S1-Recursión

Factorial de n - Básico

Dado un entero n >= 0, calcula el factorial de n, es decir, n!.

Entrada

Se te dará un entero n donde **0 <= n <= 15**

Salida

Un número entero que representa el factorial de n, es decir, n!

Ejemplo 1:

Entrada:
4
Salida:
24

Ejemplo 2:

Entrada:
6
Salida:
720

Suma de los números del 1 al n - Básico

Dada un entero n sumar todos los números desde el 1 hasta n, **calcula recursivamente** (sin usar ciclos, ni ecuaciones pitagóricas). Imprime la sumatoria.

Entrada

Se te dara un entero n donde **0 <= n <= 990**

Salida

Un número entero que indique la sumatoria desde el 1 hasta el entero n.

Ejemplo 1:

Entrada:
5
Salida:
15

Ejemplo 2:

Entrada:
10
Salida:
55

Nota: Si la entrada es 5, imprime 15 porque 1+2+3+4+5 = 15

Pares (con recursión) - Básico

Número máximo de ficheros: 1
Tipo de trabajo: Individual

Cree un programa en Python que le pida al usuario por pantalla un número (int), luego imprima todos los número pares, desde 2 hasta el número que ingreso el usuario (incluyéndolo si es par).

Ejemplo:

Si el usuario ingresa 8 el programa debe mostrar:

2
4
6
8

Pares raros (con recursión)

Número máximo de ficheros: 1
Tipo de trabajo: Individual

Cree un programa en Python que le pida al usuario por pantalla un número (int), luego imprima todos los número pares, desde 2 hasta el número que ingreso el usuario (incluyéndolo si es par). Pero ojo, nunca imprima ni el 6 ni el 8, ya que son malignos.

Ejemplo:

Si el usuario ingresa 15 el programa debe mostrar:

2
4
10
12
14

Notas:

1) Elabore su programa en un archivo de Python (llamado **paresraros.py**). Si omite la extensión le aparecerá un error de compilación. No deje espacios en blanco en el

Crear una nueva cadena cambiando la x por la y

Nombre archivo: cambiarx.py

Crea una función la cual reciba una cadena de caracteres y cambie recursivamente (sin usar ciclos) cada caracter 'x' que aparece en la cadena por el caracter 'y'.

NO usar replace porque replace, internamente, usa ciclos.

Nota: Debes recibir la cadena por pantalla, llamar la función para ejecutarla e imprimir el resultado por pantalla.

Entrada

Se te dara una cadena de tamaño n donde 0 <= n <= 300.

Salida

Una cadena que tenga el caracter 'y' donde estaba anteriormente la 'x'.

Ejemplo 1:

Entrada:
xadcxfx
Salida:
yadcyfy

Cantidad de x

Nombre archivo: cantidad.py

Dada una cadena de caracteres en minúscula, crea una función que calcule recursivamente (sin usar ciclos) el número de caracteres 'x' que aparecen en la cadena.

NO usar count porque count usa ciclos internamente.

Nota: Debes recibir la cadena por pantalla, llamar la función para ejecutarla e imprimir el resultado por pantalla.

Entrada

Se te dará una cadena de tamaño n donde **0 <= n <= 300**

Salida

Un número entero que indique la cantidad de 'x' en la cadena

Ejemplo 1:

Entrada:
xadxx
Salida:
3

Euclides

Nombre archivo: euclides.py

Actualmente, las baldosas del piso se hacen de diversos materiales como granito, cerámica, madera o mármol. Seria ideal que las baldosas se ajustaran exactamente al tamaño de la habitación, sin dejar sobrantes; de esa manera, no habría que cortar las baldosas, desperdiciando un material que a menudo es difícil de reciclar y que su extracción tuvo un impacto negativo sobre el medio ambiente. Para lograr esto, se debe elegir un tamaño de baldosa cuadrada más grande que divida exactamente tanto al largo como el alto de la habitación. Un algoritmo para lograr encontrar el tamaño de una baldosa cuadrada más grande que divida exactamente tanto el ancho como el alto de una habitación es el algoritmo recursivo de Euclides.

Nota: Llama la función para ejecutarla en caso de haberla creado e imprime el resultado por pantalla.

Entrada

Se dará la cantidad n de casos de prueba donde 0 < n < 100, en cada caso de prueba se darán dos números j y k donde 0 < j <= k < 10^3

Salida

Para cada caso de prueba se debe escribir el texto "Case #:" donde # es el numero correspondiente del caso de prueba seguido con el tamaño máximo de la baldosa que divida exactamente tanto al largo como el alto de la habitación.

