Guía de Ejercicios Prácticos 22

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos 22
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:30

Tabla de contenidos

1. Quini 6

Desarrollar un programa que permita generar el extracto del sorteo del Quini 6. El programa debe tener las siguientes opciones:

1) Cargar sorteo: cargar por teclado los 6 números sorteados. Los valores posibles van del 0 al 36, ambos inclusive (validar). Grabarlos el archivo extracto.dat, ordenados de forma ascendente.

2) Consultar: mostrar el contenido del archivo extracto.dat.

1.1. quini.py

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

import pickle

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


def ordenar_ascendente(v):
    for i in range(len(v)-1):
        for j in range(i+1, len(v)):
            if v[i] > v[j]:
                v[i],v[j] = v[j],v[i]


def cargar_sorteo():
    v = [0] *6
    for i in range(len(v)):
        v[i] = validar_entre('Ingrese el ' + str(i+1) + '° número: ',0,36)
    ordenar_ascendente(v)
    m = open("extracto.dat","wb")
    pickle.dump(v,m)
    m.close()


def mostrar_extracto():
    archivo = 'extracto.dat'
    m = open(archivo,'rb')
    v = pickle.load(m)
    print(v)
    m.close()


def main():
    opcion = -1
    while opcion!=0:
        print('*'*80)
        print('SORTEO DEL QUINI 6')
        print('1- Cargar sorteo')
        print ('2 - Consultar')
        print ('0 - Salir')
        opcion = int(input('Ingrese opción: '))
        if opcion == 1:
            cargar_sorteo()
            print('Sorteo cargado!')
        elif opcion == 2:
            mostrar_extracto()
        print('*'*80)

main()
```

2. Cabañas

Una empresa dedicada al alquiler de cabañas de veraneo desea almacenar la información referida a los n alquileres de la temporada estival en un arreglo de registros (cargar n por teclado).

Por cada alquiler, se pide guardar el DNI de la persona que hizo la reserva, monto del alquiler y un código entre 0 y 9 que indica el tipo de cabaña alquilada.

Se pide desarrollar un programa en Python que permita:

- Cargar el arreglo pedido (validar tipo de cabaña).
- Trasladar a un archivo los alquileres que registraron un monto mayor a x, siendo x un valor pasado por parámetro.
- Usando los datos del archivo, determinar y mostrar el monto total recaudado por cada tipo de cabaña posible.

2.1. cabaña.py

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


class Cabaña:
    def __init__(self, documento, monto, tipo):
        self.documento = documento
        self.monto = monto
        self.tipo = tipo


def to_string(cabaña):
    cadena = 'Reseva por Documento: {:<10}'.format(cabaña.documento)
    cadena += ' Moto a pagar: ${:<10.2f}'.format(cabaña.monto)
    cadena += ' Tipo Cabaña: ${:<5}'.format(cabaña.tipo)
    return cadena
```

2.2. principal.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from cabaña import *
import random
import pickle
import os.path

def generar_vector(n):
    v = [None] * n
    for pos in range(n):
        doc = random.randint(1000000, 50000000)
        monto = random.uniform(1000, 7500)
        tipo = random.randrange(10)
        v[pos] = Cabaña(doc, monto, tipo)
    return v


def generar_archivo(vector, monto, archivo):
    arch = open(archivo, 'wb')
    for cabaña in vector:
        if cabaña.monto > monto:
            pickle.dump(cabaña, arch)
    arch.flush()
    arch.close()


def leer_archivo(archivo):
    size = os.path.getsize(archivo)
    if size < 0:
        return None

    arch = open(archivo, 'rb')
    vc = [0] * 10
    while arch.tell() < size:
        cabaña = pickle.load(arch)
        vc[cabaña.tipo] += cabaña.monto

