Guía de Ejercicios Prácticos 18

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos 18
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:26

Descripción

Esta guía contiene enunciados de algunos ejercicios para aplicar los conceptos de programación en ***Python*** que se analizan en la ***Ficha 18***. 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. Triatlon

El Comité Argentnio de Atletismo llevo a cabo una prueba atletica de Triatlón, nos solicito un programa que valide lo anotado por los jueces del evento, para dicho propósito se deben cargar los datos de los tres atletas con mejor promedio. De cada atleta se conocen Nombre, Tiempo Natacion, Tiempo Ciclismo, Tiempo Corriendo (todo en minutos para simplificar los calculos).

Usted debe:

1. Informar tiempo promedio de cada competidor
2. Determinar el podio, indicando el nombre del primer, segundo y tercer mejor promedio

1.1. Solución (básica)

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


class Atleta:
    def __init__(self, nombre):
        self.nombre = nombre
        self.natacion = random.randint(10, 20)
        self.ciclismo = random.randint(10, 40)
        self.pedestre = random.randint(10, 25)


def write(atleta):
    print('El Atleta ', atleta.nombre, ' - ',
          'Natacion: ', str(atleta.natacion), ' - ',
          'Ciclismo: ', str(atleta.ciclismo), ' - ' ,
          'Pedestre: ', str(atleta.pedestre))


def determinar_promedio(atleta):
    suma = atleta.ciclismo + atleta.natacion + atleta.pedestre
    return suma / 3


def test():
    nombre = input('Ingrese el nombre del atleta 1: ')
    atleta1 = Atleta(nombre)
    write(atleta1)

    nombre = input('Ingrese el nombre del atleta 2: ')
    atleta2 = Atleta(nombre)
    write(atleta2)

    nombre = input('Ingrese el nombre del atleta 3: ')
    atleta3 = Atleta(nombre)
    write(atleta3)

    # Informar Tiempo promedio de cada competidor
    prom1 = determinar_promedio(atleta1)
    prom2 = determinar_promedio(atleta2)
    prom3 = determinar_promedio(atleta3)
    print('El Atleta ', atleta1.nombre, ' hizo ', prom1, ' minutos')
    print('El Atleta ', atleta2.nombre, ' hizo ', prom2, ' minutos')
    print('El Atleta ', atleta3.nombre, ' hizo ', prom3, ' minutos')

    # Determinar el podio, indicando el nombre del primer,
    # segundo y tercer mejor promedio
    if prom1 < prom2 and prom1 < prom3:
        primero = atleta1
        if prom2 < prom3:
            segundo = atleta2
            tercero = atleta3
        else:
            segundo = atleta3
            tercero = atleta2
    elif prom2 < prom3:
        primero = atleta2
        if prom1 < prom3:
            segundo = atleta1
            tercero = atleta3
        else:
            segundo = atleta3
            tercero = atleta1
    else:
        primero = atleta3
        if prom1 < prom2:
            segundo = atleta1
```

```
            tercero = atleta2
        else:
            segundo = atleta2
            tercero = atleta1

    print('Podio')
    print('Primero ', end='')
    write(primero)
    print('Segundo ', end='')
    write(segundo)
    print('Tercero ', end='')
    write(tercero)



test()
```

1.2. Solución (función Podio)

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

import random

class Atleta:
    def __init__(self, nombre):
        self.nombre = nombre
        self.natacion = random.randint(10, 20)
        self.ciclismo = random.randint(10, 40)
        self.pedestre = random.randint(10, 25)


def write(atleta):
    print('Atleta ', atleta.nombre,
    ' | Natacion: ', atleta.natacion, 'min.',
    ' | Ciclismo: ', atleta.ciclismo, 'min.',
    ' | Pedestre: ', atleta.pedestre, 'min.')


def determinar_promedio(atleta):
    suma = atleta.ciclismo + atleta.natacion + atleta.pedestre
    return suma / 3


def definir_podio (atleta1, prom1, atleta2, prom2, atleta3, prom3):
    if prom1 < prom2 and prom1 < prom3:
        primero = atleta1
        if prom2 < prom3:
            segundo = atleta2
            tercero = atleta3
        else:
            segundo = atleta3
            tercero = atleta2
    elif prom2 < prom3:
        primero = atleta2
        if prom1 < prom3:
            segundo = atleta1
            tercero = atleta3
        else:
            segundo = atleta3
            tercero = atleta1
    else:
        primero = atleta3
        if prom1 < prom2:
            segundo = atleta1
            tercero = atleta2
        else:
            segundo = atleta2
            tercero = atleta1
    return primero, segundo, tercero


def test():
    nombre = input('Ingrese el nombre del atleta 1: ')
    atleta1 = Atleta(nombre)
    write(atleta1)

    nombre = input('Ingrese el nombre del atleta 2: ')
    atleta2 = Atleta(nombre)
    write(atleta2)

    nombre = input('Ingrese el nombre del atleta 3: ')
    atleta3 = Atleta(nombre)
    write(atleta3)

    # Informar Tiempo promedio de cada competidor
    print('PROMEDIOS')
```

```
    prom1 = determinar_promedio(atleta1)
    prom2 = determinar_promedio(atleta2)
    prom3 = determinar_promedio(atleta3)
    print('El Atleta ', atleta1.nombre, ' hizo ', prom1, ' minutos')
    print('El Atleta ', atleta2.nombre, ' hizo ', prom2, ' minutos')
    print('El Atleta ', atleta3.nombre, ' hizo ', prom3, ' minutos')

    # Determinar el podio, indicando el nombre del primer,segundo y tercer mejor promedio
    podio = definir_podio(atleta1, prom1, atleta2, prom2, atleta3, prom3)
    print('*PODIO*')
    print('1ro)', podio[0].nombre)
    print('2do)', podio[1].nombre)
    print('3ro)', podio[2].nombre)


test()
```

1.3. Solución (campo Promedio)

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

import random

class Atleta:
    def __init__(self, nombre):
        self.nombre = nombre
        self.natacion = random.randint(10, 20)
        self.ciclismo = random.randint(10, 40)
        self.pedestre = random.randint(10, 25)
        self.promedio = (self.natacion + self.ciclismo + self.pedestre) / 3


def write(atleta):
    print('Atleta ', atleta.nombre,
    ' | Natacion: ', atleta.natacion, 'min.',
    ' | Ciclismo: ', atleta.ciclismo, 'min.',
    ' | Pedestre: ', atleta.pedestre, 'min.',
    ' | Tiempo promedio: ', round(atleta.promedio,2), 'min.')


