Guía de Ejercicios Prácticos 23

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos 23
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:32

Descripción

Esta guía contiene enunciados de algunos ejercicios para aplicar los conceptos de programación en ***Python*** que se analizan en la ***Ficha 23***. Los alumnos no deben subir nada al aula virtual: la guía se propone como fuente de ejercicios generales. Se sugiere intentar resolver cada uno de estos problemas ya sea trabajando solos o en grupos de estudio, y cuando las soluciones se publiquen, controlar lo hecho con las sugerencias propuestas por sus docentes. Utilice el foro del curso para plantear dudas y consultas, que cualquier alumno puede intentar responder.

Tabla de contenidos

1. Gimnasio

Un gimnasio necesita un programa que gestione a través de un menú de opciones las altas bajas y modificaciones de los socios que asisten al gimnasio, en un archivo llamado “socios.dat”.

De los socios se conocen su número (positivo menor a 99999, validar), nombre (un máximo de 40 caracteres), plan (0: Completo / 1: Gimnasio / 2:Natatorio / 3: Pilates) y monto mensual que se abona.

Se pide:

1. Realizar el alta de un nuevo socio, cargando todos los datos
2. Modificar un usuario, actualizando su plan y el monto mensual que abona
3. Realizar la baja lógica de un socio
4. Mostrar el contenido del archivo
5. Determinar el monto total que se ha cobrado por cada tipo de plan
6. Realizar la compactación del archivo, eliminando físicamente del archivo los socios dados de baja

1.1. principal.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

from socio import *
import os.path
import pickle
import io

def buscar(fd, m, numero):
    t = os.path.getsize(fd)

    fp_inicial = m.tell()
    m.seek(0, io.SEEK_SET)

    posicion = -1
    while m.tell() < t:
        fp = m.tell()
        soc = pickle.load(m)
        if soc.activo and soc.numero == numero:
            posicion = fp
            break

    m.seek(fp_inicial, io.SEEK_SET)
    return posicion


def validar_entre(desde,hasta,mensaje):
    num = int(input(mensaje))
    while num < desde or num > hasta:
        print('Inválido! Debe ser un valor entre',desde,'y',hasta)
        num = int(input(mensaje))
    return num


def validar_monto(mensaje):
    monto = float(input(mensaje))
    while monto < 0:
        print('Inválido! Debe ser un valor positivo.')
        monto = float(input(mensaje))
    return monto


def alta(fd):
    m = open(fd, 'a+b')
    print()
    numero = validar_entre(0, 99999,'Ingrese número de socio: ')
    pos = buscar(fd, m, numero)
    if pos == -1:
        # no estaba repetido... lo cargamos por teclado...
        nombre = input('Ingrese nombre: ')
        # ...ajustamos a 30 caracteres, rellenando con blancos al final...
        nombre = nombre.ljust(40, ' ')
        plan = validar_entre(0,3,'Ingrese plan: ')
        monto = validar_monto('Ingrese cuota mensual: ')
        reg = Socio(numero,nombre,plan,monto)
        # ...lo grabamos...
        pickle.dump(reg, m)
        # ...volcamos el buffer de escritura
        # para que el sistema operativo REALMENTE
        # grabe en disco el registro...
        m.flush()
        print('Registro grabado en el archivo...')
    else:
        print('Número repetido... alta rechazada...')
    m.close()


def modificacion(fd):
```

```
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    m = open(fd, 'r+b')

    print()
    numero = validar_entre(0, 99999,'Ingrese número de socio a eliminar: ')
    pos = buscar(fd, m, numero)
    if pos != -1:
        # encontrado... procedemos a cargarlo...
        m.seek(pos, io.SEEK_SET)
        reg = pickle.load(m)

        # ...mostramos el registro tal como estaba...
        print()
        print('El registro actualmente grabado es:')
        print(to_string(reg))

        plan = validar_entre(0,3,'Ingrese plan:')
        monto = validar_monto('Ingrese cuota mensual: ')

        # ...registro modificado en memoria...
        # ...ahora nos volvemos a su posición en el archivo...
        m.seek(pos, io.SEEK_SET)

        # ...y volvemos a grabar el registro modificado...
        pickle.dump(reg, m)

        print()
        print('Los datos del registro se actualizaron en el archivo...')

    else:
        print('Ese registro no existe en el archivo...')

    m.close()


def baja(fd):

    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    m = open(fd, 'r+b')

    print()
    numero = validar_entre(0, 99999,'Ingrese número de socio: ')
    pos = buscar(fd, m, numero)
    if pos != -1:
        # encontrado... procedemos a cargarlo...
        m.seek(pos, io.SEEK_SET)
        reg = pickle.load(m)

        # ...mostramos el registro tal como estaba...
        print()
        print('El registro actualmente grabado es:')
        print(to_string(reg))

        # ...chequemos si el usario está seguro de lo que hace...
        r = input('Está seguro de querer borrarlo (s/n)?: ')
        if r in ['s', 'S']:
            # lo marcamos como borrado, y ya...
            reg.activo = False
```

```
            # ...reubicamos el file pointer...
            m.seek(pos, io.SEEK_SET)

            # ...y lo volvemos a grabar...
            pickle.dump(reg,m)

            print()
            print('Registro eliminado del archivo...')

    else:
        print('Ese registro no existe en el archivo...')

    m.close()


def listado_completo(fd):
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    tbm = os.path.getsize(fd)

    m = open(fd, 'rb')

    print('Listado General de Socios')
    while m.tell() < tbm:
        reg = pickle.load(m)
        if reg.activo:
            print(to_string(reg))

    m.close()


def sumar_por_plan(fd):
    va = [0] * 4
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    tbm = os.path.getsize(fd)

    m = open(fd, 'rb')

    while m.tell() < tbm:
        reg = pickle.load(m)
        if reg.activo:
            va[reg.plan] += reg.monto

    m.close()
    return va


def depuracion(fd):
    """Optimiza el espacio físico del archivo.

    La función aplica un proceso de baja física generalizada: todos los registros que estaban
    marcados en forma lógica como eliminados, son físicamente eliminados del archivo.
    """
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    tbm = os.path.getsize(fd)
```

```
    original = open(fd, 'rb')
    temporal = open('temporal.dat', 'wb')

    print('Procediendo a optimizar el archivo', fd, '(eliminación física de registros borrados)')
    while original.tell() < tbm:
        # cargar un registro del archivo original...
        reg = pickle.load(original)

        # ...y si no estaba marcado como eliminado, grabarlo en el archivo temporal...
        if reg.activo:
            pickle.dump(reg, temporal)

    # cerrar ambos archivos...
    original.close()
    temporal.close()

    # eliminar el archivo original...
    os.remove(fd)

    # y renombrar el temporal...
    os.rename('temporal.dat', fd)


def main():
    opcion = -1
    fd = 'socios.dat'
    while opcion != 0:
        print('\nMenu de Opciones')
        print('=' * 80)
        print('1\t Alta de Socio')
        print('2\t Modificacion de Socio')
        print('3\t Baja de Socio')
        print('4\t Listado de Socios')
        print('5\t Monto Total por Plan')
        print('6\t Compactar Archivo')
        print('0\t Salir')
        opcion = int(input('Ingrese la opcion a ejecutar: '))
        if opcion == 1:
            alta(fd)
        elif opcion == 2:
            modificacion(fd)
        elif opcion == 3:
            baja(fd)
        elif opcion == 4:
            listado_completo(fd)
        elif opcion == 5:
            v = sumar_por_plan(fd)
            for i in range(len(v)):
                print('Plan',i,'Monto $',v[i])
        elif opcion == 5:
            depuracion(fd)
        print('=' * 80)


if __name__ == '__main__':
    main()
```

1.2. socio.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

class Socio:

    def __init__(self,numero, nombre, plan, monto):
        self.numero = numero
        self.nombre = nombre
        self.plan = plan
        self.monto = monto
        self.activo = True


def to_string(socio):
    txt = '{:<10}'.format(socio.numero)
    txt += '{:<45}'.format(socio.nombre)
    txt += '{:<5}'.format(socio.plan)
    txt += '${:10,.2f}'.format(socio.monto)
    return txt
```

2. Concesionaria

Desarrollar un programa controlado por menú de opciones, que permita realizar en forma completa la gestión de un archivo de registros de automoviles de una concesionaria. Por cada auto, prevea tres campos para la patente, el estado (0-vendido, 1-disponible) y el modelo (que es el año de fabricación).

El programa debe incluir opciones que permitan:

1. Realizar altas de registros en el archivo.
2. Realizar ventas de un automóvil con patente x, ingresada por teclado.
3. Mostrar el contenido completo del archivo.
4. Mostrar los datos de los autos disponibles con modelo mayor m, ingresado por teclado.

2.1. auto.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

class Auto:
    def __init__(self,patente, modelo):
        self.patente = patente
        self.modelo = modelo
        self.estado = 1

def to_string(aut):
    print('Patente:', aut.patente, end=' ')
    print('Modelo:', aut.modelo, end=' ')
    print('Estado:', aut.estado)
```

2.2. concesionario.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

from auto import *
import io
import os
import pickle
import os.path

def buscar(fd, m, p):
    """Busca un registro cuya patente coincida con p en el archivo de automoviles.

    Si lo encuentra, retorna su posición (como número de byte). Si no lo encuentra, retorna
    el valor -1.

    :param fd: El file descriptor del archivo donde se debe buscar.
    :param m: El manejador del archivo donde se debe buscar.
    :param p: La patente a buscar.
    :return la posición de byte donde fue encontrado el registro.
    """
    t = os.path.getsize(fd)

    fp_inicial = m.tell()
    m.seek(0, io.SEEK_SET)

    posicion = -1
    while m.tell() < t:
        fp = m.tell()
        aut = pickle.load(m)
        if aut.patente == p:
            posicion = fp
            break

    m.seek(fp_inicial, io.SEEK_SET)
    return posicion


def alta(fd):
    """Agrega el registro de un automovil al archivo.

    Los datos del automovil se toman por teclado. El registro será grabado sólo si el
    archivo no contenía ya un automovil con la misma patente.

    :param fd: El file descriptor del archivo donde se debe buscar.
    """
    m = open(fd, 'a+b')

    patente = input('Patente del auto a registrar (cargue 0 para salir): ')
    while patente != '0':
        # buscamos el registro con esa patente...
        pos = buscar(fd, m, patente)

        if pos == -1:
            # no estaba repetido... lo cargamos por teclado...
            modelo = int(input('Modelo: '))
            aut = Auto(patente, modelo)

            # ...lo grabamos...
            pickle.dump(aut, m)

            # ...y volcamos el buffer de escritura
            # para que el sistema operativo REALMENTE
            # grabe en disco el registro...
            m.flush()

            print('Registro grabado en el archivo...')

        else:
```

```
            print('Patente repetida... alta rechazada...')

        patente = input('Otra patente (cargue 0 para salir): ')

    print()
    print('Los nuevos registros han sido grabados...')
    input('Presione  para seguir...')
    m.close()


def modificacion(fd):
    """Modifica el contenido de un registro de un automovil del archivo.

    Solo se permitira modificar el estado de un automovil a no dispobible.
    :param fd: El file descriptor del archivo donde se debe buscar.
    """
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    m = open(fd, 'r+b')

    patente = input('Patente del automovil a modificar su estado (cargue 0 para salir): ')
    while patente != '0':
        # buscamos el registro con esa patente...
        pos = buscar(fd, m, patente)

        if pos != -1:
            # encontrado... procedemos a cargarlo...
            m.seek(pos, io.SEEK_SET)
            aut = pickle.load(m)

            # ...mostramos el registro tal como estaba...
            print()
            print('El registro actualmente grabado es:')
            to_string(aut)

            if aut.estado == 0:
                print('El automovil ya fue VENDIDO')
            else:
                aut.estado = 0

                # ...registro modificado en memoria...
                # ...ahora nos volvemos a su posición en el archivo...
                m.seek(pos, io.SEEK_SET)

                # ...y volvemos a grabar el registro modificado...
                pickle.dump(aut, m)

                print()
                print('El automovil cambio su estado a VENDIDO...')

        else:
            print('Ese registro no existe en el archivo...')

        input('Presione  para seguir...')
        patente = input('Otra patente a modificar su estado (cargue 0 para salir): ')

    m.close()

def listado_completo(fd):
    """Muestra el contenido completo del archivo.

    :param fd: El file descriptor del archivo donde se debe buscar.
    """
    if not os.path.exists(fd):
```

```
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    tbm = os.path.getsize(fd)

    m = open(fd, 'rb')

    print('Listado general de automoviles registrados:')
    while m.tell() < tbm:
        aut = pickle.load(m)
        to_string(aut)

    m.close()

    print()
    input('Presione  para seguir...')


def listado_filtrado(fd):
    """Muestra los registros de los auto con modelo mayor a m.

    :param fd: El file descriptor del archivo donde se debe buscar.
    """
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return

    tbm = os.path.getsize(fd)

    m = open(fd, 'rb')

    modelo = int(input('Modelo de automovil para filtrar: '))

    print('Listado de automoviles disponibles con modelo mayor a ' + str(modelo))
    while m.tell() < tbm:
        aut = pickle.load(m)
        if aut.estado == 1 and aut.modelo > modelo:
            to_string(aut)

    m.close()

    print()
    input('Presione  para seguir...')


def main():
    fd = 'automoviles.aut'

    op = 0
    while op != 5:
        print('Opciones del archivo de automoviles')
        print('   1. Alta de automoviles')
        print('   2. Modificación de estado de un automovil')
        print('   3. Listado completo de automoviles')
        print('   4. Listado de automoviles disponibles con modelo mayor a m')
        print('   5. Salir')
        op = int(input('\t\tIngrese número de la opción elegida: '))
        print()

        if op == 1:
            alta(fd)

        elif op == 2:
            modificacion(fd)
```

```
        elif op == 3:
            listado_completo(fd)

        elif op == 4:
            listado_filtrado(fd)

        elif op == 5:
            pass


# script principal...
if __name__ == '__main__':
    main()
```

3. Gestor de discos

Una radio solicita un programa que le permita mantener un seguimiento de las reproducciones de los discos que tiene la misma. Para ello solicita un programa que:

Cargue los datos de los discos de un archivo de texto a un vector:

El formato del archivo es el siguiente:

La primera línea contendrá solo el número de discos que hay en el archivo, y seguido están los datos de cada uno de los discos en líneas sucesivas. De cada disco se conoce: Título, Artista, Año de publicación, género (0-20) y cantidad de reproducciones

Una vez cargados los discos presente un menú con opciones para:

- Determinar cual fue el disco con mas reproducciones.

