Ayudantía Programación

Ejercicios Certamen 2

Gonzalo Fernández

IWI-131 12/11/2018

Temas de la Ayudantía de Hoy

- Ejercicios.
- Ejercicios.
- Más ejercicios.

Análisis de algoritmo

Analice el siguiente algoritmo y determine, en pocas palabras, lo que realiza. No debe usar más del espacio indicado.

```
# n numero entero
def f(n):
    l = []
    for i in range(len(str(n))-1,-1,-1):
        x = n/(10**i)
        l.append(x)
        n=n%(10**i)
    return l
```

2. [35 %] El prestamista Vito Corleone, aburrido de perseguir a sus clientes, ha solicitado la ayuda de algún astro de la programación para resolver sus problemas logísticos.

Vito cuenta con la lista `clientes`, la que tiene tuplas con el nombre de cada cliente, el monto adeudado y la fecha del último pago (también dentro de una tupla).

```
clientes = [('Don Ramon', 3500, (9, 4, 2014)), ('Miguel', 2785, (30,
    10, 2014)), ('Cesar', 100, (28, 5, 2015)), # ...
]
```

Nota: Esto es sólo un ejemplo, considere que la lista puede tener muchos clientes. Sin embargo, asuma que cada nombre de cliente aparece sólo una vez.

a) Implemente la función `deuda_total(clientes)`, que reciba como entrada la lista de tuplas `clientes`, y retorne la suma de las deudas de todos los deudores.

```
>>> deuda_total(clientes)
6385
```

2. [35 %] El prestamista Vito Corleone, aburrido de perseguir a sus clientes, ha solicitado la ayuda de algún astro de la programación para resolver sus problemas logísticos.

Vito cuenta con la lista `clientes`, la que tiene tuplas con el nombre de cada cliente, el monto adeudado y la fecha del último pago (también dentro de una tupla).

```
clientes = [('Don Ramon', 3500, (9, 4, 2014)), ('Miguel', 2785, (30,
    10, 2014)), ('Cesar', 100, (28, 5, 2015)), # ...
]
```

Nota: Esto es sólo un ejemplo, considere que la lista puede tener muchos clientes. Sin embargo, asuma que cada nombre de cliente aparece sólo una vez.

b) Implemente la función `mayor_deudor(clientes, ultimo)`, que retorne el nombre del cliente que tiene la mayor deuda, y que además su último pago haya sido realizado en el año `ultimo`. Si no hay clientes con pagos en el año `ultimo`, retorne un string vacío.

```
>>> mayor_deudor(clientes, 2014)
'Don Ramon'
```

2. [35 %] El prestamista Vito Corleone, aburrido de perseguir a sus clientes, ha solicitado la ayuda de algún astro de la programación para resolver sus problemas logísticos.

Vito cuenta con la lista `clientes`, la que tiene tuplas con el nombre de cada cliente, el monto adeudado y la fecha del último pago (también dentro de una tupla).

```
clientes = [('Don Ramon', 3500, (9, 4, 2014)), ('Miguel', 2785, (30,
    10, 2014)), ('Cesar', 100, (28, 5, 2015)), # ...
]
```

Nota: Esto es sólo un ejemplo, considere que la lista puede tener muchos clientes. Sin embargo, asuma que cada nombre de cliente aparece sólo una vez.

c) Implemente la función `pagar(clientes, pago)` que recibe la lista `clientes` y una tupla `pago` compuesta por el nombre de quien paga, el monto que cancela y la fecha en que lo hace. La función **modifica y retorna** la lista `clientes`, descontando la deuda y actualizando la fecha del último pago, según se indique en la tupla `pago`. Si la deuda llega a 0, debe borrar al cliente de esta lista. No es necesario considerar el caso en que se paga más de lo que se debe.

```
>>> pagar(clientes, ('Miguel', 85, (3, 6, 2015)))
[('Don Ramon', 3500, (9, 4, 2014)), ('Miguel', 2700, (3, 6, 2015)), ('
   Cesar', 100, (28, 5, 2015))]
>>> pagar(clientes, ('Cesar', 100, (4, 6, 2015)))
[('Don Ramon', 3500, (9, 4, 2014)), ('Miguel', 2700, (3, 6, 2015))]
```