    arch.close()
    return vc


def principal():
    menu = 'Menu de Opciones \n' \
           '================================== \n' \
           '1 - Cargar Arreglo de Reservas \n' \
           '2 - Grabar Archivo por Monto \n' \
           '3 - Mostrar Archivo \n' \
           '4 - Salir\n' \
           'Ingrese la opcion: '

    nombre_archivo = 'reservas.dat'
    opcion = 0
    vector = []
    while opcion != 4:
        opcion = int(input(menu))
        if opcion == 1:
            n = int(input('Ingrese la cantidad de elementos a generar: '))
            vector = generar_vector(n)

        elif opcion == 2:
            monto = float(input('Ingrese el monto minimo para guardar: '))
            generar_archivo(vector, monto, nombre_archivo)

        elif opcion == 3:
            listado = leer_archivo(nombre_archivo)
            if not listado is None:
                for i in range(len(listado)):
                    print('Tipo de Cabaña ', i, 'recaudo $', listado[i])
```

```
            else:
                print('El Archivo no tiene Registros')

principal()
```

3. Rapipago

Una empresa de pagos de servicios necesita un programa que permita llevar los cobros que ha realizado, para ello marco como muy importante que los datos no se pierdan. De un cobro se sabe el dia en que se cobró, el tipo de servicio (un valor de 1 a 15), el monto cobrado y el número de cuenta asociada a la persona que pagó dicho servicio.

Se desea manejar la siguiente lógica a través de un menú de opciones que cumpla:

1. Agregar un nuevo cobro al archivo de cobros.
2. Determinar el monto total para un número de cuenta X pasado por parámetro
3. Indicar el monto total acumulado para cada servicio, en el día del mes que se lo cobro (matriz de acumulación)
4. Indicar, a partir de la matriz, el día con mayor monto cobrado
5. Indicar, a partir de la matriz, el promedio cobrado para el servicio X pasado por parámetro

3.1. principal.py

```
import os.path
import pickle

__author__ = 'Catedra de Algoritmos y Estructuras de Datos'


class Cobro:
    def __init__(self, cuenta, mes, tipo, monto):
        self.cuenta = cuenta
        self.mes = mes
        self.tipo = tipo
        self.monto = monto


def validar_rango(minimo, maximo, mensaje):
    error = 'Error!!! El valor debe ser entre {} y {}. {}'.format(minimo, maximo, mensaje)
    numero = int(input(mensaje))
    while numero < minimo or numero > maximo:
        numero = int(input(error))
    return error


def generar_cobro():
    cuenta = input('Ingrese la cuenta del servicio a cobrar: ')
    tipo = validar_rango(1, 15, 'Ingresar el tipo de Servicio: ')
    dia = validar_rango(1, 31, 'Ingresar el dia en que se Cobro el Servicio: ')
    monto = float(input('Ingrese el monto cobrado: '))
    return Cobro(cuenta, dia, tipo, monto)


def grabar_cobro(cobro, archivo):
    file = open(archivo, 'ab')
    pickle.dump(cobro, file)
    file.flush()
    file.close()


def calcular_total_cuenta(cuenta, fd):
    total = 0
    size = os.path.getsize(fd)
    if size == 0:
        return False, total

    m = open(fd, 'rb')
    while m.tell() < size:
        cobro = pickle.load(m)
        if cobro.cuenta == cuenta:
            total += cobro.monto

    m.close()
    return True, total


def generar_matriz(fd):
    if not os.path.exists(fd):
        return []

    matriz = [[0] * 31 for i in range(15)]
    m = open(fd, 'rb')
    size = os.path.getsize(fd)
    while m.tell() < size:
        cobro = pickle.load(m)
        matriz[cobro.tipo - 1][cobro.dia - 1] += cobro.monto
    m.close()
    return matriz
```

