**Laboratorio No.3: Arreglos y listas encadenadas**

*Mariana Díaz Arenas 202020993 & Nicole Murillo Fonseca 202025521*

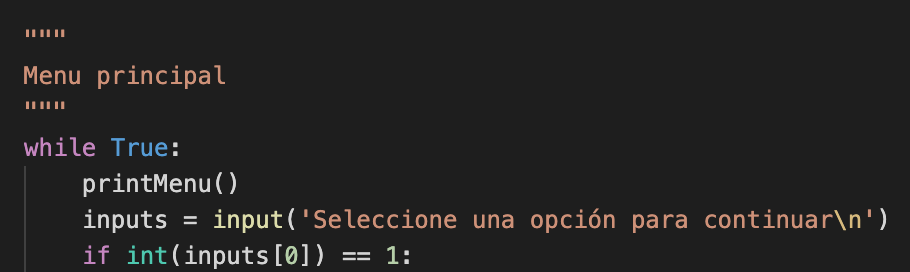
*Universidad de Los Andes*

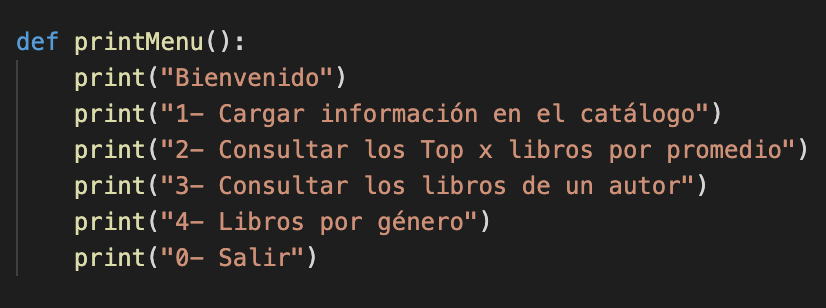
*11 de Febrero 2021*

**Paso 3:** Estudiar el ejemplo en VS Code

1. ¿Cuáles son los mecanismos de interacción (I/O: Input/Output) que tiene el view.py con el usuario?

El primer mecanismo de interacción presente en el view.py es el menú principal, desde el cual el usuario puede elegir la función que desee que ejecute el programa. Lo que el usuario ve es la segunda imagen, una variedad de opciones a ejecutar con su respectivo número; ahora bien, el usuario puede escribir el número que desea porque hay un condicional while que imprime el menú con un input mientras esté en True o la opción “0- Salir” no sea elegida. Una vez que el usuario digita un número en el input el while se encarga de buscar la función correspondiente y efectuarla.



*Imagen 1. Input del menú principal*

*Imagen 2. Print del menú principal*

1. ¿Cómo se almacenan los datos de GoodReads en el model.py?

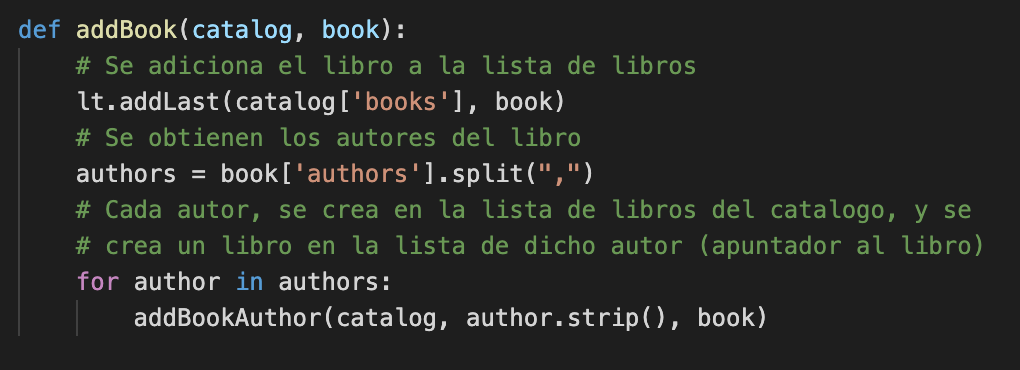
En el model.py para almacenar los datos, primero se implementa la función newCatalog() que retorna un catálogo con listas vacías para guardar posterior y respectivamente los libros, los autores, los géneros (tags) y finalmente la asociación géneros y libros (booktags).