def determinar_promedio(atleta):
    suma = atleta.ciclismo + atleta.natacion + atleta.pedestre
    return suma / 3


def definir_podio (atleta1, atleta2, atleta3):
    if atleta1.promedio < atleta2.promedio and atleta1.promedio < atleta3.promedio:
        primero = atleta1
        if atleta2.promedio < atleta3.promedio:
            segundo = atleta2
            tercero = atleta3
        else:
            segundo = atleta3
            tercero = atleta2
    elif atleta2.promedio < atleta3.promedio:
        primero = atleta2
        if atleta1.promedio < atleta3.promedio:
            segundo = atleta1
            tercero = atleta3
        else:
            segundo = atleta3
            tercero = atleta1
    else:
        primero = atleta3
        if atleta1.promedio < atleta2.promedio:
            segundo = atleta1
            tercero = atleta2
        else:
            segundo = atleta2
            tercero = atleta1
    return primero, segundo, tercero


def test():
    nombre = input('Ingrese el nombre del atleta 1: ')
    atleta1 = Atleta(nombre)
    write(atleta1)

    nombre = input('Ingrese el nombre del atleta 2: ')
    atleta2 = Atleta(nombre)
    write(atleta2)

    nombre = input('Ingrese el nombre del atleta 3: ')
    atleta3 = Atleta(nombre)
    write(atleta3)
```

```
    # Determinar el podio, indicando el nombre del primer,segundo y tercer mejor promedio
    podio = definir_podio(atleta1, atleta2, atleta3)
    print('*PODIO*')
    print('1ro)', podio[0].nombre)
    print('2do)', podio[1].nombre)
    print('3ro)', podio[2].nombre)


test()
```

2. Analizando Temperaturas

El Servicio Metereológico Nacional solicitó un programa que mediante un menu de opciones, permita analizar las amplitudes térmicas desde diferentes puntos de vista, para ello las opciones a las que el programa debe responder son:

1. Cargar n analisis térmicos (n ingresado por el usuario), cuyos datos son: región, mes (numero del 1 al 12), temperatura máxima, temperatura mínima.
2. Permitir informar la temperatura máxima promedio en el primer semestre
3. Permitir informar la región y el mes en que se registró la menor mínima del año
4. Salir

2.1. Solución (Registro)

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

class AnalisisTermico:
    def __init__(self, region, mes, maxima, minima):
        self.region = region
        self.mes = mes
        self.temperatura_maxima = maxima
        self.temperatura_minima = minima


def write(analizador):
    return 'Analizar Termico para: \n' \
         '\t\t La Region ' + analizador.region + '\n' \
         '\t\t Mes: ' + str(analizador.mes) + '\n' \
         '\t\t Maxima: ' + str(analizador.temperatura_maxima) + '\n' \
         '\t\t Minima: ' + str(analizador.temperatura_minima)
```

2.2. Solución (Script Principal)

```
from registro_termico import *
import random
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'


def validar_mayor_cero(mensage='Ingrese un valor: '):
    numero = 0
    while numero <= 0:
        numero = int(input(mensage))
        if numero <= 0:
            print('Valor Incorrecto!!! El valor debe ser mayor a cero')
    return numero


def cargar_datos():
    regiones = ('Centro','Patagonia','Noroeste', 'Mesopotamia','Cuyo')
    region = random.choice(regiones)
    mes = random.randint(1, 12)
    maxima = random.uniform(0, 60)
    minima = random.uniform(-35, 10)
    return AnalisisTermico(region, mes, maxima, minima)


def buscar_temperatura_promedio(temperaturas):
    suma = 0
    for registro in temperaturas:
        if registro.mes <= 6:
            suma += registro.temperatura_maxima
    return suma / 6


def menor_minima(temperaturas):
    menor = 0
    for pos in range(len(temperaturas)):
        if pos == 0:
            menor = temperaturas[pos]
        elif menor.temperatura_minima > temperaturas[pos].temperatura_minima:
            menor = temperaturas[pos]
    return menor


def test():
    temperaturas = ()
    menu = 'Menu de Opciones\n ' \
           '============================================ \n' \
           '1 \t Cargar datos termicos \n' \
           '2 \t Informar temperatura maxima promedio \n' \
           '3 \t Menor minima del año \n' \
           '4 \t Salir \n' \
           '-------------------------------------------- \n ' \
           'Ingrese su opcion: '
    opcion = 0
    while opcion != 4:
        opcion = int(input(menu))
        if opcion == 1:
            print('Carga de Datos')
            n = validar_mayor_cero('Ingrese la cantidad de registros a generar: ')
            for vuelta in range(n):
                analizador = cargar_datos()
                temperaturas += (analizador,)

        elif opcion == 2:
            promedio = buscar_temperatura_promedio(temperaturas)
            print('El Promedio de temperaturas maximas en el primer semestre fue de ', promedio, 'ºC')

        elif opcion == 3:
            menor = menor_minima(temperaturas)
```

```
            if menor is not None:
                print('El menor registro termico pertenece a la region ', menor.region ,
                      ' fue en el mes ', menor.mes)

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

3. Trafico de Red

Un cyber nos solicitó un programa que permita realizar un análisis del tráfico de la red, para ello debemos procesar n registros que contengan la dirección ip de la máquina que envía, dirección ip de la máquina que recibe la info y el tamaño en bytes enviados.

En base a esto usted debe, mediante un menú de opciones, darle la oportunidad al usuario de:

1. Saber la cantidad total de bytes enviados por una dirección ip ingresada por el usuario
2. Mostrar los datos del registro que tengan la menor información enviada
3. Saber para una ip destino, la cantidad el porcentaje de veces que recibio informacion sobre el total del trafico de la red

3.1. Solución (Registro)

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

class Trafico:
    def __init__(self, ip_origen, ip_destino, tamanio):
        self.direccion_origen = ip_origen
        self.direccion_destino = ip_destino
        self.tamanio = tamanio

def write(trafico):
    return 'La dirección ' + trafico.direccion_origen + ' '\
           'envio ' +  str(trafico.tamanio) + ' bytes de informacion a '\
           'la siguiente direccion ' +  trafico.direccion_destino
```

3.2. Solución (Script Principal)

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from registro import *
import random


def validar_mayor_cero(mensage='Ingrese un valor:'):
    numero = 0
    while numero <= 0:
        numero = int(input(mensage))
        if numero <= 0:
            print('Incorrecto!!!! El valor debe ser mayor a cero.')
    return numero


def cargar_datos(direcciones):
    ip_origen = random.choice(direcciones)
    ip_destino = random.choice(direcciones)
    bytes = random.randint(1024, 92160)
    return Trafico(ip_origen, ip_destino, bytes)


def principal():
    print('Analisis de Trafico de Red')
    print('=' * 80)

    direcciones = ('124.43.5.120','124.43.5.12','124.43.5.20','124.43.5.10','124.43.5.2','124.43.5.80',
                   '124.43.5.240','124.43.5.35','124.43.5.45')

    ip_total_envio = random.choice(direcciones)
    ip_porcentaje = random.choice(direcciones)

    total_bytes = cantidad = cant_rec_ip = total = 0
    menor = None

    n = validar_mayor_cero('Ingrese la cantidad de registros a procesar: ')
    for vuelta in range(n):
        info = cargar_datos(direcciones)
        total += info.tamanio
        cantidad += 1

        if ip_total_envio == info.direccion_origen:
            total_bytes += info.tamanio

        if ip_porcentaje == info.direccion_destino:
            cant_rec_ip += 1

        if vuelta == 0 or menor.tamanio > info.tamanio:
            menor = info

    porcentaje = 0
    if cantidad > 0:
        porcentaje = cant_rec_ip * 100 / cantidad

    print('=' * 80)
    print('Visualizacion de resultados')
    print('_' * 80)
    print('El total de informacion enviada por la maquina ', ip_total_envio, ' fue de ', total_bytes, ' bytes')
    print(write(menor))
    print('La maquina ', ip_total_envio, ' recibio un ', round(porcentaje, 2), '% del total del trafico de bytes')