```
def mostrar_matriz(matriz):
    tc = len(matriz[0])
    tf = len(matriz)
    celda = '|{:^8}'
    listado = celda.format('')
    for i in range(1, tc + 1):
        listado += celda.format('Dia ' + str(i + 1))

    for i in range(tf):
        listado += '\n' + celda.format('Tipo ' + str(i + 1))
        for j in range(tc):
            listado += celda.format(matriz[i][j])

    return listado


def totalizar_dia(columna, matriz):
    total = 0
    for f in range(len(matriz)):
        total += matriz[f][columna]
    return total


def mayor_dia_con_cobros(matriz):
    tc = len(matriz[0])
    mayor = totalizar_dia(0, matriz)
    dia = 0
    for c in range(1, tc):
        total = totalizar_dia(c, matriz)
        if mayor < total:
            mayor = total
            dia = c
    return dia


def totalizar_por_servicio(tipo, matriz):
    total = 0
    for c in range(len(matriz[tipo])):
        total += matriz[tipo][c]
    return total / len(matriz[tipo])


def principal():
    fd = 'cobros.dat'
    menu = 'Menu de Opciones\n' \
           '==================================================\n' \
           '1 - Cargar un Cobro en el Archivo\n' \
           '2 - Determinar Total Cobrado a Cuenta\n' \
           '3 - Determinar el Monto por Tipo Servicio y Dia\n' \
           '4 - Determinar dia con mayor monto\n' \
           '5 - Determinar Promedio Cobrado por Tipo de Servicio\n' \
           '6 - Salir\n' \
           'Ingrese su opcion: '

    matriz = []
    opcion = 0
    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            cobro = generar_cobro()
            grabar_cobro(cobro, fd)

        elif opcion == 2:
            cuenta = input('Ingrese el numero de cuenta a totalizar: ')
            res, total = calcular_total_cuenta(cuenta, fd)
            if res:
                print('El total cobrado a la cuenta ingresada fue de $', total)
```

```
            else:
                print('El Archivo no existe o no tiene registros de cobros')

        elif opcion == 3:
            matriz = generar_matriz(fd)
            if len(matriz) > 0:
                print(mostrar_matriz(matriz))
            else:
                print('No se ha generado la matriz')

        elif opcion == 4:
            dia = mayor_dia_con_cobros(matriz)
            print('El dia con mayor monto cobrado fue', dia + 1)

        elif opcion == 5:
            tipo = validar_rango(1, 15, 'Ingrese el tipo de servicio a promediar: ')
            promedio = totalizar_por_servicio(tipo, matriz)
            print('El promedio cobrado para el tipo', tipo, 'fue de $', round(promedio, 2))


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

4. Agenda

Se solicita un programa que permita agendar contactos en un archivo. De cada contacto se debe ingresar su nombre y apellido, mail y teléfono.

Se pide:

- Cargar el archivo por teclado.
- Mostrar el archivo.
- Generar a partir del archivo un vector con todos los contactos.

4.1. contacto.py

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

class Contacto:
    def __init__(self, nombre, apellido, mail, telefono):
        self.nombre = nombre
        self.apellido = apellido
        self.mail = mail
        self.telefono = telefono


def to_string(contacto):
    linea = '{:<30} {:<30} {:<15} {:<30}'
    return linea.format(contacto.nombre, contacto.apellido, contacto.telefono, contacto.mail)
```

4.2. principal.py

```
import os.path
import pickle
import random

from contacto import *

__author__ = 'Catedra de Algoritmos y Estructuras de Datos'


def menu():
    print('Menu de Opciones')
    print('=' * 60)
    print('1 \t Cargar Archivo por Teclado')
    print('2 \t Mostrar Archivo')
    print('3 \t Generar Vector de contactos')
    print('4 \t Salir')
    return int(input('Ingrese su opcion: '))


def generar_archivo(archivo, n):
    m = open(archivo, 'wb')
    for i in range(n):
        nombre = input('Ingrese el nombre del contacto: ')
        apellido = input('Ingrese el apellido del contacto: ')
        telefono = input('Ingrese el telefono del contacto: ')
        mail = input('Ingrese el mail del contacto: ')
        entrada_agenda = Contacto(nombre, apellido, mail, telefono)
        pickle.dump(entrada_agenda)
    m.flush()
    m.close()


