OBSERVACIONES DEL LA PRÁCTICA

Pablo Pedreros Cod 202112491

Laura Guiza Cod 201920926

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

Los mecanismos input son cuando el programa le pide al usuario un número que indica que opción del menú que desea realizar, o cuando le pide información para realizar las funciones, como el número de libros, el nombre del autor o una etiqueta. Por ejemplo, en la función “Consultar los libros de un autor”, en donde el programa inicia solicitando el nombre del autor por input por medio del siguiente código:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Como podemos ver, el archivo view.py convocará una función del archivo controller.py usando como parámetro el input del usuario.

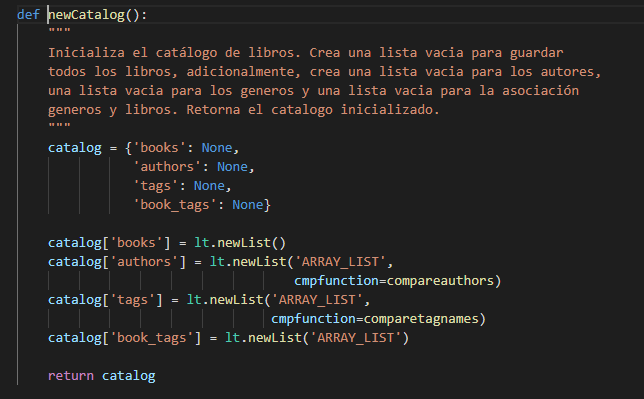
Los mecanismos Output son todas las funciones que imprime la información para el usuario, como los resultados a las funciones o el menú, que tiene el siguiente código:

Texto, Carta

Descripción generada automáticamente

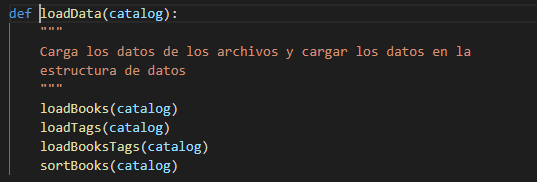
1. ¿Cómo se almacenan los datos de **GoodReads** en el **model.py**?

Cuando uno selecciona la opción de cargar los datos de GoodReads, lo primero que hace el programa es crear un catálogo con la función “newCatalog” desde el archivo model.py:



este catálogo es un diccionario con las llaves “books”, “authors”, “tags” y “book\_tags”, cada una asociada a una lista vacía del tipo arreglo.

Este catálogo vacío se llenará de datos por medio de la función loadData, que llamará cuatro funciones que ayudan a cargar los datos en el diccionario del catálogo:



La función “loadBooks” leerá el archivo .csv de los libros y, para cada libro, lo agregará a la llave “books”, creará una lista arreglo de libros para su autor (si no está creada ya) y agregará el libro a esta lista.

La función “loadTags” leerá el archivo .csv de los tags y creará para cada tag un diccionario con su nombre y su id. Estos diccionarios se almacenarán en una lista asociada a la llave “tag” del diccionario del catálogo.

La función “loadBooksTags” leerá el archivo .csv de los booktags y creará para cada booktag un diccionario con su id y id de su libro . Estos diccionarios se almacenarán en una lista asociada a la llave “book\_tags” del diccionario del catálogo.

Por último, la función sortBooks ordenará los libros basándose en sus calificaciones.

Así pues, los datos de GoodReads quedarán almacenados en un diccionario con cuatro llaves, cada una asociada a una lista arreglo de diccionarios de diferentes tipos de datos. Los diccionarios de libros estarán ordenados por sus calificaciones, y tendrán otras muchas llaves. Los diccionarios de la lista de autores tendrán tres llaves, el nombre del autor, la lista de sus libros y su calificación promedio. Los diccionarios de tags tendrán una llave para su nombre y otra para su id. Por último, los diccionarios de bookTags tendrán una llave para su id y otra para el id de su libro en GoodReads.

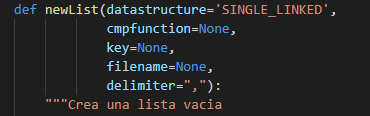
1. ¿Cuáles son las funciones que comunican el el **view.py** y el **model.py**?

Las funciones que comunican el view.py con el model.py son:

* InitCatalog() en donde llama a controller para que inicie el catálogo de los libros, llamando a model.newCatalog() y este posteriormente a model.
* LoadData(catalog) en donde model retorna los libros en la estructura de datos, los cuales fueron fueron solicitados al controlador, el controlador llama al modelo y el modelo le retorna esta información para que posteriormente controlador la procese y envié.
* PrintMenu() en donde si la opción solicitada es la 2, view solicita a controller los mejores libros por promedio según los requisitos solicitados por medio de getBestBooks(catalog, number) en donde controller llama a model para solicitar la información y luego devolverla. Si se seleccionó la opción 3, controller procesa y envia a view los libros de un autor solicitado por getBooksByAuthor (catalog, authorname) después de solicitárselo a model. Y finalmente, si seleccina la opción 4, controller envía los libros por género usando countBooksByTag(catalog, tag) de igual manera. Cabe aclarar que lo envía si la información solicitada es correcta y se encuentra cargada en la base de datos.