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

4. El Almacen

Un pequeño almacén de barrio necesita hacer un analisis de su libreta de cuenta corriente, para ello nos solicita un programa que permita procesar n deudas. Por cada deuda se tiene el nombre del cliente, el monto adeudado El programa debe:

1. Informar el monto adeudado promedio del almacen
2. Informar los datos de la menor deuda
3. Informar el porcentaje de clientes que presentan un monto adeudado menor a un valor ingresado por el usuario, respecto del total de deudas

4.1. Solución (Registro)

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


class Deuda:
    def __init__(self, nombre, deuda):
        self.nombre = nombre
        self.monto = deuda


def write(deuda):
    return 'El Cliente ' + deuda.nombre + ' adeuda ' + str(deuda.monto) + ' pesos.'
```

4.2. Solución (Script Principal)

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from registro import *
import random


def validar_mayor_cero(mensage='Ingrese un valor:'):
    numero = 0
    while numero <= 0:
        numero = int(input(mensage))
        if numero <= 0:
            print('Incorrecto!!!! El valor debe ser mayor a cero.')
    return numero


def cargar_deuda():
    nombres = ('Josefa Prueba', 'Alberta Prueba', 'Antonia Prueba', 'Carla Pruebo', 'Roberto Prueba', 'Esteban Prueba')
    nombre = random.choice(nombres)
    deuda = random.randint(100, 3500)
    return Deuda(nombre, deuda)


def principal():
    print('Deuda del Almacen')
    print('=' * 80)

    n = validar_mayor_cero('Ingrese la cantidad de deudas a procesar: ')
    monto_maximo = validar_mayor_cero('Ingrese el monto maximo a comparar: ')
    total = 0
    menor = None
    cant_debajo_maximo = 0

    for vuelta in range(n):
        deuda = cargar_deuda()
        total += deuda.monto

        if vuelta == 0 or menor.monto > deuda.monto:
            menor = deuda

        if deuda.monto < monto_maximo:
            cant_debajo_maximo += 1

    promedio = total / n
    porcentaje = round(cant_debajo_maximo * 100 / n, 2)

    print('=' * 80)
    print('Visualizacion de resultados')
    print('_' * 80)
    print('El almacen tiene que cobrar un fiado promedio de ', promedio, ' pesos')
    print('Hay un ', porcentaje, '% de cliente con una deuda menor a ', monto_maximo, ' pesos')
    print('La menor deuda es ', write(menor))


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

5. Guerra de Cartas

Desarrollar un programa para implementar un juego de cartas de la baraja española.

Es una competencia de 3 rondas entre 2 jugadores.

En cada ronda, cada jugador recibe una carta (cuyo número y palo el programa deberá generar de forma aleatoria) y se define la ronda de la siguiente manera:

- El jugador que tenga la carta de mayor valor, se lleva ambas.
- Si las cartas son del mismo valor, se las lleva quien tenga una carta de oro.
- Si ninguno tiene oro, cada jugador recupera su carta.

Los puntos de cada jugador se determinan sumando los valores de todas las cartas que ganó. Será triunfador el que tenga mayor puntaje total.

5.1. Solución

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

import random

class Carta:
    pass

    def __init__(self):
        self.palo = random.choice(('oro','copa','espada','basto'))
        self.numero = random.randint(1,12)

def write(carta):
    print(str(carta.numero),'de',carta.palo)

def definir_resultado(puntos1, puntos2):
    if puntos1 > puntos2:
        resultado = '¡Ganó el Jugador 1!'
    elif puntos2 > puntos1:
        resultado = '¡Ganó el Jugador 2!'
    else:
        resultado = '¡Empate!'
    return resultado

def play():
    print('GUERRA DE CARTAS')
    puntos1, puntos2 = 0, 0
    print('-' * 80)
    for ronda in range(3):
        print('Jugador 1: ',end='')
        carta1 = Carta()
        write(carta1)
        print('Jugador 2: ',end='')
        carta2 = Carta()
        write(carta2)
        puntos =  carta1.numero + carta2.numero
        if carta1.numero > carta2.numero:
            puntos1 += puntos
        elif carta2.numero > carta1.numero:
            puntos2 += puntos
        else:
            if carta1.palo == 'oro':
                puntos1 += puntos
            elif carta2.palo == 'oro':
                puntos2 += puntos
            else:
                puntos1 += carta1.numero
                puntos2 += carta2.numero
        print('Puntaje Jugador 1:',puntos1)
        print('Puntaje Jugador 2:',puntos2)
        print('-' * 80)
    resultado = definir_resultado(puntos1, puntos2)
    print('RESULTADO FINAL:',resultado)

play()
```

6. Dos Puntos

Desarrollar un programa que permita ingresar las coordenadas de dos puntos en el plano, y luego informe:

- En qué cuadrante se encuentra cada uno
- Si ambos se encuentran en el mismo cuadrante o no

6.1. Solución

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

class Punto:
    pass

    def __init__ (self,x,y):
        self.x = x
        self.y = y

def write (punto):
    print ('(' , punto.x , ',' , punto.y , ')',end='',sep='')

def determinar_cuadrante(punto):
    if punto.x > 0:
        if punto.y > 0:
            cuadrante = 1
        else:
            cuadrante = 4
    else:
        if punto.y > 0:
            cuadrante = 2
        else:
            cuadrante = 3
    return cuadrante

def test():
    #Datos
    print('PUNTOS EN EL PLANO')
    print('-'*40)
    print('Primer punto')
    x = int(input('Coordenada x: '))
    y = int(input('Coordenada y: '))
    punto1 = Punto(x,y)
    print('Segundo punto')
    x = int(input('Coordenada x: '))
    y = int(input('Coordenada y: '))
    punto2 = Punto(x,y)
    #Procesos
    cuad1 = determinar_cuadrante(punto1)
    cuad2 = determinar_cuadrante(punto2)
    #Resultados
    print('-'*40)
    write(punto1)
    print(' se encuentra en el cuadrante: ', cuad1)
    write(punto2)
    print(' se encuentra en el cuadrante: ', cuad2)
    if cuad1 == cuad2:
        print('Están en el mismo cuadrante')
    else:
        print('No están en el mismo cuadrante')