Ejemplo 1:

Entrada
1 60 48
Salida
Case 1: 12

SubTexto (con recursión)

Número máximo de ficheros: 1
Tipo de trabajo: Individual

Cree un programa en Python que le pida al usuario por pantalla un texto (string) (texto1). Luego pida una posición inicial (int), y luego pida una posición final (int). Luego imprima la subcadena que representa el texto ingresado, desde la posición inicial ingresada, hasta la posición final ingresada.

Ejemplo:

Si el usuario ingresa *plantita*, luego el usuario ingresa *1*, y luego el usuario ingresa *4*, el programa debe mostrar:

lant

Notas:

1) Elabore su programa en un archivo de Python (llamado **subtexto.py**). Si omite la extensión le aparecerá un error de compilación. No deje espacios en blanco en el

Más Corrupto (con recursión)

Límite de entrega: Friday, 4 de November de 2022, 12:00
Número máximo de ficheros: 1
Tipo de trabajo: Individual

Cree dentro del programa una función que se llame **mascorrupto**, y que reciba un arreglo.
El arreglo tendrá 6 elementos, los elementos en posiciones pares del arreglo (indices) serán nombres de politicos corruptos, y los impares la plata que se han robado cada uno de ellos.

Ejemplo: ["Calvarini",100,"Pinturosky",200,"Tajardini",400]. Según el ejemplo Tajardini se robo 400 trillones de dolares, y Pinturosky se robo 200.

La función debe retornar el nombre del politico más corrupto.

Notas:

1) Elabore su programa en un archivo de Python (llamado **corruptos.py**). Si omite la extensión le aparecerá un error de compilación. No deje espacios en blanco en el

S1 - Palíndromo Recursivo

Ficheros requeridos: ejercicio.py (Descargar)
Tipo de trabajo: Individual

RECURSIVO

Dado un string S retornar true si este es un palíndromo

Un palíndromo es una palabra que se lee igual de izquierda a derecha que de derecha a izquierda (EJ: amoroma, 12321, etc...)

Entrada

Lee por pantalla la primera línea, en la cual estará el string S
Salida

Imprime True si S es un palíndromo; de los contrario, False

Ejemplo 1:

Input:
reconocer
Output:
True

Puntos extra: Preguntar por padalustro

S1 - Suma Subgrupo

Nombre archivo: subgrupo.py

En el videojuego Dark Souls 2, cada jugador tiene un inventario elementos que puede cargar. Los elementos que puede cargar no deben superar un peso máximo definido en el juego. Ejemplos de estos elementos son armas y accesorios.

Un problema que ocurre con frecuencia en el juego es que uno encuentra tesoros con muchas armas y accesorios, más de los que uno puede cargar. Una solución es escoger un subconjunto de estos elementos cuya suma sea igual al peso máximo.

Implementen un algoritmo que, dados los ***n*** pesos de un grupo de elementos, determine si existe o no existe un subgrupo de elementos cuya suma sea igual a un peso máximo.

Nota: Recuerda leer por pantalla las entradas

Entrada

En la primera línea se dará un número n donde 0 < n < 10^3 y un peso k donde 0 < k < 10^9. En la segunda línea se darán ***n*** pesos ***p*** donde 0 < p < 10^3

Salida

Imprimir "YES" si existe un subgrupo de elementos cuya suma sea igual a un peso máximo en otro caso imprimir "NO"

Ejemplo 1:

Entrada
5 7 1 3 2 4 1
Salida
YES

S1 - Suma por grupos 2

Número máximo de ficheros: 1
Tipo de trabajo: Individual

RECURSIVO - Nombre de archivo: ejercicio.py

Dado un arreglo, ¿ Es posible encontrar un grupo de enteros que su suma sea igual a K?
Condición: Todos los múltiplos de 5 tienen que estar incluidos en el grupo. Si el valor inmediatamente siguiente a un múltiplo de 5 es 1, entonces no se debe elegir ese 1.

Entrada

En la primera línea estará un entero N: el número de elementos del arreglo
En las siguientes N líneas estarán los enteros que conforman el arreglo
En la última línea estará el entero K: El valor objetivo

Salida

Imprimir True si existe un subgrupo tal que sume a K, de resto False

Ejemplo 1:

Input:
4 2 5 10 4 19
Output:
true

Explicacion
El subgrupo posible:

4 + 5 + 10 = 19

S1 - Subconjuntos de una cadena

Nombre archivo: subconjuntos.py

Una empresa de helados quiere saber cuáles son las combinaciones de helado favoritas por el público, para esto primero encuentran todas las posibles formas de armar el helado. Teniendo en cuenta que cada caracter es un sabor de helado, imprime todas las posibles posibles subconjuntos de sabores que se pueden elegir para formar los helados. Es decir, subconjuntos de la cadena.Imprime todos los subconjuntos posibles que se puedan elegir para formar el helado.

