Guía de Ejercicios Prácticos 24

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos 24
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:31

Descripción

Esta guía contiene enunciados de algunos ejercicios para aplicar los conceptos de programación en ***Python*** que se analizan en la ***Ficha 24***. 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. Elecciones

Luego de las elecciones nacionales, la junta electoral debe procesar 2 archivos con el fin de obtener los datos del ganador de la elecciones. El primer archivo se llama votos.dat y contiene un listado con Votos, cada voto contiene los siguientes datos: candidato (codificado de 0 a 6), provincia (codificada de 0 a 20) y votante. Y el segundo archivo candidatos.dat que contiene un listado con Candidatos, cada candidato contiene código y nombre.

Se pide:

1. Mostrar el listado completo del archivo de votos.
2. Generar una matriz en donde cada fila es una provincia y cada columna un candidato; y en sus componentes este la cantidad de votos que recibió ese candidato en esa provincia.
3. Mostrar con la matriz el nombre del candidato que ganó las elecciones, junto con la cantidad de votos obtenida.

1.1. voto.py

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


class Voto:
    def __init__(self, candidato, provincia, dni):
        self.candidato = candidato
        self.provincia = provincia
        self.votante = dni


def to_String(v):
    print('Candidato:', v.candidato)
    print('Provincia:', v.provincia)
    print('Votante:', v.votante)
```

1.2. candidato.py

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

class Candidato:
    def __init__(self,codigo, nombre):
        self.codigo = codigo
        self.nombre = nombre
```

1.3. generador.py

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


from voto import *
import pickle
import random


def generar(m):
    a = 1
    while a == 1:
        a = 0
        votante = random.randint(0, 99999)
        for i in range(len(m)):
            if a == m[i]:
                a = 1

    return votante


def alta():
    m = open("votos.dat", 'a+b')
    votante = []
    for i in range(3000):
        v = Voto(random.randint(0, 6), random.randint(0, 20), generar(votante))
        pickle.dump(v, m)
        m.flush()
    m.close()

def alta_candidato():
    m = open("candidatos.dat", 'a+b')
    v = Candidato("Antonio Miguel Carmona", 0)
    pickle.dump(v, m)
    m.flush()
    v.nombre = "Ada Colau"
    v.codigo = 1
    pickle.dump(v, m)
    m.flush()
    v.nombre = "Alberto Fernández Díaz"
    v.codigo = 2
    pickle.dump(v, m)
    m.flush()
    v.nombre = "Fernando Giner"
    v.codigo = 3
    pickle.dump(v, m)
    m.flush()
    v.nombre = "Juan Espadas"
    v.codigo = 4
    pickle.dump(v, m)
    m.flush()
    v.nombre = "Elena Martínez"
    v.codigo = 5
    pickle.dump(v, m)
    m.flush()
    v.nombre = "Alicia Morales"
    v.codigo = 6
    pickle.dump(v, m)
    m.flush()

    m.close()


