Universidad de SanAndrés

Registro de deudas

I110 - Introducción al Pensamiento Computacional

Trabajo Práctico 3

Fecha de entrega: 17 de noviembre, 23:59 hs

Entregas tarde: Se restarán 2 puntos por cadía día de entrega tarde

1. Objetivo

La temática de este trabajo se relaciona con la contabilidad, asumiendo que en una amplia vivienda coexisten una gran cantidad de inquilinos y debe llevarse un registro de los gastos comunes que se realizan. Regularmente, alguno de esos inquilinos hace un gasto en nombre de una o varias personas, las cuales contraen una deuda con la vivienda. Todos los gastos y cualquier otro tipo de información relevante se almacenan en un archivo de texto en un formato específico. El objetivo de este trabajo es tomar dicho archivo con la información registrada sobre los gastos y procesarla para analizar la evolución de la deuda de cada persona respecto a la vivienda. Para ello, se debe leer el archivo, reestructurar la información de manera conveniente, y graficar la evolución de sus deudas a lo largo de los años. A diferencia del trabajo anterior, donde el procedimiento era considerablemente guiado, tendrán más libertad para formular la estructura del código de acuerdo a su conveniencia. Aún asi, se explicarán ciertas funciones útiles.

2. Deudas compartidas

Como se mencionó previamente, cada persona puede en algún momento realizar un gasto a nombre de una o varias personas. Para no perder el rastro de los gastos se guardan en un archivo de texto "transacciones.txt". Este archivo tiene las siguientes peculiaridades:

1. La primer línea del archivo tiene los nombres de todos aquellos que viven en el departamento al momento de crear el archivo, es decir, los que son los inquilinos "originales". Por ejemplo:

   ```
   persona-A persona-B persona-C persona-D persona-E persona-F
   ```

2. Luego de la primer línea pueden haber 4 tipos de líneas. En todas el primer dato es la fecha, y luego:

   1. **Una deuda repartida entre algunos:** Si por ejemplo, el gasto del primero de mayo de 2023 de $1800 la pagó *persona-A* , y ese gasto solo se reparte entre *persona-A*, *persona-B* y *persona-C*, la línea del archivo sería la siguiente:

      ```
      2023-05-01 persona-A 1800 persona-A persona-B persona-C
      ```

      Primero esta la fecha, luego el que pagó, luego el monto, y luego a los que les corresponde la deuda, incluyendo a la persona que pagó (si corresponde), todo separado por espacios.

   2. **Una deuda repartida entre todos:** Si la deuda le corresponde pagar a todos, la línea es de la siguiente forma:

      ```
      2023-05-10 persona-E 2000 ~
      ```

      Nuevamente, primero esta la fecha, luego el que pagó (en este caso *persona-E*) y luego el monto. Como ahora la deuda le corresponde a todos, en vez de poner los nombres de todos se pone específicamente el caracter ~. Nuevamente, todo separado por espacios.

   3. **Una deuda repartida entre todos menos algunos:** Si la deuda le corresponde pagar a todos menos a algunos, la línea es de la siguiente forma:

      ```
      2023-06-12 persona-F 1204 ~ persona-B
      ```

      Aquí también primero esta la fecha, luego el que pagó (en este caso *persona-F*) y luego el monto. Como la deuda le corresponde a todos menos algunos se agrega el ~ y luego los nombres de aquellos que **no deben pagar esta deuda**. En este caso a la *persona-B* no le corresponde pagar la deuda. Nuevamente, todo separado por espacios.

4. **Incorporación de un nuevo inquilino:** Como a lo largo de los años se puede sumar gente a la vivienda, esta línea indica que alguien se sumo y que a partir de ese momento puede pagar deudas, deber deudas, y ser considerado como parte del "todos".

   ```
   2023-08-01 * persona-G
   ```

   En este caso primero esta la fecha, luego un asterisco que indica que se está sumando alguien, y luego el nombre del la persona que se suma.