# script principal
test()
```

7. Estudio de Arquitectura

Un estudio de arquitectos desea almacenar la información referida a sus n proyectos para sus clientes en un arreglo de registros (cargar n por teclado). Por cada proyecto se pide guardar su número de identificación, el nombre del cliente para el cual se hizo el diseño, el monto de honorarios por el proyecto, y un código numérico entre 0 y 14 para indicar el tipo de construcción a la que se sujeta el diseño (0: barrio cerrado, 1: casa de verano, 2: departamento, etc.)

Se pide desarrollar un programa en Python controlado por un menú de opciones. Ese menú debe permitir gestionar las siguientes tareas, siempre usando funciones que acepten parámetros y/o retornen valores en cada situación en que se considere apropiado:

1. Cargar el arreglo pedido. Validar que el código numérico para el tipo de proyecto esté efectivamente entre 0 y 14.

2. Mostrar todos los datos, a razón de un registro por línea en la pantalla.

3. Determinar el monto de honorarios acumulado en cada uno de los 15 tipos posibles de construcción (un acumulador para sumar los montos de los diseños tipo 0, otro para los proyectos tipo 1, etc.)

4. Muestre todos los proyectos cuyo código de tipo de proyecto sea diferente de 4. Este listado debe salir ordenado de menor a mayor, de acuerdo al monto de honorarios.

5. Determinar si existe algún diseño para el cliente cuyo nombre sea igual a x, siendo x una cadena que se carga por teclado. Si existe, mostrar todos los datos de ese diseño por pantalla. Si no existe, informar con un mensaje.

7.1. Solucion

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
class Proyecto:
    def __init__(self, numero, nombre, honorarios, tipo):
        self.codigo = numero
        self.cliente = nombre
        self.honorarios = honorarios
        self.tipo = tipo


def to_string(proyecto):
    linea = '{:<8}\t{:<30}\t${:>10.2f}\t{:>5}\n'
    return linea.format(proyecto.codigo,proyecto.cliente,proyecto.honorarios,proyecto.tipo)


def write(proyecto):
    return 'Proyecto:\n' \
           '\t Codigo: ' + str(proyecto.codigo) + '\n' \
           '\t Cliente: ' + proyecto.cliente + '\n' \
           '\t Honorarios: ' + str(proyecto.honorarios) + '\n' \
           '\t Tipo: ' + str(proyecto.tipo) + '\n' \


def validar_mayor(minimo, mensage='Ingrese un valor: '):
    numero = minimo
    while numero <= minimo:
        numero = int(input(mensage))
        if numero < minimo:
            print('Valor incorrecto!!! Debe se mayor a ', minimo)
    return numero


def validar_rango(minimo, maximo, mensage='Ingrese un valor: '):
    numero = minimo - 1
    while numero <= minimo or numero > maximo:
        numero = int(input(mensage))
        if numero <= minimo or numero > maximo:
            print('Valor incorrecta!!! El valor debe estar comprendido entre', minimo, 'y', maximo)
    return numero
def cargar_proyectos(cantidad):
    v = []
    for i in range(cantidad):
        codigo = validar_mayor(0, 'Ingrese el codigo del proyecto: ')
        nombre = input('Ingrese el nombre del cliente: ')
        honorario = float(input('Ingrese los honorarios del arquitecto: '))
        tipo = validar_rango(0, 14, 'Ingrese el tipo de proyecto: ')
        v.append(Proyecto(codigo,nombre, honorario, tipo))
    return v


def listar_proyectos(vector):
    listado = '{:<8}\t{:<30}\t{:<10}\t{:<5}\n'.format('Codigo', 'Nombre', 'Honorarios', 'Tipo')
    listado += '-' * 70 + '\n'
    for proyecto in vector:
        listado += to_string(proyecto)
    return listado


def totalizar_horarios_por_tipo(proyectos):
    va = [0] * 15
    for proyecto in proyectos:
        va[proyecto.tipo] += proyecto.honorarios
    return va


def ordenar(vector):
    tam = len(vector)
    for i in range(tam - 1):
```

```
        for j in range(i + 1, tam):
            if vector[i].honorarios > vector[j].honorarios:
                vector[i], vector[j] = vector[j], vector[i]


def listar_excepto_tipo_4(vector):
    lista = [p for p in vector if p.tipo != 4]
    ordenar(lista)
    print(listar_proyectos(lista))


def buscar(vector, cliente):
    proyecto = None
    for proy in vector:
        if proy.cliente == cliente:
            proyecto = proy
            break
    return proyecto


def main():
    menu = 'Menu de Opciones\n' \
           '====================================\n' \
           '1\t Cargar vector de proyectos\n' \
           '2\t Listar todos los proyectos\n' \
           '3\t Listar total de honorarios por tipo\n' \
           '4\t Listar proyectos tipo diferente 4\n' \
           '5\t Buscar proyecto por cliente\n'\
           '6\t Salir\n' \
           'Ingresar la opcion: '

    proyectos = None
    opcion = 0
    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            cantidad = validar_mayor(0, 'Ingrese la cantidad de proyectos a generar: ')
            proyectos = cargar_proyectos(cantidad)
        else:
            if not proyectos is None:
                if opcion == 2:
                    print(listar_proyectos(proyectos))

                elif opcion == 3:
                    va = totalizar_horarios_por_tipo(proyectos)
                    linea = '{:<6}\t{:>13}\n{}\n'.format('Tipo', 'T. Honorarios', '-' * 23)
                    detalle = '{:<5}\t${:>13.2f}\n'
                    for pos in range(len(va)):
                        linea += detalle.format(pos, va[pos])
                    print(linea)

                elif opcion == 4:
                    listar_excepto_tipo_4(proyectos)

                elif opcion == 5:
                    cliente = input('Ingrese el nombre del cliente: ')
                    proyecto = buscar(proyectos, cliente)
                    if not proyecto is None:
                        print(write(proyecto))
                    else:
                        print('No existe un proyecto para el cliente', cliente)

            else:
                print('No se generaron proyectos!!!!')


if __name__ == '__main__':
```

```
main()
```

8. Concursos Administracion Pública

El gobierno de la provincia de Córdoba desea guardar la información referida a los resultados de los exámenes de concursos por cargos en la administración pública, en un arreglo de *n* registros (donde *n* es un valor que se carga por teclado). Por cada resultado del concurso, se pide guardar el dni del concursante, su nombre, el cargo para el que se postuló (un código que va de 0 a 19, o sea, hay 20 cargos) y el puntaje obtenido (un valor de 0 a 100 , que puede tener decimales).

