factorial - factorial

Dado un entero n >= 1, retorne el factorial de n, el cual es n*(n-1) *(n-2)*…*1. Calcular el resultado de forma recursiva (sin ciclos).

bunnyEars - orejasDeConejo

Tenemos un número de conejitos y cada conejito tiene dos grandes orejas. Queremos calcular el número de orejas en todos los conejitos de forma recursiva (sin ciclos o multiplicación).

fibonacci - fibonacci

La secuencia fibonacci es una parte famosa de las matemáticas, y resulta que tiene una definición recursiva. Los primeros dos valores en la secuencia son 0 y 1 (esencialmente 2 casos base). Cada uno de los siguientes valores es la suma de los 2 valores previos, por lo que la secuencia completa es 0, 1, 1, 2, 3, 5, 8, 13, 21 y así sucesivamente. Defina un método recursivo fibonacci(n) que retorne el n numero Fibonacci, con n=0 representando el inicio de la secuencia.

bunnyEars2 – orejasDeConejo2

Tenemos conejitos posicionados en una línea, enumerados 1,2,…, n. Los conejos impares (1, 3, …) tienen 2 orejas (lo normal). Los conejos pares (2, 4, …) tienen 3 orejas, ya que cada uno tienen un pie levantado. Recursivamente retorne en número de "orejas" en la línea de conejos, contando hasta el conejito especificado (entero bunnies) (sin ciclos, ni multiplicación).

triangle - triangulo

Tenemos un triángulo hecho de bloques. La fila superior tiene 1 bloque, la siguiente fila abajo tiene 2 bloques, la siguiente fila tiene 3 bloques y así sucesivamente. Calcular recursivamente (sin ciclos o multiplicación) el número total de bloques en un triángulo como este con el número de filas dadas.

sumDigits - sumarDigitos

Dado un entero no negativo n, retorne la suma de sus dígitos recursivamente (sin ciclos). Note que el modulo (%) por 10 obtiene el dígito más a la derecha (126 % 10 es 6), mientras que dividiendo (/) por 10 removemos el dígito más a la derecha (126 / 10 es 12).

count7 – contar7

Dado un entero no negativo n, retorne la cantidad de ocurrencias de 7 como dígito, así que por ejemplo para 717 retornaría 2. (Sin ciclos). Tenga en cuenta que mod (%) por 10 da el dígito más a la derecha (126% 10 es 6), mientras que dividir (/) por 10 elimina el dígito más a la derecha (126/10 es 12).

count8 – contar8

Dado un entero no negativo n, calcule de forma recursiva (sin ciclos) la cantidad de ocurrencias de 8 como un dígito, excepto que un 8 con otro 8 inmediatamente a su izquierda cuenta por 2. Por ejemplo para 8818 sería 4. Tenga en cuenta que mod (%) por 10 produce el dígito más a la derecha (126% 10 es 6), mientras que dividir (/) por 10 elimina el dígito más a la derecha (126/10 es 12).

powerN – potenciaN

Dados dos enteros base y n, ambos >= 1, calcule recursivamente (sin ciclos) el valor de base elevada a la n-ésima potencia. Por ejemplo (3, 2) es 9 (3 al cuadrado).

countX - contarX

Dada una cadena, calcule recursivamente (sin ciclos) el número de caracteres 'x' (minúscula) que hay en esta.

countHi - contarHi

Dada una cadena, calcule recursivamente (sin ciclos) el número de veces que aparece "hi" (en minúscula) en la cadena.

changeXY - cambiarXY

Dada una cadena, calcule recursivamente (sin ciclos) una nueva cadena donde todos los caracteres 'x' (minúscula) han sido cambiados por el carácter 'y' (minúscula).

changePi – cambiarPi

Dada una cadena, calcule recursivamente (sin ciclos) una nueva cadena donde todas las apariciones de "pi" han sido remplazadas por "3.14".

noX - noX

Dada una cadena, calcule recursivamente una nueva cadena donde todos los caracteres 'x' han sido removidos.

array6 – array6

Dado un arreglo de enteros, calcule recursivamente si el arreglo contiene un 6. Vamos a utilizar la convención de considerar solo la parte del arreglo que comienza en el índice dado. De este modo, el llamado recursivo puede pasar index+1 para desplazarse por el arreglo. El llamado recursivo inicial será con índice 0.

array11 – array11

Dado un arreglo de enteros, calcule recursivamente el número de veces que aparece el número 11 en el arreglo. Vamos a utilizar la convención de considerar solo la parte del arreglo que comienza en el índice dado. De este modo, el llamado recursivo puede pasar index+1 para desplazarse por el arreglo. El llamado recursivo inicial será con índice 0.

array220 – array220

Dado un arreglo de enteros, calcule recursivamente si el arreglo contiene en algún lado un valor que en el arreglo esta seguido de sí mismo multiplicado por 10. Vamos a utilizar la convención de considerar solo la parte del arreglo que comienza en el índice dado. De este modo, el llamado recursivo puede pasar index+1 para desplazarse por el arreglo. El llamado recursivo inicial será con índice 0.

allStar - todosEstrella

Dada una cadena, calcule recursivamente una nueva cadena donde todos los caracteres adyacentes están ahora separados por un '*'.

pairStar - paresEstrella

Dada una cadena, calcule recursivamente una nueva cadena donde caracteres iguales que son adyacentes en la cadena original son separados uno del otro por un '*'.

endX – finX

Dada una cadena, calcule recursivamente una nueva cadena donde todos los caracteres 'x' (minúscula) han sido movidos al final de la cadena.

countPairs - contarPares

Diremos que un "par" en una cadena son dos instancias de un mismo carácter separados por otro carácter. Así en "AxA" las A hacen un par. Los pares pueden superponerse, por lo que "AxAxA" contiene 3 pares - 2 para A y 1 para x. Recursivamente calcule el número de pares en la cadena dada.

countAbc - contarAbc

Cuente recursivamente el número total de sub cadenas "abc" and "aba" que aparecen en la cadena dada.

count11 – contar11

Dada una cadena, calcule de forma recursiva (sin ciclos) el número de subcadenas "11" en la cadena. Las subcadenas "11" no deben superponerse.

stringClean - limpiarCadena

Dada una cadena, retorne recursivamente una cadena "limpia" en donde caracteres adyacentes iguales han reducido a un solo carácter. Ej.: "yyzzza" produce "yza".

countHi2 – contarHi2

Dada una cadena, compute de forma recursiva el número de veces que aparece "hi" (minúscula) en la cadena, sin embargo, no cuentan los "hi" que tienen una 'x' inmediatamente antes de él.

parenBit - parenBit

Dada una cadena que contiene un solo par de paréntesis, calcular de forma recursiva una nueva cadena hecha de sólo los paréntesis y su contenido, por lo que "xyz(abc)123" produciría "(abc)"

nestParen - parenAnidados

Dada una cadena, retorne true si esta es un anidamiento de cero o más pares de paréntesis, como por ejemplo "(())" o "((()))". Sugerencia: compruebe los primeros y últimos caracteres, después repetir con lo que está dentro de ellos.

strCount - contarStr

Dada una cadena y una subcadena sub no vacía, compute de forma recursiva el número de veces que sub aparece en la cadena, sin que las subcadenas se superpongan.

strCopies - copiasStr

Dada una cadena y una subcadena sub no vacía, compute de forma recursiva si cómo mínimo n copias de sub aparecen en la cadena en algún lugar. Las cadenas pueden superponerse. N será no negativo.

strDist - strDist

Dada una cadena y una subcadena sub no vacía, compute de forma recursiva la mayor subcadena que comienza y termina con sub y retorne su longitud.