def generar_archivo_random(archivo, n):
    nombres = ('Carlos', 'Juan', 'Esteban', 'Jazmin', 'Serio', 'Luis', 'German')
    apellidos = ('Garcia', 'Perez', 'Lopez', 'Martinez', 'Prueba', 'Pruebon', 'Pruebita')
    m = open(archivo, 'wb')
    for i in range(n):
        nombre = random.choice(nombres)
        apellido = random.choice(apellidos)
        telefono = '035' + str(random.randint(0, 9)) + str(random.randint(100000, 999999))
        mail = nombre + '.' + apellido + '@test.com'
        entrada_agenda = Contacto(nombre, apellido, mail, telefono)
        pickle.dump(entrada_agenda, m)
    m.flush()
    m.close()


def validar_mayor(minimo, mensage):
    valor = int(input(mensage))
    while valor <= minimo:
        print('El valor es incorrecto!!! Debe ser mayor a', minimo)
        valor = int(input(mensage))
    return valor


def mostrar_archivo(archivo):
    size = os.path.getsize(archivo)
    if size <= 0:
        return 'No hay registro para mostrar'

    listado = 'Listado de Contactos de la Agenda. \n{}'.format('=' * 110)
    listado += '\n{:<30} {:<30} {:<15} {:<30}'.format('Nombre', 'Apellido', 'Telefono', 'Email')
    listado += '\n{}'.format('-' * 110)
    m = open(archivo, 'rb')
    while m.tell() < size:
        contacto = pickle.load(m)
        listado += '\n' + to_string(contacto)
```

```
    m.close()
    return listado


def generar_vector(archivo):
    vector = []
    size = os.path.getsize(archivo)
    if size <= 0:
        return vector

    m = open(archivo, 'rb')
    while m.tell() < size:
        contacto = pickle.load(m)
        vector.append(contacto)
    m.close()
    return vector


def mostrar_vector(vector):
    listado = 'Listado de Contactos de la Agenda. \n{}'.format('=' * 110)
    listado += '\n{:<30} {:<30} {:<15} {:<30}'.format('Nombre', 'Apellido', 'Telefono', 'Email')
    listado += '\n{}'.format('-' * 110)
    for contacto in vector:
        listado += '\n' + to_string(contacto)
    return listado


def principal():
    print('Generacion de Agenda')
    archivo = 'contactos.dat'

    opcion = -1
    while opcion != 4:
        opcion = menu()
        print()
        if opcion == 1:
            tam = validar_mayor(0, 'Ingrese la cantidad de contactos: ')
            rand = input('Desea agregar contactos en forma aleatoria (S/N): ')
            if rand == 'S':
                generar_archivo_random(archivo, tam)
            else:
                generar_archivo(archivo, tam)

        elif opcion == 2:
            listado = mostrar_archivo(archivo)
            print(listado)

        elif opcion == 3:
            vector = generar_vector(archivo)
            print(mostrar_vector(vector))
        print('\n\n')

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

5. Series

Un amigo tiene una lista de series que le gustaría ver, guardadas en un archivo “series.csv” y nos pide ayuda para poder manejar dicha lista. El archivo posee la siguiente información de cada serie:

- Título o nombre
- Género: 0-Infantil, 1-Comedia, 2-Romántico, 3-Drama, 4-Ciencia Ficción, 5-Otros.
- Idioma Original: 0-Español, 1: Inglés, 2: Francés, 3: Portugués, 4:Otros.
- Cantidad de temporadas.
- Duracion total (en minutos)

En primer lugar, cargar el contenido del archivo en un vector de registros y ordenarlo por título.