**

*Imagen 3. Función newCtalog()*

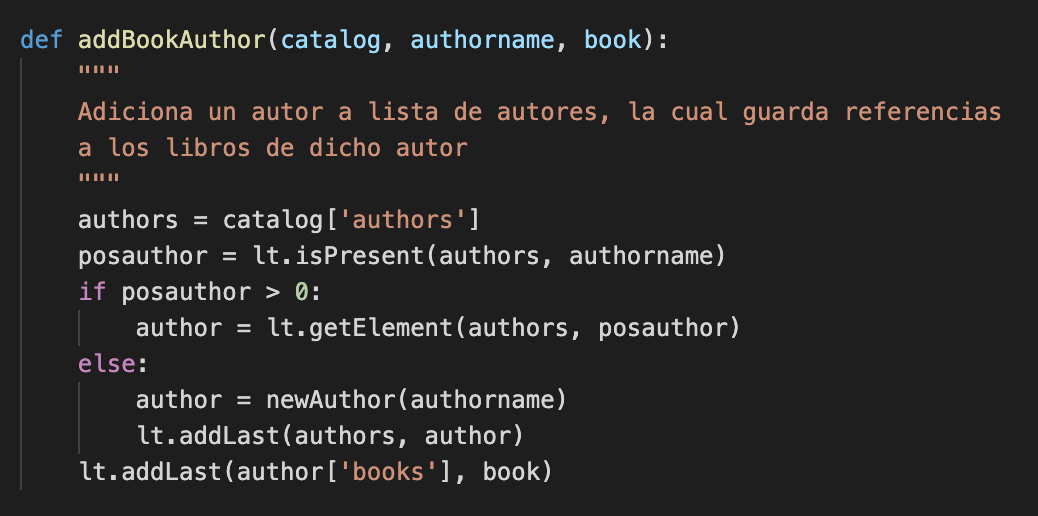
Una vez que se creó el catálogo vacío se empieza a agregar la información en las listas respectivas por medio de las funciones addBook() para agregar los libros, addBookAuthor() para agregar los autores, addTag() para agregar los géneros (tags) y por último la función addBookTag() para agregara la asociación de géneros y libros (booktags).

En la función addBook() se agrega el nombre del libro en la lista vacía de libros y los autores se dividen en otra lista denominada ‘authors’ con el método Split. Luego ese autor es agregado al catalogo con la siguiente función definida en el model.py addBookAuthor().

**

*Imagen 4. Función addBook()*

En la siguiente función, addBookAuthor(), implementada dentro de addBook(), se revisa si el nombre del autor que se quiere agregar, ya está es la lista de autores del catálogo, si está se usa lt.getElement para agregar ese autor a la lista pero si no está se usa otra función, que será explicada más adelante, para crear un nuevo autor.



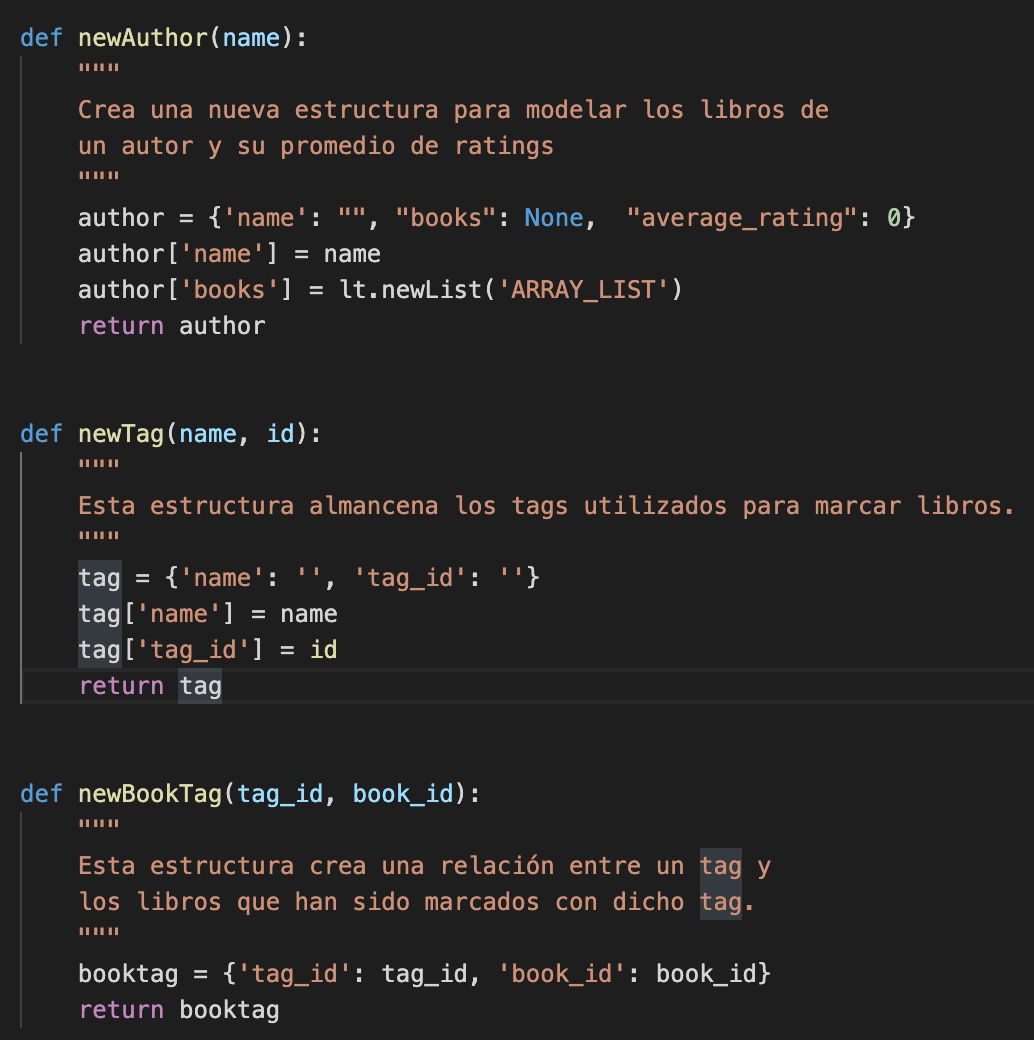
*Imagen 5. Función addBookAuthor()*

Las siguientes dos funcionas addTag() y addBookTag() agregan respectivamente un género o tag a la lista de tags del catálogo y la asociación géneros y libros (booktags) a la lista de book\_tags en el catálogo, utilizando el método addlast(). Además utilizan las funciones newTag() y newBookTag que serán explicadas a continuación.



