PRÁCTICA DE SOLEMNE

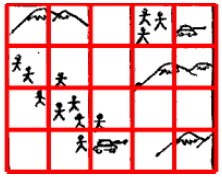
P1. Considere que existe una biblioteca llamada **`wartigray`**, la que tiene datos de la guerra del Tigray. En particular, nos interesan dos métodos de esta biblioteca:

`dias()` devuelve el número de días que duró la guerra;

`bajas(d)` devuelve el total de bajas al día **`d`** (`0 <= d < dias()`).

Escriba un programa que busque e imprima el día con más bajas.

P2. Un campo de batalla es representado como una lista de listas de rivales, como sigue:



0	0	0	3	4
2	1	0	0	0
1	3	1	0	0
0	1	4	0	0

```
[[0, 0, 0, 3, 4],
 [2, 1, 0, 0, 0],
 [1, 3, 1, 0, 0],
 [0, 1, 4, 0, 0]]
```

Además, considere que tiene una lista de blancos de la forma:

`[[i1,j1], [i2,j2], [i3,j3], ...]`

donde cada par `[i,j]` es de índices que refieren a una posición en el campo.

Implemente una función, de nombre **`ataque`**, que reciba como parámetros un campo de batalla y un lista de blancos. La función debe retornar la cantidad de bajas por ataque.

Problema 3

Se están creando planillas de cuatro columnas acerca de artes marciales. Por ejemplo:

Arte Marcial	Dificultad	Popularidad	Representante
Aikido	9	2	steven@seagal.cl
Karate	4	9	chuck@norris.cl
Taekwondo	7	7	pata@face.cl
...	...	...	...

Esta base de datos se implementa como una lista de filas, ej:

```
[['Aikido', 9, 2, 'steven@seagal.cl'],
 ['Karate', 4, 9, 'chuck@norris.cl'],
 ['Taekwondo', 7, 7, 'pata@face.cl'],
                    ...]
```

Preguntas

a. Implemente la función **`mas_facil(bdatos)`** que retorna el nombre del arte marcial más sencillo.

b. Implemente la función **`mas_conveniente(bdatos)`** que retorna el nombre del arte marcial de mayor razón **popularidad / dificultad**.

c. Implemente la función **`populares(bdatos)`** que retorna **la lista** con todas las artes marciales con **popularidad ≥ 5**.

Solución Problema 1

Procedamos cómo siempre lo hacemos:

1. Debemos comenzar cumpliendo con **la formalidad del problema**: escribir **import**, **def**, los argumentos, etc.
2. Debemos **identificar cómo es y cómo se entrega el resultado**, lo que nos permite definir las variables que debemos definir, sus valores iniciales y si respondemos con **print** o **return**
3. Debemos **crear e implementar el algoritmo**, que son los pasos o estrategia para llegar al resultado. Al respecto,
    1. Primero debemos pensar qué acciones debemos realizar para llegar al resultado y comentar esto en la respuesta y
    2. Luego debemos escribir el código que implementa la estrategia de solución pensada.

Para este problema la **formalidad** nos obliga a (#1) importar la biblioteca **wartigray** (o sea, comenzamos la respuesta con **import wartigray**). Luego identificamos qué se desea obtener: **el día** con más **bajas**. Como son dos datos (que van mano a mano), (#2) definimos ambas variables al principio. Al final, (#6) respondemos con un **print** que reporta el día.

Respecto del algoritmo, hacemos esto: (#2) comenzamos con una respuesta tentativa. Luego, (#3, #4) revisamos cada día viendo las bajas ocurridas y (#5) contrastamos con la respuesta tentativa. (#5) Si hubo más bajas que en la respuesta tentativa, cambiamos la

respuesta tentativa por el día analizado en cuestión. Siguiendo el procedimiento anterior, debemos quedarnos con el día con más bajas.

```
# 1. importamos la biblioteca
import wartigray

# 2. variables de respuesta
top_dia = 0
top_mue = 0

# 3. obtenemos nro de dias
n_dias = wartigray.dias()

# 4. buscamos en dias validos
for d in range(0, n_dias):
    # 5. vemos si es el "mejor" dia hasta ahora
    if top_mue < wartigray.bajas(d):
        top_dia = d
        top_mue = wartigray.bajas(d)

# 6. reportamos
print("El dia", top_dia, "fue el dia con mas muertos, con", top_mue, "muertes")
```

