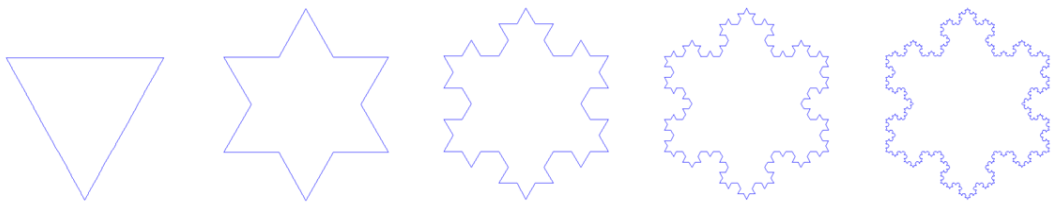
CS1114 Section 4: To Understand Recursion...
February 13th, 2013

"To understand recursion, you must first understand recursion."



In lecture last time we saw an example of *recursion* with the quicksort algorithm. A recursive algorithm (or function) is one that calls itself one or more times. In this exercise, you will explore a few simples examples of recursion, and see how algorithms can sometimes be implemented using either loops or recursion. Once you wrap your head around recursion, you will find that for certain problems, the recursive implementation is more elegant and easier to program.

1 Matlab Preliminaries

You can find files from the demo in `/courses/cs1114/section/recursion/`. The transcript is in `demo.m` and the factorial function is in `fact.m`.

2 Computing sums with recursion

For your first task, we will compute the "sumall" function. The sumall function takes a positive integer n, and is defined as the sum of all numbers between 1 and n:

$$\texttt{sumall}(n) = \sum_{i=1}^{n} i.$$

1. Write a function `sumall1` that takes a positive integer n and computes `sumall`(n). This function should use a **for loop**, and its header should be:

   ```
   function [ sum ] = sumall1(n)
   % Compute the sum of all numbers between 1 and n using a for loop
   ```

   This function should be saved in a file called `sumall1.m` (remember that you can type `edit sumall1.m` in Matlab to open an editor). Test your function by calling `sumall1` with several numbers. Remember from class that this sum is also just $n(n+1)/2$, you can use this fact to verify that your function is correct.

2. Now you will implement the same function using recursion. To do so, notice that `sumall`$(n) = n +$ `sumall`$(n-1)$ (at least for $n > 1$). Thus, `sumall` can be written as a function of itself with a slightly smaller input, a classic sign that we can use recursion. Use this fact to implement a **recursive** Matlab function `sumall2` that, just like `sumall1`, takes a positive integer n and computes `sumall`(n). The header of this function should be:

```
function [ sum ] = sumall2(n)
% Compute the sum of all numbers between 1 and n using recursion
```

Be careful to write your code so that it terminates eventually. In other words, remember that you need a base case!

3 Fibonacci Numbers

The **Fibonacci numbers** are the numbers in the following sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 \ldots$$

By definition, the first two Fibonacci numbers are 0 and 1, and each remaining number is the sum of the previous two. In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation:

$$\begin{aligned} F_1 &= 0 \\ F_2 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \text{ for all } n > 2 \end{aligned}$$

The Fibonacci sequence is named after Leonardo of Pisa.

3. Write a Matlab function `fibo1` that takes a positive number$n$ and computes the nth Fibonacci number, F_n. This function should use a **for loop**. The header of this function should be:

```
function [ fn ] = fibo1(n)
% Compute the nth Fibonacci number using a for loop
```

4. The nth Fibonacci number can also be computed using recursion based on the recurrence given above. Note that this recursion will have 2 base cases: $F_1 = 0$ and $F_2 = 1$.

   Use this fact to implement a **recursive** Matlab function `fibo2` that computes the nth Fibonacci number, F_n The header of this function should be:

```
function [ fn ] = fibo2(n)
% Compute the nth Fibonacci number using recursion
```

4 Palindrome

A palindrome is a sequence of characters that is the same whether you read it forwards or backwards. Some examples of palindromes are:

```
r a c e c a r
r a t s   l i v e   o n   n o   e v i l   s t a r
```

Often in presenting palindromes, spacing and punctuation are ignored, which allows for more examples and easier reading:

```
"Go hang a salami, I'm a lasagna hog!"
"Never odd or even"
"Drab as a fool, aloof as a bard."
"T. Eliot, top bard, notes putrid tang emanating, is sad.
   I'd assign it a name: gnat dirt upset on drab pot-toilet."
```

For our purposes, we'll be assuming that all punctuation and spacing has been stripped from a string before it is given to us. You will write two functions—one iterative, one recursive—that determine whether a string is a palindrome or not.

5. Write a function `pal1` that takes a string and returns `true` if it is a palindrome, and false otherwise. Do the computation using a `for` loop. The function header should be:

   ```
   function [ is_pal ] = pal1(str)
   % Determine whether the input str is a palindrome using a for loop
   ```

   Be sure to test your function on inputs that are palindromes (e.g., `pal1(racecar)`, `pal1(anna)`) and some inputs that are not (e.g., `pal1(racecah)`, `pal1(warsaw)`, etc.). Also make sure it works on very short strings!

