Programación – Certamen 2 - Martes 13 de Noviembre de 2018

Nombre

Rol — Paralelo Delaunay

1. **[20 %]** Realice el ruteo del siguiente programa e indique qué es lo que imprime. Cada vez que el valor de una variable cambie, escríbalo en una nueva fila de la tabla. Recuerde que si una variable es de tipo string, su valor debe ir entre comillas simples ' '.
***Importante**: La tabla tiene suficientes filas.*

```
def f1(lista, d, p=0):
    l = []
    for n in lista:
        if n[p] < d:
            l.append(n)
    return l

lista = [('d',4)]
lista.append(('x',8))
lista  = f1(lista,5,1)
print lista

lista.append(('c' ,9))
lista = f1(lista,'d')
print lista
```

global	f1				
lista	lista	d	p	l	n

Análisis de Algoritmo

En base al contexto del programa anterior, analice la función **f1** y determine, **en a lo más 30 palabras**, lo que realiza.

Programación – Certamen 2 - Martes 13 de Noviembre de 2018

Nombre

Rol — Paralelo Delaunay

2.

Programación – Certamen 2 - Martes 13 de Noviembre de 2018

Nombre

Rol — Paralelo Delaunay

3.

Programación – Certamen 2 - Martes 13 de Noviembre de 2018

2. **[40 %] NOTA:** Puede hacer uso de la función definida en la pregunta 1.

La conocida aplicación "SpotiPhy"necesita determinar cuales son los álbumes y los artistas mejores evaluados en su plataforma. Para lograrlo dispone de una lista de tuplas que tienen la siguiente forma: $(cancion, album, artistas, fecha\_evaluacion, nota)$, en donde canción y álbum están en formato string, artistas es una lista que contiene los nombres de los artistas (una canción puede tener más de un artista) en formatro string, la fecha es una tupla (A,M,D) y la nota es un entero entre 1 y 5. A continuación se muestra un ejemplo de la estructura *evaluaciones*:

```
evaluaciones = [
('Phypocrita', 'Hasta abajo', ['Pynuel'], (2018, 6, 30), 4),
('Sin Phyjama', 'Nuevo Estilo', ['Becky T', 'Turize'], (2018, 4, 14), 3),
('Vaina Crazy', 'del Weno', ['Ozune', 'Turize'], (2018, 7, 18), 5),
('Phypocrita', 'Hasta abajo', ['Pynuel'], (2018, 8, 20), 5),
('Quiero Beber Agua', 'Hasta abajo', ['Pynuel'], (2018, 2, 25), 5),
 ...]
```

Notar que una misma canción pudo ser evaluada en distintos momentos. De acuerdo a la información anterior, se le solicita implementar las siguientes funciones:

a.- Escriba la función `notas_por_artista(lista)`, donde `lista` es del tipo `evaluaciones`. Esta función debe retornar un diccionario donde la llave es un **string** con el nombre del artista y como valor tienen una **lista** de enteros que corresponden a todas las evaluaciones de canciones en las que ha participado. Recuerde que una misma canción **puede tener varias evaluaciones** y todos estos valores deben estar en la lista.

```
>>> print notas_por_artista(evaluaciones)
{'Ozune': [5], 'Pynuel': [4, 5, 5], 'Becky T': [3], 'Turize': [3, 5]}
```

b.- Escriba la función `artistas_hit(lista, fecha)`, donde `lista` es del tipo `evaluaciones` y `fecha` es una **tupla** con formato (A,M,D). La función debe retornar una **lista** con los 3 artistas que con mayor probabilidad producen hits, **ordenada de forma descendente según la probabilidad**. Esta probabilidad se obtiene al dividir la cantidad de evaluaciones con nota 4 ó 5 del artista por la cantidad total de evaluaciones que posee dicho artista de sus canciones, en ambos considerar las evaluaciones hasta la `fecha` consultada, inclusive. Por lo tanto la probabilidad siempre será un número entre 0 y 1. Si hay menos de 3 artistas con evaluaciones anteriores a la `fecha`, entregue una lista con todos los artistas posibles. Tenga en consideración que los 3 mejores artistas seleccionados deben tener probabilidad $\geq$ a todo el resto de los artistas en SpotiPhy.

```
>>> print artistas_hit(evaluaciones,(2018,8,30))
['Pynuel', 'Ozune', 'Turize']
>>> print artistas_hit(evaluaciones,(2018,2,28))
['Pynuel']
```

NOTA: Los puntos suspensivos en las estruturas de datos indican que existen más datos.