3. **[40 %]** El villano más malvado sobre la faz de la tierra, Ultron, tiene catalogado a todos los superhéroes de la tierra de la siguiente forma:

```
# nombre: [detalle, edad, habilidad, (min-poder, max-poder)]
superheroes = {
'Iron man': ['mk42', 50 , 'uni-rayo',  (45, 95)],
'Thor': ['hijo de odin', 10000 , 'mjolnir',  (50, 100)],
'Condorito':['de pelotillehue', 40, 'washington', (1, 10)],
'Chapulin Colorado': ['no contaban con mi astucia', 40, 'chipote
   chillon', (40, 90)],
# ...
}
# nombre : nombre_asociados
asociados = {
'Iron man': set(['Thor', 'Black Widow', 'Hawkeye', 'Hulk']),
'Thor':  set(['Iron man', 'Hulk', 'Chapulin Colorado']),
'Condorito' : set(['Don Chuma', 'Hulk']),
'Chapulin Colorado' : set(['Condorito', 'Thor']),
# ...
}
```

a) Desarrollar la función `diferencias_poder(superheroes, diferenciapoder, umbral)` que reciba el diccionario `superheroes`, el valor `diferenciapoder` y un valor de `umbral`. La función debe retornar una lista de tuplas de todos los superhéroes que tengan una diferencia de poder (diferencia min-poder y max-poder) mayor o igual a `diferenciapoder` y un min-poder superior al `umbral`. Cada tupla debe contener nombre, detalle, habilidad y max-poder.

```
>>> diferencias_poder(superheroes, 30, 39)
[('Iron man', 'mk42', 'uni-rayo', 95), ('Thor', 'hijo de odin', '
   mjolnir', 100), ('Chapulin Colorado', 'no contaban con mi astucia',
   'chipote chillon', 90)]
```

3. **[40 %]** El villano más malvado sobre la faz de la tierra, Ultron, tiene catalogado a todos los superhéroes de la tierra de la siguiente forma:

```
# nombre: [detalle, edad, habilidad, (min-poder, max-poder)]
superheroes = {
'Iron man': ['mk42', 50 , 'uni-rayo',  (45, 95)],
'Thor': ['hijo de odin', 10000 , 'mjolnir',  (50, 100)],
'Condorito':['de pelotillehue', 40, 'washington', (1, 10)],
'Chapulin Colorado': ['no contaban con mi astucia', 40, 'chipote
   chillon', (40, 90)],
# ...
}
# nombre : nombre_asociados
asociados = {
'Iron man': set(['Thor', 'Black Widow', 'Hawkeye', 'Hulk']),
'Thor':  set(['Iron man', 'Hulk', 'Chapulin Colorado']),
'Condorito' : set(['Don Chuma', 'Hulk']),
'Chapulin Colorado' : set(['Condorito', 'Thor']),
# ...
}
```

b) Ultron necesita saber quién es amigo de quien. Dos asociados se consideran amigos si cada uno tiene al otro en el diccionario asociados. Genere una función `amigos(asociados)` que reciba el diccionario `asociados` y retorne un diccionario con los amigos, donde la llave es el superhéroe y como valor un conjunto con los amigos de éste. Si un superhéroe no tiene amigos, no se agrega.

```
>>> amigos(asociados)
{'Iron man': set(['Thor']), 'Chapulin Colorado': set(['Thor']), 'Thor':
    set(['Iron man', 'Chapulin Colorado'])}
```