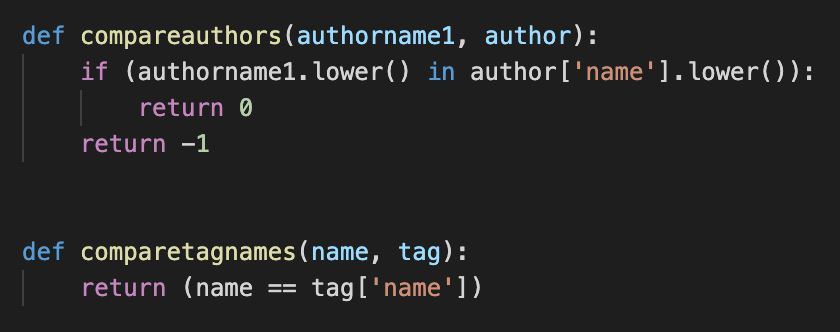
*Imagen 6. Funciones addTag() y addBookTag()*

Las funciones newAuthor(), newTag() y newBookTag sirven para crear datos y retornar un diccionario con la información específica y requerida para cada categoría: autor, género (tags) y la asociación género y libro (booktags). En la imagen 7 se pueden observar los datos de cada diccionario. Estas funciones son implementadas por las mencionadas anteriormente si el nombre del autor, género (tags) o la asociación género y libro (booktag) aún no existen en sus respectivas listas.



*Imagen 7. Funciones de creación (diccionarios)*

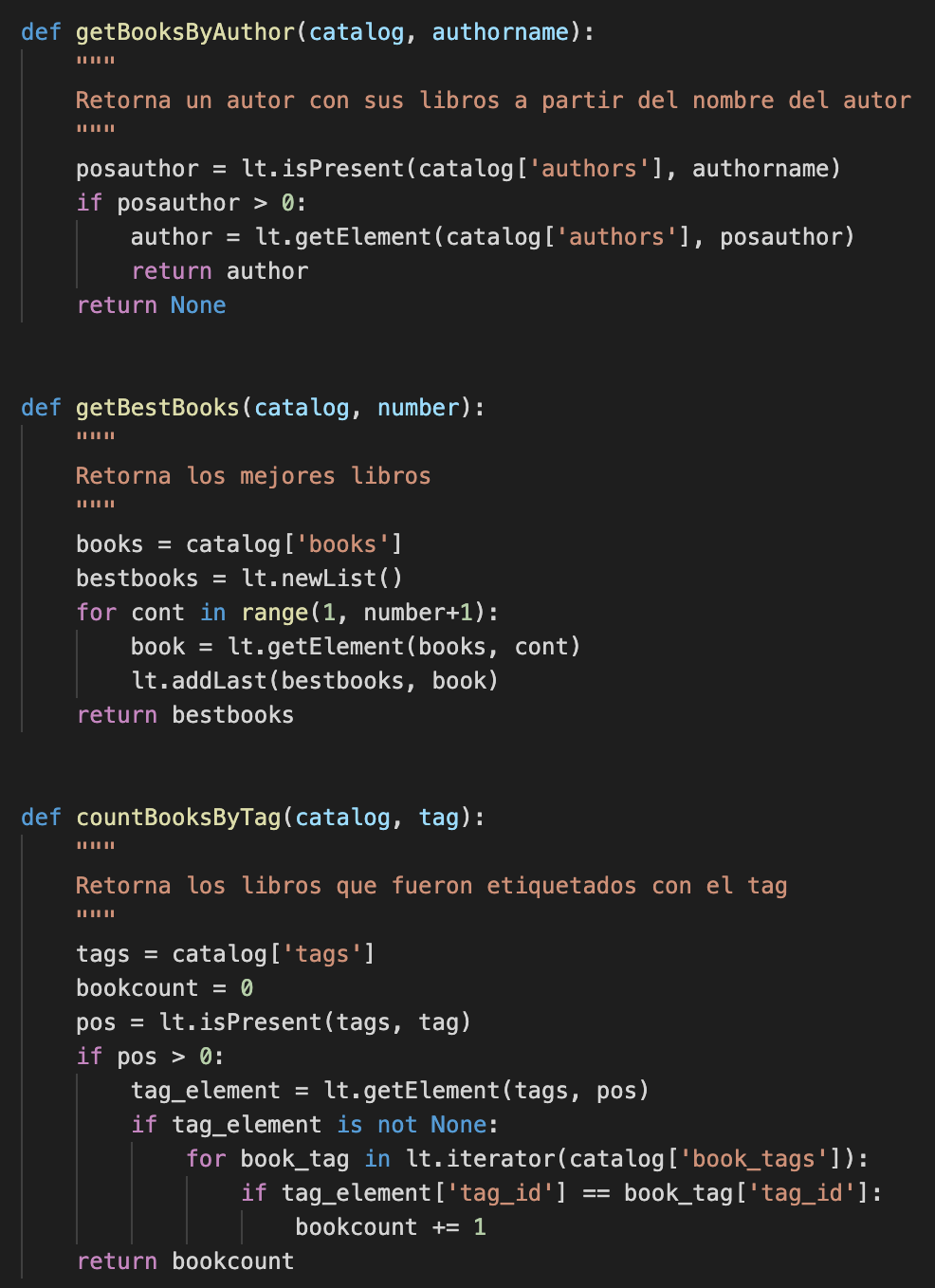
Finalmente, hay unas funciones definidas de consulta; compareauthors() y comparetagnames(), estas las podemos observar en la función newCatalog() en la creación de listas de ‘autores’ y ‘tags’, como parámetro del método newlist().