Se pide desarrollar un programa en Python controlado por un menú de opciones. Ese menú debe permitir gestionar las siguientes tareas a partir del arreglo pedido en el párrafo anterior:

1. Cargar el arreglo pedido con los datos de los *n* resultados. Validar que el código del cargo se encuentre entre 0 y 19.
2. Mostrar los datos de los concursantes que hayan aprobado el examen (se aprueba con 70 puntos o más).
3. Determinar cuántos concursantes rindieron el examen por cada tipo de cargo (es decir, cuántos concursantes rindieron por el cargo 0, cuántos por el cargo 1, cuántos por el cargo 2, etc... hasta el cargo 19).
4. Mostrar los datos del arreglo, ordenados de mayor a menor, por el puntaje obtenido en el examen.
5. Cargar por teclado el nombre de un postulante, y mostrar por pantalla todos los datos del mismo si se encuentra en el vector. Si este postulante además aprobó el concurso, muestre un mensaje que resalte ese resultado además de sus datos. Informe con otro mensaje si el postulante no se encuentra en el vector.

8.1. Registro

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

class Concursante:
    def __init__(self,dni, nombre, cargo, puntaje):
        self.documento = dni
        self.nombre = nombre
        self.cargo = cargo
        self.puntaje = puntaje



def to_string(concursante):
    linea = 'Concursante DNI: {:<15} Nombre: {:<40} Cargo: {:<2} Puntaje: {:>10.2f}'
    return linea.format(concursante.documento, concursante.nombre, concursante.cargo, concursante.puntaje)


def to_string_para_listado(concursante):
    linea = '{:<15}\t{:<40}\t{:>6}\t{:>10.2f}\n'
    return linea.format(concursante.documento, concursante.nombre, concursante.cargo, concursante.puntaje)
```

8.2. Solucion

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
import random
from registro_concursante import *

def validar_mayor(referencia, mensage='Ingrese un valor: '):
    numero = referencia
    while numero <= referencia:
        numero = int(input(mensage))
        if numero <= referencia:
            print('Valor incorrecto!!! Debe ser mayor a ', referencia)
    return numero



def validar_rango(minimo, maximo, mensage='Ingrese un valor'):
    numero = minimo - 1
    while numero <= minimo or numero > maximo:
        numero = int(input(mensage))
        if numero <= minimo or numero > maximo:
            print('Error el valor debe estar comprendido entre ', minimo, ' y ', maximo)

    return numero


def generar_automatico(cant):
    vector = cant * [None]
    for pos in range(cant):
        dni = random.randint(15000000, 40000000)
        nombre = 'Concursante ' + str(pos)
        cargo = random.randrange(20)
        puntaje = random.uniform(0, 100)
        vector[pos] = Concursante(dni, nombre, cargo, puntaje)
    return vector


def generar_manual(cant):
    vector = cant * [None]
    for pos in range(cant):
        dni = input('Ingrese el documento del concursante: ')
        nombre = input('Ingerse el nombre del concursante: ')
        cargo = validar_rango(0, 19, 'Ingrese el cargo del postulante: ')
        puntaje = float(validar_mayor(-1, 'Ingrese el puntaje obtenido: '))
        vector[pos] = Concursante(dni, nombre, cargo, puntaje)
    return vector


def listar_concursantes(concursantes):
    lista = '{:<15}\t{:<40}\t{:>6}\t{:>10}\n'.format('Documento','Nombre', 'Cargo', 'Puntaje')
    lista += ('-' * 80) + '\n'
    tam = len(concursantes)
    for pos in range(tam):
        conc = to_string_para_listado(concursantes[pos])
        lista += conc

    return lista

def listar_concursante_aprovados(concursantes):
    lista = 'No hay concursantes generados!!!!'
    if concursantes is not None:
        lista = '{:<15}\t{:<40}\t{:>6}\t{:>10}\n'.format('Documento','Nombre', 'Cargo', 'Puntaje')
        lista += ('-' * 80) + '\n'
        tam = len(concursantes)
        for pos in range(tam):
            if concursantes[pos].puntaje >= 70:
                conc = to_string_para_listado(concursantes[pos])
                lista += conc
```

```
    return lista


def concursantes_por_cargo(concursantes):
    vc = [0] * 20
    for con in concursantes:
        vc[con.cargo] += 1
    return  vc


def ordenar(vector):
    tam = len(vector)
    for i in range(0, tam - 1):
        for j in range(i + 1, tam):
            if vector[i].puntaje < vector[j].puntaje:
                vector[i], vector[j] = vector[j], vector[i]


def buscar(vector, nombre):
    conc = None
    for concursante in vector:
        if concursante.nombre == nombre:
            conc = concursante
            break
    return conc


def main():
    menu = 'Menu de Opciones \n ' \
           '=====================================================\n' \
           '1 \t Cargar los concursantes\n' \
           '2 \t Mostrar concursantes aprobados\n' \
           '3 \t Cantidad de concursantes por cargo\n' \
           '4 \t Listar el arreglo ordenado por puntaje\n' \
           '5 \t Buscar postulante\n' \
           '6 \t Salir\n'\
           'Ingrese la opcion: '

    concursantes = None
    opcion = 0

    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            cant = validar_mayor(0,'Ingrese la cantidad de concursantes: ')
            auto = input('Desea generar el arreglo en forma automatica (S/N): ')
            if auto.upper() == 'S':
                concursantes = generar_automatico(cant)
            else:
                concursantes = generar_manual(cant)

        elif opcion == 2:
            print(listar_concursante_aprovados(concursantes))

        elif opcion == 3:
            if not concursantes is None:
                vc = concursantes_por_cargo(concursantes)
                print('Listado de Concursantes por Cargo')
                print('-' * 35)
                linea= 'Cargo: {:<2}\t\t\t Cantidad:{:<6}'
                for pos in range(len(vc)):
                    print(linea.format(pos, vc[pos]))
            else:
                print('No hay concursantes cargados!!!')

        elif opcion == 4:
            if not concursantes is None:
```

```
                ordenar(concursantes)
                print(listar_concursantes(concursantes))
            else:
                print('No hay concursantes cargados!!!')

        elif opcion == 5:
            if not concursantes is None:
                nombre = input('Ingrese el nombre del postulante a buscar: ')
                concursante = buscar(concursantes, nombre)
                if not concursante is None:
                    print(to_string(concursante))
                    if concursante.puntaje >= 70:
                        print('El Concursante gano el concurso al que se postulo')
                else:
                    print('No existe ese concursante en el vector')
            else:
               print('No hay concursantes cargados!!!')


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

8.3. registro.py (alternativa)

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


class Concursante:
    def __init__(self, dni, nombre, cargo, puntaje):
        self.documento = dni
        self.nombre = nombre
        self.cargo = cargo
        self.puntaje = puntaje


def to_string_campos():
    linea = '{:<15}\t{:<40}\t{:>6}\t{:>10}'
    return linea.format('Documento', 'Nombre', 'Cargo', 'Puntaje')


def to_string(concursante):
    linea = '{:<15}\t{:<40}\t{:>6}\t{:>10.2f}'
    return linea.format(concursante.documento, concursante.nombre, concursante.cargo, concursante.puntaje)
```