3 Cálculo de la deuda

Asuma que tenemos el siguiente archivo de transacciones:

```
Juan Maria Pedro Rosana
2023-01-01 Pedro 1000 ~
2023-01-03 Maria 2000 Maria Juan
2023-01-03 Rosana 1500 ~ Pedro
2023-03-01 * Alejandro
2023-03-05 Maria 2500 ~
```

Si quisieramos calcular la deuda luego del 5 de marzo de 2023, el procedimiento sería el siguiente. Comienzan las deudas en 0:

Deudas:

```
Juan: 0   Maria: 0   Pedro: 0   Rosana: 0
```

El primero de enero de 2023 Pedro paga 1000 por todos. Como son 4 inquilinos la deuda es 1000/4 = 250. Como Pedro ya pagó 1000 su deuda es 250 - 1000 = -750.

Deudas:

```
Juan: 250   Maria: 250   Pedro: -750   Rosana: 250
```

Dos días después Maria paga 2000 de una deuda que le corresponde a ella y a Juan, es decir, 1000 cada uno. La deuda de Maria es ahora 250 - 2000 + 1000 = -750. Juan ahora debe 250 + 1000 = 1250.

Deudas:

```
Juan: 1250   Maria: -750   Pedro: -750   Rosana: 250
```

Ese mismo día, Rosana paga 1500 por algo que deben todos menos Pedro, es decir, 3 personas. La deuda individual es entonces 1500/3 = 500. La deuda de Rosana es entonces 250 + 500 - 1500 = -750. Juan y Maria suman 500 de deuda.

Deudas:

```
Juan: 1750   Maria: -250   Pedro: -750   Rosana: -750
```

El primero de marzo se suma Alejandro al departamento. La deuda queda entonces:

Deudas:

```
Juan: 1750   Maria: -250   Pedro: -750   Rosana: -750   Alejandro: 0
```

Por último, María paga 2500 por algo que le corresponde a todas las 5 personas actuales, es decir, 500 cada uno. Las deudas quedan entonces:

Deudas:

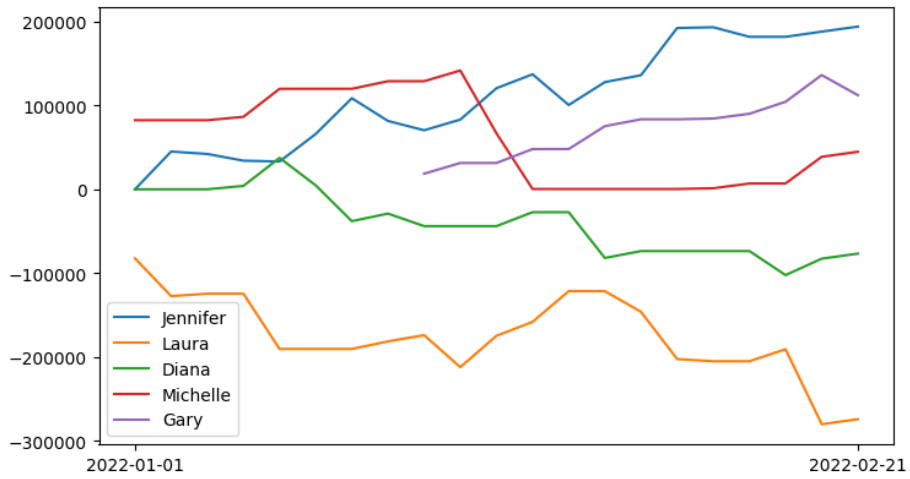
```
Juan: 2250   Maria: -2250   Pedro: -250   Rosana: -250   Alejandro: 500
```

Noten que Alejandro solo tiene como deuda aquellas que le correspondan luego de que se agregue al grupo.

4. Requerimientos del trabajo

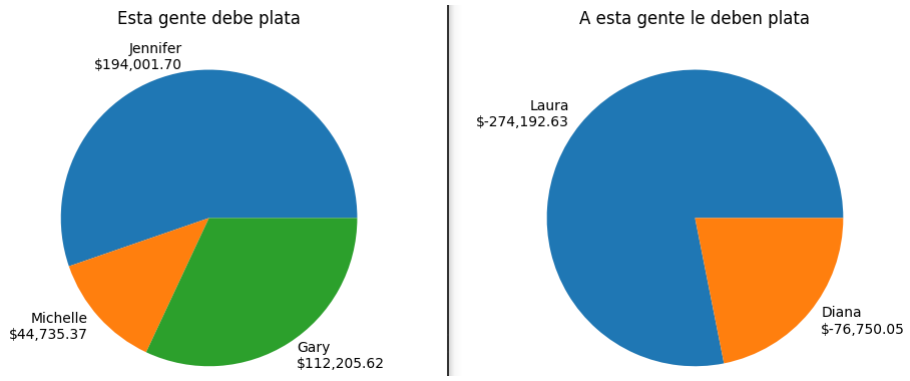
Para este trabajo les pediremos dos funciones:

1. Una función que reciba el nombre de un archivo de transacciones y haga un gráfico de la evolución de las deudas. El eje X serían las fechas, y el eje Y sería la deuda en pesos de cada persona. Debería haber una línea por persona. Si una persona se suma en el medio, su línea de deuda debe aparecer únicamente despues de su primer deuda. Para que esté más "limpio" el eje X, solo se deben poner la primer y última fecha. Para el archivo de prueba "transacciones_simple.txt" el gráfico quedaría así:



> Nota: Si en un mismo dia se hacen varias transacciones, en el gráfico debe aparecer **únicamente** el estado de deduas al finalizar ese día

2. Una función que reciba el nombre de un archivo de transacciones y una fecha:
    1. Si la fecha es anterior a la primer fecha del archivo, debe imprimir por pantalla que la fecha ingresada no es válida
    2. Caso contrario, se deben hacer dos gráficos circulares con informacion corresponiente a la fecha ingresada. El primer gráfico con aquellos a los que se les debe plata, y el segundo con aquellos que deben plata. Para cada pedazo del círculo se debe informar el nombre y la deuda (para mejorar la legibilidad se debe poner el valor de las deudas con comillas). Para el caso de "transacciones_simple.txt" en el último día quedarían de la siguiente forma:



> Nota: Si en un mismo dia se hacen varias transacciones, en el gráfico debe aparecer **únicamente** el estado de deduas al finalizar ese día. Si la fecha seleccionada no registra transacciones, **aun asi hay deudas**. En esos casos deberá hacer los gráficos con las deudas de la fecha con transacciones más cercana (en el pasado).

5. Recomendaciones

5.1 Organización del código

Antes de empezar a efectivamente escribir código, les recomendamos que planteen en papel una organización general del mismo. Lean bien la consigna y evalúen que funciones podrían servir para reducir el código repetido y modularizar el trabajo. Por ejemplo, una pregunta clave que deben responder es cómo van a organizar la información que extraigan del archivo, en una estructura adecuada en python. ¿Cuál es la mejor / mas cómoda / mas entendible forma de organizar una especie de tabla de deudas a lo largo del tiempo? ¿Cómo hago para que me quede lo más fácil posible el acceso a cada dato, considerando los objetivos del trabajo? Hay varias respuestas adecuadas, todo depende de cómo se organice el código.

Además, es clave que presten atención a la modularización del código. Las dos funciones que se piden realizar, ¿tienen tareas comunes? En caso afirmativo, ¿puedo tomar los procesos que se repiten, y encapsularlos en otras nuevas funciones? ¿En cuántas funciones lo puedo modularizar, conviene tener una sola o más?

5.2 Archivos de prueba

Junto a esta consigna se les entregan dos archivos de prueba:

1. transacciones_simple.txt
2. transacciones_largo.txt

Estos archivos pueden ser usados para probar su código a medida que lo van trabajando. Sin embargo tal vez les resulte cómodo probar con archivos más cortos, con 2 o 3 personas, 2 o 3 deudas. Archivos sin agregar inquilinos, archivos sin deudas generales. Para ello pueden simplemente tomar el archivo de transacciones simple, y crear un nuevo archivo de texto con sólo las primeras transacciones. De esta manera, pueden probar el código primero teniendo solamente pagos de una persona, luego pueden agregar pagos con deuda para todos, luego eventos donde se suma una persona a la vivienda, etc. Esto permite ir probando el código de a partes, ya que al intentar hacer un código que funcione con todo lo pedido puede llevar a pequeños errores difíciles de encontrar.

5.3 Uso de funciones

En línea con lo dicho previamente, si bien ambas funciones solicitadas en este trabajo pueden tener tareas repetidas, si usamos funciones podemos aprovechar para evitar repetir código entre ellas. Sería de conveniencia entonces generar nuevas funciones para ocuparse de sub-tareas. Por ejemplo, una función que se encarge de abrir el archivo y convertir la información en un formato que nos sea más conveniente de usar. Luego, dentro de esa función o en una aparte se pueden ejecutar subfunciones que se encargen de procesar cada caso en particular del archivo de texto (todos los tipos posibles de pago, el agregado de nuevos inquilinos). De esta manera podemos probar los distintos casos por separado y luego unificar todo.

6. Extensiones [OPCIONAL]

Si usted desea, puede agregar funcionalidades más allá de las que se piden, siempre y cuando no interfieran con las funcionalidades básicas. Si estas funcionalidades "extra" funcionan correctamente y no afectan a la consigna base, el docente podrá a su discreción darle hasta 2 puntos adicionales, que levantan la nota en caso que no sea la máxima (si la nota base era un 8 pero se cumplen con todos los puntos opcionales se puede subir a un 10).

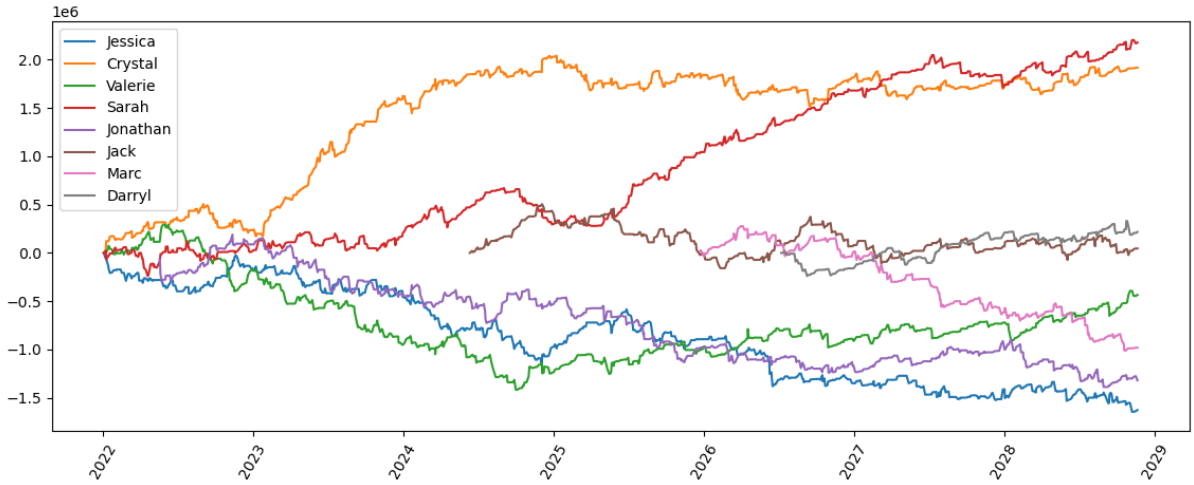
A continuación se muestran ideas de "features" adicionales que usted podría agregar, si quisiera, ordenados de menor a mayor complejidad.

1. [★] Ahora en el archivo de transacciones aparecerá también el comando para eliminar gente de los gastos comunes:

   ```
   2023-03-01 - Alejandro
   ```

   En este caso se esta eliminando a *Alejandro* de la lista, lo cual se indica con un guión en lugar de un asterisco. Esto no quiere decir que su deuda se elimine, simplemente ya no se lo considera para los gastos generales (tampoco deberia aparecer con gastos individuales). Modifique su código para contemplar estos casos

2. [★★] Para mejorar el gráfico de dedudas temporales, utilize la libreria Datetime. Así no solo se espacian correctamente las fechas, sino que automáticamente se eligen las fechas que iran en el eje x. También rote las fechas 60 grados. Para *transacciones_largo.txt* queda:

3. [★ ★ ★] Haga una función que reciba el nombre del archivo de transacciones y genere un informe llamado "*reporte.txt*". Por ejemplo, para el archivo "transacciones_largo.txt" el formato del informe debe ser el siguiente:

   ```
   Registro de deudas al día de 2028-11-20

   Deudores:
       Crystal debe dar $1,917,510.20
       Sarah debe dar $2,178,502.00
   ```

```
    Jack debe dar $46,103.82
    Darryl debe dar $217,224.28

Acreedores:
    Jessica debe recibir $1,626,492.84
    Valerie debe recibir $434,507.56
    Jonathan debe recibir $1,318,589.69
    Marc debe recibir $979,750.21


Intercambios:
    Crystal debe pagarle a Jessica $1,626,492.84
    Crystal debe pagarle a Valerie $291,017.36
    Sarah debe pagarle a Valerie $143,490.20
    Sarah debe pagarle a Jonathan $1,318,589.69
    Sarah debe pagarle a Marc $716,422.11
    Jack debe pagarle a Marc $46,103.82
    Darryl debe pagarle a Marc $217,224.28
```

Notese que la primer línea informa el último día registrado en el archivo. Luego se informan los deudores, seguido por los acreedores. Finalmente en se informan los intercambios de dinero necesario para que se cancelen todas las deudas

6. Código de Ética

Recuerde que se considera falta grave a la ética universitaria: "El plagio, la omisión de citar las fuentes de ideas, razonamiento o información que no resulten de la propia elaboración; la presentación en un curso, como si fueran originales, de trabajos realizados en otros cursos; el uso de fuentes o materiales no autorizados en exámenes finales o parciales; el aprovechamiento indebido del trabajo de otros integrantes de la comunidad universitaria; la facilitación de materiales que permitan a otro fraguar un conocimiento o habilidad (capacidad) que no se posee y cualquier otro acto que implique un engaño sobre las propias habilidades o capacidades." - Código de Ética de la Universidad de San Andrés.

7. Modalidad de Trabajo y Formato de entrega

- El TP se podrá realizar en grupos de hasta 2 personas.
- Se debe realizar una entrega digital a través del Campus Virtual de la materia, compuesta por el archivo con el código fuente en .py.
- **El nombre del archivo deberá seguir el formato** *tp3_apellido1_apellido2.py*
- El código entregado deberá estar debidamente comentado de manera que el docente que corrige el trabajo entienda lo que se está haciendo y por qué decidieron hacerlo de tal o cual manera. Además, todas las funciones deben contener su docstring, indicando parámetros de entrada y salida, tipos de datos, y breve descripción de la función. Abajo se muestra un ejemplo de un código con comentarios adecuados. El programa en cuestión debe imprimir en pantalla los números múltiplos de 5 hasta que el usuario decida que no se impriman más. Observe cómo está comentado para que se comprenda el flujo. Ustedes deberán comentar su código de manera similar para que el corrector entienda lo que querían hacer.

```
def multi_5():
    """
    La función multi_5 imprime numeros múltiplos de 5 hasta que el usuario
    decida no imprimir más. Para que eso suceda deberá ingresar un valor
    distinto de 'si'. Una vez que se deja de imprimir el programa devuelve
    el último valor al que se llegó
    Entrada:
        - None
    Salida:
        - final [int]: Último numero al que se llegó
    """

    # creamos la variable "numero" que guardará el número a imprimr en cada
    # ciclo. Como vamos a sumarle 5 cada vez que se corra el ciclo la variable
    # comienza en 0 para que en el primer ciclo se le sume 5 y se imprima 5
    numero = 0

    # Creamos la variable "seguir" que indicará si se quiere seguir
    # imprimiendo números Para que el ciclo se haga por lo menos una vez se
    # comienza con 'si'
    seguir = 'si'
```

```
    # El ciclo while seguirá siempre y cuando "seguir" sea 'si'
    while seguir == 'si':
        numero += 5 # se le suma 5 para ir por múltiplos de 5
        print(numero)
        seguir = input('¿Quiere seguir?    ') # Se le pregunta al usuario si
        # quiere seguir y se guarda la respuesta en la variable "seguir"
        # Si el usuario ingresa 'si' se repetirá el ciclo

    # Una vez que el usuario ingrese algo que no sea 'si' se rompe el ciclo y
    # se termina el código
    print('Adiós')
    return numero

# Utilizamos la función
valor = multi_5()
```