*Imagen 8. Funciones para comparar elementos utilizados en la función newCatalog()*

1. ¿Cuáles son las funciones que comunican el view.py y el model.py?

Las funciones que comunican el view.py y el model.py son las funciones de consulta: getBookByAuthor(), getBestBooks() y CountBooksByTag().

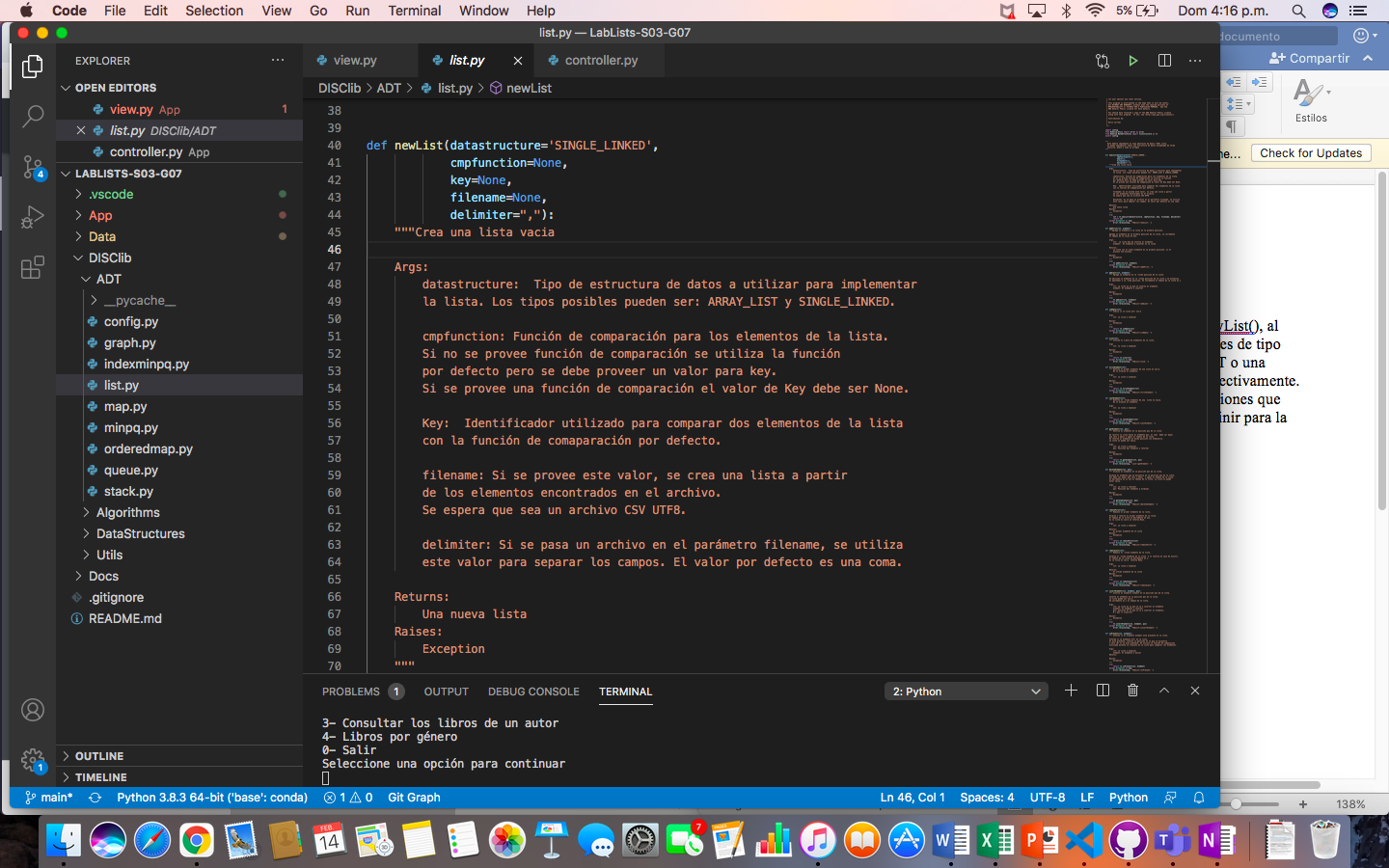


*Imagen 9. Funciones que conectan al model.py con el view.py*

**Paso 4:** Estudiar el uso de listas

1. ¿Cómo se crea una lista?

