**Introducción a la Programación**

**Recuperatorio Segundo Parcial**

- El examen se aprueba con 6 puntos
- Utilizar este archivo fuente de base para la programación. Ya cuenta con los def y las signaturas correctas.
- 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.**
- **Todo el parcial se puede resolver con las herramientas básicas de imperativo vistas en clase. No está permitido el uso de listas por comprensión, funciones de librerías externas o funciones nativas no vistas en clase (por ejemplo: enumerate, zip, count, remove, index, reversed, etc)**

---

**1) Billetera Virtual [2 puntos]**

Las billeteras virtuales son sistemas financieros (pero no necesariamente bancarios) en los cuales los usuarios pueden cargar dinero y utilizarlo "virtualmente", realizando pagos y recargas de forma sencilla. Éstas transacciones quedan registradas en el sistema de la billetera virtual. Por ejemplo, el Sistema Único de Boleto Electrónico (SUBE) es un sistema de billetera virtual.

En el presente ejercicio verificaremos el histórico de transacciones (el historial) de una billetera virtual. En esta billetera sólo se puede: recargar saldo (r), pagar un viaje (v), ver el balance actual (s), y salir (x). Cada transacción queda grabada con su correspondiente caracter en una lista que representa el historial. En historial tendremos una secuencia de caracteres donde:

**v**: Realiza un viaje (todos los viajes son de $56)
**r**: Recarga saldo (todas las recargas son de $350)
**s**: Visualiza el saldo actual (no modifica el saldo)
**x**: El usuario decide terminar el programa

Implementar la función *verificar_transacciones()* que dada una secuencia de caracteres *s*, devuelve el saldo de la billetera al momento de terminar el programa. La finalización del programa está determinada por: (1) aparición de una *x*, (2) el usuario está intentando hacer un pago sin saldo suficiente (en esta billetera virtual no se permite saldo negativo), (3) no hay más transacciones en la lista.

```
problema verificar_transacciones (in s: String) : Z {
  requiere: {|s|>0}
  requiere: {s sólo puede contener los caracteres "r", "x", "v" o "s"}
  asegura: {res >= 0}
  asegura: {res = ($350 * #ap_antes_corte("r", s)) - ($56 * #ap_antes_corte("v", s))}
}
```

problema #ap_antes_corte(in *c*: char, in *s*: String): Z {
  **requiere:** {True}
  **asegura:** {res = cantidad de veces que aparece *c* desde el inicio hasta que: aparece una x, o que el cálculo del saldo se hace negativo}
}

Ejemplo 1: dado el siguiente historial
*s* = "ssrvvrrvvsvvsxrvvv"
se debería devolver *res* = 714

Ejemplo 2: dado el siguiente historial
*s* = "ssrvvvvsvvsvvv"
se debería devolver *res* = 14 (en este caso el programa termina porque el saldo no alcanza para realizar un viaje que está entre las transacciones)

---

**2) Hace calor [2 puntos]**

El cambio climático es innegable. En las últimas décadas hemos registrado aumentos en las temperaturas medias del planeta, lo que está generando una gran cantidad de efecto en la climatología de todo el mundo: lluvias más caudalosas, temperaturas más extremas, etc. Las ciencias de la atmósfera vienen estudiando estos cambios hace muchos años, y la conclusión con amplio consenso científico es que este cambio es producto de la actividad humana: lo que se llama "cambio climático antropogénico". Un grupo de estudiantes de la Licenciatura en Ciencias de la Atmósfera (carrera que se estudia en esta Facultad) cuenta con una lista de tuplas de las temperaturas mínimas y máximas diarias de los últimos años en la Ciudad de Buenos Aires. Implementar la función valor_minimo() que dado este listado de tuplas devuelva el valor mínimo, entre las mínimas alcanzadas.