Ejemplo:

La cadena "cvf" significan los helados chocolate, vainilla, fresa.

Estos tres sabores de helados pueden ser armados de la siguiente forma: (vacío)-c - v - f - cv - cf - vf - cvf

Las anteriores combinaciones son subconjuntos de la cadena.

Entrada

En la primera línea se te dará una cadena n, donde 0 < n < 300 y cada caracter es un sabor de helado. Ten en cuenta que debes formar todos los subconjuntos que correspondan a la cadena dada. (Incluye el vacío)

Salida

Imprime todas las posibilidades de formar el helado a partir de los sabores que se dan.

Ejemplo 1:

Input:
"sna"
Output:
sna sn sa s na n a -> Espacio que indica el subconjunto vacío, debes imprimir el vacío

S1 - ¿De cuántas maneras?

Nombre archivo: maneras.py

Existe un juego en el que un jugador puede ganar 3, 5 o 7 puntos en un solo turno. Realiza una función de forma recursiva la cual calcule de cuántas maneras puede el jugador obtener un total de ***N*** puntos.

Nota: Debes recibir el valor de N por pantalla, llamar la función para ejecutarla e imprimir el resultado por pantalla.

Entrada

Recibe el número ***N*** entero el cual indica el total de puntos a obtener donde -20 < N <=70

Salida

Retorna la cantidad de maneras con las cuales se logra obtener la puntuación **N.**

Nota: Debes recibir la cadena por pantalla, llamar la función para ejecutarla e imprimir el resultado por pantalla.

Ejemplo 1:

Input:
10
Output:
3

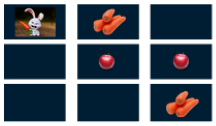
Input:
12
Output:
3

Puntos extra: Preguntar por padalustro

S1 - El conejo

Nombre archivo: conejo.py

Un conejo está ubicado en la celda **(0,0)** de un tablero **A** de tamaño **n x m**. Si el conejo está en la casilla **(i , j)**, el conejo puede avanzar *-únicamente-* horizontalmente a la casilla **(i + 1, j)** o verticalmente a la casilla **(i, j + 1)**. En algunas celdas **(ik , jk)**, hay manzanas que le darán un grado de satisfacción **d** al conejo y, en otras celdas **(ij , jh)** hay zanahorias que le darán satisfacción **k** al conejo. Donde hay manzanas estará el carácter '*****' y donde hay zanahorias estará el carácter '**#**'. Dónde no hay ni zanahorias ni manzanas, está el carácter '**.**' ¿Cuál es la mayor satisfacción que puede tener el conejo, si el conejo recorre el tablero de la manera anteriormente mencionada y tiene que llegar a la posición **(n-1, m-1)** del tablero?



Entrada

En la primera línea se te dará dos números n y m donde 0 < n <= m < 15, en la siguiente dos números d y k donde 0 < d <= k < 10^3 seguido por la representación del tablero n*m

Salida

Un número entero de la satisfacción máxima que puede tener el conejo al llegar a la posición **(n-1, m-1)** del tablero

Ejemplo 1:

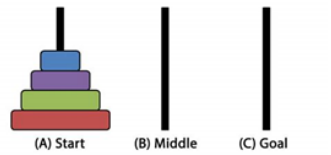
Input:
3 3 5 2 . # . . * * . . #
Output:
14

S1 - Torres de Hanoi

Nombre archivo: torres.py

La torre de Hanoi es un famoso rompecabezas en el que tenemos tres varas y N discos. El objetivo del rompecabezas es mover toda la pila a otra vara. Se te da el número de discos N. Inicialmente, estos discos están en la vara 1. Tienes que imprimir todos los pasos del movimiento de los discos para que todos los discos lleguen a la tercera vara. Además, necesitas encontrar el total de movimientos.

Nota: Los discos están dispuestos de tal manera que el disco superior está numerado 1 y el disco inferior está numerado N. Además, todos los discos tienen diferentes tamaños y un disco más grande no puede ser puesto en la parte superior de un disco más pequeño.