Para crear una lista con el ATD se debe crear una función utilizando el comando newList(), al cual se le deben agregar cinco parámetros, donde cuatro son opcionales y uno es de tipo obligatorio. En este caso, se pueden crear dos tipos de listas, un ARRAY\_LIST o una SINGLE\_LINKED correspondientes a un arreglo o una lista encadenada, respectivamente. Así mismo, los argumentos de cmpfunction, key, filename y delimiter son opciones que permiten personalizar adecuadamente una lista pero que no es necesarias definirlas para la creación de la lista.



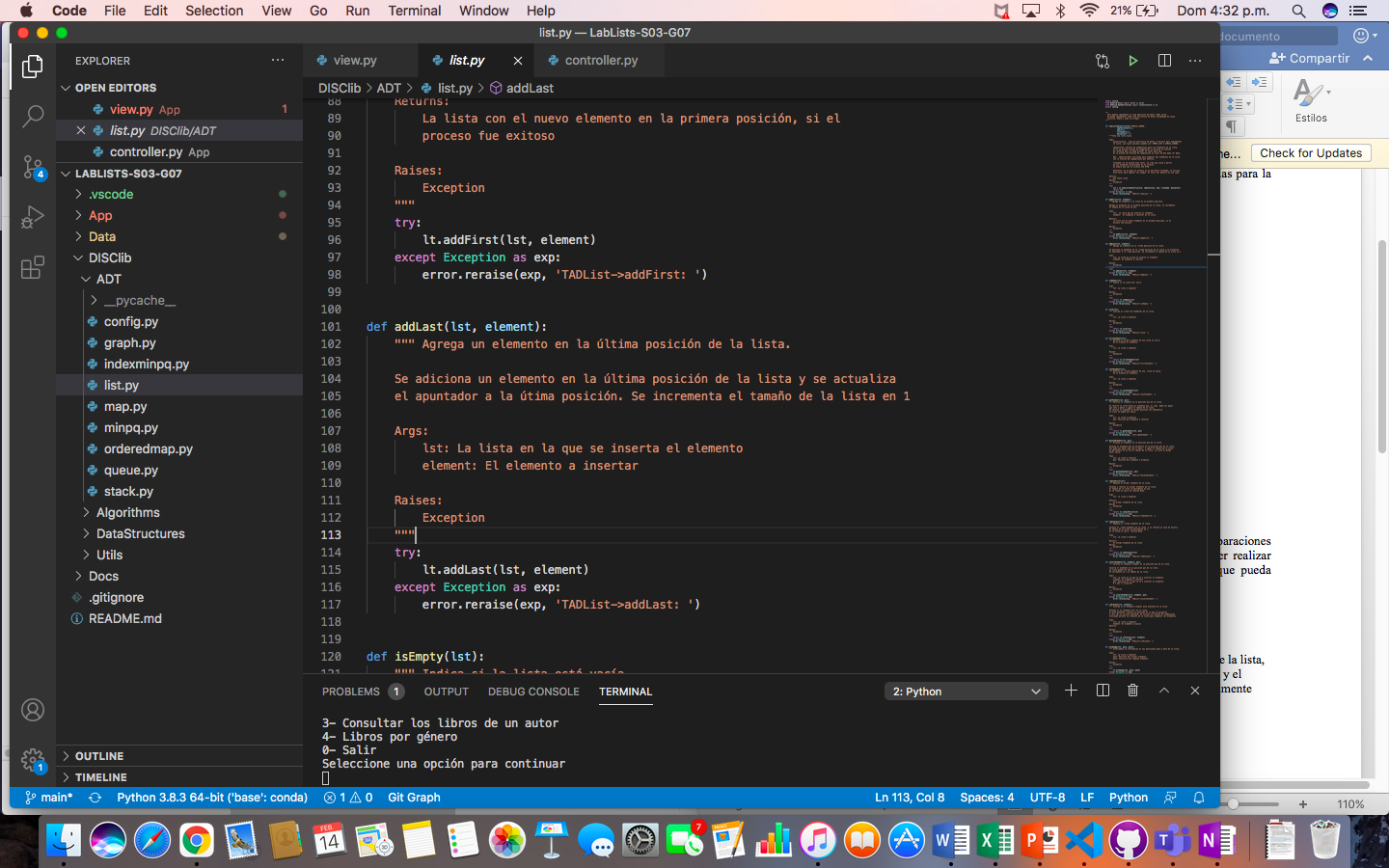
*Imagen 10. newList()*

1. ¿Qué hace el parámetro cmpfunction = None en la función newList()?

Teniendo en cuenta que el parámetro cmpfunction() funciona para realizar comparaciones de tipo (definidas por el usuario) dentro de la lista, si este está en None, False etc. significa que no se va a agregar a la lista.

