REKURSION

FACTORIAL

Given n of 1 or more, return the factorial of n, which is n * (n-1) * (n-2) ... 1. Compute the result recursively (without loops).

```
factorial(1) → 1
factorial(2) → 2
factorial(3) → 6
```

BUNNIES

We have bunnies standing in a line, numbered 1, 2, ... The odd bunnies (1, 3, ..) have the normal 2 ears. The even bunnies (2, 4, ..) we'll say have 3 ears, because they each have a raised foot. Recursively return the number of "ears" in the bunny line 1, 2, ... n (without loops or multiplication).

```
bunnyEars2(0) → 0
bunnyEars2(1) → 2
bunnyEars2(2) → 5
```

COUNT-7

Given a non-negative int n, return the count of the occurrences of 7 as a digit, so for example 717 yields 2. (no loops). Note that mod (%) by 10 yields the rightmost digit (126 % 10 is 6), while divide (/) by 10 removes the rightmost digit (126 / 10 is 12).

```
count7(717) → 2
count7(7) → 1
count7(123) → 0
```

COUNT-X

Given a string, compute recursively (no loops) the number of lowercase 'x' chars in the string.

```
countX("xxhixx") → 4
countX("xhixhix") → 3
countX("hi") → 0
```

ARRAY-11

Given an array of ints, compute recursively the number of times that the value 11 appears in the array. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0.

```
array11([1, 2, 11], 0) → 1
array11([11, 11], 0) → 2
array11([1, 2, 3, 4], 0) → 0
```

PAIR STAR

Given a string, compute recursively a new string where identical chars that are adjacent in the original string are separated from each other by a "*".

```
pairStar("hello") → "hel*lo"
pairStar("xxyy") → "x*xy*y"
pairStar("aaaa") → "a*a*a*a"
```

TRIANGLE

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
```

ARRAY 220

Given an array of ints, compute recursively if the array contains somewhere a value followed in the array by that value times 10. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0.

```
array220([1, 2, 20], 0) → true
array220([3, 30], 0) → true
array220([3], 0) → false
```

END X

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"
```

STR COPIES

Given a string and a non-empty substring sub, compute recursively if at least n copies of sub appear in the string somewhere, possibly with overlapping. N will be non-negative.

```
strCopies("catcowcat", "cat", 2) → true
strCopies("catcowcat", "cow", 2) → false
strCopies("catcowcat", "cow", 1) → true
```

FIBONACCI

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
```

SUM DIGITS

Given a non-negative int n, return the sum of its digits recursively (no loops). Note that mod (%) by 10 yields the rightmost digit (126 % 10 is 6), while divide (/) by 10 removes the rightmost digit (126 / 10 is 12).

```
sumDigits(126) → 9
sumDigits(49) → 13
sumDigits(12) → 3
```

POWER N

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
```

ARRAY 6

Given an array of ints, compute recursively if the array contains a 6. We'll use the convention of considering only the part of the array that begins at the given index. In this way, a recursive call can pass index+1 to move down the array. The initial call will pass in index as 0.

```
array6([1, 6, 4], 0) → true
array6([1, 4], 0) → false
array6([6], 0) → true
```