Entrada

Se te da la cantidad N de discos donde 1 <= N <= 16

Salida

Tienes que imprimir todos los pasos con el siguiente formato “Mover disco #D de la vara #1 a la vara #2” seguido por la cantidad total de movimiento

#D: número del disco

#1 : origen

#2: destino

Ejemplo 1:

Entrada
2
Salida
Mover disco 1 de la vara 1 a la vara 2 Mover disco 2 de la vara 1 a la vara 3 Mover disco 1 de la vara 2 a la vara 3 3

S1 - Permutaciones

Nombre archivo: permutaciones.py

Una empresa de ciberseguridad tiene como objetivo descifrar la contraseña de sus clientes para así probar la seguridad actual con la que cuentan sus nuevos clientes y así brindarles la mejor seguridad.

Implementa un algoritmo recursivo para descifrar la contraseña probando todas las permutaciones posibles de los caracteres de un arreglo de caracteres. Teniendo en cuenta, por simplicidad, que las letras de la contraseña no se repiten y que una contraseña es una permutación de la cadena de caracteres ingresada por el usuario.

Explicación permutación (Definición de 2 min y ejemplos):

Entrada

En la primera línea se dan los caracteres a permutar.

Salida

Imprime todas las permutaciones posibles de la contraseña en **formato string**

Ejemplo 1:

Input:
a b c
Output:
abc acb bac bca cab cba

S1 - Subsecuencia común más larga

Número máximo de ficheros: 1
Tipo de trabajo: Individual

Problema - Nombre de archivo: ejercicio.py

Dados dos strings, retornar la longitud de la subsequencia común más larga.

Una subsequencia de un string es un subtring formado al borrar caracteres del original SIN modificar el orden.

Entrada

En la primera línea se encuentra el primer string

En la segunda línea se encuentra el segundo string

Salida

Imprimir la longitud de la LCS.

Ejemplo 1:

Input:
AZBMNCPD ABUYCED
Output:
4

Explicación:
La subsecuencia común más larga es ABCD

S1 - Equilibra la balanza

En un parque de diversiones hay un juego llamado equilibra la balanza, en este juego participan N personas, las cuales deben dividirse en dos grupos de tamaño similar, y cada grupo debe subirse a un platillo de la balanza. La forma de ganar el juego es organizando estratégicamente los grupos teniendo en cuenta los pesos de las personas, tales que en un solo intento la balanza quede equilibrada o lo más equilibrada posible. Para ganar el juego debes crear una función *equilibra* que reciba un arreglo con el peso de cada persona (NO recibirlo por pantalla usando *input*), luego, formar los dos grupos estratégicamente y retornar una tupla con ambas listas ordenadas de forma ascendente (NO imprimirlas en pantalla). Recuerda que una balanza se equilibra si los pesos que hay en cada platillo son lo más iguales posibles. Y recuerda que el número de personas puede ser par o impar.

Ejemplo 1:

Entrada

personas=[80, 59,60,81,57,58,76,75]

Salida

[58,60,75,80], [57,59,76,81]

Nota: la suma de los pesos de cada grupo son 273 los dos, por lo que su diferencia es de 0, es decir, la balanza está equilibrada.

S1 - María Paulina quiere ir a N

Número máximo de ficheros: 1
Tipo de trabajo: Individual

Problema - Nombre de archivo: ejercicio.py

María quiere llegar hasta N (O más) de un camino representado por un arreglo, para esto ella tiene una energía M y puede dar 0, 1, 2, ..., K saltos de posición a posición.

Tenemos un entero N, y un arreglo "arr" que tiene N espacios. En cada espacio del arreglo tenemos un entero X. Si estamos en la casilla i, moverse a la casilla i + k necesita abs(arr[i] - arr[i+k]) de energía.

Devolver True si es posible llegar hasta N con el arreglo, saltos y energía disponible, False si no es posible.

Entrada

En la primera línea está el tamaño del arreglo N

En las siguientes N líneas estarán los valores de energía de cada casilla

En la siguiente línea estará la energía M

En la última línea estará el número de saltos disponible K

Salida

Imprimir True si se puede llegar hasta N, de resto False

Ejemplo 1 (Lo que está en paréntesis es solo para aclarar):

Input:
7 (La N) 1 2 2 1 3 2 1 4 (La energía M) 2 (Los saltos disponibles K)
Output:
True

Explicación:

[(1), 2, (2), (1), 3, (2), 1] (finaliza)

María puede pasar de la casilla 1 (Que vale 1) a la casilla 3 (vale 2) entonces gasta abs(1 -2)=1 de energía, le queda 3.

De ahí solo avanza una casilla para gastar abs(2 -1)=1 de energía, le queda 2.

El tercer paso es dar un salto de 2 para llegar a la casilla 6 y gastar abs(1-2)=1 de energía, le queda 1.

Finalmente desde la 6 ya puede llegar a N sin gastar más energía, por lo que acaba.

