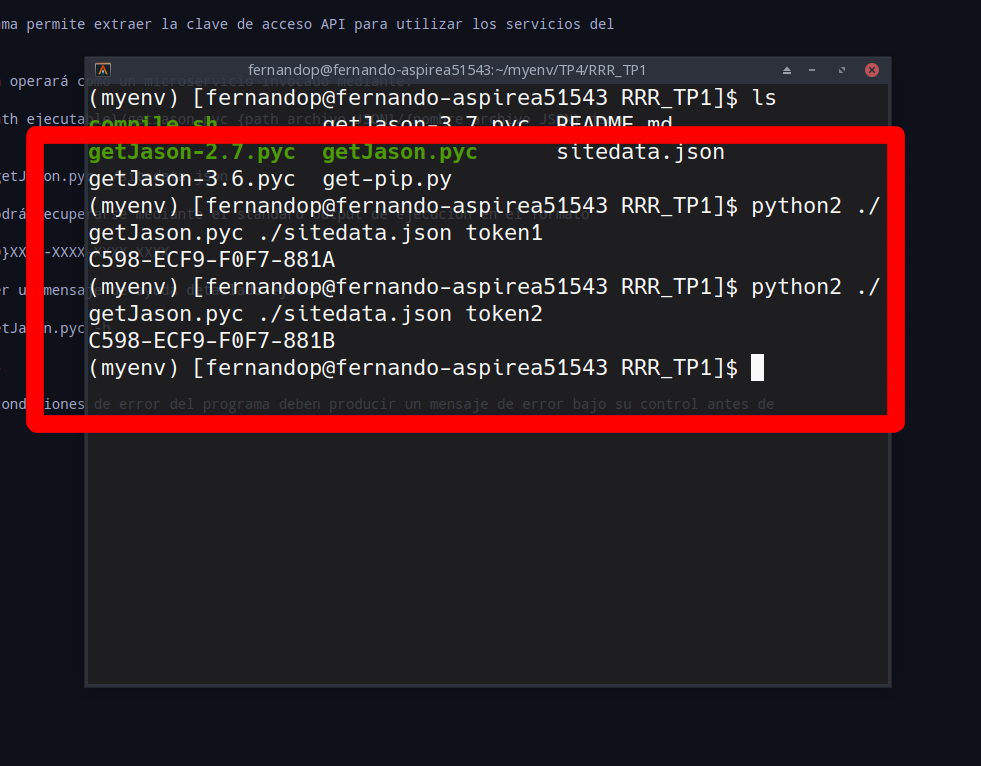
**TRABAJO PRÁCTICO N°4 - ING. SOFTWARE II**

**ALUMNO**: FERNANDO PÉREZ

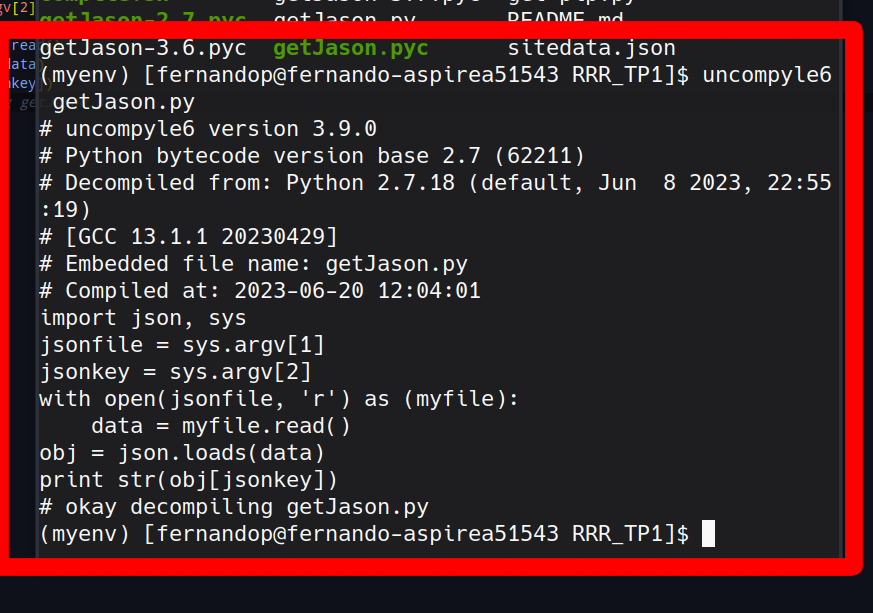
**DOCENTES**: Dr. Pedro E. Colla y Hernán Sanchez

