Simulacro Python - Tema 1

Importante

- El parcial se aprueba con 6 puntos
- Descargar y usar el siguiente **template** de funciones a implementar acá
- Lista de funciones permitidas acá
- 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)

Implementar las siguientes especificaciones en Python.

---

1) Ejercicio 1 [2.25 puntos]

problema maximas_cantidades_consecutivos (in *v*: seq⟨Z⟩) : Diccionario⟨Z,Z⟩ {
requiere: { True }
asegura: { Las claves de *res* son exactamente los números que aparecen al menos una vez en *v* }
asegura: { Para cada clave *k* de *res*, su valor es igual a la máxima cantidad de apariciones consecutivas de *k* en *v*, donde dicha cantidad de apariciones es mayor o igual 1. }
}

---

2) Ejercicio 2 [2.25 puntos]

problema maxima_cantidad_primos(in *A*: seq⟨seq⟨Z⟩⟩) : Z {
requiere: { Todas las filas de *A* tienen la misma longitud }
asegura: { Existe alguna columna *c* en A para la cual *res* de sus elementos son números primos }
asegura: { Todas las columnas de A tienen a lo sumo *res* elementos que son números primos }
}

---

3) Ejercicio 3 [2.25 puntos]

Dadas las siguientes definiciones:

- Tupla positiva: par de números enteros en el cual el producto de ambos números es positivo. Ejemplo: (2,3)
- Tupla negativa: par de números enteros en el cual el producto de ambos números es negativo. Ejemplo: (3,-2)
- Tupla nula: par de números enteros en el cual el producto de ambos números es igual a cero. Ejemplo: (0,0)

Se pide resolver:

problema tuplas_positivas_y_negativas(inout *c*: Cola⟨ZxZ⟩) {
requiere: { *c* no tiene tuplas repetidas }
modifica: { *c* }
asegura: { *c* contiene todas las tuplas positivas y todas las tuplas negativas de *c@pre* y además no contiene ninguna otra tupla }
asegura: { No hay niguna tupla negativa en *c* que aparezca

antes que alguna tupla positiva }
asegura: { Para todas las tuplas positivas *t1* y *t2* en *c*, con *t1* != *t2*, si *t1* aparece antes que *t2* en *c@pre*, entonces *t1* aparece antes que *t2* en *c* }
asegura: { Para todas las tuplas negativas *t1* y *t2* en *c*, con *t1* != *t2*, si *t1* aparece antes que *t2* en *c@pre*, entonces *t1* aparece antes que *t2* en *c* }
}

---

4) Ejercicio 4 [2.25 puntos]

problema resolver_cuenta(in *s*: string): R {
requiere: { Cada caracter de *s* es '+', '-', '.' (separador decimal) o es un dígito }
requiere: { No hay dos operadores consecutivos en *s* (los operadores son '+' y '-') }
requiere: { El último caracter de *s* es un dígito }
requiere: { El primer caracter de *s* es un dígito o un operador }
requiere: { En las posiciones adyacentes a cada aparición de '.' en *s*, hay un dígito }
requiere: { Entre un operador y el operador inmediato siguiente hay a lo sumo un separador decimal }
requiere: { Antes del primer operador hay a lo sumo un separador decimal }
requiere: { Después del último operador hay a lo sumo un separador decimal }
asegura: { *res* es igual al resultado de la operación aritmética representada por *s* }
}
Hint: las funciones int o float permiten convertir strings en números enteros o flotantes respectivamente.
Ejemplo: Para el input "+10" se debe devolver 10.

---

5) Pregunta teórica (1 punto)

Conteste marcando la opción correcta.

La principal diferencia entre testing de caja blanca y de caja negra es que en testing de caja blanca tenemos acceso a:

- ○ Al código fuente del programa que queremos testear
- ○ A la documentación del programa que queremos testear
- ○ Al Control Flow Graph (CFG) del programa que queremos testear

---

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:
Seleccionar archivo Ningún archivo seleccionado

Enviar