1. ¿Qué hace la función addLast()?

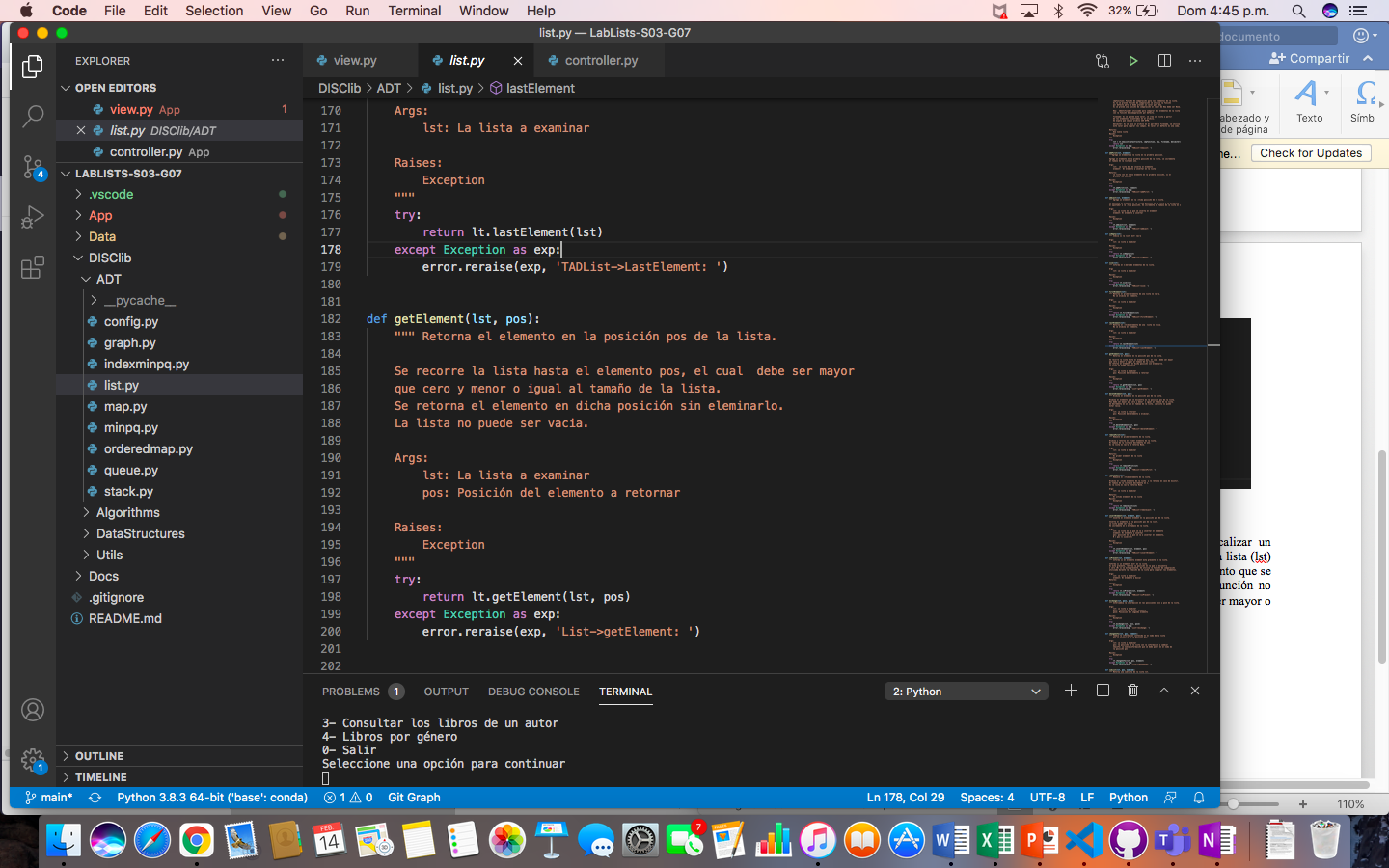
La función addLast () permite agregar elementos en la última posición de la lista, por lo cual, pide como parámetros la lista donde se desea agregar el elemento (lst) y el elemento que se quiere agregar a la lista (element). Igualmente, la lista automáticamente incrementa el size en uno para ajustarlo a la modificación.



*Imagen 11. addLast()*

1. ¿Qué hace la función getElement()?