**UADER-FCYT**

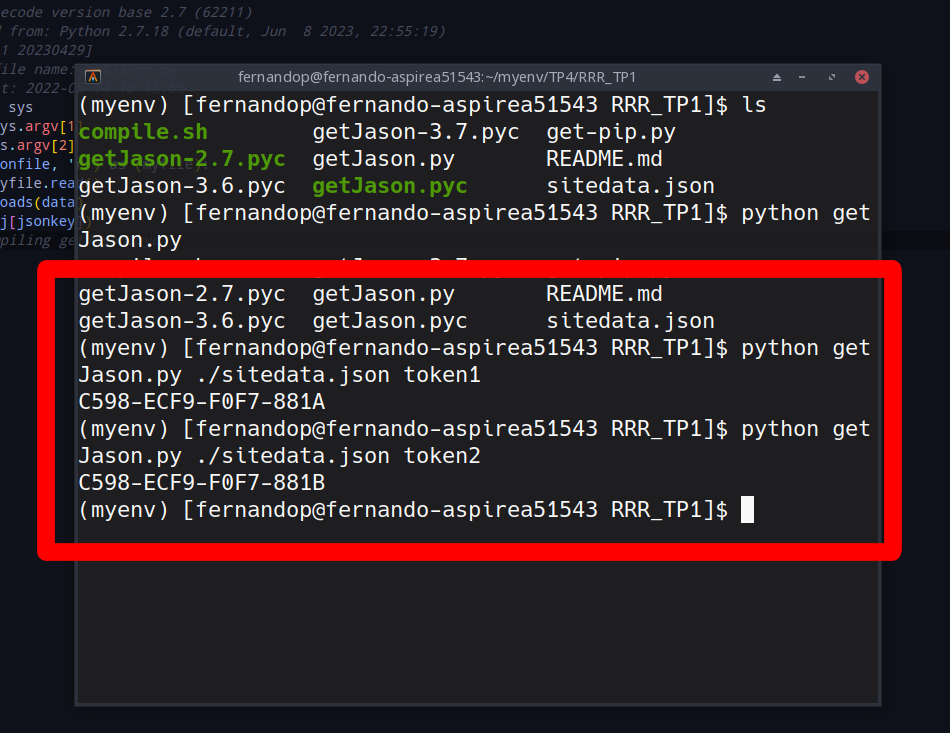
**1\_b)** Al instalar los programas y solicitar las claves al programa getjason.pyc junto al archivo .json de prueba se obtienen las siguientes claves como se muestra a continuación:



**1\_d)** Luego de haber instalado el paquete Python uncompyle6 se ejecuta sobre el programa getJason.pyc y se obtiene lo siguiente:



Luego el programa nuevo getJason.py lo ejecutamos para ver si se ejecuta igual que el programa original y corroboramos que funciona perfectamente como se muestra en la siguiente imagen:

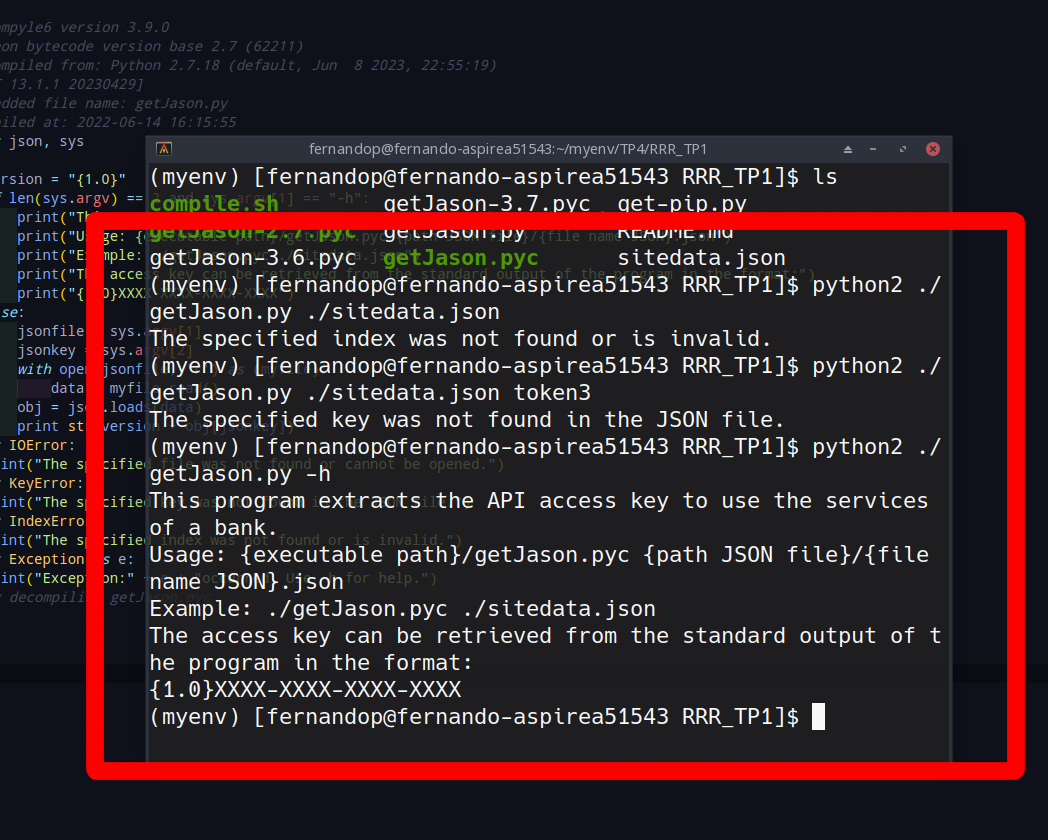


**1\_e)** Las diferencias encontradas entre la documentación del programa y el nuevo programa getJason.py es que los tokens obtenidos no se muestran con el contenido {0.1} que representa la versión del mismo y tampoco se muestra un mensaje de ayuda al ingresar el comando ./getJason.pyc -h.

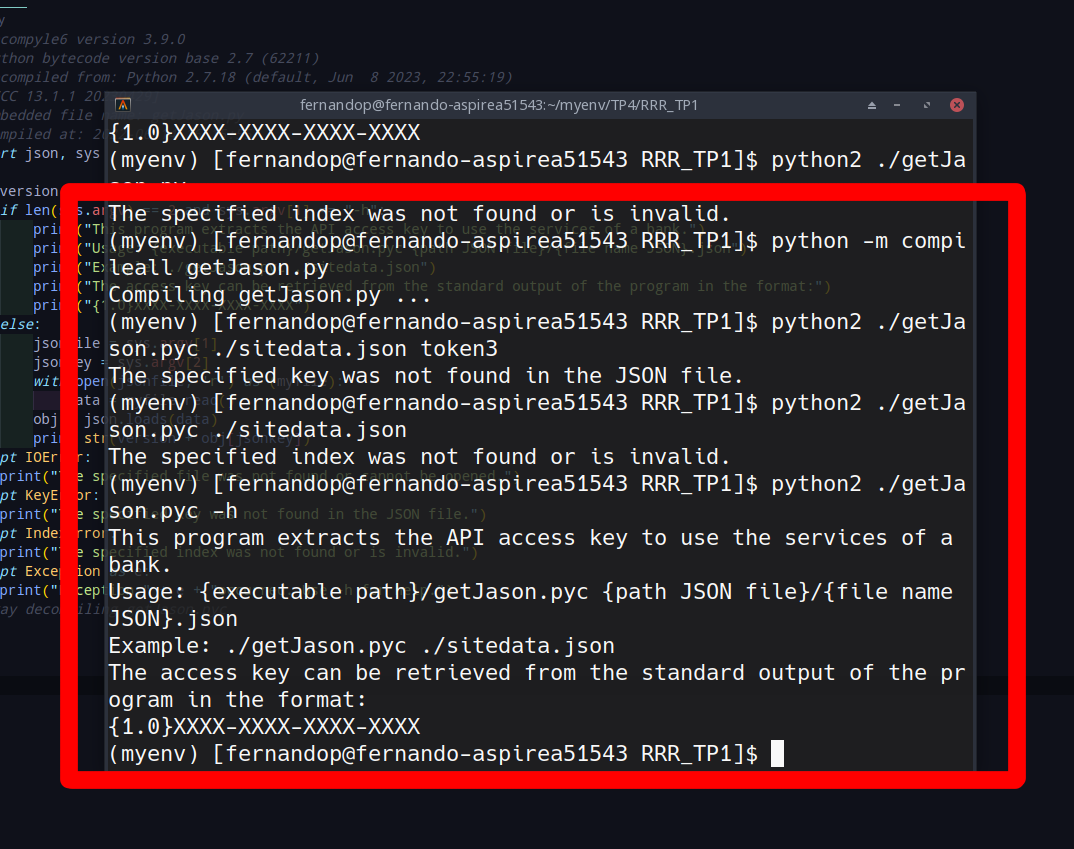
**1\_f)** Se modifica el programa getJason.py para que actúe como indica la documentación, se muestra a continuación el código modificado:



Y acá validando su funcionamiento:



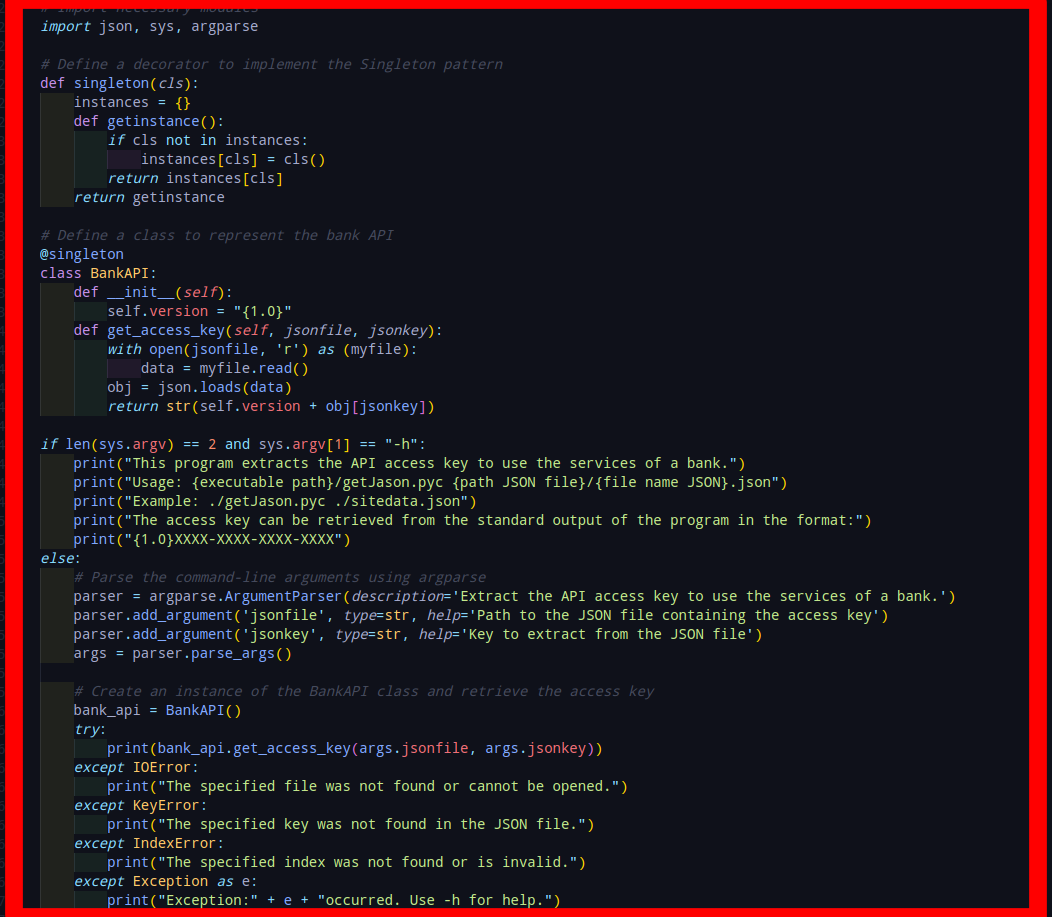
**1\_g)** Se genera la nueva versión del programa getJason.pyc y se verifica su validación como se muestra en la siguiente imagen en la cual corroboramos que se controlen las excepciones:



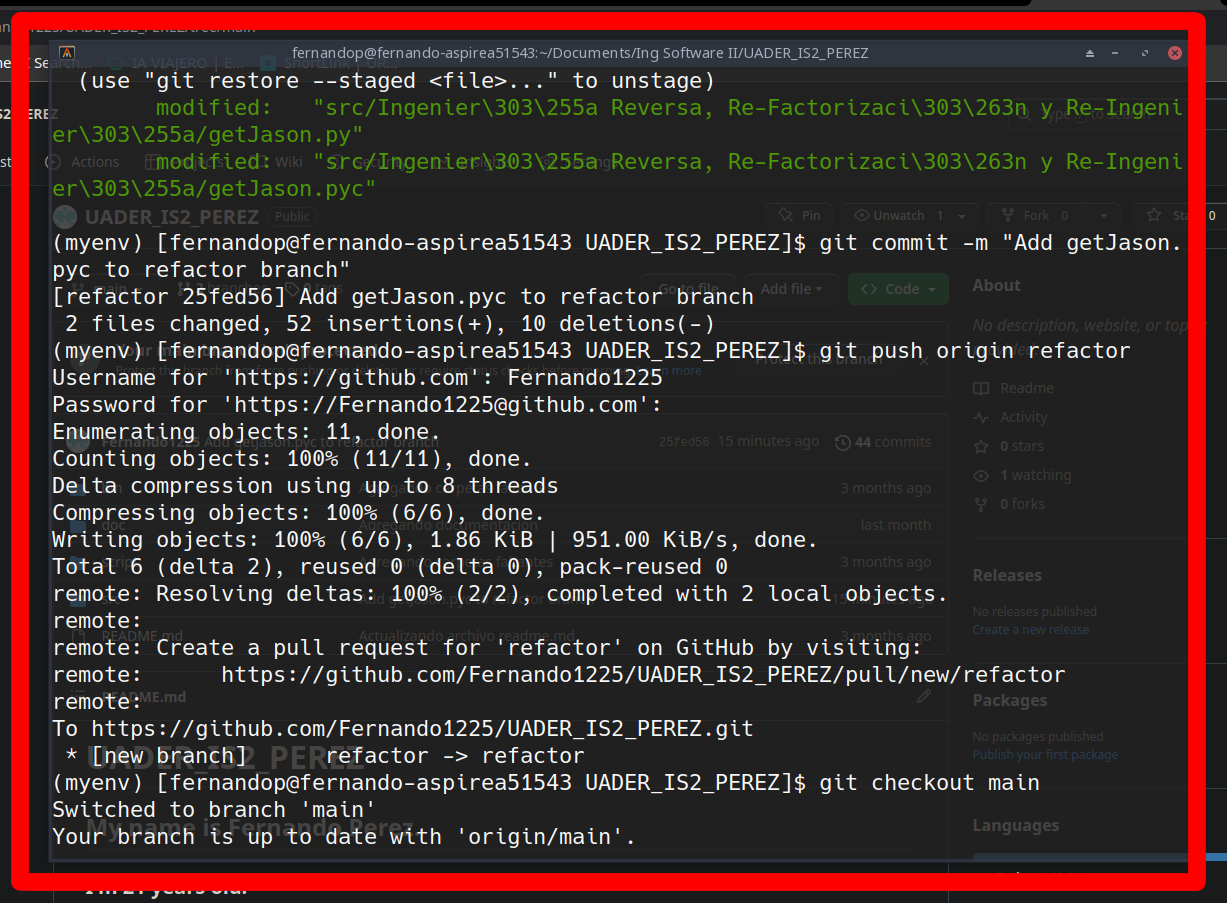
**2\_b)** Al programa anterior getJason.py se le aplica una transformación para que utilice un patrón de diseño Singleton y nos asegure que solo una instancia será creada y esto se logra modificando el programa anterior creando una clase Singleton que cheque si ya hay una instancia o sino cree una, sobre la clase BankAPI que esta última clase lo que hace es mediante un método obtener la clave de acceso denominada token, y hace lo mismo que la versión anterior del programa, chequea que el token solicitado esté y sino lanze una excepción controlada. Se muestra imagen del código a continuación:



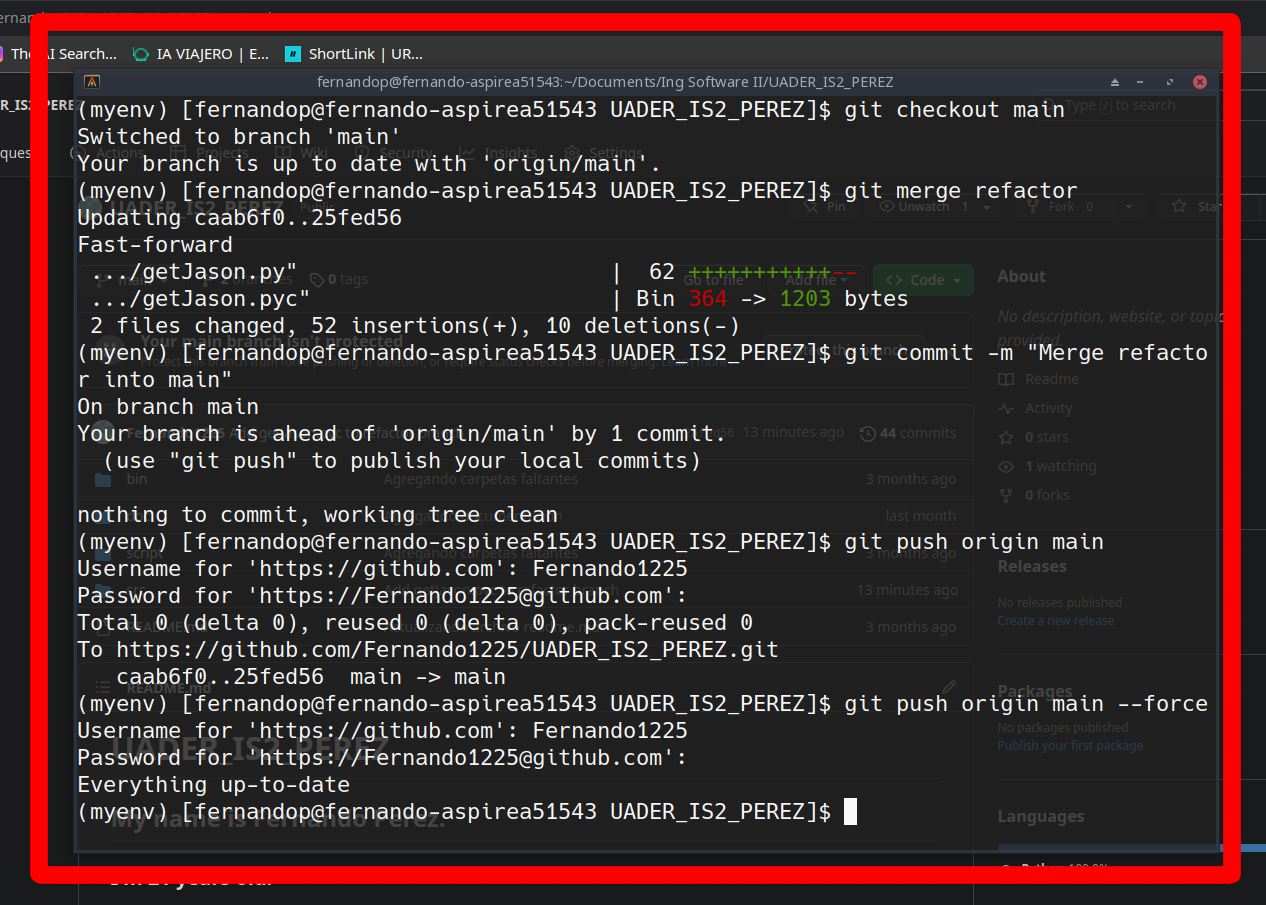
**2\_c)** Al programa getJason.py modificado anteriormente se le mejoran las condiciones de chequeo de parámetros externos de ejecución mediante el uso del módulo argparse que nos permite definir los argumentos que se esperan en la línea de comandos y validarlos siguiendo ciertas reglas, como en éste caso si los argumentos proporcionados son la ruta de acceso al archivo .json y si se proporciona una cadena de carácteres que representen el token que se desea extraer de dicho archivo, así nos aseguramos que el programa nunca va a terminar con un error de sistema y si con error controlado. Se muestra a continuación imagen del código:



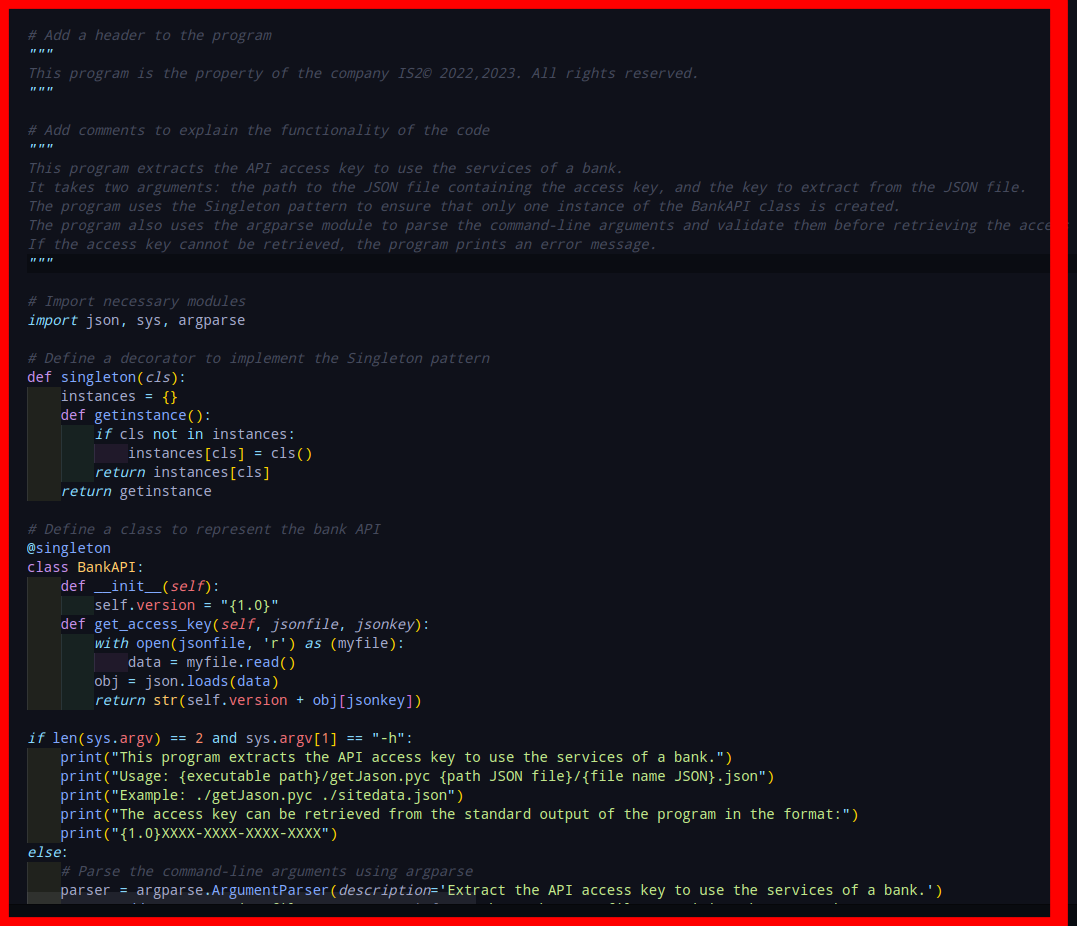
**2\_d)** Para hacer la convergencia entre el programa original y su versión refactorizada creamos una nueva branch llamada «refactor» en la que subimos los archivos getJason.py y getJaspon.pyc como se muestra a continuación:



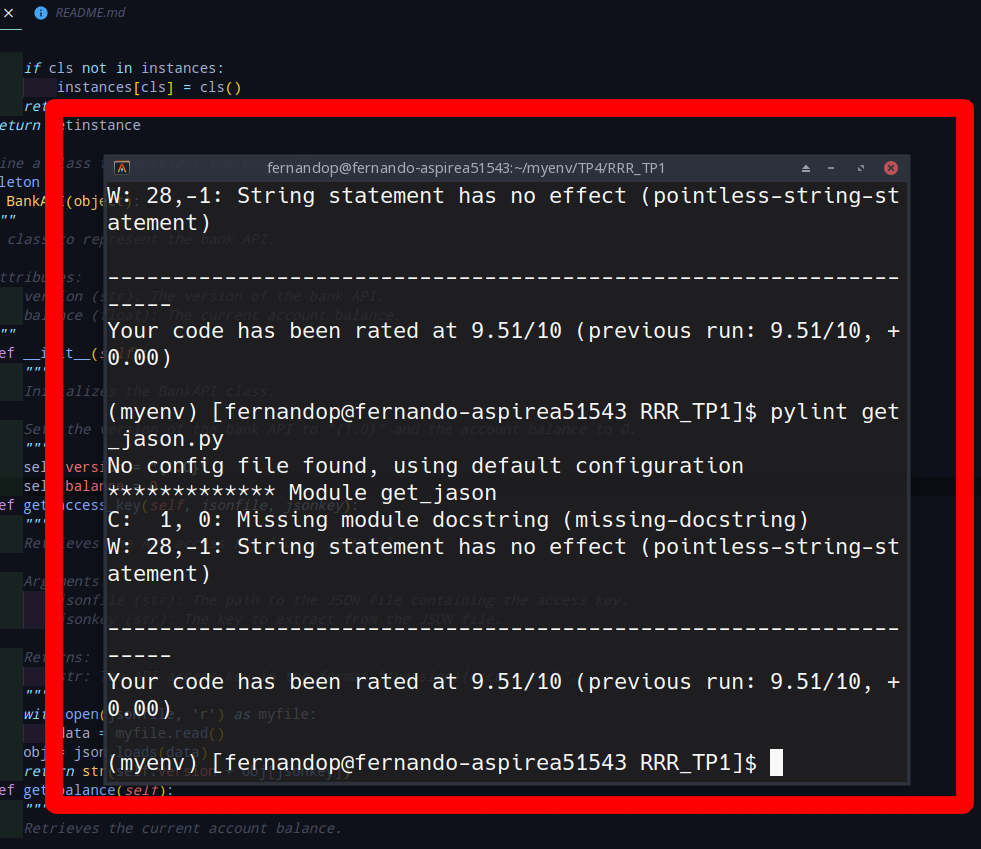
Una vez que modificamos los archivos y corroboramos su correcto funcionamiento hacemos un merge con la rama principal denominada «main» en la que se encuentran los archivos con la versión anterior para actualizarlos como se muestra a continuación:



**2\_e)** Se agregan comentarios al programa anterior getJason.py de su funcionamiento y una carátula donde se especifica la propiedad de la compañía y el funcionamiento general del código como se muestra a continuación en la parte superior:



**2\_f)** Se revisó el programa con el analizador estático pylint y se corrigieron algunas observaciones dando por final el siguiente puntaje:



(La modificación del nombre del archivo se debe al cumplimiento de unas de las solicitudes de modificación de pylint)

**3\_b)** En el siguiente método llamado «make\_payment» se chequea si existe saldo o no en una respectiva cuenta y de ser así se hace el pago, caso contrario muestra un mensaje de error diciendo que el saldo es insuficiente, como se muestra a continuación:



**3\_c)** En el siguiente método se selecciona la cuenta que se desea utilizar para las transacciones, se muestra el código a continuación:



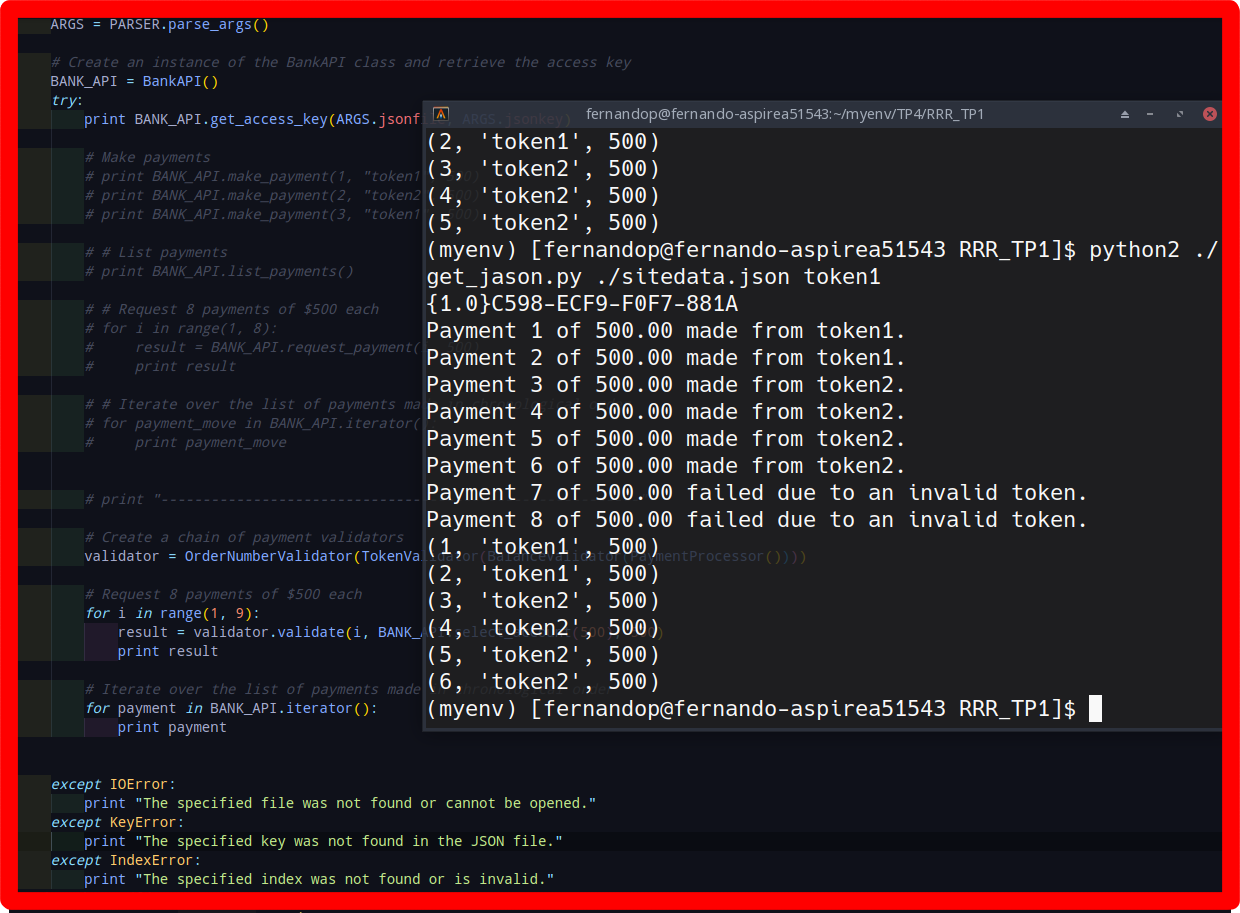
**3\_d\_e)** Se crean clases para controlar las dos cuentas y realizar pedidos de pagos e ir chequeando si las respectivas cuentas tienen o no fondo para hacer la transacción haciendo uso del patrón «cadena de comandos», se muestra ejemplo de código a continuación:

|

Y otra clase para validar los pagos como se muestra en la imagen siguiente:



Y a continuación se muestra un ejemplo de su funcionamiento:



**GITHUB REPO:** <https://urlis.net/repo-tp4>