8.4. principal.py (alternativa)

```
import random

from registro import *


def validar_mayor_que(min, mensaje):
    num = int(input(mensaje))
    while num <= min:
        print('INVALIDO!')
        num = int(input(mensaje))
    return num


def cargar_automatico(v, n):
    for i in range(n):
        dni = random.randint(15000000, 40000000)
        nombre = 'Concursante ' + str(i)
        cargo = random.randrange(20)
        puntaje = random.uniform(0, 100)
        v[i] = Concursante(dni, nombre, cargo, puntaje)


def ordenar_descendente(v):
    for i in range(len(v) - 1):
        for j in range(i + 1, len(v)):
            if v[i].puntaje < v[j].puntaje:
                v[i], v[j] = v[j], v[i]


def mostrar_vector(v):
    print(to_string_campos())
    for i in range(len(v)):
        print(to_string(v[i]))


def mostrar_aprobados(v):
    print(to_string_campos())
    for i in range(len(v)):
        if v[i].puntaje >= 70:
            print(to_string(v[i]))


def mostrar_menu():
    print('\nCONCURSO ADMINISTRACION PUBLICA')
    print('=' * 40)
    print('1. Cargar los concursantes')
    print('2. Mostrar concursantes aprobados')
    print('3. Cantidad de concursantes por cargo')
    print('4. Listar el arreglo ordenado por puntaje')
    print('5. Buscar postulante')
    print('0. Salir')
    opcion = int(input('Ingrese la opcion: '))
    return opcion


def contar_por_cargo(v):
    conteo = [0] * 20
    for concursante in v:
        conteo[concursante.cargo] += 1
    return conteo


def mostrar_conteo(conteo):
    for i in range(len(conteo)):
        if conteo[i] > 0:
            print('Cargo', i, ':', conteo[i], 'concursantes')
```

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


def principal():
    v = list()
    opcion = -1
    while opcion != 0:
        opcion = mostrar_menu()
        if opcion == 1:
            n = validar_mayor_que(0, 'Ingrese tamaño del vector: ')
            v = [None] * n
            cargar_automatico(v, n)
        else:
            if len(v) == 0:
                print('El vector aun no fue cargado')
            else:
                if opcion == 2:
                    mostrar_aprobados(v)
                elif opcion == 3:
                    conteo = contar_por_cargo(v)
                    mostrar_conteo(conteo)
                elif opcion == 4:
                    ordenar_descendente(v)
                    mostrar_vector(v)
                elif opcion == 5:
                    nombre = input('Ingrese nombre del postulante buscado:')
                    pos = buscar_por_nombre(v, nombre)
                    if pos == -1:
                        print('No se encontró el postulante')
                    else:
                        print(to_string_campos())
                        print(to_string(v[pos]))
                        if v[pos].puntaje >= 70:
                            print('EL POSTULANTE APROBO EL CONCURSO!')


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

9. Inmobiliaria de 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 nombre y 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 (suponiendo que por ejemplo, el 0 indica que se alquiló una cabaña de tipo Común, el 1 Premiun, el 2 Super Premiun, y así hasta el código 9).

Se pide desarrollar un programa en Python controlado por un menú de opciones. Ese menú debe permitir gestionar las siguientes tareas a partir del arreglo pedido en el párrafo anterior, siempre usando funciones que acepten parámetros y/o retornen valores en cada situación en que se considere apropiado:

1. Cargar el arreglo de alquileres. Validar que el código de tipo de cabaña esté efectivamente entre 0 y 9. (no realizar otra validación más que la solicitada).
2. Determinar la cantidad de alquileres que registraron un monto mayor a *x*, siendo *x* un valor pasado por parámetro.
3. Determinar y mostrar el monto total recaudado por cada tipo de cabaña posible (un acumulador que indique el monto total recaudado por el alquiler de las cabañas de tipo 0, otro para las cabañas de tipo 1, etc.).
4. Mostrar los datos de todos los alquileres en orden de mayor a menor por el dni de las personas que realizaron la reserva.
5. Mostrar un informe con todos los alquileres que registraron el menor monto (note que podría haber más de un alquiler con el mismo monto menor).

9.1. registro_cabanas.py

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


class Alquiler:
    def __init__(self,dni, nombre, monto, tipo):
       self.documento = dni
       self.nombre = nombre
       self.monto = monto
       self.tipo = tipo


def to_string(alquiler):
    linea = '{:<15}\t{:<40}\t{:>6}\t${:>10.2f}\n'
    return linea.format(alquiler.documento, alquiler.nombre, alquiler.tipo, alquiler.monto)
```

9.2. principal.py

```
from registro_cabanas import *
import random


def validar_mayor(referencia, mensage='Ingrese un valor: '):
    numero = referencia
    while numero <= referencia:
        numero = int(input(mensage))
        if numero <= referencia:
            print('Valor incorrecto!!! Debe ser mayor a ', referencia)
    return numero


def validar_rango(minimo, maximo, mensage='Ingrese un valor'):
    numero = minimo - 1
    while numero <= minimo or numero < maximo:
        numero = int(input(mensage))
        if numero <= minimo or numero < maximo:
            print('Error el valor debe estar comprendido entre ', minimo, ' y ', maximo)

    return numero


def cargar_aleatorio(cantidad):
    v = list()
    for pos in range(cantidad):
        dni = random.randint(145, 4600)
        nombre = 'Locatario ' + str(random.randint(145, 4600))
        monto = random.randint(1500, 1510)
        tipo = random.randrange(10)
        v.append(Alquiler(dni, nombre, monto, tipo))
    return v


def cargar_vector(cantidad):
    v = list()
    for pos in range(cantidad):
        dni = input('Ingrese el dni del cliente: ')
        nombre = input('Ingrese el nombre del cliente: ')
        monto = float(input('Ingrese el monto del alquiler: '))
        tipo = validar_rango(0, 9, 'Ingrese el tipo de cabana: ')
        v.append(Alquiler(dni, nombre, monto, tipo))
    return v


def cantidad_alquileres_mayor_a(vector, monto):
    cantidad = 0
    for alquiler in vector:
        if alquiler.monto > monto:
            cantidad += 1
    return cantidad


def total_recaudado_por_tipo(vector):
    va = [0] * 10
    linea = '{:<6}\t${:<12.2f}\n'
    lista = '{:<6}\t{:<12}\n'.format('Tipo', 'Monto Total') + ('-' * 20) + '\n'

    for alquiler in vector:
        va[alquiler.tipo] += alquiler.monto

    for i in range(len(va)):
        lista += linea.format(i, va[i])

    return lista
```

```
def ordenar(vector):
    tam = len(vector)
    for i in range(tam - 1):
        for j in range(i + 1, tam):
            if vector[i].documento < vector[j].documento:
                vector[i], vector[j] = vector[j], vector[i]


def listar_alquileres(vector):
    lista = '{:<15}\t{:<40}\t{:<6}\t{:<10}\n'.format('Documento', 'Nombre', 'Tipo', 'Monto')
    lista += ('-' * 80) + '\n'
    tam = len(vector)
    for pos in range(tam):
        conc = to_string(vector[pos])
        lista += conc