- Determinar la cantidad de discos de cada género

- Actualizar la cantidad de reproducciones de un disco con título x, agregando la cantidad y siendo x e y ambos ingresados por teclado

- Agregar un nuevo disco

- Eliminar un disco con título x y artista y (siendo ambos ingresados por teclado)

- Al salir, actualizar el archivo con los datos del vector.

3.1. discos.py

```
class Disco:
    def __init__(self, titulo, artista, anio, genero, reproducciones):
        self.titulo = titulo
        self.artista = artista
        self.anio = anio
        self.genero = genero
        self.reproducciones = reproducciones


def write(disco):
    print("Disco - Titulo:{} Artista:{} Año:{} Genero:{} Reproducciones:{}"
          .format(disco.titulo, disco.artista, disco.anio,
                  disco.genero, disco.reproducciones)
          )


def cargar_disco():
    titulo = input("Ingrese titulo: ")
    artista = input("Ingrese artista: ")
    anio = int(input("Ingrese año: "))
    genero = int(input("Ingrese género: "))
    reproducciones = int(input("Ingrese reproducciones: "))
    return Disco(titulo, artista, anio, genero, reproducciones)
```

3.2. gestor.py

```
import discos


def mostrar_vector(vec):
    for x in vec:
        discos.write(x)


def cargar_discos():
    file = open("datos.txt", "rt")
    n = int(file.readline())
    v = [None] * n
    for i in range(n):
        titulo = file.readline().strip()
        artista = file.readline().strip()
        anio = int(file.readline())
        genero = int(file.readline())
        reproducciones = int(file.readline())
        v[i] = discos.Disco(titulo, artista, anio, genero, reproducciones)
    file.close()
    return v


def menu():
    print("1 _ Mas reproducciones")
    print("2 _ Cantidad por genero")
    print("3 _ Actualizar reproducciones")
    print("4 _ Agregar disco")
    print("5 _ Eliminar disco")
    print("6 _ Salir")
    return int(input("Ingrese opción: "))


def agregar_disco(v):
    print("Ingrese datos del disco: ")
    disco = discos.cargar_disco()
    v.append(disco)


def grabar_discos(v):
    file = open("datos.txt", "wt")
    n = len(v)
    file.write(str(n) + "\n")
    for disco in v:
        file.write(disco.titulo + "\n")
        file.write(disco.artista + "\n")
        file.write(str(disco.anio) + "\n")
        file.write(str(disco.genero) + "\n")
        file.write(str(disco.reproducciones) + "\n")
    file.close()


def eliminar_disco(v, x, y):
    for i in range(len(v)):
        if v[i].titulo == x and v[i].artista == y:
            del v[i]
            break


def disco_mas_reproducido(v):
    mayor = None
    for i in range(len(v)):
        if i == 0 or v[i].reproducciones > mayor.reproducciones:
            mayor = v[i]
    return mayor
```

```
def cantidad_por_genero(v):
    cont = [0] * 21
    for disco in v:
        cont[disco.genero] += 1
    return cont


def incrementar_reproducciones(v, x, cant):
    for disco in v:
        if disco.titulo == x:
            disco.reproducciones += cant
            break


def main():
    v = cargar_discos()
    mostrar_vector(v)
    op = 0
    while op != 6:
        op = menu()
        if op == 1:
            mayor = disco_mas_reproducido(v)
            print("El disco con mas reproducciones es: ")
            discos.write(mayor)
        elif op == 2:
            cantidades = cantidad_por_genero(v)
            for i in range(len(cantidades)):
                if cantidades[i] != 0:
                    print("La cantidad del género", i, "es :", cantidades[i])
        elif op == 3:
            x = input("Ingrese titulo: ")
            cant = int(input("Ingrese artista: "))
            incrementar_reproducciones(v, x, cant)
        elif op == 4:
            agregar_disco(v)
        elif op == 5:
            x = input("Ingrese titulo: ")
            y = input("Ingrese artista: ")
            eliminar_disco(v, x, y)
        elif op == 6:
            grabar_discos(v)


if __name__ == '__main__':
    main()
```

3.3. datos.txt

```
3
And Justice For All...
Metallica
1987
2
12345
In Utero
Nirvana
1994
13
83457
Deja vu
Gustavo Ceratti
2010
5
10854
```