alta_candidato()
alta()
```

1.4. votacion.py

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

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


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 de votos:')
    while m.tell() < tbm:
        v = pickle.load(m)
        to_String(v)
    m.close()

    print()
    input('Presione <Enter> para seguir...')

def generar_matriz(fd):

    matriz = [[0] * 7 for f in range(21)]
    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('Generando Matriz')
    while m.tell() < tbm:
        v = pickle.load(m)
        matriz[v.provincia][v.candidato] += 1
    m.close()

    print()

    for t in range(len(matriz)):
        for c in range(1, len(matriz[t])):
             print("Cantidad " + str(matriz[t][c]) + " -- Provincia "+str(t)+" -- Candidato " + str(c))

    input('Presione <Enter> para seguir...')
    return matriz

def ganador(m):
    cont = [0] * 7
    for t in range(len(m)):
        for c in range(1, len(m[t])):
             cont[c] += m[t][c]
    may = 0
    for i in range(len(cont)):
        if cont[i] > cont[may]:
            may = i
    return may, cont[may]
```

```
def buscar_nombre(fd, id):

   nombre = "none"
   if not os.path.exists(fd):
        print('El archivo', fd, 'no existe... use la opción 1 para crearlo y grabarle registros...')
        print()
        return nombre

   m = open(fd, 'rb')
   t = os.path.getsize(fd)
   while m.tell() < t:
        c = pickle.load(m)
        if c.codigo == id:
            nombre = c.nombre
            break
   m.close()
   return nombre


def main():
    votos_fd = "votos.dat"
    candidatos_fd =  "candidatos.dat"
    listado_completo(votos_fd)
    matriz = generar_matriz(votos_fd)
    datos = ganador(matriz)
    print("El candidato que gano las elecciones es " + buscar_nombre(candidatos_fd, datos[0])
          + " con " + str(datos[1]) + " votos")

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

2. Gala de beneficencia

Desarrollar un programa que permita generar estadísticas sobre un evento solidario organizado en la ciudad

Se cuenta con un archivo de invitados. Por cada invitado se registra: nombre, mesa que ocupó (0 al 12), ONG a la que ayuda (código del 0 al 9), monto de donación realizada.

Las ONG que se ayudan son:

0	Missing Children	5	Aldeas
1	Caritas	6	Fundaleu
2	PUPI	7	Cimientos
3	Médicos Sin Fronteras	8	Uniendo Caminos
4	Vida Silvestre	9	Adoptare

Programar un módulo separado para generar el archivo de invitados por primera vez.

Al iniciar el programa, cargar en un vector el contenido del archivo, y luego implementar un menú con las siguientes opciones:

1. Mostrar el listado completo de invitados.
2. Informar cantidad de invitados por ONG y por mesa, sólo cuando la cantidad de invitados sea mayor a cero. Mostrar los nombres de las ONGs, en lugar de su código.
3. Mostrar el nombre del invitado que realizó la mayor donación, para una cierta ONG que se ingresa por teclado (validar).
4. Guardar en un archivo todas las donaciones que correspondan a una ONG, cuyo código se ingresa por teclado. El nombre del archivo debe ser donacionesX (donde X es el código de ONG).
5. Buscar el archivo de la ONG x (x se ingresa por teclado, validar). Si existe, recorrerlo y determinar el total recaudado por mesa.

2.1. registros

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

class Invitado:

    def __init__(self,nombre, mesa, ong, monto):
        self.nombre = nombre
        self.mesa = mesa
        self.ong = ong
        self.monto = monto


def to_string(invitado):
    txt = '{:<30}'.format(invitado.nombre)
    txt += '{:<5}'.format(invitado.mesa)
    txt += '{:<30}'.format(describir_ong(invitado.ong))
    txt += '${:<8.2f}'.format(invitado.monto)
    return txt


def describir_ong(codigo):
    ongs = ['Missing Children','Caritas','PUPI','Médicos Sin Fronteras','Vida Silvestre',
            'Aldeas','Fundaleu','Cimientos','Uniendo Caminos','Adoptare']
    return ongs[codigo]
```

2.2. generador.py

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

from registros import *
import pickle
import random


def generar_archivo():
    nombres = ['Marcelo Tinelli', 'Susana Gimenez', 'Mirtha Legrand','Juana Viale', 'Mariana Fabbiani',
               'Mauricio Macri', 'Juliana Awada', 'Jorge Rial', 'Guillermo Andino', 'Alejandro Fantino',
               'Sebastian Yatra', 'Tini Stoessel', 'Lali Esposito', 'Jimena Baron', 'Paulo Londra'
               'Carolina Ardohain','Nicole Neuman', 'Paula Chaves', 'Valeria Mazza',
               'Abel Pintos', 'Luciano Pereyra', 'Soledad Pastorutti']
    m = open('invitados.dat', 'wb')
    for nombre in nombres:
        mesa = random.randrange(13)
        ong = random.randrange(10)
        monto = random.randrange(1000, 100000, 100)
        pickle.dump(Invitado(nombre, mesa, ong, monto), m)
    m.close()


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

3. Administrador de Gastos

Una empresa cuenta con 3 sucursales numeradas desde la 0 a la 2, y necesita gestionar sus gastos por mes. De cada gasto se registra lo siguiente:

- Código.
- Descripción.
- Mes (1-12).
- Sucursal (0-2)
- Importe.

Se necesita hacer lo siguiente, trabajando desde un menu de opciones:

1. Cargar los n registros de gastos en un vector.
2. Mostrar el vector de gastos.
3. Generar un archivo con aquellos gastos cuyo importe supere cierto valor.
4. Mostrar el archivo.
5. Generar una matriz de acumulación a partir del archivo generado en el punto 3,que represente el gasto total por mes y sucursal.
6. A partir de la matriz, totalizar los gastos de un determinado mes.

3.1. gasto.py

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

import random

class Gasto:
    # constructor que genera los datos aleatorios
    def __init__(self):
        self.codigo = random.randint(1, 99)
        self.descripcion = random.choice(["Luz","Agua","Expensas","Gas","Telefono"])
        self.mes = random.randint(1, 12)
        self.sucursal=random.randint(0,2)
        self.importe = random.random() * 1000 + 150