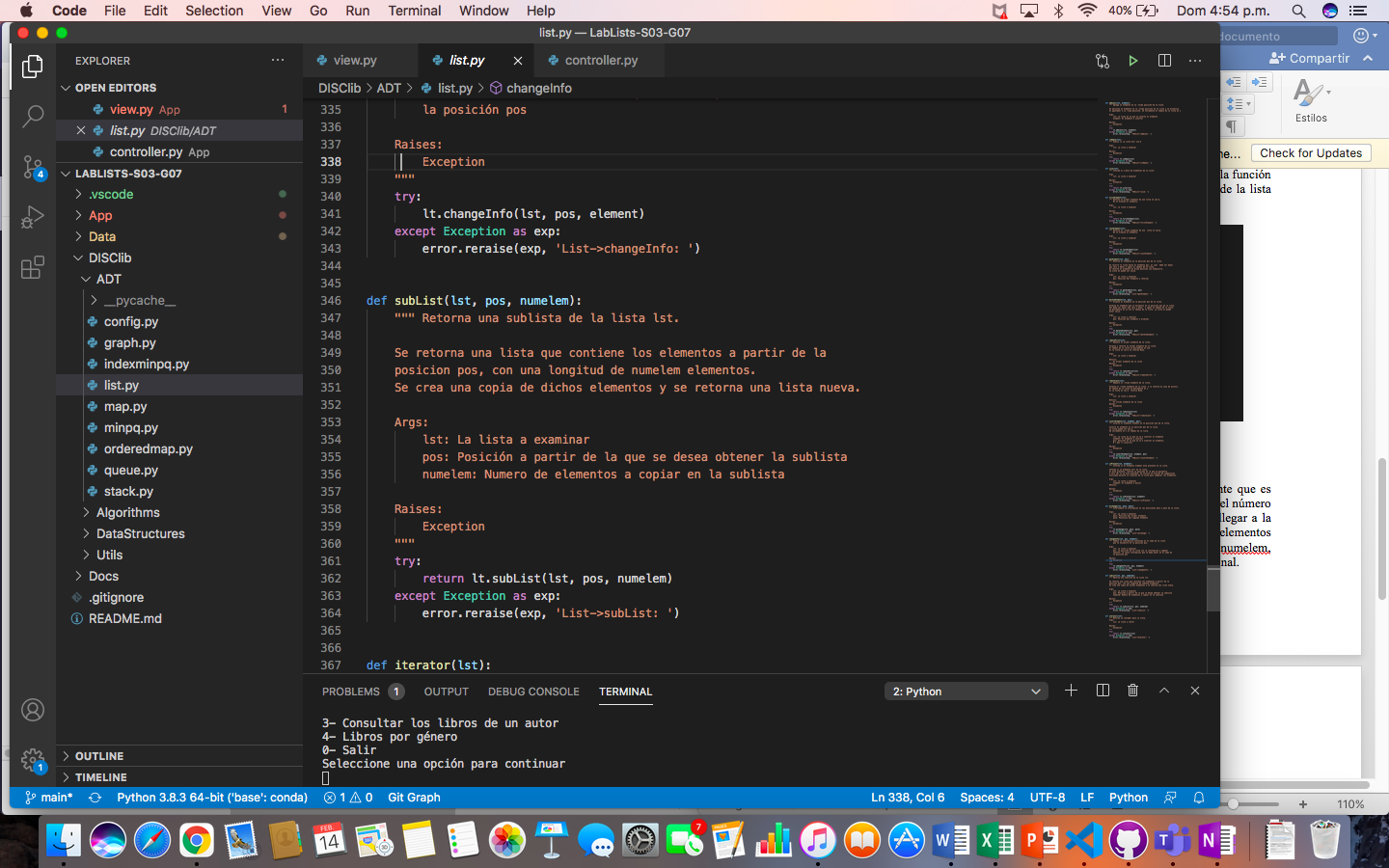
Esta función se encarga de recorrer la lista elemento por elemento para localizar un determinado elemento que se pase por parámetro, por lo cual, la función recibe la lista (lst) de la cual se quiere obtener el elemento y también se recibe la posición del elemento que se desea buscar (pos), el parámetro de posición debe ser mayor o igual a cero y menor o igual al tamaño de la lista. Se pide que la lista no esté vacía porque en ese caso, la función no funcionaría correctamente, así mismo, al obtener el elemento, no lo elimina de la lista principal, simplemente lo muestra.



*Imagen 12. getElement()*

1. ¿Qué hace la función subList()?

El propósito de esta función es crear una nueva lista a partir de una ya existente que es indicada por parámetro, por lo cual, recibe una lista (lst), una posición(pos) y el número de elementos que se desea copiar (numelem). La función recorre la lista hasta llegar a la posición que se indica y cuando llega a esta, inicia una nueva lista con todos los elementos hasta que haya recorrido el número de elementos especificados en el parámetro numelem, cuando finaliza esto crea una copia con todos los elementos en el rango creado y retorna esta nueva lista sin modificar la original.

