**Happy Numbers**

Recursion can be quite mind bending at first! The only way to get better at it is with practice (to understand recursion, you must first understand recursion). In this lab, the recursion problems get a bit trickier.

A recursive method must have a base case! A base case is a problem so simple, you can solve it immediately. Otherwise (if the base case hasn't been reached), your method should attempt to break to the problem up into smaller pieces, such that you're now trying to solve a smaller subset of a larger, more complex problem.

1. Write a recursive method that determines if the String parameter is a palindrome. Always start with your base case, a problem so simple it can be solved immediately! If you haven't reached your base case, attempt to break the problem up into smaller pieces until you do.
2. Write a recursive method that will return true if a number is prime.

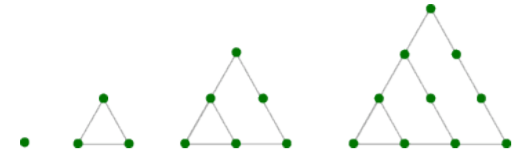
/\* Your method will need two parameters. Think about how the prime checker algorithm works with a for loop, then convert the algorithm to use recursion \*/

1. Complete the method public void timesTwo(int n) that prints the value of n factored into some number of 2s multiplied by a single odd number. You can assume n will be even.

timesTwo(80) >>> 2 \* 2 \* 2 \* 2 \* 5

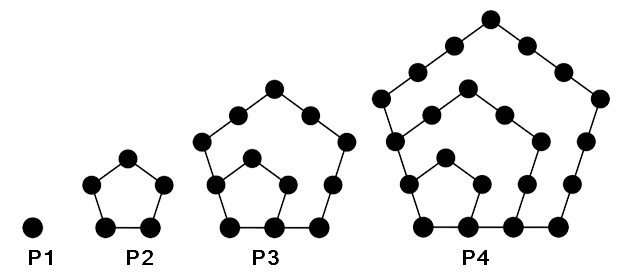
timesTwo(68) >>> 2 \* 2 \* 17

1. Early members of the Pythagorean Society defined "figurate numbers" to be the number of dots in certain geometrical configurations. The first four "triangular numbers" are 1, 3, 6, and 10:



Find a recursive expression for the nth triangular number (write a method int tri(n) that will find T(n)). **Massive hint:** number of dots in the new row is the side length of the resulting triangle.

1. A "pentagonal number" is similarly one that corresponds to dots arranged into a pentagon. Here are the first four pentagonal numbers and their corresponding pentagons:



P(1) = 1

P(2) = 5

P(3) = 12

P(4) = 22

The pentagonal number P(N) has sides made of N dots. To figure out how many dots P(N) has, find out how many P(N-1) has and add the dots for the new sides of size N.

Find the recursive formula for P(N) and write a method int penta(int n) that implements it.

1. (Riddle) A merchant has five children, and half of them are daughters. How is this possible?
2. Write a recursive method that will return the sum of all elements in an integer array.

//another strange but interesting thing to do – you'll need an 'index' parameter

1. Complete the following methods that will reverse all the letters in the String parameter.
   1. String reverseString(String str) – returns a backwards version of str.
   2. void printReverse(String str) – *prints* (only) a backwards version of str.
2. (Riddle) Penny has five children. The first is named January. Second kid is February. Her third is called March. Fourth is April. What is the name of the fifth.
3. Complete the method boolean isPowerOfN(int num, int base), that will return true if num is a power of base, and false if it is not. Reminder – any number to the power of 0 equals 1.

isPowerOfN(9, 3) >>> true

isPowerOfN(25, 3) >>> false

1. Write a recursive method that will return a version of the String parameter where only a pair of curly brackets and what is between the curly brackets remain.

curlyString("what's {all this} then") >>> {all this}

1. Write a recursive method void printPattern2(int n) that will print the pattern below (shown for n = 16). Do your best to solve this problem on your own! If you're totally stumped, see [here](https://youtu.be/FzJ6Uhw9VZU).

16 11 6 1 -4 1 6 11 16 //an extension of a problem in the previous lab

1. Write a recursive method that will print squares in the following manner:

printSquares(5) >>> 25, 9, 1, 4, 16 //odds descending, then evens ascending

printSquares(8) >>> 49, 25, 9, 1, 4, 16, 36, 64

1. Write a recursive method String wordWrap(String line, int width) that performs a word wrap, moving to the next line whenever the String is longer than width. For example, a call to wordWrap("hello, how are you doing today?", 13)) returns (including line breaks):

hello, how

are you doing

today? //String's two-parameter lastIndexOf() method may be useful

**Happy Numbers**

The definition of a "happy" number, per Wikipedia:

*A****happy number****is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number either equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1 (4, 16, 37, 58, 89, 145, 42, 20, 4, 16, 37, ...)*

*Those numbers for which this process ends in 1 are happy numbers, while those that do not end in 1 are "sad" numbers.*

Write a method boolean isHappy(int n) that will return true if a number is happy. Your solution should not use *any* loops. You can assume n will always be greater than zero.