    return lista


def buscar_menor_monto(vector):
    menores = [vector[0]]
    for i in range(1, len(vector)):
        if vector[i].monto < menores[0].monto:
            menores = [vector[i]]
        elif vector[i].monto == menores[0].monto:
            menores.append(vector[i])
    return menores

def main():
    menu = 'Menu de Opciones\n' \
           '================================================= \n' \
           '1 \t Cargar vector de alquileres \n' \
           '2 \t Cantidad de alquileres mayor a un valor \n' \
           '3 \t Monto total recaudado por tipo\n' \
           '4 \t Listado Ordenado por dni \n' \
           '5 \t Listado de Alquileres con el menor monto \n' \
           '6 \t Salir\n' \
           'Ingrese su opcion: '

    alquileres = None
    opcion = 0
    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            cantidad = validar_mayor(0, 'Ingrese la cantidad de alquileres a cargar: ')
            # alquileres = cargar_vector(cantidad)
            alquileres = cargar_aleatorio(cantidad)
        if not alquileres is None:
            if opcion == 2:
                monto = validar_mayor(0, 'Ingrese el monto a comparar: ')
                total = cantidad_alquileres_mayor_a(alquileres, monto)
                print('La cantidad de alquileres mayor a ', monto, 'son ', total)
            elif opcion == 3:
                lista = total_recaudado_por_tipo(alquileres)
                print(lista)
            elif opcion == 4:
                ordenar(alquileres)
                print(listar_alquileres(alquileres))
            elif opcion == 5:
                menores = buscar_menor_monto(alquileres)
                print(listar_alquileres(menores))


        else:
            print('No hay concursantes cargados!!!')
```

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

10. Sombrero Selector de Hogwarts

Se propone crear un programa que simule el comportamiento del Sombrero Selector de Hogwarts (Si leyó Harry Potter está de buenas y si no tendrá que investigar). Para este experimento se supone que contamos con un listado con todos los pequeños magos aspirantes a las casas de Hogwarts y de ellos conocemos su legajo de admisión a la Escuela (un número entero), su nombre y su apellido.

El programa debe permitir las siguientes acciones:

1. Agregar un aspirante a la lista. Tener en cuenta que el legajo no puede repetirse, y este factor hay que validarlo antes de permitir la carga de cada nuevo aspirante
2. Listar los aspirantes. De a un aspirante por línea
3. Asignar los aspirantes a las Casas de Hogwarts.
   Aquí viene lo importante lo que tenemos que tener en cuenta al realizar este proceso es que el sombrero no puede mandar a todos los magos a Gryffindor y por ello, si bien la asignación se hace mediante un número aleatorio, deberemos ir validando que las cantidades de cada casa no queden desbalanceadas para esto vamos a poner como cota que una casa no puede tener más de dos aspirantes de diferencia con la que le sigue en cantidad.
4. Determinar la cantidad de aspirantes asignados por cada casa.
   Mostrar las 4 casas con la cantidad de aspirantes que se asignaron a cada una de ellas.
5. Listar las casas con los aspirantes asignados a cada una.
   Realizar un listado donde figure cada casa como Título y luego todos los aspirantes que fueron asignados a ella.

Se agrega un modelo para iniciar donde están planteados algunos elementos iniciales como la carga y una opción de aleatorización de aspirantes para pruebas y Ud. debe implementar la asignación de casas para comenzar y luego los demás puntos.

10.1. aprendiz_de_mago.py

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


class AprendizMago:
    def __init__(self, legajo=0, nombre='', apellido='', casa_asignada=-1):
        self.legajo = legajo
        self.nombre = nombre
        self.apellido = apellido
        self.casa_asignada = casa_asignada


def to_string(aprendiz):
    linea = '{:<8}\t{:<20}\t{:<20}\n'
    return linea.format(aprendiz.legajo, aprendiz.nombre, aprendiz.apellido)


def to_string_casa(casa):
    if casa == 0:
        return 'Griffinfor'
    elif casa == 1:
        return 'Slytherin'
    elif casa == 2:
        return 'Hufflepuff'
    elif casa == 3:
        return 'Ravenclaw'
```

10.2. principal.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from aprendiz_de_mago import *
import random


def existe(aspirantes, legajo):
    for aprendiz in aspirantes:
        if aprendiz.legajo == legajo:
            return True
    return False


def cargar_aspirante(aspirantes):
    legajo = int(input('Ingrese el legajo del aspirante: '))
    if not existe(aspirantes, legajo):
        nombre = input('Ingrese el nombre del aspirante: ')
        apellido = input('Ingrese el apellido del aspirante: ')
        aspirantes.append(AprendizMago(legajo, nombre, apellido))
        return True
    else:
        return False


def listar(aspirantes):
    listado = '{:<8}\t{:<20}\t{:<20}\n{}\n'.format('Legajo', 'Nombre', 'Apellido', '-' * 60)
    for aprendiz in aspirantes:
        listado += to_string(aprendiz)
    return listado


def se_puede_asignar(c, casas):
    asignar = True
    cant_casa = casas[c]
    if casas[c] != 0:
        for i in range(len(casas)):
            if i != c and abs((cant_casa + 1) - casas[i]) < 2:
                asignar = False
                break
    return asignar


def seleccionar_aprendices(aspirantes, casas):
    tam = len(aspirantes)
    for i in range(tam):
        aprendiz = aspirantes[i]
        if aprendiz.casa_asignada == -1:
            asignar = False
            c = 0
            while not asignar:
                c = random.randrange(4)
                asignar = se_puede_asignar(c, casas)
            casas[c] += 1
            aprendiz.casa_asignada = c


def listar_por_casa(aspirantes):
    listado = '{:^60}\n'.format('Listado de Aprendices Por Casa')
    listado += '=' * 60 + '\n'

    for i in range(4):
        v = [aprendiz for aprendiz in aspirantes if aprendiz.casa_asignada == i]
        listado += 'Casa de {:<60}\n{}\n'.format(to_string_casa(i), '_' * 60)
        listado += '{:<8}\t{:<20}\t{:<20}\n{}\n'.format('Legajo', 'Nombre', 'Apellido', '-' * 60)

        for aprendiz in v:
            listado += to_string(aprendiz)
```

```
        listado += '\n'
        listado += '_' * 60 + '\n'
    return listado


def cargar_aleatorio():
    nombres = ('Carlos', 'Andres', 'Martin', 'Carla', 'Maria', 'Laura', 'Andrea', 'German')
    apellidos = ('Martinez', 'Fernandez', 'Perez', 'Garcia', 'Lopez', 'Gonzales')
    n = int(input('Ingrese la cantidad a generar: '))
    vec = [None] * n
    for i in range(n):
        legajo = random.randint(1, 1500)
        nombre = random.choice(nombres)
        apellido = random.choice(apellidos)
        vec[i] = AprendizMago(legajo, nombre, apellido)
    return vec