3. **[40 %]** El villano más malvado sobre la faz de la tierra, Ultron, tiene catalogado a todos los superhéroes de la tierra de la siguiente forma:

```
# nombre: [detalle, edad, habilidad, (min-poder, max-poder)]
superheroes = {
'Iron man': ['mk42', 50 , 'uni-rayo',  (45, 95)],
'Thor': ['hijo de odin', 10000 , 'mjolnir',  (50, 100)],
'Condorito':['de pelotillehue', 40, 'washington', (1, 10)],
'Chapulin Colorado': ['no contaban con mi astucia', 40, 'chipote
   chillon', (40, 90)],
# ...
}
# nombre : nombre_asociados
asociados = {
'Iron man': set(['Thor', 'Black Widow', 'Hawkeye', 'Hulk']),
'Thor':  set(['Iron man', 'Hulk', 'Chapulin Colorado']),
'Condorito' : set(['Don Chuma', 'Hulk']),
'Chapulin Colorado' : set(['Condorito', 'Thor']),
# ...
}
```

c) Gracias a la manipulación mental de Scarlet Witch, Ultron logrará que los superhéroes amigos que posean una diferencia de poder superior a 40 y además que superen un umbral de min-poder de 30 luchen entre ellos. Se pide generar una función `versus(superheroes, asociados)` que reciba el diccionario `superheroes`, el diccionario `asociados` y retorne una lista de tuplas con los enfrentamientos de los amigos. NOTA: no se deben repetir las parejas.

```
>>> versus(supeheroes, asociados)
[('Iron man', 'Thor'), ('Chapulin Colorado', 'Thor')]
```

2. [40 %] Una organización internacional de conservación construyó un domo de cristal para mantener un micro clima en una cierta superficie. Para esto, se instalaron miles de paneles de cristal, contando cada uno de ellos con varios sensores que reportan información diariamente a un computador central. Cada sensor transmite una vez al día una tupla con su medición, indicando la fecha, su identificador único y el valor de la medición. Como ejemplo, una medición sería: `(2016, 3, 27, 'aa:01:10', 0.0021)`.

Todas estas mediciones son colocadas en una lista en el computador central, la cual es mantenida **ordenada descendentemente** por el valor medido. Esto facilita la recuperación posterior de mediciones, debido a que por el gran volumen del registro resulta imposible ordenarlo con cualquiera de los algoritmos disponibles en la actualidad. A modo de **ejemplo**, un posible registro sería:

```
registro = [
(2016, 1, 25, 'd2:00:10', 0.0112),
(2015, 10, 24, '3e:15:c2', 0.0105),
(2015, 11, 3, '28:0b:5c', 0.0009), # ...
]
```

a) Construir la función `agregar_medicion(registro, A, M, D, id, val)`, la que agrega la información capturada al `registro`. Considere `A` el año, `M` el mes y `D` el día de la medición.

```
>>> agregar_medicion(registro, 2016, 3, 27, 'aa:01:10', 0.0021)
>>> print registro
[(2016, 1, 25, 'd2:00:10', 0.0112), (2015, 10, 24, '3e:15:c2', 0.0105),
    (2016, 3, 27, 'aa:01:10', 0.0021), (2015, 11, 3, '28:0b:5c',
   0.0009)]
```

2. [40 %] Una organización internacional de conservación construyó un domo de cristal para mantener un micro clima en una cierta superficie. Para esto, se instalaron miles de paneles de cristal, contando cada uno de ellos con varios sensores que reportan información diariamente a un computador central. Cada sensor transmite una vez al día una tupla con su medición, indicando la fecha, su identificador único y el valor de la medición. Como ejemplo, una medición sería: `(2016, 3, 27, 'aa:01:10', 0.0021)`.