Solución Problema 2

De nuevo, (#1) obedecemos las formalidades y definimos la firma de la función (**def** ataque...), considerando que recibe dos argumentos/parámetros. Como se pide por la cantidad total de bajas realizadas, (#2) creamos la variable **bajas** como **bajas=0** y (#3) escribimos **return bajas** al final.

Para el algoritmo: (#4) revisamos cada par de índices en la lista de listas de blancos (o sea, escribimos **for par in blancos** o similar), (#5) extraemos los índices de cada par de índices y (#6) sumamos a la respuesta (a **bajas**) los rivales en **campos[i][j]**.

```
# 1. firma de la funcion
def ataque( campo, blancos ):
    # 2. variable para la respuesta
    bajas = 0
    # 4. recorremos la lista de blancos
    for par in blancos:
        # 5. extraemos los indices desde par (par es de tipo [i,j])
        i = par[0]
        j = par[1]
        # 6. sumamos los rivales alcanzados
        bajas += campo[i][j]
    # 3. retornamos la respuesta
    return bajas
```

Solución Problema 3

Este problema tiene tres partes. Ojo que las partes **a** y **b** se parecen al problema 1.

P3a (los comentarios explican el razonamiento)

```
# 1. definimos la firma de la funcion (nombre, argumentos)
def mas_facil(B):
    # 2. identificamos las variables que necesitamos responder
    r_nom = B[0][0] # B[0] es la primera fila, B[0][0] es el nombre
    r_dif = B[0][1] # B[1] es la primera fila, B[0][1] es la dificultad
    # 3. revisamos PARA CADA fila en B (o sea, usamos for)
    for fila in B:
        # 4. extraemos los campos relevantes de cada fila
        nom = fila[0]
        dif = fila[1]
        # 5. revisamos si esta arte marcial es mas facil que la nuestra
        if dif > r_dif:
            # 6. si asi es, guardamos como nueva respuesta candidata
            r_dif = dif
            r_nom = nom
    # 3. retornamos la respuesta
    return r_nom
```

P3b (los comentarios explican el razonamiento)

```
# 1. definimos la firma de la funcion (nombre, argumentos)
def mas_conveniente(B):
    # 2. identificamos las variables que necesitamos responder
    r_nombre = B[0][0] # B[0] es la primera fila, B[0][0] es el nombre
    r_razon  = B[0][2]/B[0][1] # popularidad/dificultad para B[0]
    # 3. revisamos PARA CADA fila en B (o sea, usamos for)
    for fila in B:
        # 4. extraemos los campos relevantes de cada fila
        nombre = fila[0]
        razon  = fila[2]/fila[1]
        # 5. revisamos si esta arte marcial es mas facil que la nuestra
        if razon > r_razon:
            # 6. si asi es, guardamos como nueva respuesta candidata
            r_razon  = razon
            r_nombre = nombre
    # 3. retornamos la respuesta
    return r_nombre
```

P3c

En este caso, se debe construir una lista en la respuesta. Entonces hay que recorrer todas las filas e identificar, por cada fila, si cumple con la popularidad. Si así es, el nombre del arte marcial debe anexarse a la respuesta.

```
# 1. definimos la formalidad
def populares(B):
    # 2. identificamos que vamos a responder con una lista
    resp = []
    # 4. revisamos cada fila de la lista de listas
    for fila in B:
        # 5. chequeamos si el arte marcial tiene popularidad >= 5
        if fila[2] >= 5:
            # 6. anexa el nombre del arte marcial a la respuesta
            resp.append( fila[0] )
    # 3. retornamos la lista que hemos construido
    return resp
```