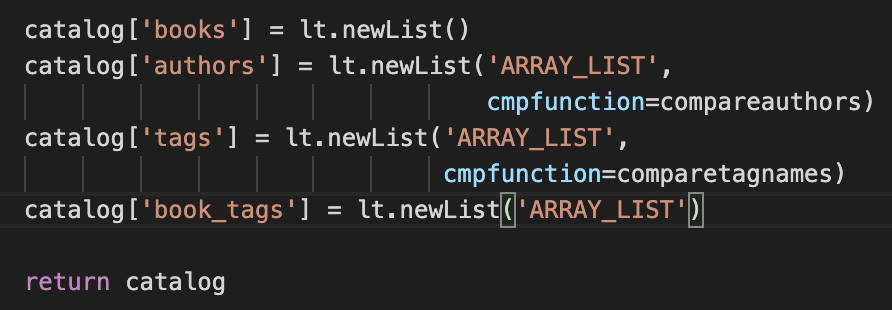


*Imagen 13. subList()*

**Paso 5:** Cambios en el uso de TAD lista (ADT list)

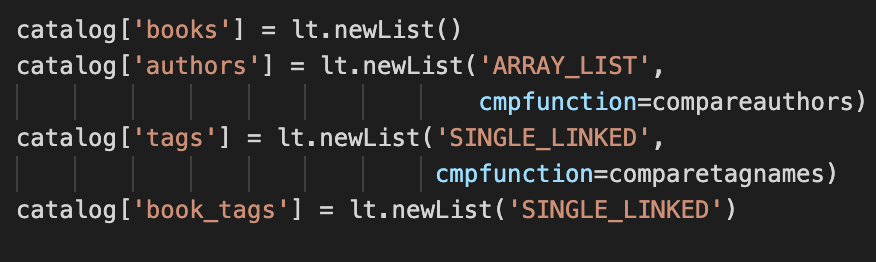
1. ¿Observó algún cambio en el comportamiento del programa al cambiar la implementación del parámetro “ARRAY\_LIST” a “SINGLE\_LINKED”?

Cuando se implementó el programa con ARRAY\_LIST el tiempo transcurrido fue de varios minutos.



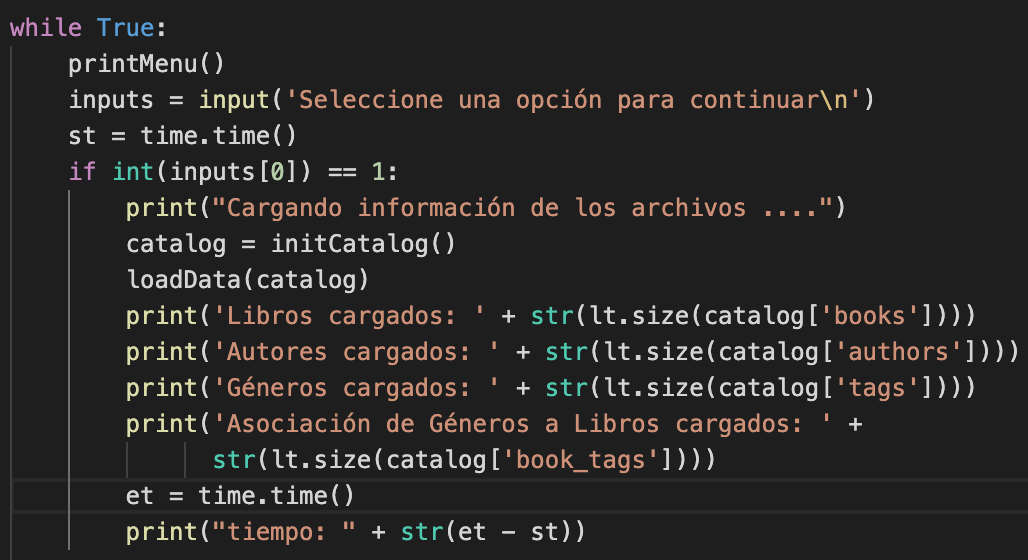
*Imagen 14. Model.py con ARRAY\_LIST*

Sin embargo, cuando se cambió en el model.py a SINGLE\_LINKED las listas asociadas con el manejo de los tags y los booktags, parecía el tiempo de ejecución del programa era más extenso que al usar ARRAY\_LIST.



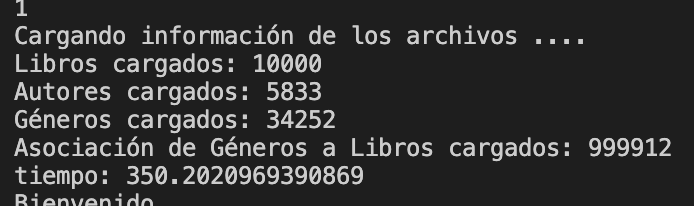
*Imagen 15. Model.py con SINGLE\_LINKED*

Por esta razón, decidimos usar la función time() para verificar nuestra hipótesis de que el cambio a SINGLE\_LINKED hizo que el tiempo de ejecución fuera mayor.

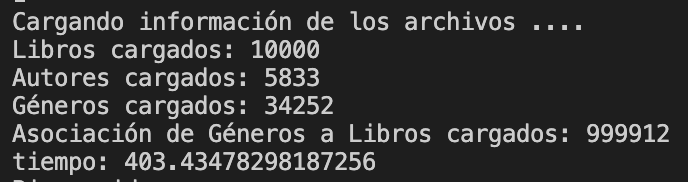


*Imagen 16. Función time()*

Cuando volvimos a ejecutar los programas para ARRAY\_LIST y para SINGLE\_LINKED los tiempos arrojados fueron respectivamente: 350.20 segundos y 403.44 segundos.



*Imagen 17. Tiempo para ARRAY\_LIST*



*Imagen 18. Tiempo para SINGLE\_LINKED*

Como lo sospechábamos, al usar SINGLE\_LINKED o una lista encadenada en vez de un arreglo, pudimos observar que el tiempo de ejecución fue mayor por casi 54 segundos más.