1) Atención por guardia (1 punto)
Desde el Hospital Fernandez nos pidieron solucionar una serie de problemas relacionados con la información que maneja
sobre los pacientes y el personal de salud. En primer lugar debemos resolver en qué orden se deben atender los pacientes
que llegan a la guardia. En enfermería, hay una primera instancia que clasifica en dos colas a los pacientes: una urgente y
otra postergable (esto se llama hacer triage). A partir de dichas colas que contienen la identificación del paciente, se pide
devolver una nueva cola según la siguiente especificación.

problema orden_de_atencion (in urgentes: cola <Z>, in postergables: cola <Z>): cola <Z> {
requiere: {no hay elementos repetidos en urgentes}
requiere: {no hay elementos repetidos en postergables}
resquiere: {la intersección entre postergables y urgentes es vacía}
requiere: {|postergables| = |urgentes|}
asegura: {no hay repetidos en res}
asegura: {res es permutación de la concatenación de urgentes y postergables}
asegura: {Si urgentes no es vacía, en tope de res hay un elemento de urgentes}
asegura: {En res no hay dos seguidos de urgentes}
asegura: {En res no hay dos seguidos de postergables}
asegura: {Para todo c1 y c2 de tipo "urgente" pertenecientes a urgentes si c1 aparece antes que c2 en urgentes entonces c1
aparece antes que c2 en res}
asegura: {Para todo c1 y c2 de tipó "postergable" pertenecientes a postergables si c1 aparece antes que c2 en postergables
entonces c1 aparece antes que c2 en res}
}

2) Chicken Game (3 puntos)
El juego del gallina es una competición en la que dos participantes conducen un vehículo en dirección al del contrario; si
alguno se desvía de la trayectoria de choque pierde y es humillado por comportarse como un "gallina". Se hizo un torneo
para ver quién es el menos gallina. Juegan todos contra todos una vez y van sumando puntos, o restando. Si dos jugadores
juegan y se chocan entre sí, entonces pierde cada uno 5 puntos por haberse dañado. Si ambos jugadores se desvían, pierde
cada uno 10 puntos por gallinas. Si uno no se desvía y el otro sí, el gallina pierde 15 puntos por ser humillado y el ganador
suma 10 puntos! En este torneo, cada persona que participa tiene una estrategia predefinida para competir: o siempre se
desvía, o nunca lo hace. Se debe programar la función 'torneo_de_gallinas' que recibe un diccionario (donde las claves
representan los nombres de los participantes que se anotaron en el torneo, y los valores sus respectivas estrategias) y
devuelve un diccionario con los puntajes obtenidos por cada jugador.

problema torneo_de_gallinas(in estrategias: dict<String, String>) : dict<String,Z> {
requiere: {estrategias tiene por lo menos 2 elementos(jugadores)}
requiere: {Las claves de estrategias tienen longitud mayor a 0}
requiere: {Los valores de estrategias sólo pueden ser los strings "me desvío siempre" ó "me la banco y no me desvío"}
asegura: {Las claves de res y las claves de estrategias son iguales}
asegura: {para cada jugador p perteneciente a claves(estrategias), res[p] es igual a la cantidad de puntos

que obtuvo al
finalizar el torneo, dado que jugó una vez contra cada otro jugador}
}

3) Cuasi Ta-Te-Ti (2 puntos)
Ana y Beto juegan al Ta-Te-Ti-Facilito. El juego es en un tablero cuadrado de lado entre 5 y 10. Cada jugador va poniendo su
ficha en cada turno. Juegan intercaladamente y comienza Ana. Ana pone siempre una "X" en su turno y Beto pone una "O" en
el suyo. Gana la persona que logra poner 3 fichas suyas consecutivas en forma vertical. SI el tablero está completo y no ganó
nadie, entonces se declara un empate. El tablero comienza vacío, representado por " " en cada posición.
Notar que dado que juegan por turnos y comienza Ana poniendo una "X" se cumple que la cantidad de "X" es igual a la
cantidad de "O" o bien la cantidad de "X" son uno más que la cantidad de "O".
Se nos pide implementar una función en python 'problema_quien_gano_el_tateti_facilito' que determine si ganó alguno, o si
Beto hizo trampa (puso una 'O' cuando Ana ya había ganado).

problema quien_gano_el_tateti_facilito(in tablero: seq<seq<Char>>) : Z{
requiere: {tablero es una matriz cuadrada}
requiere: {5 <= |tablero[0]| <= 10]}
requiere: {tablero sólo tiene 'X', 'O' y '' (espacio vacío) como elementos}
requiere: {En tablero la cantidad de 'X' es igual a la cantidad de 'O' o bien la cantidad de 'X' es uno más que la cantidad de
'O'}
asegura: {res = 1 <==> hay tres 'X' consecutivas en forma vertical (misma columna) y no hay tres 'O' consecutivas en forma
vertical(misma columna)}
asegura: {res = 0 <==> no hay tres 'O' ni hay tres 'X' consecutivas en forma vertical}
asegura: {res = 3 <==> hay tres 'X' y hay tres 'O' consecutivas en forma vertical (evidenciando que beto hizo trampa)}
}

4) Cuántos palíndromos sufijos (2 puntos)
Decimos que una palabra es palíndromo si se lee igual de izquierda a derecha que de derecha a izquierda. Se nos pide
programar en python la siguiente función:

problema cuantos_sufijos_son_palindromos(in texto:String) : Z{
requiere: -
asegura: {res es igual a la cantidad de palíndromos que hay en el conjunto de sufijos de texto}
}

Nota: un sufijo es una subsecuencia de texto que va desde una posición cualquiera hasta el final de la palabra. Ej: "Diego",
el conjunto de sufijos es: "Diego", "iego", "ego", "go", "o". Para este ejercicio no consideramos a "" como sufijo de ningún
texto.