problema valor_minimo (in *s*: seq<(R,R)>): R {
  **requiere:** {|*s*| > 0}
  **requiere:** {En cada tupla de *s* el primero valor es menor que el segundo}
  **asegura:** {*res* pertenece a alguna tupla de *s*, en la primera posición}
  **asegura:** {No existe ningún valor en las primeras posiciones de las tuplas de *s* que sea menor que *res*}
}

Por ejemplo, dado
*s* = [(1.0, 5.2), (10.4, 15.1), (19.7, 28.9), (25.4, 35.6), (-3.1, 1.3)]
se debería devolver *res* = -3.1

---

**3) El MERVAL [3 puntos]**

En la bolsa de valores de Argentina (denominada MERVAL) operan un gran número de empresas. Las empresas cuyas acciones cotizan en la bolsa tienen un nombre identificador. Por ejemplo, "YPF Sociedad Anónima" en la bolsa es: YPF; "Banco Galicia" es GGAL; Aluminio Argentino (Aluar, la empresa siderúrgica) es ALUA; Loma Negra Compañía industrial de Cemento es LOMA. Durante una jornada los precios de las acciones de estas empresas se ven modificados.

En el presente ejercicio vamos a trabajar con un diccionario (*valores_diarios*) que registrará el precio de las acciones de diferentes empresas en diferentes momentos de un determinado mes. El diccionario tendrá como clave los nombres de las empresas y como valores, listas de tuplas, donde cada tupla (dupla) codifica el día (como entero) y el valor de la acción (como real). Es posible que exista más de un registro por día.

Implementar la función *valores_extremos()* que dado un diccionario *valores_diarios* devuelva un nuevo diccionario, con las mismas claves, pero que los valores sean tuplas que indiquen el mínimo y el máximo alcanzado durante el periodo registrado.

problema valores_extremos(in *cotizaciones_diarias*: dict⟨String,seq⟨(Z x R)⟩⟩): dict⟨String,(R x R)⟩ {
  **requiere:** {Cada valor de *cotizaciones_diarias* es una secuencia de tuplas, donde los primeros elementos de dichas tuplas son enteros del 1 al 31}
  **requiere:** {Cada valor de *cotizaciones_diarias* es una secuencia de tuplas, de longitud mayor estricto que 0}
  **asegura:** {*res* tiene como claves exactamente las mismas claves que *cotizaciones_diarias*}
  **asegura:** {Cada valor de *res* es una tupla de (mínimo, máximo), donde mínimo y máximo son los valores extremos alcanzados por las cotizaciones de cada empresa}
}

Por ejemplo, dada la siguiente cotización:

cotizaciones_diarias = {"YPF" : [(1,10),(15, 3), (31,100)], "ALUA" : [(1,0), (20, 50), (31,30)]}
resultado_esperado es: {"YPF" : (3,100), "ALUA" : (0,50)}

---

**4) Sudoku [3 puntos]**

El sudoku es un juego moderno, inventado en el Siglo XX, que consiste en una grilla de 9x9 celdas. Para resolverlo es necesario completar las grillas con los dígitos del 1 al 9 haciendo que cada dígito aparezca exactamente una vez en cada fila y en cada columna (para este ejemplo estaremos obviando una regla extra que tiene el sudoku original por la cual los dígitos no se pueden repetir dentro de cuadrantes de 3x3 que se encuentran en la grilla) Implementar la función *es_sudoku_valido()* que dado un tablero de sudoku (de 9x9) semi-completo devuelve True si en cada una de sus filas no hay números del 1 al 9 repetidos, y en cada una de sus columnas no hay números de 1 al 9 repetidos. Las celdas vacías se marcarán con valor 0.

problema es_sudoku_valido(in *m*:seq⟨seq⟨Z⟩⟩ ) : Bool {
  **requiere:** {todos los elementos de *m* tienen longitud 9}
  **requiere:** {|m| = 9}
  **requiere:** {todos los elementos en todas las secuencias de m son números del 0 al 9}
  **asegura:** {(*res* = true) <=> en cada fila de m no se repiten números del 1 al 9}
  **asegura:** {(*res* = true) <=> en cada columna de m no se repiten números del 1 al 9}
}

