Universidad Jorge Tadeo Lozano
Estructuras de Datos
Prueba de Programación: Segundo momento
2025-2 (**Hasta Diccionarios y Listas**)

Fecha: 17 de septiembre de 2025
Profesor: José Alejandro Franco Calderón
Monitor: Ludwig Alvarado Becerra

Cabito Organizadito

Autor: Ludwig Alvarado

Cabito (figura 1 a la izquierda) es la mascota de la Universidad Jorge Tadeo Lozano. Él es el sucesor de Cabo (figura 1 a la derecha, lo vamos a llamar Don Cabo), un perrito muy querido por toda la comunidad Tadeísta (al igual que Cabito) hace muchos años... Cabito llego como un compañero de Don Cabo y es el que hoy en día conocemos en la Tadeo. Estos dos amados perritos en el corto tiempo que estuvieron juntos se pasaron algunas *mañas*, en especial Don Cabo (como gran mentor) a Cabito.

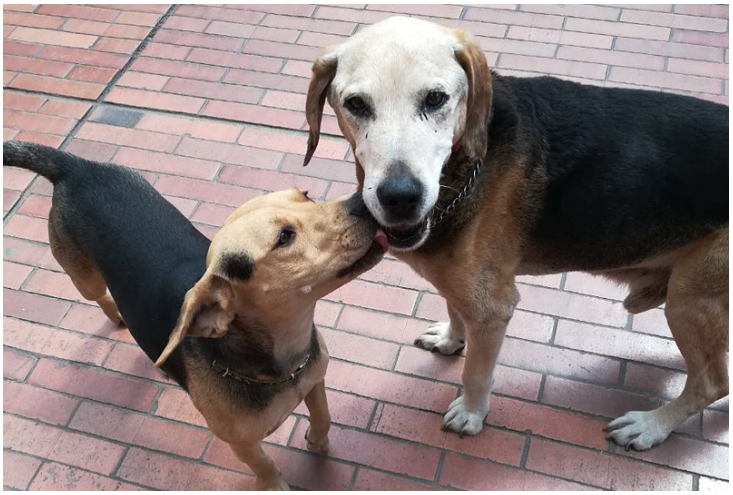

Figure 1: Cabito y Cabo. Autor de la foto: Universidad Jorge Tadeo Lozano

Entre una de las muchas *mañas*, está la comida... Don Cabo en un día de enseñanzas a Cabito le dijo: "*guau guau, guauuu guau guau woof, woof guau wooooof...*" En español esto quiere decir "Cabito, cuando tengas hambre, pídele comida a esos ingenuos estudiantes..." Siguiendo sus aprendizajes, Cabito mira con sus tiernos ojos a los estudiantes en busca de comida y ellos le dan (**¡Nunca le debes dar comida a Cabito!**).

Cada vez que alguien le da comida a Cabito él se va a su *guarida secreta* y coloca la comida que le han dado en el suelo y organiza las unidades de comida por filas. Cabito sigue el siguiente patrón; en la primera fila coloca 1 unidad de comida, en la segunda fila coloca 2 unidades de comida, y así sucesivamente, entonces en la fila i Cabito coloca i unidades de comida.

Al final de cada día Cabito quiere saber cuántas filas de comida puede organizar y tu misión es escribir un programa para ayudarlo.

Entrada del programa

Cabito te va a dar un número entero ($1 \leq n \leq 100$), la cantidad total de unidades de comida que logró recolectar a lo largo del día. Recuerda que Cabito organiza su comida

por filas y en la primera fila debe haber 1 unidad de comida, en la segunda fila 2 unidades de comida, en la tercera fila 3 unidades de comida y así sucesivamente...

Salida del programa

Lo que tu programa debe hacer es imprimir un número entero, la cantidad de filas que Cabito puede formar con el n dado. Si Cabito te da $n = 3$ (tres unidades de comida), en la primera fila hay una unidad de comida y en la segunda dos. Por lo tanto, lo que debes imprimir es `2` (porque hay 2 filas en total). Sin embargo, hay un detalle extra, en la fila i **deben haber mínimo** i unidades de comida. Es decir, en el caso de que Cabito te de $n = 4$ la comida se debe organizar así; en la primera fila una unidad de comida, en la segunda fila **3** unidades de comida. Esto se debe a que para tener una tercera fila deben haber **mínimo** 3 unidades de comida, por lo tanto, para el caso $n = 4$ tu programa debe imprimir `2`.

Casos de prueba de ejemplo

Prueba #1

Entrada	Salida
3	2

Prueba #2

Entrada	Salida
6	3

Prueba #3

Entrada	Salida
4	2

Notas

En la primera prueba, Cabito reparte `3` unidades: pone 1 en la primera fila y 2 en la segunda, usando en total `2` filas.

En la tercera prueba, con `4` unidades: coloca 1 en la primera fila y 3 en la segunda. No alcanza para una tercera (que requeriría 6 unidades en total, vea prueba 2), así que también se usan `2` filas.

Aspectos Importantes

1. La prueba se resolverá en **2 momentos**: primero en papel, sin editor de código (**valdrá 3.0 puntos**). Segundo, pasará el algoritmo desarrollando en papel al editor de código (**valdrá 2.0 puntos**).

2. Hagan uso de los conceptos aprendidos hasta el tema de **diccionarios y listas**.

3. Asuman que el valor que se va a ingresar es de tipo entero positivo.

4. Pueden hacer uso del concepto de listas por comprensión.

5. Pueden usar solo la función `range()`, no se permite el uso de otras funciones, si tienen dudas pregunten al profesor si es permitida, de lo contrario si la usa y no es permitida (-1.0 puntos).

6. No importe ninguna librerías externas, no las necesita, en caso de hacerlo (-1.0 puntos).

7. Si utilizan declaraciones `break` o `continue` (-0.2 puntos).

8. No preocuparse por validaciones de errores, asuma que los datos se suministran de manera correcta.

9. Prestar atención al formato de entrada como al formato de salida para la escritura y presentación del algoritmo.