Luego, implementar un menú de opciones que permita:

1. Listar el contenido del vector, mostrando una línea por serie (usar género y el idioma en lugar de sus códigos).
2. Ingresar por teclado un idioma x. Generar un archivo cuyo nombre tenga la forma “SeriesX.dat” (reemplazando x por el número del idioma seleccionado) conteniendo todas las series de ese idioma que tengan más de una temporada. Mostrar el nuevo archivo generado.
3. Buscar en el vector una serie con el título x (x se ingresa por teclado). Si la serie existe, mostrar sus datos. Si no, informar con un mensaje.
4. Determinar la duración total de las series en idioma español por cada género disponible.
5. A partir del vector determinar la cantidad de series por género y por idioma. Para eso se debe utilizar una matriz de conteo. Mostrar las cantidades sólo cuando sean mayores a 0.

5.1. registro.py

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


class Serie:
    def __init__(self, titulo, genero, idioma, temporadas, duracion):
        self.titulo = titulo
        self.genero = genero
        self.idioma = idioma
        self.temporadas = temporadas
        self.duracion = duracion


def csv_to_Serie(linea):
    linea = linea[:-1]
    v = linea.split(',')
    return Serie(v[0], int(v[1]), int(v[2]), int(v[3]), int(v[4]))


def to_string(reg):
    return '{:<30}{:<20}{:<12}{:>4}{:>6}'.format(reg.titulo, describir_genero(reg.genero), describir_idioma(reg.idioma),
                                                 reg.temporadas, reg.duracion)


def to_string_titulos():
    return '{:<30}{:<20}{:<12}{:<4}{:>6}'.format('Titulo', 'Genero', 'Idioma', 'Temp', 'Durac')


def describir_idioma(cod):
    idiomas = ('0-Español', '1-Ingles', '2-Frances', '3-Portugues', '4-Otros')
    return idiomas[cod]


