Recuperatorio Python - Tema 2

Importante

- El parcial se aprueba con 6 puntos
- Utilizar este archivo fuente de base para la programación. Ya cuenta con los def y las signaturas correctas.
- Lista de funciones permitidas aca
- Para testear el código pueden usar este archivo que ya cuenta con todo lo necesario para desarrollar sus propios tests (este archivo no se entrega)
- **Para aprobar el parcial es requisito indispensable que todos los programas pasen los tests del archivo del punto anterior**

---

1) Gestión de ventas [2 puntos]

En una tienda llamada "SuperVentas", se desea desarrollar un programa para gestionar las ventas totales por empleado y producto. El programa debe procesar una lista de tuplas donde cada tupla contiene el nombre de un empleado, el nombre de un producto vendido y la cantidad total vendida de ese producto por el empleado.

Se pide implementar una función en python, que respete la siguiente especificación:

problema gestion_ventas (in *ventas_empleado_producto*: seq⟨(String x String x Z)) : dict⟨String, seq⟨(String x Z)⟩⟩ {
  requiere: { Las primeras componentes de *ventas_empleado_producto* tienen longitud mayor estricto a cero}
  requiere: { Las segundas componentes de *ventas_empleado_producto* tienen longitud mayor estricto a cero}
  requiere: { Las terceras componentes de *ventas_empleado_producto* son mayores estricto que 1 }
  requiere: { No hay 2 tuplas en *ventas_empleado_producto* que tengan la primera y segunda componente iguales (mismo empleado y mismo producto) }
  asegura: {*res* tiene como claves solo los primeros elementos de las tuplas de *ventas_empleado_producto* (o sea, un empleado)}
  asegura: {El valor en *res* de un empleado es una lista de tuplas donde cada tupla contiene como primera componente el nombre del producto y como segunda componente la cantidad vendida por el empleado de ese

producto según *ventas_empleado_producto*}
asegura: { Para toda clave (empleado) en *res*, en su valor (lista de tuplas) no hay 2 tuplas que tengan la misma primera componente (producto) }
}

---

2) Cantidad dígitos impares [2 puntos]

Se pide implementar una función en Python llamada cantidad_digitos_impares que respete la siguiente especificación:

problema cantidad_digitos_impares (in *numeros*: seq⟨Z⟩) : Z {
requiere:{Todos los elementos de *numeros* son mayores iguales a 0}
asegura: {*res* es la cantidad total de digitos impares que aparecen en cada uno de los elementos de *numeros*}
}

Por ejemplo, si la lista de números es [57, 2383, 812, 246], entonces el resultado esperado sería 5 (los dígitos impares son 5, 7, 3, 3 y 1).

---

3) Reordenar cola de carpetas [2 puntos]

En una empresa de gestión documental llamada "Archivos Rápidos", se manejan carpetas que llegan para su almacenamiento o envío. Cada carpeta está representada por una tupla (id_carpeta, numero_paginas), donde id_carpeta es un identificador único de la carpeta y numero_paginas es el número de páginas que contiene.

Se pide implementar una función en Python llamada reordenar_cola_primero_numerosas que respete la siguiente especificación:

problema reordenar_cola_primero_numerosas(in *carpetas*: Cola⟨(String x Z)⟩, in umbral:Z): Cola⟨(String x Z)⟩{
requiere: {no hay repetidos en las primeras componentes (Ids) de *carpetas*}
requiere: {todos las segundas componentes (número de páginas) de *carpetas* son mayores estricto a cero}
requiere: {*umbral* es mayor o igual a cero}
asegura: {los elementos de *res* son exactamente los mismos que los elementos de *carpetas*}
asegura: {|*res*| = |*carpetas*|}
asegura: {no hay un elemento en *res*, cuyo número de páginas sea menor o igual que el *umbral*, que aparezca primero que otro elemento en *res* cuyo número de páginas sea mayor que el *umbral*)}
asegura: {Para toda carpeta *c1* y carpeta *c2* cuyas

número de páginas son menores o iguales que el *umbral* y pertenecen a *carpetas*, si *c1* aparece primero que *c2* en *carpetas* entonces *c1* aparece primero que *c2* en *res*}
asegura: {Para toda carpeta *c1* y carpeta *c2* cuyas número de páginas son mayores que el *umbral* y pertenecen a *carpetas*, si *c1* aparece primero que *c2* en *carpetas* entonces *c1* aparece primero que *c2* en *res*}
}

---

4) Matriz cuasi decreciente [2 puntos]

Se desea verificar si una matriz es cuasi decreciente por columnas. Esto es que el máximo de cada columna sea mayor estricto que el máximo de la columna siguiente

Para ello se pide desarrollar una función en Python que implemente esta idea respetando la siguiente especificación:

matriz_cuasi_decreciente (in *matriz*: seq<seq<Z>>): Bool {
requiere: {|*matriz*| > 0}
requiere: {|*matriz*[0]| > 0}
requiere: {todos los elementos de *matriz* tienen la misma longitud}
asegura: {*res* es igual a True <=> para todo 0<=i<|*matriz*[0]|-1, el máximo de la columna i de *matriz* > el máximo de la columna i + 1 de *matriz* }
}

---

5) Preguntas teóricas (2 puntos)

Conteste marcando la opción correcta.

A) ¿Qué es un "alcance" (scope) en Python? (0.5 punto)

- ○ La sección del código donde una variable puede ser modificada.
- ○ La cantidad de memoria asignada a una variable
- ○ El contexto en el cual una variable es accesible

B) ¿Cuál es la principal diferencia entre una lista y una tupla en Python? (0.75 punto)

- ○ Las listas son ordenadas y las tuplas no.
- ○ Las listas pueden cambiar de tamaño después de su creación y las tuplas no.
- ○ Las listas pueden contener elementos duplicados y las tuplas no.

C) ¿Qué es el testing de caja blanca? (0.75 punto)

- ○ Un tipo de prueba que se centra en la funcionalidad externa del software sin conocer su implementación interna.
- ○ Un método de prueba utilizado únicamente para aplicaciones web.
- ○ Un tipo de prueba que se basa en el conocimiento del código fuente y su estructura interna.

Adjunta el archivo con tu solución:

Solo se puede adjuntar 1 archivo de extensión .py.
En caso de haber desarrollado tests propios, no deben ser entregados.

Seleccionar archivo a enviar:

Choose File No file chosen

Enviar