Un grupo de amigos apasionados por las salas de escape, esas aventuras inmersivas donde tienen 60 minutos para salir de una habitación resolviendo enigmas, llevan un registro meticuloso de todas las salas de escape que hay en Capital. Este registro indica si han visitado una sala y si pudieron o no salir de ella. Un 0 significa que no fueron, un 61 que no lograron salir a tiempo, y un número entre 1 y 60 representa los minutos que les tomó escapar exitosamente. Con estos datos, pueden comparar sus logros y desafíos en cada nueva aventura que emprenden juntos.

1) Promedio de salidas [2 puntos]

Dado un diccionario donde la clave es el nombre de cada amigo y el valor es una lista de los tiempos (en minutos) registrados para cada sala de escape en Capital, escribir una función en Python que devuelva un diccionario. En este nuevo diccionario, las claves deben ser los nombres de los amigos y los valores deben ser tuplas que indiquen la cantidad de salas de las que cada persona logró salir y el promedio de los tiempos de salida (solo considerando las salas de las que lograron salir)

problema promedio_de_salidas (in registro: dict String, seq Z) : dict String, Z x R {
  requiere: {registro tiene por lo menos un integrante}
  requiere: {Todos los integrantes de registro tiene por lo menos un tiempo}
  requiere: {Todos los valores de registro tiene la misma longitud}
  requiere: {Todos los tiempos de los valores de registro están entre 0 y 61 inclusive}
  asegura: {res tiene las mismas claves que registro}
  asegura: {El primer elemento de la tupla de res para un integrante, es la cantidad de salas con tiempo mayor estricto a 0 y menor estricto a 61 que figuran en sus valores de registro}
  asegura: {El segundo elemento de la tupla de res para un integrante, si la cantidad de salas de las que salió es mayor a 0: es el promedio de salas con tiempo mayor estricto a 0 y menor estricto a 61 que figuran en sus valores de registro; sino es 0.0}
}

2) Tiempo más rápido [1 punto]

Dada una lista con los tiempos (en minutos) registrados para cada sala de escape de Capital, escribir una función en Python que devuelva la posición (índice) en la cual se encuentra el tiempo más rápido, excluyendo las salas en las que no haya salido (0 o mayor a 60).

problema tiempo_mas_rapido (in tiempos_salas: seq Z): Z {
  requiere: {Hay por lo menos un elemento en tiempos_salas entre 1 y 60 inclusive}
  requiere: {Todos los tiempos en tiempos_salas están entre 0 y 61 inclusive}
  asegura: {res es la posición de la sala en tiempos_salas de la que más rápido se salió (en caso que haya más de una, devolver la primera, osea la de menor índice)}
}

3) Racha más larga [3 puntos]

Dada una lista con los tiempos (en minutos) registrados para cada sala de escape a la que fue una persona, escribir una función en Python que devuelva una tupla con el índice de inicio y el índice de fin de la subsecuencia más larga de salidas exitosas de salas de escape consecutivas.

problema racha_mas_larga (in tiempos: seq Z): Z x Z {
  requiere: {Hay por lo menos un elemento en tiempos entre 1 y 60 inclusive}
  requiere: {Todos los tiempos en tiempos están entre 0 y 61 inclusive}
  asegura: {En la primera posición de res está la posición (índice de la lista) de la sala que inicia la racha más larga}
  asegura: {En la segunda posición de res está la posición (índice de la lista) de la sala que finaliza la racha más larga}
  asegura: {El elemento de la primer posición de res en tiempos es mayor estricto 0 y menor estricto que

61}
  asegura: {El elemento de la segunda posición de res en tiempos es mayor estricto 0 y menor estricto que 61}
  asegura: {La primera posición de res es menor o igual a la segunda posición de res }
  asegura: {No hay valores iguales a 0 o a 61 en tiempos entre la primer posición de res y la segunda posición de res}
  asegura: {No hay otra subsecuencia de salidas exitosas, en tiempos, de mayor longitud que la que está entre la primer posición de res y la segunda posición de res}
  asegura: {Si hay dos o más subsecuencias de salidas exitosas de mayor longitud en tiempos, res debe contener la primera de ellas.}
}

4) Escape en solitario [2 puntos]

Dada una matriz donde las columnas representan a cada amigo y las filas representan las salas de escape, y los valores son los tiempos (en minutos) registrados para cada sala (0 si no fueron, 61 si no salieron, y un número entre 1 y 60 si salieron), escribir una función en Python que devuelva los índices de todas las filas (que representan las salas) en las cuales el primer, segundo y cuarto amigo no fueron (0), pero el tercero sí fue (independientemente de si salió o no).

problema escape_en_solitario (in amigos_por_salas: seq seq Z): seq Z {
  requiere: {Hay por lo menos una sala en amigos_por_salas}
  requiere: {Hay 4 amigos en amigos_por_salas}
  requiere: {Todos los tiempos en cada sala de amigos_por_salas están entre 0 y 61 inclusive}
  asegura: {La longitud de res es menor igual que la longitud de amigos_por_salas}
  asegura: {Por cada sala en amigos_por_salas cuyo primer, segundo y cuarto valor sea 0, y el tercer valor sea distinto de 0, la posición de dicha sala en amigos_por_salas debe aparecer res}
  asegura: {Para todo i pertenciente a res se cumple que el primer, segundo y cuarto valor de amigos_por_salas[i] es 0, y el tercer valor es distinto de 0}
}

5) Preguntas teóricas (2 puntos)

Conteste marcando la opción correcta.

A) ¿Cuál es el propósito de la sentencia 'else' en una estructura 'if' en Python? (0.75 punto)

Definir una variable local.
Ejecutar un bloque de código si todas las condiciones anteriores son falsas. -> CORRECTA
Iniciar un ciclo 'while'.

B) ¿Qué hace la sentencia 'break' en un ciclo en Python? (0.75 punto)

Reinicia el ciclo desde el principio.
Termina el ciclo inmediatamente. -> CORRECTA
Salta la siguiente iteración del ciclo.

C) ¿Qué diferencia hay entre coverage de sentencias y coverage de branches? (0.5 punto)

Coverage de sentencias verifica cada línea de código, mientras que coverage de branches verifica las posibles salidas de las decisiones lógicas. -> CORRECTA
Coverage de branches verifica cada línea de código, mientras que coverage de sentencias verifica las posibles salidas de las decisiones lógicas.
No hay diferencia, ambos se refieren al mismo concepto.