def describir_genero(cod):
    generos = ('0-Infantil', '1-Comedia', '2-Romantico', '3-Drama', '4-Ciencia Ficcion', '5-Otros')
    return generos[cod]
```

5.2. generador.py

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

import random


def generar_archivo_csv():
    titulos = ['Friends', 'Lost', 'House of Cards', 'Mad Men', 'Vis a Vis',
               'La casa de papel', 'Breaking Bad', 'Dexter', 'This is us', 'The Crown',
               'House', 'Los Simpsons', 'Cobra Kai', 'Narcos', 'Stranger Things']
    m = open('series.csv', 'wt')
    for tit in titulos:
        genero = str(random.randrange(6))
        idioma = str(random.randrange(5))
        temporadas = str(random.randint(1, 10))
        duracion = str(random.randint(60, 600))
        linea = '{},{},{},{},{}\n'.format(tit, genero, idioma, temporadas, duracion)
        m.write(linea)
    m.close()


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

5.3. principal_base

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


def mostrar_menu():
    print('-' * 80)
    print('CATÁLOGO DE SERIES')
    print('1. Mostrar vector')
    print('2. Generar archivo por idioma')
    print('3. Buscar por título')
    print('4. Duración de series en español por género')
    print('5. Contar por género e idioma')
    print('0. Salir')
    print('-' * 80)


def principal():
    opcion = -1
    while opcion != 0:
        mostrar_menu()
        opcion = int(input('Ingrese su opción: '))


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

5.4. principal.py

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

import os
import pickle
from registro import *


def mostrar_menu():
    print('-' * 80)
    print('CATÁLOGO DE SERIES')
    print('1. Mostrar vector')
    print('2. Generar archivo por idioma')
    print('3. Buscar por título')
    print('4. Duración de series en español por género')
    print('5. Contar por género e idioma')
    print('0. Salir')
    print('-' * 80)


def leer_csv(fd):
    v = list()
    if os.path.exists(fd):
        f = open(fd, "rt")
        for linea in f:
            serie = csv_to_Serie(linea)
            v.append(serie)
        f.close()
    else:
        print('El archivo', fd, 'no existe')
    return v


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


def ordenar_por_titulo(v):
    for i in range(len(v) - 1):
        for j in range(i + 1, len(v)):
            if v[i].titulo > v[j].titulo:
                v[i], v[j] = v[j], v[i]


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


def generar_archivo(v, idioma, fd):
    f = open(fd, 'wb')
    for reg in v:
        if reg.idioma == idioma and reg.temporadas > 1:
            pickle.dump(reg, f)
    f.close()


def mostrar_archivo(fd):
    hay_datos = False
    if os.path.exists(fd):
        f = open(fd, "rb")
        size = os.path.getsize(fd)
        while f.tell() < size:
            hay_datos = True
```

```
            reg = pickle.load(f)
            print(to_string(reg))
        f.close()
    if not hay_datos:
        print('El archivo', fd, 'está vacío')


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


def sumar_duracion_por_genero(v):
    va = [0] * 6
    for reg in v:
        if reg.idioma == 0:
            va[reg.genero] += reg.duracion
    return va


def mostrar_acumulador(va):
    for i in range(len(va)):
        print(describir_genero(i), ':', va[i], 'minutos')


def contar_por_genero_idioma(v):
    mc = [[0] * 5 for f in range(6)]
    for reg in v:
        mc[reg.genero][reg.idioma] += 1
    return mc


def mostrar_matriz(m):
    for i in range(len(m)):
        fila = describir_genero(i)
        for j in range(len(m[i])):
            if m[i][j] > 0:
                if fila != "":
                    print(fila)
                    fila = ''
                print('\t', describir_idioma(j), ':', m[i][j])


def principal():
    v = leer_csv('series.csv')
    if len(v) > 0:
        ordenar_por_titulo(v)
        opcion = -1
        while opcion != 0:
            mostrar_menu()
            opcion = int(input('Ingrese su opción: '))
            if opcion == 1:
                mostrar_vector(v)
            elif opcion == 2:
                idioma = validar_entre(0, 4, 'Ingrese idioma: ')
                fd = "Series" + str(idioma) + ".dat"
                generar_archivo(v, idioma, fd)
                mostrar_archivo(fd)
            elif opcion == 3:
```

```
                titulo = input('Ingrese título a buscar: ')
                pos = buscar_binario(v, titulo)
                if pos == -1:
                    print('No se encontró esa serie')
                else:
                    print('Serie encontrada!', to_string(v[pos]))
            elif opcion == 4:
                va = sumar_duracion_por_genero(v)
                mostrar_acumulador(va)
            elif opcion == 5:
                mc = contar_por_genero_idioma(v)
                mostrar_matriz(mc)


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

6. Club Deportivo

Un club deportivo necesita procesar los pagos realizados por sus socios.

Para ello, se cuenta con un archivo denominado cuotas.dat, donde cada registro contiene: número de socio, deporte que realiza (0: Natacion/1: Basquet/2: Karate/3: Futbol/ 4: Patin), día del mes en que pagó, valor de la cuota. Si el socio aún no abonó, el día del mes debe ser 0.

Se debe cargar el archivo en un vector y luego implementar un menú con las siguientes opciones:

1. Consulta: mostrar el contenido del vector
2. Cobro: buscar un socio y deporte ingresados por teclado. Si el registro existe, registrar el día y valor de pago (también ingresado por teclado). Si no, agregar un nuevo registro en el vector con esos datos. Informar el cambio realizado
3. Morosos: generar un archivo de texto conteniendo los registros de los socios que aun no pagaron la cuota
4. Totales: indicar cantidad de cobros por cada deporte, y cuál es el deporte con mayor cantidad de participantes
5. Grabar: reemplazar el archivo pagos.dat con el contenido del vector. Mostrar el contenido del archivo.

6.1. registro

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

class Cuota:
    def __init__(self, socio, deporte, dia, valor):
        self.socio = socio
        self.deporte = deporte
        self.dia = dia
        self.valor = valor

def to_string(reg):
    return '{:<6}{:<10}{:>4}  ${:>5}'.format(reg.socio, describir_deporte(reg.deporte), reg.dia,reg.valor)

def to_string_titulos():
    return '{:<6}{:<10}{:>4}{:>8}'.format('Socio', 'Deporte', 'Dia','Valor')

def describir_deporte(cod):
    deportes = ['Natacion','Basquet','Karate','Futbol','Patin']
    return str(cod) + '-' + deportes[cod]
```