1. ¿Cómo se crea una lista?

Inicialmente toca usar la línea de import config as cf, e insertar la línea de importar de la librería, luego crearemos la lista con la función newList que se define en list.py. Esta lista la asignaremos a una variable, por ejemplo: lista=lt.newList(v,w,x,y,z), donde podemos modificar estos 5 argumentos:

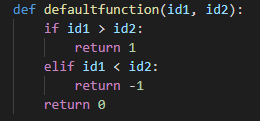


* Datastructure: es el tipo de estrucutra de datos que usaremos en la lista que creamos, puede ser SINGLE\_LINKED o ARRAY\_LIST. Cada una manejará los datos de forma diferente pero ambas corresponden al TAD lista.
* Cmpfunction: es la función de comparación entre los elementos de la lista, si no se llena este parámetro se utilizará la función de comparación por defecto.
* Key: Solo es necesario si se va a usar la función de comparación por defecto, es el identificador que se usaría para comparar los elementos de la lista.
* Filename: si queremos hacer la lista para los elementos de un archivo, ponemos acá el nombre o ruta del archivo y se creará automáticamente la lista con los datos de ese archivo.
* Delimiter: si hacemos la lista sobre un archivo, este será el valor que separe los elementos de la lista. Por defecto es una coma.

NewList llamará a la función newList del arhcivo liststrcuture y, dependiendo de los parámetros, esta llamará la función newList del archivo arraylist.py o singlelinkedlist.py

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

Si el parámetro tiene None como valor, la función que comparará los elementos de la lista será defaultfucntion, que funciona de la siguiente manera:

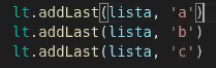


Si se cambiara el valor del parámetro, se podría usar otra función de comparación.

1. ¿Qué hace la función **addLast()**?

La función addLast(), definida en list.py, sirve para agregar un elemento a la última posición de una list. Se usa de la siguiente manera: lt.addLast(lista, x), siendo x el elemento que se quiere agregar y lista la lista a la que se agrega.

Por ejemplo:



En este caso tendríamos que, si lista hubiese estado vacía, ahora tendría [‘a’,’b’,’c’] agregados a la última posición uno después de otro.

La forma en la que se agregan estos elementos dependerá de la estructura de datos que usa la lista.

1. ¿Qué hace la función **getElement()**?

La función getElement() sirve para retornar un elemento en una posición solicitada de la lista. En donde se recorre la lista hasta la posición del elemento a retornar, se retorna el elemento en dicha posición sin eliminarlo. A esta función se le tienen que ingresar dos argumentos, la lista a examinar, que no puede estar vacía, y la posición del elemento a retornar, número que debe ser mayor que cero y menor o igual al tamaño de la lista.

Un ejemplo en el programa sería:



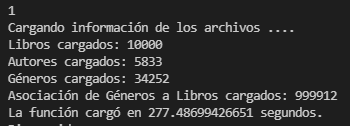
1. ¿Qué hace la función **subList()**

La función subList() retorna una lista nueva a partir de la lista indicada a examinar. Esta lista o sublista es la copia de un número ingresado de elementos de la lista a partir de una posición ingresada.

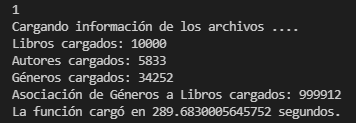
Los parámetros a ingresar son 3: (x, y, z) siendo x la lista a examinar; y, la posición a partir de la que se desea obtener la sublista y z el número de elementos a copiar en la sublista.

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

El cambio más importante de una estructura a otra es el espacio empleado en memoria y el tiempo de ejecución. Con la estructura ARRAY\_LIST, medimos que la función 1 carga los datos en el siguiente tiempo:



La misma función con el mismo archivo, pero usando SINGLE\_LINKED para los tags y book\_tags tardó el siguiente tiempo:



Luego de medirlo varias veces, el programa usando SINGLE\_LINKED list se demoró aproximadamente 10 o 12 segundos más que con ARRAY\_LIST, pues el recorrido por un array siempre será más rápido.

El segundo cambio es que el espacio que ocupa en memoria es un poco más alto con el SINGLE\_LINKED list, pues cada elemento tiene que almacenar también una referencia al siguiente elemento.