1) Códigos filtrados [2 puntos]
El hijo del dueño de la veterinaria, cuya actividad principal es ver tik toks, cree que los productos
cuyos código de barras terminimoan en números primos son especialmente auspiciosos y deben ser destacados
en la tienda. Luego de convencer a su padre de esta idea, solicita una función en python que facilite
esta gestión.

Se pide implementar una función que, dada una secuencia de enteros, cada uno representando un código
de barras de un producto, cree y devuelva una nueva lista que contenga únicamente aquellos números de
la lista original cuyos últimos tres dígitos formen un número primo (por ejemplo, 101, 002 y 011).

Nota: un número primo es aquel que solo es divisible por si mismo y por 1. Algunos ejemplos de hasta
tres dígitos son 2, 3, 4, 101, 103, 107, etc.

problema filtrar_codigos_primos(in codigos_barra: seq<Z>) : seq<Z> {
requiere: {Todos los enteros de codigos_barra tienen, por lo menos, 3 dígitos}
requiere: {No hay elementos repetidos en codigos_barra}
asegura: {los últimos 3 dígitos de cada uno de los elementos de res forman un número primo}
asegura: {Todos los elementos de codigos_barra cuyos últimos 3 dígitos forman un número primo
están en res}
asegura: {Todos los elementos de res están en codigos_barra}
}

2) Cambios de stock de stock_productos [2 puntos]

En la veterinaria "Exacta's pets", al finalizar cada día, el personal registra en papeles los nombres y
la cantidad actual de los productos cuyo stock ha cambiado. Para mejorar la gestión, desde la dirección
de la veterinaria han pedido desarrollar una solución en Python que les permita analizar las
fluctuaciones del stock.

Se pide implementar una función que reciba una lista de tuplas, donde cada tupla contiene el nombre de
un producto y su stock en ese momento. La función debe procesar esta lista y devolver un diccionario
que tenga como clave el nombre del producto y como valor una tupla con su mínimo y máximo stock histórico
registrado.

problema stock_productos(in stock_cambios: seq<<String X Z>>): dict<String, <Z X Z>>{
requiere: {Todos los elementos de stock_cambios están formados por un string no vacío y un entero >= 0}
asegura: {res tiene como claves solo los primeros elementos de las tuplas de stock_cambios (o sea, un
producto)}
asegura: {res tiene como claves todos los primeros elementos de las tuplas de stock_cambios}
asegura: {El valor en res de un producto es una tupla de cantidades. Su primer elemento es la menor
cantidad de ese producto en stock_cambios y como segundo valor el mayor}
}

3) Matriz de responsables por turnos [2 puntos]

Las personas responsables de los turnos están anotadas en una matriz donde las columnas representan los

días, en orden de lunes a domingo, y cada fila a un rango de una hora. Hay cuatro filas para los turnos
de la mañana (9, 10, 11 y 12 hs) y otras cuatro para la tarde (14, 15, 16 y 17).

Para hacer más eficiente el trabajo del personal de la veterinaria, se necesita analizar si quienes
quedan de responsables, están asignadas de manera continuada en los turnos de cada día.

Para ello se pide desarrollar una función en Python que, dada la matriz de turnos, devuelva una lista
de tuplas de Bool, una por cada día. Cada tupla debe contener dos elementos. El primer elemento
debe ser
True sí y solo sí todos los valores de los turnos de la mañana para ese día son iguales entre sí. El
segundo elemento debe ser True sí y solo sí todos los valores de los turnos de la tarde para ese día
son iguales entre sí. Siempre hay una persona responsable en cualquier horario de la veterinaria.

problema un_responsable_por_turno(in grilla_horaria: seq<seq<String>>): seq<(Bool x Bool)> {
requiere: {|grilla_horaria| = 8}
requiere: {Todos los elementos de grilla_horaria tienen el mismo tamaño (mayor a 0 y menor a 8)}
requiere: {No hay cadenas vacías en las listas de grilla_horaria}
asegura: {|res| = |grilla_horaria[0]|}
asegura: {El primer valor de la tupla en res[i], con i:Z, 0 <= i < |res| es igual a True <==> los primeros
4 valores de la columna i de grilla_horaria son iguales entre sí}
asegura: {El segundo valor de la tupla en res[i], con i:Z, 0 <= i < |res| es igual a True <==> los últimos
4 valores de la columna i de grilla_horaria son iguales entre sí}
}

4) Subsecuencia más larga [2 puntos]

Con el objetivo de organizar el flujo de pacientes, en la veterinaria se anotan los tipos de mascotas
que van ingresando al local. Se necesita identificar las consultas que involucran solo a perros y gatos.
Por eso, se decide desarrollar una función en Python que encuentre la secuencia más larga de
consultas
consecutivas que solo contenga los tipos de mascota "perro" o "gato".

Se pide implementar una función que, dada una secuencia de Strings, que representan los tipos de
animales
atendidos, devuelva el índice donde comienza la subsecuencia más larga que cumpla con estas
condiciones.

problema subsecuencia_mas_larga(in tipos_pacientes_atendidos: seq<String>): Z{
requiere: {tipos_pacientes_atendidos tiene, por lo menos, un elemento "perro" o "gato"}
asegura: {res es el índice donde empieza la subsecuencia más larga de tipos_pacientes_atendidos
que
contenga solo elementos "perro" o "gato"}
asegura: {Si hay más de una subsecuencia de tamaño máximo, res tiene el índice de la primera}
}