Programación—Certamen 2 (CSSJ) - Martes 15 de Mayo de 2018

Nombre

Rol — Paralelo

1. [20 %] Realice el ruteo del siguiente programa e indique qué es lo que imprimen. 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, debe colocar su valor entre comillas simples ' '.

Importante: *La tabla tiene suficientes filas.*

```
def buscar(lista,n=-1):
    if n!=-1:
        c = 0.0
        for x in lista:
            if x==n:
                c+=1
        return c/len(lista)
    else:
        li = []
        for x in lista:
            if x not in li:
                li.append(x)
        return li
res = [3,5,3,1,3]
for i in range(1,-1,-1):
    if i!=0:
        r = buscar(res)
        print r
    else:
        r = buscar(res,r[i])
        print r
```

global			buscar				
res	i	r	lista	n	c	x	li

Análisis de algoritmo

En base al algoritmo que acaba de rutear, escriba en no más de 20 palabras, qué es lo que hace la función **buscar**, cuando recibe más de un parámetro.

Programación—Certamen 2 (CSSJ) - Martes 15 de Mayo de 2018

Nombre

Rol — Paralelo

2. **[40 %] NOTA:** Puede hacer uso de las funciones definidas en la pregunta 1.

Leonardo Parkas aburrido de su mala suerte ha decidido apostar en las ruletas del casino de Pythonia, *Pytichello*. Gracias a la ayuda de su fiel amigo PyMundo, ha conseguido obtener la información de los resultados de las ruletas en los juegos anteriores, los cuales están disponibles en un **diccionario**, cuya llave es un **string** con el identificador de la ruleta y el valor es una **lista**, donde cada elemento de la lista corresponde a un número que ha sido ganador en algún juego anterior en la ruleta. Habrán tantos números como juegos se hayan llevado a cabo en la ruleta.

```
resultados = {'r1':[6,3,12,3,7,6],
              'r9':[12,8,12],
              'r2':[7,1,15,4,7,11,7,1],
              ...}
```

En las ruletas de Pytichello los números a los que se puede apostar van desde el 1 hasta el 15. El máximo de números a apostar por juego es de 5. Como Parkas quiere asegurar su apuesta, le ha solicitado a usted implemente unas funciones que le entreguen información relevante para hacer su apuesta, para lo cual solicita lo siguiente:

a) Implemente la función **estadisticas(ruleta, diccio)** la cual recibe 2 parámetros, un **string** `ruleta` con el id de la ruleta y el diccionario `diccio`, con la información de los resultados por ruleta. Esta función debe retornar un diccionario donde cada llave corresponderá a un número ganador y su respectivo valor corresponderá a la frecuencia que se repitió dicho número, es decir, la cantidad de veces que apareció el número dividido por la cantidad total de números que se han jugado en la ruleta.

```
>>> print estadisticas('r2',resultados)
{11: 0.125, 1: 0.25, 15: 0.125, 4: 0.125, 7: 0.375}
```

b) Implemente la función **k_esimo_mejor(k, diccio, ruleta)** la cual recibe 3 parámetros, **un entero k**, un **diccionario** con la información de los resultados de los juegos en las distintas ruletas y un **texto** con el ID de la ruleta a consultar. Esta función debe retornar los `k` primeros números que más se han repetido en los juegos para la ruleta consultada. En caso de haber empate, retorne cualquiera de los empatados. Si el valor `k` es mayor que la cantidad de números ganadores que posee la ruleta, debe retornar la lista completa, respetando el orden por frecuencia de mayor a menor. Si el número `k` no es válido, **retorne 'error'**.

```
>>> print k_esimo_mejor(3,resultados,'r2')
[7, 1, 15]
>>> print k_esimo_mejor(6,resultados,'r1')
[6, 3, 12, 7]
>>> print k_esimo_mejor(-8,resultados,'r2')
error
```

Programación—Certamen 2 (CSSJ) - Martes 15 de Mayo de 2018

Nombre

Rol — Paralelo

3. [40 %] En la ciudad de Pythonia todos los trenes realizan el mismo recorrido, pasando por cada una de las estaciones de la Línea P. El orden de las estaciones se encuentra en la lista `linea_P`. Sin embargo, el tiempo entre una estación y otra depende del tipo de tren. Los trenes *Expreso* tardan 2 minutos en ir desde una estación a otra, los trenes *Normal* demoran 5 minutos, y los trenes *Corto* tardan 7 minutos. **Asuma que el tiempo que el tren se detiene es despreciable y no será considerado para este problema**. Los tipos de trenes se encuentran en el diccionario `tipos`, donde la llave es el tipo y el valor una lista con los identificadores únicos de los trenes que pertenecen a ese tipo. Finalmente, el diccionario `trenes` contiene como llave el identificador único del tren y su valor es una tupla que indica su horario de salida `('HH','MM')` **desde la primera estación de la línea**, la *Estación Central* de Pythonia.

```
linea_P = [
'Estacion Central',
'Puente Sal y Santo',
'Pytronato',
'La Pysterna',
'Pynstein',
'Estadio Nacional',
'Pio-Pio',
'Plaza Python Alto',
...
]
```

```
trenes = {
'E07':('09','03'),
'F07':('10','14'),
'H00':('09','23'),
'B00':('08','00'),
'G00':('08','46'),
'C01':('13','59'),
'F08':('11','02'),
... }
```

```
tipos = {
'Expreso':['E07','F07','F08',...], 'Normal':['G00','H00', ...],
'Corto':['B00','C01', ...]}
```

Desarrolle la función `proximos_trenes(est, hor)` donde `est` es un string con el nombre de la estación en la que se encuentra el pasajero y `hor` es una tupla en formato ('HH','MM') que indica la hora actual. La función debe retornar un diccionario que tenga como llave el identificador de todos los trenes que **aún no hayan pasado** por la estación actual `est` a la hora actual `hor`. El valor de cada llave corresponde a los minutos que faltan para el arribo del tren a la estación actual `est`. En caso de no haber trenes para la hora consultada, se debe retornar un diccionario vacio.

```
>>> print proximos_trenes('Pytronato', ('09','06'))
{'E07': 1, 'F07': 72, 'H00': 27, 'F08': 120, 'C01': 307}

>>> print proximos_trenes('Pio-Pio', ('11','00'))
{'C01': 221, 'F08': 14}

>>> print proximos_trenes('Puente Sal y Santo', ('15','00'))
{}
```