Todas estas mediciones son colocadas en una lista en el computador central, la cual es mantenida **ordenada descendentemente** por el valor medido. Esto facilita la recuperación posterior de mediciones, debido a que por el gran volumen del registro resulta imposible ordenarlo con cualquiera de los algoritmos disponibles en la actualidad. A modo de **ejemplo**, un posible registro sería:

```
registro = [
(2016, 1, 25, 'd2:00:10', 0.0112),
(2015, 10, 24, '3e:15:c2', 0.0105),
(2015, 11, 3, '28:0b:5c', 0.0009), # ...
]
```

a) Construir la función `agregar_medicion(registro, A, M, D, id, val)`, la que agrega la información capturada al `registro`. Considere `A` el año, `M` el mes y `D` el día de la medición.

```
>>> agregar_medicion(registro, 2016, 3, 27, 'aa:01:10', 0.0021)
>>> print registro
[(2016, 1, 25, 'd2:00:10', 0.0112), (2015, 10, 24, '3e:15:c2', 0.0105),
    (2016, 3, 27, 'aa:01:10', 0.0021), (2015, 11, 3, '28:0b:5c',
   0.0009)]
```

2. [40 %] Una organización internacional de conservación construyó un domo de cristal para mantener un micro clima en una cierta superficie. Para esto, se instalaron miles de paneles de cristal, contando cada uno de ellos con varios sensores que reportan información diariamente a un computador central. Cada sensor transmite una vez al día una tupla con su medición, indicando la fecha, su identificador único y el valor de la medición. Como ejemplo, una medición sería: `(2016, 3, 27, 'aa:01:10', 0.0021)`.

Todas estas mediciones son colocadas en una lista en el computador central, la cual es mantenida **ordenada descendentemente** por el valor medido. Esto facilita la recuperación posterior de mediciones, debido a que por el gran volumen del registro resulta imposible ordenarlo con cualquiera de los algoritmos disponibles en la actualidad. A modo de **ejemplo**, un posible registro sería:

```
registro = [
(2016, 1, 25, 'd2:00:10', 0.0112),
(2015, 10, 24, '3e:15:c2', 0.0105),
(2015, 11, 3, '28:0b:5c', 0.0009), # ...
]
```

b) Construir la función `corregir_medicion(registro, A, M, D, id, nvo_valor)` que corrige el valor medido de un sensor. La función recibe como parámetro el `registro`, el año, mes y día de la medición a corregir, el identificador único y el nuevo valor medido. Considere que luego de esta actualización el registro debe permanecer ordenado como se indicó en el enunciado.

```
>>> corregir_medicion(registro, 2016, 3, 27, 'aa:01:10', 0.0221)
>>> print registro
[(2016, 3, 27, 'aa:01:10', 0.0221), (2016, 1, 25, 'd2:00:10', 0.0112),
   (2015, 10, 24, '3e:15:c2', 0.0105), (2015, 11, 3, '28:0b:5c',
   0.0009)]
```

2. [40 %] Una organización internacional de conservación construyó un domo de cristal para mantener un micro clima en una cierta superficie. Para esto, se instalaron miles de paneles de cristal, contando cada uno de ellos con varios sensores que reportan información diariamente a un computador central. Cada sensor transmite una vez al día una tupla con su medición, indicando la fecha, su identificador único y el valor de la medición. Como ejemplo, una medición sería: `(2016, 3, 27, 'aa:01:10', 0.0021)`.

Todas estas mediciones son colocadas en una lista en el computador central, la cual es mantenida **ordenada descendentemente** por el valor medido. Esto facilita la recuperación posterior de mediciones, debido a que por el gran volumen del registro resulta imposible ordenarlo con cualquiera de los algoritmos disponibles en la actualidad. A modo de **ejemplo**, un posible registro sería:

```
registro = [
(2016, 1, 25, 'd2:00:10', 0.0112),
(2015, 10, 24, '3e:15:c2', 0.0105),
(2015, 11, 3, '28:0b:5c', 0.0009), # ...
]
```

c) Construir la función `sensores_sobre_umbral(registro, u)`, la que recibe un valor de medición usado como umbral y también el `registro`. Esta función entrega un listado con todos aquellos sensores, cuyo valor medido se encontró por sobre el umbral `u` entregado. Este listado no debe contener identificadores duplicados.

```
>>> sensores_sobre_umbral(registro, 0.01)
['aa:01:10', 'd2:00:10', '3e:15:c2']
```

```
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

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

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}
```

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],
              ...}
```

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
```

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: