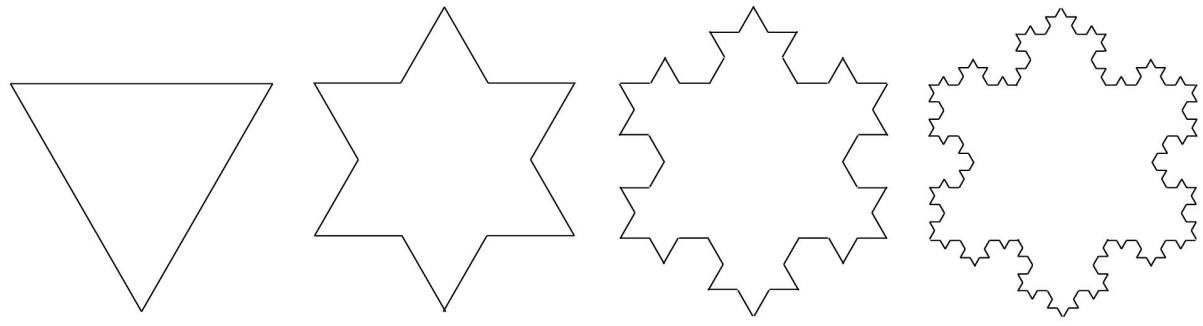
Fall's got you feelin some type of way, but what you're really looking forward to is snow! But since it never snows in Atlanta, you're going to have to settle for a MATLAB function that plots snowflakes instead.

You may or may not have heard of fractals before, but they are incredibly complex structures that occur everywhere in nature. From trees to computer networks to city planning, fractals have been used to model all sorts of things. You can read more about them here.

For this problem you will be creating a plot of a well-known fractal, namely, the Koch Snowflake. The Koch Snowflake is created by starting with an equilateral triangle and recursively replacing every line segment with 4 line segments as follows:



Line segments $\overline{ax}$, $\overline{xy}$, $\overline{yz}$, and $\overline{zb}$ are all the same length. Additionally $\triangle xyz$ is an equilateral triangle. Based on this rule, the first few recursions of the Koch Snowflake look like this:



`0` `1` `2` `3`

The first input to the function determines how many times you should recurse. So `snowflake(0)` will be the first triangle pictured above, `snowflake(3)` will be the snowflake on the right and `snowflake(6)` will have even more detail. The top left, top right and bottom point of the starting triangle should be `(-1, 0)`, `(1, 0)` and `(0, -√3)`, respectively.

In case your geometry is a little rusty, you have been provided with a helper function, `triPoints()`, that will produce the coordinates for points x, y, and z given points a and b. You can read more on how to use this function by typing help triPoints in the Command Window.

Notes:

- Use `axis equal` and `axis off`.
- Plot in a black line.
- Because of the computational complexity of plotting the Koch Snowflake, you will probably not want to test your code on an input larger than 12 or 13. The Koch Snowflake with a recursive depth of 13 has over 200 million line segments. With a recursive depth of 19 there will be more line segments than there are stars in the Milky Way!

Hints:

- Focus on creating one side of the triangle first, then extend your idea to the remaining sides.

EXTRA CREDIT

Function Name: `cornMaze`

Inputs:

1. *(char)* A character array representing a maze

Outputs:

1. *(char)* The character array maze solved

Function Description:

It's Fall, the leaves are changing, there is a chill in the air, and while you are at the farm helping the farmer out with his hay moving problem, you decide to go try your hand at the corn maze. After a few minutes of frustration from not knowing where to turn and being sure that you are walking in circles, you decide there must be a better way!

Write a function in MATLAB called `cornMaze` that takes in a maze and solves it. The maze will be the input to the function and it will be a character array. This array will be rectangular and the walls of the maze are represented by the character '#', the spaces (possible paths) of the maze are represented by the character ' ' (space), and there is also an 'O' (capital O, not zero) and an 'X' on the border of the maze. The 'O' represents the entry point of the maze (the start) and the 'X' represents the exit point (the destination) of the maze. Your job is to write a function that updates the character array with a path from 'O' to 'X'. The path should be represented with the character 'o' (lowercase 'o').

For example, the maze on the left below could be an input to the maze and what is on the right would be your output.

```
#####          #####
# # #          # # #
#   X          #oooX
O # #          Oo# #
#####          #####
```

Notes:

- The mazes will only have one possible solution
- Diagonal moves are not allowed
- To generate additional test cases, we have provided a function called `makeMaze` that will randomly generate a maze of the input dimensions. You can type `help makeMaze` to see details on how to use the function.
- If you like this problem and are interested in a challenge, try to write your own function that generates a maze! It requires similar logic but is a much harder problem. (This is just for funzies, not required!)

Hints:

- Image you were actually in a maze and could not see anything besides the paths available from your current location. What methodical strategy might you devise to find your way through the maze?