La tarea menciona la instalación de un programa llamado getJason.pyc, el cual es heredado de un sistema obsoleto y no se dispone del código fuente. Este programa permite recuperar un token de API para acceder a microservicios bancarios de un banco específico, almacenado en un archivo JSON. El objetivo es reutilizar el programa para que pueda recuperar cualquier clave existente en el archivo JSON, utilizando la clave "token1" como valor predeterminado.

Para instalar los elementos contenidos en el archivo RRR\_TP4.tar.gz, incluyendo el programa getJason.pyc, el archivo JSON de prueba y la documentación existente, se deben seguir los siguientes pasos:

1. Descomprimir el archivo RRR\_TP4.tar.gz en una ubicación deseada.
2. Verificar que Python 2.7 esté instalado en el sistema.
3. Colocar el archivo getJason.pyc, el archivo JSON y la documentación en la misma carpeta.

Para ejecutar el programa getJason.pyc y verificar la correspondencia entre la documentación y el comportamiento real, se pueden seguir los siguientes pasos (asumiendo que se encuentra en la misma carpeta que los archivos mencionados):

1. Abrir una terminal o línea de comandos.
2. Navegar a la carpeta donde se encuentran los archivos.
3. Ejecutar el siguiente comando: python getJason.pyc.
4. Seguir las instrucciones proporcionadas por el programa y verificar si el comportamiento coincide con lo documentado.
5. Realizar una prueba utilizando diferentes claves del archivo JSON para asegurar que se recuperan los valores correspondientes.

Para instalar el paquete Python uncompyle6, se puede utilizar el administrador de paquetes pip. Ejecuta el siguiente comando en la terminal o línea de comandos: pip install uncompyle6.

Para obtener el código fuente de getJason.py a partir de getJason.pyc utilizando uncompyle6, se puede ejecutar el siguiente comando en la terminal o línea de comandos: uncompyle6 getJason.pyc > getJason.py. Esto generará un archivo getJason.py legible con el código fuente.

Las posibles razones de las diferencias encontradas entre el código obtenido y la documentación pueden incluir

* errores en la documentación original.
* Modificaciones realizadas en el programa original después de compilar getJason.pyc.
* Falta de actualización o discrepancia entre la documentación y la implementación real del programa.

Para modificar el programa getJason.py de acuerdo con la documentación y los requisitos de reutilización, se deben seguir las pautas proporcionadas en la documentación. Esto puede incluir la adición de funcionalidades para recuperar cualquier clave existente en el archivo JSON y establecer "token1" como valor predeterminado si no se proporciona una clave.

Para generar una nueva versión de getJason.pyc, se puede utilizar el módulo compileall proporcionado por Python. Ejecuta el siguiente comando en la terminal o línea de comandos: python -m compileall getJason.py. Esto creará un archivo getJason.pyc actualizado.

Para permitir que la clase resultante pueda ejecutarse desde la línea de comandos y mejorar las condiciones de verificación de los parámetros externos, se deben implementar controles robustos. Esto implica realizar una validación adecuada de los parámetros de entrada, manejar errores de forma controlada y garantizar que el programa finalice con errores del programa en lugar de errores del sistema.

Para lograr la convergencia entre el programa original y su versión refactorizada, se puede utilizar la estrategia "Branching by abstraction". Esto implica crear una versión refactorizada del programa que implemente las mejoras y funcionalidades deseadas, mientras se mantiene la versión original intacta. Luego, se pueden aplicar técnicas de despliegue gradual o enrutamiento para migrar gradualmente a la nueva versión, evaluando su funcionamiento y realizando ajustes según sea necesario.

Utilice el analizador estático de código pylint para realizar una revisión del programa y corregir las observaciones hasta que el puntaje otorgado sea de 8 o superior. Esto implica seguir las recomendaciones y prácticas de estilo de código de Python, asegurarse de utilizar nombres descriptivos de variables y funciones, mantener una estructura clara y legible, entre otros aspectos.

- En el contexto de un cambio organizacional más profundo, se plantea una intervención de reingeniería en el programa getJason.py:

En el proceso de pagos anterior, un empleado decidía qué banco liberar un pago y elegía un token correspondiente. Sin embargo, se propone automatizar este proceso, verificando que exista saldo suficiente en la cuenta respectiva y realizando los pagos de forma balanceada.

Para automatizar la decisión del banco y token correspondiente, se puede crear un nuevo componente que, ante una solicitud de pago, seleccione automáticamente la cuenta desde la cual realizarlo. La información de relación entre el banco (token) y la clave se encuentra en el archivo sitedata.json.

Para integrar este nuevo componente, que dado un nombre de token (banco) proporciona la clave, se puede extender la funcionalidad del objeto Singleton existente en el programa. Esto permitirá seleccionar automáticamente la cuenta adecuada para realizar un pago, basándose en el saldo disponible.

Se creará una clase utilizando el patrón de diseño "cadena de comando" para controlar las dos cuentas correspondientes a "token1" y "token2". Estas cuentas tendrán saldos iniciales de $1000 y $2000 respectivamente.

Y por último se realizarán pedidos de pago por un monto de $500. Estos pagos se alternarán entre ambas cuentas que tengan saldo suficiente. La clase deberá proporcionar como salida el número de pedido, el token utilizado y el monto del pago realizado. Además, se deberá implementar una función de "listado" que muestre todos los pagos realizados en orden cronológico, utilizando un patrón iterator para recorrer los registros.