IMPORTANTE: Además de lo visto para el certamen 1, puede utilizar los comandos de Python vistos en las diapositivas de clases como por ejemplo:

```
range(x,y=0,z=1). Para una lista L: L.append(x), L.index(x), L.insert(x,y),
sum(L), len(L), L.reverse(), L.sort(), x in L, L[x], L[x:y]. Para un
diccionario D: D.values(), D.keys(), D.items(), D[x], x in D, len(D)
```

Programación – Certamen 2 - Martes 13 de Noviembre de 2018

3. **[40 %]** PyCornerShop es una aplicación que le permite a sus usuarios conseguir productos del supermercado desde la comodidad del hogar usando sus teléfonos móviles. La información de los repartidores y usuarios se encuentra almacenada en dos diccionarios llamados `repartidores` y `usuarios` respectivamente. El diccionario `repartidores` tiene los nombres de los repartidores como llave y como valor una tupla que contiene la ubicación del repartidor en el plano de la ciudad y su estado de disponibilidad: `True` (disponible) o `False` (no disponible). A continuación, se muestra un ejemplo de este diccionario:

```
repartidores = {'rayo macuin': ((10, 2), True), 'reparti dhor': ((9, 3), True),
                'eliseo al-azar': ((5, 5), False), ...}
```

El diccionario `usuarios` contiene el código del usuario como llave y como valor la ubicación del mismo en el plano de la ciudad. A continuación se muestra un ejemplo:

```
usuarios = {1221: (5, 2), 441: (8, 2), 587: (10, 1), ...}
```

Existe un diccionario llamado `visitas` donde cada llave es el nombre de un repartidor y como valor tiene una lista de tuplas. Cada tupla se compone de un código de usuario y de la cantidad de veces que dicho repartidor (llave del diccionario) visitó a dicho usuario. Tenga en consideración que si un usuario nunca ha solicitado reparto, no aparecerá en el diccionario visitas. A continuación un ejemplo:

```
visitas = {'rayo macuin': [(1221, 5), (441, 8), (587, 2)],
           'reparti dhor': [(1221, 2), (441, 5), (587, 3)],
           'eliseo al-azar': [(1221, 8), (441, 2), (587, 1)],
            ...}
```

Para que la compañía PyCornerShop pueda funcionar de manera eficiente, le solicita a Ud. que implemente una función llamada `buscar_repartidor(repartidores,usuarios,visitas,codigo)` que reciba como parámetros los diccionarios antes mencionados y un código de usuario (formato `int`). La función debe retornar una lista de repartidores disponibles y cercanos, ordenados de manera descendente, según el número de visitas que estos hayan realizado al usuario requerido. **Un repartidor será considerado cercano si está ubicado a menos de 4 km del usuario** y para esto considere que las tuplas de ubicación tienen valores enteros medidos en *km*. Finalmente, si no hay repartidores cercanos, la función debe retornar una lista vacía.

Recuerde que los puntos suspensivos en los contenedores indican que existen más datos. Además considere los siguientes ejemplos de ejecución para construir su solución:

```
>>> print buscar_repartidor(repartidores,usuarios,visitas,587)
['reparti dhor', 'rayo macuin']
>>> print buscar_repartidor(repartidores,usuarios,visitas,441)
['rayo macuin', 'reparti dhor']
>>> print buscar_repartidor(repartidores,usuarios,visitas,1221)
[]
```

Nota: Para calcular la distancia d entre un usuario ubicado en (x_1, y_1) y un repartidor ubicado en (x_2, y_2) se debe utilizar la ecuación: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.