6.2. generador.py

```
import pickle
import random

from registro import *


def generar():
    valores = [2000, 850, 700, 1200, 900]
    m = open('cuotas.dat', 'wb')
    for i in range(10):
        socio = i + 1000
        deporte = random.randrange(4)
        dia = random.randint(1, 30)
        if dia > 20:
            dia = 0
        valor = valores[deporte]
        pickle.dump(Cuota(socio, deporte, dia, valor), m)
    m.close()


if __name__ == '__main__':
    generar()
    print('Archivo generado')
```

6.3. principal.py

```
import os
import pickle

from registro import *


def leer_archivo(fd):
    v = []
    if not os.path.exists(fd):
        print('El archivo', fd, 'no existe')
    else:
        m = open(fd, 'rb')
        size = os.path.getsize(fd)
        while m.tell() < size:
            v.append(pickle.load(m))
        m.close()
    return v


def mostrar_vector(v):
    print(to_string_titulos())
    for cuota in v:
        print(to_string(cuota))


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


def mostrar_menu():
    print('-' * 80)
    print('CLUB DEPORTIVO')
    print('1 - Consulta')
    print('2 - Cobro')
    print('3 - Morosos')
    print('4 - Deportes')
    print('5 - Grabar')
    print('0 - Salir')
    op = int(input('Ingrese una opción:'))
    print('-' * 80)
    return op


def validar_entre(minimo, maximo, mensaje):
    num = int(input(mensaje))
    while num < minimo or num > maximo:
        print('INVALIDO!')
        num = int(input(mensaje))
    return num


def contar_por_deporte(v):
    cd = [0] * 5
    for cuota in v:
        cd[cuota.deporte] += 1
    return cd


def buscar_mayor(cd):
    may = 0
    for i in range(1, len(cd)):
        if cd[i] > cd[may]:
            may = i
    return may
```

```
def generar_archivo_morosos(v):
    m = open('morosos.txt', 'wt')
    for cuota in v:
        if cuota.dia == 0:
            m.write(to_string(cuota) + '\n')
    m.close()


def grabar_vector(v, fd):
    m = open(fd, 'wb')
    for cuota in v:
        pickle.dump(cuota, m)
    m.close()


def principal():
    v = leer_archivo('cuotas.dat')

    if len(v) == 0:
        print('El vector no se pudo cargar')
    else:
        opcion = mostrar_menu()
        while opcion != 0:
            if opcion == 1:
                mostrar_vector(v)
            elif opcion == 2:
                socio = int(input('Ingrese socio: '))
                deporte = validar_entre(0, 4, 'Ingrese deporte (0-4): ')
                dia = int(input('Ingrese dia: '))
                valor = int(input('Ingrese monto a cobrar $: '))
                pos = buscar_secuencial(v, socio, deporte)
                if pos == -1:
                    v.append(Cuota(socio, deporte, dia, valor))
                    print('Se agrego un nuevo registro')
                else:
                    v[pos].dia = dia
                    v[pos].valor = valor
                    print('Se registró el pago')
            elif opcion == 3:
                generar_archivo_morosos(v)
                print('Archivo generado')
            elif opcion == 4:
                cd = contar_por_deporte(v)
                may = buscar_mayor(cd)
                print('El deporte con más socios es', describir_deporte(may))
            elif opcion == 5:
                grabar_vector(v, 'cuotas.dat')
            opcion = mostrar_menu()


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