def to_string(gasto):
    renglon = ''
    renglon += '{:>5}'.format(gasto.codigo)
    renglon += ' '
    renglon += '{:<10}'.format(gasto.descripcion)
    renglon += '{:>5}'.format(gasto.mes)
    renglon += ' '
    renglon += '{:>10}'.format(gasto.sucursal)
    renglon += ' '
    renglon += '{:>6}'.format('{:.2f}'.format(gasto.importe))
    print(renglon)
```

3.2. principal.py

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


from gasto import *
import os.path
import pickle

def mostrar_menu():
    print("\nMenu de opciones:")
    print("1.Cargar gastos (en vector)")
    print("2.Mostrar gastos (vector)")
    print("3.Generar archivo de gastos (a partir del vector)")
    print("4.Mostrar archivo de gastos")
    print("5.Gasto total por mes y sucursal(generacion de matriz a partir del archivo)")
    print("6.Gasto total por mes (a partir de la matriz generada)")
    print("0.SALIR")

def generar_gastos():
    n = 0
    while n <= 0:
        n = int(input("Ingrese cantidad de gastos (mayor a cero): "))

    vec = n * [None]
    for i in range(0, n):
        vec[i] = Gasto()
    return vec

def mostrar_listado_gastos(v):
    print("Codigo Descripcion Mes Sucursal Importe")
    print("-"*50)
    for gasto in v:
        to_string(gasto)
    print("-"*50)

def grabar_archivo(v, fd, ref):
    m = open(fd, 'wb')
    for i in range(0,len(v)):
        if v[i].importe > ref:
            pickle.dump(v[i], m)
    m.close()

def mostrar_archivo(fd):
    tam_arch = os.path.getsize(fd)
    arch = open(fd,'rb')
    while arch.tell() < tam_arch:
        gasto = pickle.load(arch)
        to_string(gasto)
    arch.close()

def importe_total_por_mes_sucursal(fd):
    mat = 12 * [0]
    for i in range(0,12):
        mat[i]= 3 * [0]
    tam_arch = os.path.getsize(fd)
    arch = open(fd,'rb')
    while arch.tell() < tam_arch:
        gasto = pickle.load(arch)
        mes=gasto.mes-1
        suc=gasto.sucursal
        mat[mes][suc]+=gasto.importe
    arch.close()
    return mat

def mostrar_matriz(mat):
    for i in range(0,len(mat)):
        print("")
        for j in range(0,len(mat[i])):
```

```
            print("Mes: ", (i+1), "Sucursal: ", (j),"- Gasto total: $", '{:.2f}'.format(mat[i][j]))

def totalizar_costos_mes(mat,mes):
    total=0
    for i in range(len(mat[mes-1])):
        total+=mat[mes-1][i]
    return total

def main():
    arreglo_generado = False
    nombre_archivo = "gastos.dat"

    opc = -1
    while opc != 0:
        mostrar_menu()
        opc = int(input("Ingrese su eleccion:"))
        if opc == 1:
            vec = generar_gastos()
            arreglo_generado = True
            print("Gastos generados.")
        elif opc == 2:
            if arreglo_generado:
                mostrar_listado_gastos(vec)
            else:
                print("\nPrimero debe cargar los gastos!")
        elif opc == 3:
            if arreglo_generado:
                ref=float(input("Ingrese el importe de referencia para generar el archivo de gastos: "))
                grabar_archivo(vec, nombre_archivo, ref)
                print("\nArchivo generado!")
            else:
                print("\nPrimero debe cargar los gastos!")
        elif opc == 4:
            if os.path.exists(nombre_archivo):
                mostrar_archivo(nombre_archivo)
            else:
                print("\nPrimero debe generar el archivo!")
        elif opc == 5:
            if os.path.exists(nombre_archivo):
                mat = importe_total_por_mes_sucursal(nombre_archivo)
                mostrar_matriz(mat)
            else:
                print("\nPrimero debe generar el archivo!")
        elif opc == 6:
            if os.path.exists(nombre_archivo):
                mes=0
                while mes<1 or mes>12:
                    mes = int(input("Ingrese el mes sobre el que quiere totalizar los gastos:"))
                mat = importe_total_por_mes_sucursal(nombre_archivo)
                total=totalizar_costos_mes(mat,mes)
                print("El gasto total durante el mes ",mes," es de: $",total)
            else:
                print("\nPrimero debe generar el archivo!")
        elif opc == 0:
            print ("--- Programa finalizado ---")


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

4. Empresa de Transporte

Una empresa de transporte desea un programa que le permita realizar una serie de informes y estadísticas. Para ello dio a conocer de cada viaje el número del boleto, el nombre del chofer, el costo del boleto, el origen (hasta 10 localidades), el destino (hasta 10 localidades), y el tipo de servicio (hasta 5 tipos diferentes). En base a estos datos realizar un programa manejado por menú de opciones que que permita:

1. Cargar un arreglo con n viajes, donde n es ingresado desde el teclado, con la salvedad que el arreglo debe mantenerse ordenado, por lo que la inserción de un viaje debe respetar dicho orden
2. Determinar el total cobrado para un chofer X pasado por parametro
3. Generar y mostrar un vector con todos los viajes de un servicio X pasado por parámetro
4. Grabar el vector del punto anterior en un archivo, a razón de un viaje por vez
5. Informar la cantidad de viajes hechos de una localidad a otra, para ello armar una matriz en la que cada componente contendrá las cantidad de viajes

4.1. viaje.py

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


class Viaje:
    def __init__(self,numero, chofer, costo, origen, destino,tipo_servicio):
        self.numero = numero
        self.chofer = chofer
        self.costo = costo
        self.origen = origen
        self.destino = destino
        self.tipo_servicio = tipo_servicio


def to_string(viaje):
    cadena = 'Viaje Numero {:<10}'.format(viaje.numero)
    cadena += '- Chofer: {:<20}'.format(viaje.chofer)
    cadena += ' - Origen: {:<2}'.format(viaje.origen)
    cadena += ' - Destino: {:<2}'.format(viaje.destino)
    cadena += ' - Servicio: {:<2}'.format(viaje.tipo_servicio)
    cadena += ' - Costo: {:<8}'.format(viaje.costo)

    return cadena
```

4.2. principal.py

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


from viaje import *
import random
import pickle

__author__ = 'Catedra de Algoritmos y Estructuras de Datos'


def validar_mayor_a(minimo, mensaje):
    numero = int(input(mensaje))
    while numero <= minimo:
        print('Error!! Debe ser mayor a', minimo)
        numero = int(input(mensaje))
    return numero


def add_in_order(viaje, vector):
    tam = len(vector)

    izq, der = 0, tam - 1
    pos = der
    while izq <= der:
        med = (izq + der) // 2
        if vector[med].numero == viaje.numero:
            pos = med
            break

        if viaje.numero < vector[med].numero:
            der = med - 1
        else:
            izq = med + 1

    if izq > der:
        pos = izq

    vector[pos:pos] = [viaje]


def generar_arreglo(n, vector):
    for i in range(n):
        numero = random.randint(1, 10000)
        chofer = "Chofercito N°" + str(random.randint(1, 10000))
        origen = random.randrange(10)
        destino = random.randrange(10)
        servicio = random.randrange(5)
        costo = random.randint(100, 350)
        viaje = Viaje(numero, chofer, costo, origen, destino, servicio)
        add_in_order(viaje, vector)


def mostar(vector):
    print('Listado de Viajes')

    for viaje in vector:
        print('\n' + to_string(viaje))


def generar_por_servicio(vector, servicio):
    v = []
    for viaje in vector:
        if viaje.tipo_servicio == servicio:
            v.append(viaje)
    return v
```

```
def validar_rango(minimo, maximo, mensaje):
    numero = int(input(mensaje))
    while numero < minimo or numero > maximo:
        print('Error!!! El valor debe estar entre', minimo, 'y', maximo)
        numero = int(input(mensaje))
    return numero


def totalizar_por_chofer(chofer, vector):
    total = 0
    for viaje in vector:
        if viaje.chofer == chofer:
            total += viaje.costo

    return total


def genera_archivo(vservicios):
    arch = 'viajes_servicio.dat'
    f = open(arch, 'wb')

    for viaje in vservicios:
        pickle._dump(viaje, f)

    f.flush()
    f.close()


def generar_matriz(vector):
    mat = [[0] * 10 for i in range(10)]
    for viaje in vector:
        f = viaje.origen
        c = viaje.destino
        mat[f][c] += 1
    return mat


def main():
    menu = 'Menu de Opciones\n' \
           '========================================\n' \
           '1 - Generar Arreglo de viajes\n' \
           '2 - Totalizar por Chofer\n' \
           '3 - Generar y Mostrar Vector de Servicion\n' \
           '4 - Generar Archivo\n' \
           '0 - Salir\n' \
           'Ingrese su opcion: '

    vector = []
    vservicios = []
    opcion = -1
    while opcion != 0:
        opcion = int(input(menu))
        if opcion == 1:
            n = validar_mayor_a(0, 'Ingrese la cantidad de viajes: ')
            generar_arreglo(n, vector)

        if len(vector) != 0:
            if opcion == 2:
                chofer = input('Ingrese el nombre del chofer a buscar: ')
                total = totalizar_por_chofer(chofer, vector)
                print('El total de viajes cobrados en viajes del chofer', chofer, 'fueron', total)

            elif opcion == 3:
                servicio = validar_rango(0, 4, 'Ingrese el tipo de servicio a buscar: ')
                vservicios = generar_por_servicio(vector, servicio)
                mostar(vservicios)
```

```
            elif opcion == 4:
                if len(vservicios) == 0:
                    print('Se debe generar el vector con el punto 3')
                else:
                    genera_archivo(vservicios)

            elif opcion == 5:
                matriz = generar_matriz(vector)
                for i in range(matriz):
                    for j in random(matriz[i]):
                        print('La Cantidad de viajes del origen', i, 'al destino', j, 'fueron:', matriz[i][j])

        else:
            print('Debe primero ejecutar la opcion 1')


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

5. Farmacia

Generar un archivo llamado stock.dat conteniendo un registro por cada artículo de la farmacia, con los siguientes campos: código, descripción, precio unitario, stock actual, stock ideal.

Crear un programa que cargue el contenido del archivo en un vector (ordenado alfabéticamente por descripción de artículo) y luego ofrezca un menú con las siguientes opciones:

1. Listado: mostrar el contenido del vector
2. Compras: ingresar un código de artículo. Si existe, ingresar la cantidad comprada y sumarla al stock actual. Si no existe, informarlo.
3. Ventas: ingresar un código de artículo. Si existe, ingresar la cantidad que se desea vender; controlar que exista suficiente stock, restar la cantidad vendida e informar el monto de la venta (cantidad por precio unitario). Si el artículo no existe o no hay suficiente stock, informarlo.
4. Actualización: reemplazar el contenido del archivo stock.dat con los datos del vector
5. Pedido: crear un archivo de texto llamado pedido.txt, conteniendo todos los artículos que necesitan reposición (stock actual < stock ideal). Por cada línea indicar: artículo, cantidad a pedir, costo estimado.

5.1. insercion.py

```
# Se recorre el arreglo p con un ciclo for, controlando que cada casilla sea menor que el nuevo registro.
# Si se encuentra uno mayor (o incluso igual) al que se quiere agregar, se detiene el ciclo,
# asignando en pos el índice de la casilla que contenía al registro mayor.
# Al terminar el ciclo, se agrega en pos el nuevo elemento y se desplaza el resto del vector hacia la derecha.

def add_in_order_seq(p, nuevo):
    n = len(p)
    pos = n
    for i in range(n):
        if nuevo.campo < p[i].campo:
            pos = i
            break
    p[pos:pos] = [nuevo]


# La variable pos comienza valiendo n, el tamaño del arreglo.
# Si existiese un registro igual al que se quiere agregar, el valor de pos cambia
# para tomar el valor del índice c de la casilla que lo contiene, y el ciclo de búsqueda corta con break.
# Al terminar el ciclo, se chequea si el valor de izq terminó siendo mayor que der.  Significa que ese registro no existía en el
arreglo.
# Por lo tanto, el valor de pos cambia para tomar el valor de izq, y se inserta el nuevo registro.

def add_in_order_bin(p, nuevo):
    n = len(p)
    pos = n
    izq, der = 0, n - 1
    while izq <= der:
        c = (izq + der) // 2
        if p[c].campo == nuevo.campo:
            pos = c
            break
        if nuevo.campo < p[c].campo:
            der = c - 1
        else:
            izq = c + 1
    if izq > der:
        pos = izq
    p[pos:pos] = [nuevo]
```

5.2. articulo.py

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


class Articulo:

    def __init__(self, codigo, nombre, precio, stock_actual, stock_ideal):
        self.codigo = codigo
        self.nombre = nombre
        self.precio = precio
        self.stock_actual = stock_actual
        self.stock_ideal = stock_ideal


def to_string(reg):
    txt = '{:<5}'.format(reg.codigo)
    txt += '{:<20}'.format(reg.nombre)
    txt += '${:>8.2f}'.format(reg.precio)
    txt += '{:>8}'.format(reg.stock_actual)
    txt += '{:>8}'.format(reg.stock_ideal)
    return txt
```

5.3. generador.py

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

from articulo import *
import pickle
import random


def generar_archivo(fd):
    articulos = ['Alcohol', 'Gasa', 'Algodon', 'Jeringa', 'Aspirina',
                 'Vitaminas', 'Dentifrico', 'Bicarbonato',
                 'Amoxidal', 'Novalgina', 'Agua oxigenada', 'Pañales', 'Mamadera',
                 'Chupete', 'Crema', 'Champú', 'Acondicionador']
    m = open(fd, 'wb')
    for i in range(len(articulos)):
        precio = random.randrange(10, 200)
        actual = random.randrange(20)
        ideal = random.randrange(20)
        pickle.dump(Articulo(i, articulos[i], precio, actual, ideal), m)
    m.close()


if __name__ == '__main__':
    fd = 'stock.dat'
    generar_archivo(fd)
    print('Archivo', fd, 'generado.')
```

5.4. principal.py

```
from articulo import *
import os
import pickle

__author__ = 'Catedra de Algoritmos y Estructuras de Datos'


def add_in_order(v, articulo):
    n = len(v)
    izq, der = 0, n - 1
    pos = n
    while izq <= der:
        med = (izq + der) // 2
        if v[med].nombre == articulo.nombre:
            pos = med
            break

        if articulo.nombre < v[med].nombre:
            der = med - 1
        else:
            izq = med + 1

    if izq > der:
        pos = izq

    v[pos:pos] = [articulo]


def cargar_vector(fd):
    v = []
    if os.path.exists(fd):
        m = open(fd, 'rb')
        t = os.path.getsize(fd)
        while m.tell() < t:
            articulo = pickle.load(m)
            add_in_order(v, articulo)
        m.close()

    return v


def menu():
    print('-'*60)
    print('FARMACIA - Menu de Opciones')
    print('1 - Mostrar el Vector')
    print('2 - Comprar')
    print('3 - Vender')
    print('4 - Actualizar archivo')
    print('5 - Realizar Pedido')
    print('0 - Salir')
    print('-' * 60)
    return int(input('Ingrese su opcion: '))


def mostrar_vector(v):
    print('Listado de Articulos')
    for i in range(len(v)):
        print(to_string(v[i]))


def buscar(v, codigo):
    n = len(v)
    for i in range(n):
        if v[i].codigo == codigo:
            return i
    return -1
```

```
def actualizar_archivo(v, fd):
    m = open(fd, 'wb')
    for articulo in v:
        pickle.dump(articulo, m)
        m.flush()
    m.close()


def generar_pedido(v):
    m = open('pedido.txt', 'w')
    linea = '{:<30}\t{:<10}\t${:<10.2f}\n'
    for articulo in v:
        if articulo.stock_actual < articulo.stock_ideal:
            cantidad = articulo.stock_ideal - articulo.stock_actual
            cadena = linea.format(articulo.nombre, cantidad, cantidad * articulo.precio)
            m.write(cadena)
    m.close()


def principal():
    fd = 'stock.dat'
    v = cargar_vector(fd)
    opcion = -1
    while opcion != 0:
        opcion = menu()
        if opcion == 1:
            mostrar_vector(v)
        elif opcion == 2:
            codigo = int(input('Ingrese el codigo de articulo a buscar: '))
            pos = buscar(v, codigo)
            if pos == -1:
                print('El Articulo no existe!!!')
            else:
                cantidad = int(input('Ingrese la cantidad comprada: '))
                v[pos].stock_actual += cantidad
                print('Stock Actualizado!!!')
                print(to_string(v[pos]))
        elif opcion == 3:
            codigo = int(input('Ingrese el codigo de articulo a buscar: '))
            pos = buscar(v, codigo)
            if pos == -1:
                print('El Articulo no existe!!!')
            else:
                cantidad = int(input('Ingrese la cantidad vendida: '))
                if v[pos].stock_actual >= cantidad:
                    v[pos].stock_actual -= cantidad
                    monto = cantidad * v[pos].precio
                    print('Stock Actualizado!!!')
                    print('El monto de la venta es $', monto)
                else:
                    print('No hay suficientes unidades para realizar la venta')
                print(to_string(v[pos]))
        elif opcion == 4:
            actualizar_archivo(v, fd)
        elif opcion == 5:
            generar_pedido(v)
            print('Pedido generado en el archivo')


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

6. Aspirantes

Una escuela de la ciudad de Córdoba desea un programa que le permita procesar la información de los aspirantes a ingresar en el ciclo lectivo 2021.

Los candidatos ya han rendido un examen de ingreso, y la información se encuentra en el archivo aspirantes.csv, conteniendo para cada candidato: dni, nombre, apellido, nota obtenida (4 al 10 inclusive)

Se solicita cargar los datos en un vector (que debe generarse ordenado por DNI) y luego implementar el siguiente menú de opciones

1. Mostrar: listar la información completa de todos los aspirantes

2. Contar por nota: informar cuántos aspirantes obtuvieron cada una de las notas posibles

3. Asignar vacantes: ingresar por teclado la cantidad de vacantes disponibles (debe ser menor a la cantidad de aspirantes). Luego, proceder a asignarlas de la siguiente manera:

- Filtrar los aspirantes que obtuvieron 9 o 10. Si son menos que la cantidad de vacantes, incorporarlos a la lista de ingresantes. Si son más, sortear entre ellos las vacantes disponibles.
- Si quedaran vacantes disponibles, sortearlas entre el resto de los aspirantes.
- Mostrar el listado de aspirantes seleccionados.

4. Consultar por DNI: a partir de un DNI que se ingresa por teclado, validar si corresponde o no a un aspirante. En caso afirmativo, informar además si ha quedado seleccionado como ingresante (o indicar que aún no se han asignado las vacantes)

Antes de salir del programa, guardar la lista de ingresantes (si fue generada) en un archivo binario llamado ingresantes.dat.

6.1. registros

```
class Aspirante:
    def __init__(self, dni, nombre, apellido, nota):
        self.dni = dni
        self.nombre = nombre
        self.apellido = apellido
        self.nota = nota

    def __str__(self):
        return '{:<9}{:<20}{:>3}'.format(self.dni, self.nombre + ' ' + self.apellido.upper(), self.nota)


def aspirante_to_csv(reg):
    linea = '{},{},{},{}\n'.format(reg.dni, reg.nombre, reg.apellido, str(reg.nota))
    return linea


def csv_to_aspirante(linea):
    if linea[-1] == '\n':
        linea = linea[:-1]
    campos = linea.split(',')
    return Aspirante(campos[0], campos[1], campos[2], int(campos[3]))
```

6.2. generador

```
import random
from registros import *


def generar_csv():
    n = 100
    noms = ('Benjamin', 'Isabella', 'Martina', 'Catalina', 'Bautista', 'Sofia', 'Olivia', 'Felipe',
            'Emma', 'Valentino', 'Benicio', 'Joaquín', 'Delfina', 'Lorenzo', 'Francesca', 'Valentina', 'Emilia',
            'Mateo', 'Santino', 'Tomás', 'Francisco', 'Juana', 'Josefina', 'Santiago', 'Simon', 'Guillermina',
            'Ignacio', 'Victoria', 'Guadalupe', 'Agustin', 'Julieta', 'Pedro', 'Renata', 'Alma', 'Jazmin', 'Ciro',
            'Thiago', 'Lola', 'Pilar', 'Lucia', 'Lautaro', 'Bruno', 'Malena', 'Julia', 'Dante', 'Valentin', 'Salvador',
            'Nicolas', 'Maximo', 'Mia', 'Tiziano', 'León', 'Camila', 'Clara', 'Bianca', 'Milo', 'Facundo', 'Vicente',
            'Helena', 'Zoe', 'Agustina', 'Ambar', 'Ramiro', 'Manuel', 'Nina', 'Julián', 'Bastian', 'María Paz',
            'Paulina', 'Mora', 'Jeremías', 'Amparo', 'Morena', 'Franco', 'Matias', 'Felicitas', 'Luca', 'Genaro',
            'Lucas', 'Augusto', 'Mia Jazmin', 'Fausto', 'Camilo', 'Vera', 'Elena', 'Giovanni', 'Lucio', 'Juliana',
            'Antonella', 'Antonia', 'Agostina', 'Matilda', 'Martin', 'Constanza', 'Alejo', 'Paloma', 'Ian', 'Gael',
            'Lara', 'Luciano', 'Federico', 'Milagros', 'Lisandro', 'Pía', 'Francesco', 'Lourdes', 'Noah', 'Sara',
            'Alfonsina', 'Lupe', 'Sebastián', 'Candelaria', 'Octavio', 'Jonás', 'Geronimo', 'Teo', 'Abril', 'Tobias',
            'Gino')
    aps = (
    'González', 'Rodríguez', 'Gómez', 'Fernández', 'López', 'Díaz', 'Martínez', 'Pérez', 'García', 'Sánchez', 'Romero',
    'Sosa', 'Torres', 'Álvarez', 'Ruiz', 'Ramírez', 'Flores', 'Benítez', 'Acosta', 'Medina', 'Herrera', 'Suárez',
    'Aguirre', 'Giménez', 'Gutiérrez', 'Pereyra', 'Rojas', 'Molina', 'Castro', 'Ortiz', 'Silva', 'Núñez', 'Luna',
    'Juárez', 'Cabrera', 'Ríos', 'Morales', 'Godoy', 'Moreno', 'Ferreyra', 'Domínguez', 'Carrizo', 'Peralta',
    'Castillo', 'Ledesma', 'Quiroga', 'Vega', 'Vera', 'Muñoz', 'Ojeda', 'Ponce', 'Villalba', 'Cardozo', 'Navarro',
    'Coronel', 'Vázquez', 'Ramos', 'Vargas', 'Cáceres', 'Arias', 'Figueroa', 'Córdoba', 'Correa', 'Maldonado', 'Paz',
    'Rivero', 'Miranda', 'Mansilla', 'Farias', 'Roldán', 'Méndez', 'Guzmán', 'Agüero', 'Hernández', 'Lucero', 'Cruz',
    'Páez', 'Escobar', 'Mendoza', 'Barrios', 'Bustos', 'Ávila', 'Ayala', 'Blanco', 'Soria', 'Maidana', 'Acuña', 'Leiva',
    'Duarte', 'Moyano', 'Campos', 'Soto', 'Martín', 'Valdez', 'Bravo', 'Chávez', 'Velázquez', 'Olivera', 'Toledo')
    f = open('aspirantes.csv', 'wt')
    for i in range(n):
        dni = 47000000 + i
        nombre = random.choice(noms)
        apellido = random.choice(aps)
        nota = random.randint(4, 10)
        linea = aspirante_to_csv(Aspirante(dni, nombre, apellido, nota))
        f.write(linea)
    print(n, 'registros generados')
    f.close()


