Names: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

More fun with recursion. Develop these recursive methods from CodingBat.com. On your own, go to CodingBat and try them. We will select one or two to demo during class today.

Work with a partner!

1. We have a number of bunnies and each bunny has two big floppy ears. We want to compute the total number of ears across all the bunnies recursively (without loops or multiplication).

bunnyEars(0) → 0
bunnyEars(1) → 2
bunnyEars(2) → 4

First – What is your base case? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next – What is the refinement? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public int bunnyEars(int bunnies)

{

}

2. Given **base** and **n** that are both 1 or more, compute recursively (no loops) the value of base to the n power, so powerN(3, 2) is 9 (3 squared).

powerN(3, 1) → 3
powerN(3, 2) → 9
powerN(3, 3) → 27

First – What is your base case? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next – What is the refinement? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public int powerN(int base, int n)

{

}

3. Given a string, compute recursively a new string where all the lowercase 'x' chars have been moved to the end of the string.

endX("xxre") → "rexx"
endX("xxhixx") → "hixxxx"
endX("xhixhix") → "hihixxx"

First – What is your base case? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next – What is the refinement? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public String endX(String str)

{

}

4. We have triangle made of blocks. The topmost row has 1 block, the next row down has 2 blocks, the next row has 3 blocks, and so on. Compute recursively (no loops or multiplication) the total number of blocks in such a triangle with the given number of rows.

triangle(0) → 0
triangle(1) → 1
triangle(2) → 3

First – What is your base case? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next – What is the refinement? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public int triangle(int rows) {

}

5. The fibonacci sequence is a famous bit of mathematics, and it happens to have a recursive definition. The first two values in the sequence are 0 and 1 (essentially 2 base cases). Each subsequent value is the sum of the previous two values, so the whole sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21 and so on. Define a recursive fibonacci(n) method that returns the nth fibonacci number, with n=0 representing the start of the sequence.

fibonacci(0) → 0
fibonacci(1) → 1
fibonacci(2) → 1

First – What are your base cases? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Next – What is the refinement? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public int fibonacci(int n)

{

}