You will need a couple private "helper" methods to perform the recursion. The isHappy() method is simply the "client" method; the helper methods will do the necessary recursion.

One of the helper methods should return the "next" number in the series (the sum of the squares of the current number's digits). This is usually done with a while loop:

int num = 16, next = 0;

while (num > 0) {

next += Math.pow(num % 10, 2);

num /= 10;

} //next == 62 + 12 or 37

However, you'll be using recursion instead. You will detect infinite loops by remembering which numbers you've already encountered (e.g. using an ArrayList to remember all the numbers in the current series).

isHappy(19) >>> true

isHappy(230) >>> true

isHappy(231) >>> false

**(Advanced) Maze Solver**

A maze is a two-dimensional structure with an entrance and (sometimes) an exit. In this program, you are tasked with determining if the given maze has an exit. **All the mazes will have the entrance point at the top-left corner. The exit point can be anywhere in the far-right column.** A valid path will be a continuous block of 1's that connect the top left corner to any spot on the far-right column. A valid path can only be connected horizontally or vertically. Diagonal connections are not legal.

Using the **Maze.java** and **MazeRunner.java** starter code, attempt to solve the mazes in the text files (each file contains a maze - **make sure to look at it first so you understand the input format of the mazes**). Loop through all the mazes (create a new Maze object for each), and print exit found if there is a path leading to an exit out of the maze. Print exit not found if there is not a path leading out of the maze. Also, print out the maze itself (use an overridden toString() method).

Algorithm pseudo-code:

if (r and c are in bounds and current spot is a 1)

if we are at the exit

mark exit found as true

else

save value in current spot

mark current spot as visited (not 0 or 1)

4 recursive calls, up / down / left / right

return current spot to its original state

1 0 0 0 0 1 1

1 1 1 1 0 1 0

0 0 1 0 0 1 0

0 1 1 1 0 1 0

0 1 0 1 0 1 0

0 1 0 1 1 1 0

0 1 0 1 0 0 1 exit found

After you get the basic algorithm down, print the number of steps taken for the exit path. Example:

1 0 0 0 1

1 1 1 1 0

0 0 1 0 1

0 1 1 1 1

0 0 0 0 0 exit found – 8 steps

**(Advanced) Printing the path itself**

If the maze has an exit, print **the path itself** (doesn't have to be a 'P' character). Example:

**P** 0 0 0 1

**P** **P** **P** 1 0

0 0 **P** 0 1

0 1 **P** **P** **P**

0 0 0 0 0 exit found – 8 steps

This is most likely going to require some extra storage to avoid printing "P" at every path location.

**(Advanced) Tougher recursion problems**

1. Complete a version of the timesTwo() method shown previously, except that it now prints the odd number in the middle of all the 2s. The number of 2s on either side of the odd number should be equal if possible – if not possible, the "extra" 2 should appear at the beginning of the expression. Examples:

timesTwo2(96) >>> 2 \* 2 \* 2 \* 3 \* 2 \* 2

timesTwo2(80) >>> 2 \* 2 \* 5 \* 2 \* 2

timesTwo2(68) >>> 2 \* 17 \* 2

1. Complete the method public List<String> anagrams(String prefix, String s, List<String> list), that returns a List of all possible anagrams of s.

anagrams("", "ABC", new ArrayList<String>()) >>> [ABC, ACB, BAC, BCA, CAB, CBA]

1. Complete the method public void backup(String dir, String dest) that recursively searches a directory specified by *dir* and copies all source code found (.java files) to a new folder in *dest*. Of course, use caution when moving files around! The File class will be very useful for this method.
2. Solve the problem at [projecteuler.net/problem=18](https://projecteuler.net/problem=18) using recursive backtracking find the maximum sum.
3. Write a recursive method that will find the number of possible phone numbers in a particular area code, given this list of starting numbers that are illegal: [555, 666, 911, 411, 311, 211].
4. [uva.onlinejudge.org/index.php?option=com\_onlinejudge&Itemid=8&category=645&page=show\_problem&problem=565](https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=645&page=show_problem&problem=565)
5. Write a method that will demonstrate a solution (can be graphical or text-based) to the Towers of Hanoi problem with *n* disks. In the Towers of Hanoi, there are *n* disks on 3 pegs. You can only move one disk at a time (from any peg to any other peg), and you may not stack a smaller disk on top of a larger disk.