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

6.3. interfaz

```
from estructuras import *


def mostrar_menu():
    print('=' * 50)
    print('GESTION DE ASPIRANTES 2021')
    print('1. Mostrar aspirantes')
    print('2. Contar por nota')
    print('3. Asignar vacantes')
    print('4. Consultar por DNI')
    print('0. Salir')
    print('=' * 50)
    opcion = int(input('Ingrese su opcion: '))
    return opcion


def validar_entre(desde, hasta, mensaje):
    num = int(input(mensaje))
    while num < desde or num > hasta:
        num = int(input('Inválido! ' + mensaje))
    return num


def mostrar_conteo(conteo):
    for i in range(len(conteo)):
        print('Nota', i + 4, ':', conteo[i])


def principal():
    va = leer_csv('aspirantes.csv')
    vi = list()
    if len(va) == 0:
        return
    opcion = -1
    while opcion != 0:
        opcion = mostrar_menu()
        if opcion == 1:
            mostrar_vector(va)
        elif opcion == 2:
            conteo = contar_por_nota(va)
            mostrar_conteo(conteo)
        elif opcion == 3:
            vacantes = validar_entre(0, len(va), 'Ingrese vacantes disponibles: ')
            vi = asignar_vacantes(va, vacantes)
            mostrar_vector(vi)
        elif opcion == 4:
            dni = input('Ingrese DNI a buscar: ')
            pos = buscar_binario(va, dni)
            if pos == -1:
                print('No existe un aspirante con ese DNI')
            else:
                print('Aspirante encontrado:', va[pos])
                if len(vi) == 0:
                    print('Aún no se han sorteado las vacantes')
                else:
                    pos = buscar_secuencial(vi, dni)
                    if pos == -1:
                        print('No se encuentra entre los ingresantes')
                    else:
                        print('Es un nuevo ingresante. Felicitaciones!')
        elif opcion == 0:
            if len(vi) > 0:
                guardar_archivo(vi, 'ingresantes.dat')
                print('Se guardó la lista de ingresantes')
            print('Hasta pronto!')
```

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

6.4. estructuras

```
import os
import pickle
import random
from registros import *


def leer_csv(fd):
    v = list()
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe.')
    else:
        f = open(fd, 'rt')
        for linea in f:
            reg = csv_to_aspirante(linea)
            add_in_order(v, reg)
        f.close()
    return v


def add_in_order(v, nuevo):
    izq, der = 0, len(v) - 1
    while izq <= der:
        c = (izq + der) // 2
        if v[c].dni == nuevo.dni:
            pos = c
            break
        if nuevo.dni < v[c].dni:
            der = c - 1
        else:
            izq = c + 1
    if izq > der:
        pos = izq
    v[pos:pos] = [nuevo]


def mostrar_vector(v):
    for reg in v:
        print(reg)


def guardar_archivo(v, fd):
    f = open(fd, 'wb')
    for reg in v:
        pickle.dump(reg, f)
    f.close()


def filtrar_por_nota(va, desde, hasta):
    aux = list()
    for reg in va:
        if reg.nota >= desde and reg.nota <= hasta:
            aux.append(reg)
    return aux


def asignar_vacantes(va, vacantes):
    vi = list()
    # Filtrar alumnos con 9 o 10
    sorteo = filtrar_por_nota(va, 9, 10)
    # Si son menos que las vacantes, asignarlos directamente y definir un nuevo conjunto
    if len(sorteo) <= vacantes:
        vi[:] = sorteo[:]
        vacantes -= len(sorteo)
        sorteo = filtrar_por_nota(va, 0, 8)
    for i in range(vacantes):
        pos = random.randrange(0, len(sorteo))
        vi.append(sorteo[pos])
```

```
        del (sorteo[pos])
    return vi


def contar_por_nota(va):
    conteo = [0] * 7
    for reg in va:
        conteo[reg.nota - 4] += 1
    return conteo


def buscar_binario(v, dni):
    izq, der = 0, len(v) - 1
    while izq <= der:
        c = (izq + der) // 2
        if v[c].dni == dni:
            return c
        if dni < v[c].dni:
            der = c - 1
        else:
            izq = c + 1
    return -1


def buscar_secuencial(v, dni):
    for i in range(len(v)):
        if v[i].dni == dni:
            return i
    return -1
```