S1 - Paint!

RECURSION

Una imagen esta siendo representada por una matriz MxN llena de enteros, donde imagen[i][j] representa un pixel de la imagen.

Se van a dar 3 valores, X, Y y ColorNuevo con los que debes realizar un algoritmo de relleno.

Un algoritmo de relleno consiste en tomar el valor del pixel imagen[X][Y] y reemplazar a este y a todos sus pixeles adyacentes (y a los adyacentes de los adyacentes, etc...) que tengan el mismo color al del pixel imagen[X][Y] por el ColorNuevo.

Un pixel adyacente solo son los de arriba, abajo, izquierda y derecha, no diagonalmente.

Entrada

No hay que recibir nada, la funcion dada tiene la matriz a recibir, el X y Y (La coordenada de inicio) y el ColorNuevo como parametro.

Salida

Retornar la matriz modificada, NO CREAR UNA NUEVA, operar sobre la original.

Ejemplo 1:

Input:
matriz = [[1,1,1],[1,1,0],[1,0,1]] X=1 Y=1 ColorNuevo = 2
Output:
[[2,2,2],[2,2,0],[2,0,1]]

Representacion mas grafica: (Empezamos en el 1 entre "[ ]")
1 1 1 -> 2 2 2
1 [1] 0 -> 2 2 0
1 0 1 -> 2 0 1

S1 - Laberinto - Demandante

S1 - Laberinto - Demandante

Ficheros requeridos: camino.py (Descargar)
Tipo de trabajo: Individual

Rodolfo tiene problemas para encontrar su casa, él sabe las posiciones donde esta parado, y donde esta su casa, pero como esta en un laberinto no sabe como llegar por lo cual le pide ayuda a usted. Un laberinto en forma de matriz rectangular binaria, encuentren la longitud del camino más corto en el laberinto desde un origen dado hasta un destino dado. El camino sólo puede construirse con celdas que tengan valor 1, y en cualquier momento, sólo podemos movernos en un paso de las cuatro direcciones. El laberinto se recibe como parámetro como una lista de listas de Python. Por otro lado, el origen y el destino se reciben como parámetro en forma de tupla de enteros.

Nota: En caso de no haber camino retornar -1

Ejemplo

Entrada

origen = (0, 0)

destino = (7, 5)

laberinto =

[1, 1, 1, 1, 1, 0, 0, 1, 1, 1],

[0, 1, 1, 1, 1, 1, 0, 1, 0, 1],

[0, 0, 1, 0, 1, 1, 1, 0, 0, 1],

[1, 0, 1, 1, 1, 0, 1, 1, 0, 1],

[0, 0, 0, 1, 0, 0, 0, 1, 0, 1],

[1, 0, 1, 1, 1, 0, 0, 1, 1, 0],

[0, 0, 0, 0, 1, 0, 0, 1, 0, 1],

[0, 1, 1, 1, 1, 1, 1, 1, 0, 0],

[1, 1, 1, 1, 1, 0, 0, 1, 1, 1],

[0, 0, 1, 0, 0, 1, 1, 0, 0, 1]

Salida

12

Explicacion:

[~~1, 1, 1~~, 1, 1, 0, 0, 1, 1, 1],

[0, 1, ~~1~~, 1, 1, 1, 0, 1, 0, 1],

[0, 0, ~~1~~, 0, 1, 1, 1, 0, 0, 1],

[1, 0, ~~1~~, ~~1~~, 1, 0, 1, 1, 0, 1],

[0, 0, 0, ~~1~~, 0, 0, 0, 1, 0, 1],

[1, 0, 1, ~~1~~, ~~1~~, 0, 0, 1, 1, 0],

[0, 0, 0, 0, ~~1~~, 0, 0, 1, 0, 1],

[0, 1, 1, 1, ~~1~~, ~~1~~, 1, 1, 0, 0],

[1, 1, 1, 1, 1, 0, 0, 1, 1, 1],

[0, 0, 1, 0, 0, 1, 1, 0, 0, 1]

Ficheros requeridos

camino.py

```
1 def camino(laberinto, origen, destino):
2     # codigo aqui
```

***S1* - Las N reinas**

Número máximo de ficheros: 1
Tipo de trabajo: Individual

RECURSIVO - Nombre de archivo: ejercicio.py

Dado un tablero de tamaño NxN, es posible ubicar N reinas cada una en una columna distinta sin que se amenacen entre ellas? Devolver el numero de posibilidades

Entrada

En la primera linea estara el entero N
Salida

Imprimir el número de tableros validos para el tamaño N

Ejemplo 1:

Input:
8
Output:
92