OBSERVACIONES DEL LA PRACTICA

Estudiante 1 Cod XXXX

Estudiante 2 Cod XXXX

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

RTA/ Para poder dar respuesta a esta pregunta consideramos pertinente la división de la ejecución de este programa en tres etapas que se repiten de manera cíclica tantas veces como el usuario desee:

* Primer Output: En esta sección el programa se inicializa:
* while True:
* printMenu()

Tan pronto como esto sucede, y como se puede ver en el código enseñado anteriormente, se ejecuta la funcion x, la cual se encarga de, como su nombre lo indica, mostrarle al usuario el menú con las diversas opciones que tiene en el programa, realizando el Output de la siguiente manera:

def printMenu():

    print("Bienvenido")

    print("1- Cargar información en el catálogo")

    print("2- Consultar los Top x libros por promedio")

    print("3- Consultar los libros de un autor")

    print("4- Libros por género")

    print("0- Salir")

* Input: Esta sección del código, a pesar de ser la más corta, tiene el crucial papel de recoger la información del usuario para saber que funciones ejecutar, que información se está solicitando, o en el mejor de los casos, parar la ejecución del programa. La recolección de dicha información la realiza la siguiente sección del código:
* inputs = input('Seleccione una opción para continuar\n')
* Segundo Output: Esta es la parte más elaborada del código, pues una vez se conoce la opción elegida por el usuario, inicia la ejecución de un condicional que guiará al programa por las diferentes funciones e imprimirá la información del Output final, el condicional es el siguiente:
* if int(inputs[0]) == 1:
* print("Cargando información de los archivos ....")
* catalog = initCatalog()
* loadData(catalog)
* print('Libros cargados: ' + str(lt.size(catalog['books'])))
* print('Autores cargados: ' + str(lt.size(catalog['authors'])))
* print('Géneros cargados: ' + str(lt.size(catalog['tags'])))
* print('Asociación de Géneros a Libros cargados: ' +
* str(lt.size(catalog['book\_tags'])))
* elif int(inputs[0]) == 2:
* number = input("Buscando los TOP ?: ")
* books = controller.getBestBooks(catalog, int(number))
* printBestBooks(books)
* elif int(inputs[0]) == 3:
* authorname = input("Nombre del autor a buscar: ")
* author = controller.getBooksByAuthor(catalog, authorname)
* printAuthorData(author)
* elif int(inputs[0]) == 4:
* label = input("Etiqueta a buscar: ")
* book\_count = controller.countBooksByTag(catalog, label)
* print('Se encontraron: ', book\_count, ' Libros')
* else:
* sys.exit(0)

Como se puede observar, cada una de las opciones tiene 3 elementos clave:

(opcional)Input Adicional: Si la opción elegida requiere de más información que la ya solicitada, esta recolección de datos será lo primero que hará el programa.

Ejecución de Funciones: Cada una de las funciones necesitará realizar procesos internos para poder recoger la información que se quiere mostrar, y dichos procesos no se incluyen en el código del menú principal, por lo que se especifica su ejecución externa.

Impresión final de la información: una vez ejecutados los procesos necesarios, se enseña al usuario la información solicitada, ya bien sea con un “print” directo dentro del menú principal, o bien con una función adicional que se encargue de imprimir la información.

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

RTA/ Para el almacenamiento de datos, lo primero que se debe tener en cuenta es el diccionario principal en el cual se almacenan todas las estructuras de datos que albergan la información. Este diccionario se define en el código de la siguiente manera:

catalog = {'books': None,

               'authors': None,

               'tags': None,

               'book\_tags': None}

Una vez definido el diccionario, se procede a especificar las estructuras de datos que van a corresponder a cada una de las llaves, con el código que se muestra a continuación:

    catalog['books'] = lt.newList()

    catalog['authors'] = lt.newList('ARRAY\_LIST',

                                    cmpfunction=compareauthors)

    catalog['tags'] = lt.newList('ARRAY\_LIST',

                                 cmpfunction=comparetagnames)

    catalog['book\_tags'] = lt.newList('ARRAY\_LIST')

Adicionalmente, cabe resaltar que, para las categorías de autores, tags y book\_tags, se especifica qué tipo de lista va a ser, haciéndole explícito que se van a utilizar arreglos para cada uno de estos casos.

Posteriormente, y una vez creadas las estructuras de datos necesarias, se especifica una función cuya labor va a ser precisamente llenar este diccionario, de la siguiente manera:

def addBook(catalog, book):

Arranca adicionando el libro a la lista correspondiente:

lt.addLast(catalog['books'], book)

Luego, identifica y separa los autores de la obra:

authors = book['authors'].split(",")

Después, se crea un libro en la lista de libros de dichos autores:

for author in authors:

        addBookAuthor(catalog, author.strip(), book)

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

RTA/ Para poder dar respuesta a esta pregunta, se hizo una búsqueda en el view.py de todas las menciones a funciones del controlador, y se encontraron las siguientes 5 menciones:

    return controller.initCatalog() 🡪 Renglón 51

    controller.loadData(catalog) 🡪 Renglón 58

    books = controller.getBestBooks(catalog, int(number)) 🡪 Renglón 102

author = controller.getBooksByAuthor(catalog, authorname) 🡪 Renglón 107

book\_count = controller.countBooksByTag(catalog, label) 🡪 Renglón 112

Posteriormente, se hizo el análisis de cada una de las funciones, para saber si estas a su vez usaban funciones del model.py, y así poder asegurar que efectivamente son funciones de conexión, y para cuatro ellas efectivamente había un explícito llamado al model, como se puede observar a continuación (funciones recortadas a conveniencia para mostrar solo lo mencionado):

def initCatalog():

    catalog = model.newCatalog()

def getBestBooks(catalog, number):

    bestbooks = model.getBestBooks(catalog, number)

def getBooksByAuthor(catalog, authorname):

    author = model.getBooksByAuthor(catalog, authorname)

def countBooksByTag(catalog, tag):

    return model.countBooksByTag(catalog, tag)

Por otra parte, para la función loadData(), si bien esta no tiene ninguna mención directa al modelo, si ejecuta subfunciones que tienen injerencia en el modelo, por lo que a pesar de no citarse anteriormente, se considera que, junto con las otras 4 funciones son las 5 funciones encargadas de conectar el view.py y el model.py.

1. ¿Cómo se crea una lista?

Para crear una lista, tomando en cuenta el código de list.py, solo se necesita ejecutar la función newList(), que se presenta a continuación:

def newList(datastructure='SINGLE\_LINKED', cmpfunction=None, key=None,

            filename=None, delimiter=","):

    try:

        lst = lt.newList(datastructure, cmpfunction, key, filename, delimiter)

        return lst

    except Exception as exp:

        error.reraise(exp, 'TADList->newList: ')

En esta función para poder crear la lista se necesita especificar qué tipo de estructura de datos queremos, así como los métodos de comparación de datos para procesos posteriores de tratamiento de datos (para ello están tanto “cmpfunction” como “key”), el nombre de un archivo y su forma elemento que ayudara a la separación de las trazas de datos. Así mismo, a partir de la utilización de un “try” y un “exception”, se asegura que la creación de esta lista de manera errónea no detenga la ejecución del programa. Finalmente, cabe mencionar que esta función solo crea la lista a partir de la mención de otra función en otro documento.

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

RTA/ La funcionalidad de este parámetro es especificar un método de comparación de datos para tratar la información de la lista: En caso de usarse la función y colocar un parámetro que señale un método de comparación, este será el que se use. En caso contrario, la igualación a None le permite al programa ejecutar una función por defecto y seguir con el análisis de la información.

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

RTA/ La función addLast(), como su nombre lo indica, añade un elemento a la lista al final de la misma, partir del siguiente código:

def addLast(lst, element):

    try:

        lt.addLast(lst, element)

    except Exception as exp:

        error.reraise(exp, 'TADList->addLast: ')

En este fragmento de código se puede observar cómo la función recibe 2 parámetros: La lista en su estado inicial y el elemento que se quiere agregar. Posteriormente ejecuta un un “try” y un “exception” para evitar el colapso del programa, y se puede observar como el programa acude a una función auxiliar de otro programa para poder llevar a cabo la tarea principal.

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

RTA/ La función getElement(), como su nombre lo indica, sirve para capturar un elemento de una posición específica sin eliminarlo de la lista, con el código que se observa a continuación:

def getElement(lst, pos):

    try:

        return lt.getElement(lst, pos)

    except Exception as exp:

        error.reraise(exp, 'List->getElement: ')

Como se puede evidenciar, la función recibe como parámetros tanto la lista como la posición que se desea consultar, posteriormente recorre la lista hasta la posición determinada(siempre que esta sea menor que el tamaño de la lista) y devuelve dicho valor. Para realizar lo anterior, al igual que otras funciones se apoya tanto en el “try, exception”, como en el uso de funciones auxiliares de otros programas que lleven a cabo la tarea.

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

RTA/ La función Sublist(), como su nombre lo indica, crea una sub-lista a partir de la lista original que encierra los datos que se le piden por parámetro, a partir del siguiente código:

def subList(lst, pos, numelem):

    try:

        return lt.subList(lst, pos, numelem)

    except Exception as exp:

        error.reraise(exp, 'List->subList: ')

En consecuencia, este código tiene 3 parámetros: La lista, la posición a partir de la cual quiero empezar a hacer la recolección de datos, y la cantidad de datos que se desea tenga la sub-lista. Finalmente, al igual que en los casos anteriores, la función se apoya tanto en un “try, exception”, como en el uso de funciones auxiliares de otros programas que lleven a cabo la tarea.

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

RTA/ Para poder dar respuesta a este punto, se hizo necesaria la implementación de una librería que permitiese la medición del tiempo de ejecución de la función encargada de armar la estructura de datos, por lo que cito la librería “time” de la siguiente manera:

from time import process\_time

Posteriormente, se utilizó de manera conveniente antes y después de la ejecución de la funciones disponibles en el menu del view.py, de manera que se permitiese tomar mediciones del tiempo en dichos momentos, de la siguiente forma:

t1 = process\_time()

loadData(catalog)

t2 = process\_time()

Para finalmente imprimir la resta de dichos tiempos, obteniendo la siguiente información en cada uno de los casos(NOTA: Para mayor exactitud de las mediciones, y siguiendo las recomendaciones dadas en clase, se tuvo a la maquina de prueba exclusivamente concentrada en Visual Studio Code).

Maquina de prueba:

*Nombre del dispositivo HP-DESKTOP-R1CO9SE*

*Procesador: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz*

*RAM instalada: 8,00 GB (7,88 GB usable)*

*Identificador de dispositivo: AFBE0A90-16F9-4F45-9A75-9B2548EAFC28*

*Id. del producto: 00327-70000-00001-AA981*

*Tipo de sistema: Sistema operativo de 64 bits, procesador basado en x64*

* 1. Para el caso puntual de la función que carga la información en el catálogo, los resultados fueron los siguientes:

**ARRAY\_LIST**

Cargando información de los archivos ....

Libros cargados: 149

Autores cargados: 156

Géneros cargados: 34252

Asociación de Géneros a Libros cargados: 999

Time = 0.15625 seg

**SINGLE\_LINKED**

Cargando información de los archivos ....

Libros cargados: 149

Autores cargados: 156

Géneros cargados: 34252

Asociación de Géneros a Libros cargados: 999

Time = 0.1875 seg

* 1. Para el caso puntual de la función que arroja el Top X de libros organizados por promedio, el resultado fue el siguiente:

**ARRAY\_LIST:**

Buscando los TOP ?: 99

Time = 0.015625seg

**SINGLE\_LINKED:**

Buscando los TOP ?: 99

Time = 0.0seg

* 1. Para el caso puntual de la función que consulta los libros de un determinado autor, el resultado fue el siguiente:

**ARRAY\_LIST:**

Nombre del autor a buscar: J.K. Rowling

Autor encontrado: J.K. Rowling

Promedio: 0

Total de libros: 7

Time = 0.0seg

**SINGLE\_LINKED:**

Nombre del autor a buscar: J.K. Rowling

Autor encontrado: J.K. Rowling

Promedio: 0

Total de libros: 7

Time = 0.0seg

Así las cosas, se puede observar que para el usuario la principal diferencia entre cada una de las estructuras de datos es la cantidad de tiempo que se tiene que esperar para su formación y organización. Sin embargo, cabe señalar que estos ejemplo no marcan una pauta para una tendencia homogenea en todos los casos, por lo que se puede observar que hay funciones en las que alguna de las estructuras de datos responde mejor que otra, así como hubo una función donde no hubo diferencia. En conclusión, se puede decir que dependiendo de lo que se quiera hacer con la información se debe escoger uno u otro métodos de organización y almacenamiento de los datos.