6. Next, write function `pal2` that gives the same result using a recursive approach. The header should be:

   ```
   function [ is_pal ] = pal1(str)
   % Determine whether the input str is a palindrome using recursion
   ```

Test this function to make sure it behaves the same as `pal1`.

5 Parity

We've seen how to compute the parity of a number directly (by computing its modulus with two), but we can also approach this problem recursively. Here, you'll write two functions that are *mutually recursive*—that is, rather than a function that calls itself, you'll write a pair of functions that call each other.

7. Write two functions, `isEven` and `isOdd`, that return true if a given natural number is even or odd, respectively. **The only mathematical operations you may use are subtraction and comparison with zero.** The function headers should be:

   ```
   function [ result ] = isEven(n)
   % Return true if n is even, and false otherwise.
   ```

   and

   ```
   function [ result ] = isOdd(n)
   % Return true if n is odd, and false if it is even.
   ```

6 Bonus (If You Have Time)

6.1 Turtle Graphics

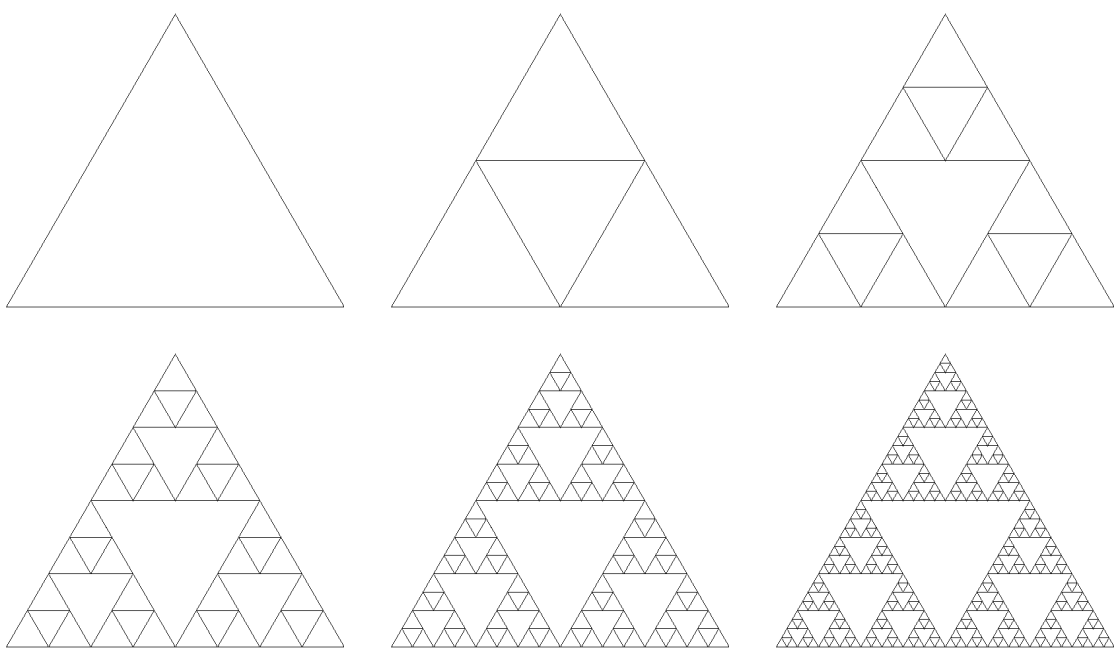
Turtle Graphics is a classic drawing interface that uses a virtual turtle (not so different from our virtual robots) to draw lines. The following commands control the turtle:

```
Turtle() - initialize a turtle object
turtleForward() - move a given distance forward, drawing a line along
                  the way if the pen is down
turtleTurn() - rotate the turtle to change the direction it's facing
turtleDown() - lower the pen so the turtle draws when it moves
turtleUp() - raise the pen so the turtle doesn't draw when it moves
```

These functions can be found at `/courses/cs1114/section/recursion/turtle/`. You should either add this directory to your Matlab path or copy the functions to your working directory so Matlab will be able to find them when you call them.

Calling `help` on these functions will give you some more detailed documentation. In particular, the `Turtle()` function's documentation explains the general usage.

6.2 Sierpinski Triangles



The Sierpinski Gasket, or Sierpinski Triangle, is a recursive structure made of triangles. In principle, it is infinitely detailed; in practice, we have to draw it down to a certain level

of detail and then stop to keep our program from running forever. The Sierpinski Triangle is shown in the figure above, drawn at detail levels 1 through 6.

8. Your task is to write a function `sierpinski` that uses Turtle graphics to draw a Sierpinski Triangle given a side length and a level of detail. The function header should be:

```
function [turtle] = sierpinski(turtle, side, level)
% draw a Sierpinski Gasket using <turtle> with side length <side>
% and a maximum detail level <level>
```

   For an extra challenge, write your function without calling `turtleUp` (remember the pen starts off in a down position, so all forward movement will result in lines being drawn). Have fun!