Por ejemplo, dada la secuencia de secuencias:

*m* = [
[1, 2, 3, 4, 5, 6, 7, 8, 9],
[9, 8, 7, 6, 4, 5, 3, 2, 1],
[0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 4, 0, 0, 0],
[0, 0, 0, 0, 6, 0, 0, 0, 0],
[0, 0, 0, 5, 0, 0, 0, 0, 0],
[0, 0, 4, 0, 0, 0, 0, 0, 0],
[0, 3, 0, 0, 0, 0, 0, 0, 0],
[2, 0, 0, 0, 0, 0, 0, 0, 0]
]
se debería devolver *res* = true

Consejo: Para probar con matrices pueden usar:
matriz_ceros = [[0]*9]*9
matriz_fila_1_distinta = [list(range(1,10))] + [[0]*9]*8

## Resolucion

```
from typing import List, Tuple, Dict



# PRIMERO PONGO TODAS LAS FUNCIONES AUXILIARES QUE SE NECESITAN PARA CORRER EL CÓDIGO:

def el_minimo(e: int, s: [int]) -> bool:
    for i in range(len(s)):
        if not (e <= s[i]):
            return False
    return True



def devuelvemin(s: [int]) -> int:
    res: int = 0
    for elem in s:
        if el_minimo(elem, s):
```

```
            res = elem
    return res


def el_max(e: int, s: [int]) -> bool:
    for i in range(len(s)):
        if not (e >= s[i]):
            return False
    return True


def devuelvemax(s: [int]) -> int:
    res: int = 0
    for elem in s:
        if el_max(elem, s):
            res = elem
    return res


def cantApariciones(e: int, s: [int]) -> int:
    contador: int = 0
    for elem in s:
        if elem == e:
            contador += 1
    return contador


def cantAparicionesEnPosicion(e: int, pos: int, s: [[int]]) -> int:
    contarposicionI: int = 0
```

```
    for fila in s:
        for i in range(len(fila)):
            if fila[i] == e and i == pos:
                contarposicionI += 1
    return contarposicionI



# Ejercicio 1
def verificar_transacciones(s: str) -> int:
    res: int = 0
    for c in s:
        if c == "r":
            res += 350
        if c == "v":
            res -= 56
            if res < 0:
                res += 56
                return res
        if c == "x":
            return res
    return res



# Ejercicio 2
def valor_minimo(s: List[Tuple[float, float]]) -> float:
    res: int = 0
    temperaturas: [int] = []
    for tupla in s:
        for valores in tupla:
```

```
            temperaturas.append(valores)
    for temperatura in temperaturas:
        if el_minimo(temperatura, temperaturas):
            res = temperatura
    return res



# Ejercicio 3
def valores_extremos(
    cotizaciones_diarias: Dict[str, List[Tuple[int, float]]]
) -> Dict[str, Tuple[float, float]]:
    res: dict = {}
    for empresas in cotizaciones_diarias.keys():
        res[empresas] = ()
    for empresas in cotizaciones_diarias.keys():
        valoresempresa = []
        for tuplas in cotizaciones_diarias[empresas]:
            valoresempresa.append(tuplas[1])
        res[empresas] = (devuelvemin(valoresempresa), devuelvemax(valoresempresa))
    return res



# Ejercicio 4
def es_sudoku_valido(m: List[List[int]]) -> bool:
    for fila in m:
        for elem in fila:
            if elem != 0 and cantApariciones(elem, fila) > 1:
                return False
    for elem in [1, 2, 3, 4, 5, 6, 7, 8, 9]:
```

```
    for i in range(0, 9):
        if cantAparicionesEnPosicion(elem, i, m) > 1:
            return False
return True
```