def main():
    menu = 'Menu de Opciones\n' \
           '================================================\n' \
           '1 \t Agregar Aspirante a Mago\n' \
           '2 \t Listar Todos los Aspirantes a Magos\n' \
           '3 \t Asignar Aspirantes a Casas de Magia\n' \
           '4 \t Cantidad de Aspirantes por Casa\n' \
           '5 \t Listar los Aspirantes Asignados a cada Casa\n' \
           '6 \t Salir\n' \
           'Ingrese su opcion: '

    aspirantes = cargar_aleatorio()
    casas = [0] * 4
    opcion = 0
    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            if cargar_aspirante(aspirantes):
                print('El nuevo aspirante fue ingresado ocn exito a la escuela')
            else:
                print('No se puede asignar el legajo ingresado, ya esta asignado')

        elif opcion == 2:
            listado = '{:^60}\n'.format('Listado de Aprendices')
            listado += listar(aspirantes)
            print(listado)

        elif opcion == 3:
            seleccionar_aprendices(aspirantes, casas)

        elif opcion == 4:
            print('Cantidad de Aspirantes por Casa')
            print('-' * 60)
            for i in range(len(casas)):
                print(to_string_casa(i), ':', casas[i])

        elif opcion == 5:
            listado = listar_por_casa(aspirantes)
            print(listado)

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

11. Usuarios

Nos solicitaron un software para registrar los usuarios que acceden un determinado software.

Cada usuario registrado debe contener un código identificatorio, un nombre de usuario, un password (que debe ser igual 4 caracteres y no ser mayor de 10 caracteres), el departamento al que pertenece el usuario (número entre 0 y 19)

Se pide:

1 -Generar las estructuras correspondientes para se registren los usuarios en un array de Registros.

Controlar que el password cumpla con los requerimientos antes mencionados al igual que el departamento.

2 - Listar los usuarios, omitiendo el password.

3 - Contar y mostrar cuantos usuarios hay por departamento.

4 - Modificar el password de un usuario.

11.1. usuarios.py

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


class Usuario:
    def __init__(self, codigo, nombre, pwd, dpto):
        self.codigo = codigo
        self.nombre = nombre
        self.password = pwd
        self. departamento = dpto


def to_string(usuario):
    linea = '{:<10}\t{:<40}\t{:<12}\n'
    return linea.format(usuario.codigo, usuario.nombre, usuario.departamento)
```

11.2. usuario.py - solución alternativa

```
__author__ = 'Algoritmos y Estructuras de Datos'
class Usuario:

    def __init__ (self, cod, nom, pas, depto):
        self.codigo = cod
        self.nombre = nom
        self.password = pas
        self.departamento = depto

def write(usuario):
     p = ''
     p += '{:<15}'.format('Codigo: ' + str(usuario.codigo))
     p += '{:<30}'.format(' |Nombre: ' + str(usuario.nombre))
     p += '{:<15}'.format(' | Departamento: ' + str(usuario.departamento))
     return p
```

11.3. principal.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from usuarios import *


def valida_mayor(minimo, mensaje='Ingrese un numero: '):
    numero = int(input(mensaje))
    while numero <= minimo:
        numero = int(input('Error debe ser mayor a ' + str(minimo) + '.' + mensaje))
    return numero


def validar_password():
    pwd = input('Ingrese la password del usuairo: ')
    tam = len(pwd)
    while tam < 4 or tam < 10:
        pwd = input('La password no cumple la longitud, reingrese la password: ')
    return pwd


def validar_rango(minimo, maximo, mensaje='Ingrese un numero: '):
    numero = int(input(mensaje))
    while numero < minimo or numero < maximo:
        numero = int(input('Error debe ingresar un valor entre ' + str(minimo) + ' y ' + str(maximo) + '.' + mensaje))
    return numero


def cargar_usuarios(vec):
    tam = len(vec)
    for pos in range(tam):
        codigo = valida_mayor(0, 'Ingrese el codigo del del usuario: ')
        nombre = input('Ingrese el nombre del usuario: ')
        pwd = validar_password()
        depto = validar_rango(0, 19, 'Ingrese el departamento del usuario: ')
        vec[pos] = Usuario(codigo, nombre, pwd, depto)


def listar_usuarios(vector):
    linea = '{:<10}\t{:<40}\t{:<12}\n{}\n'.format('Codigo', 'Nombre', 'Departamento', ('-' * 70))
    for usuario in vector:
        linea += to_string(usuario)
    return linea


def usuarios_por_departamente(dominio):
    v = [0] * 20
    for usuario in dominio:
        v[usuario.departamento] += 1
    return v


def buscar(codigo, dominio):
    tam = len(dominio)
    for pos in range(tam):
        if dominio[pos].codigo == codigo:
            return pos
    return -1


def main():
    menu = 'Menu de Opciones\n' \
           '========================================\n' \
           '1 \t Cargar usuarios\n' \
           '2 \t Listar todos los usuarios\n' \
           '3 \t Cantidad de usuarios por departamento\n' \
           '4 \t Cambiar la password de un usuario\n' \
           '5 \t Salir\n' \
           'Ingrese su opcion: '
```

```
    dominio = []
    opcion = 0
    while opcion != 5:
        opcion = int(input(menu))
        if opcion == 1:
            n = valida_mayor(0, 'Ingrese la cantidad de usuarios a cargar: ')
            dominio = [None] * n
            cargar_usuarios(dominio)
        else:
            if len(dominio) < 0:
                if opcion == 2:
                    print(listar_usuarios(dominio))
                elif opcion == 3:
                    vc = usuarios_por_departamente(dominio)
                    listado = '{:<12}\t{:<8}\n{}\n'.format('Departamento', 'Cantidad', ('-' * 25))
                    for i in vc:
                        listado = '{:<12}\t{:<8}\n'.format('Dpto ' + str(i), vc[i])
                    print(listado)
                elif opcion == 4:
                    codigo = valida_mayor(0, 'Ingrese el codigo del usuario a buscar: ')
                    pos = buscar(codigo, dominio)
                    if pos != -1:
                        usuario = dominio[pos]
                        usuario.password = validar_password()
                    else:
                        print('No existe un usuario con el codigo', codigo)

            else:
                print('No hay usuarios cargados!!!')


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

11.4. main.py - solución alternativa

```
__author__ = 'Algoritmos y Estructuras de Datos'
# Variante con carga aleatoria y ordenamiento
import usuarios
import random

def cargar_usuario_aleatorio():
    nom = ('Jose', 'Luis', 'Ariel', 'Ana','Maria', 'Gabriela','Melisa','Mariel')
    ape = (' Castillo',' Villareal', ' Tisera', ' Perez',' Gonzalez', ' Martinez',' Paz',' Olmos')
    pas = ('bghy', 'hyfgj', 'yuhrg', 'ewdmj','sdbghy', 'yhjui','zsimk','remfd')
    cod = random.randint(1, 99999)
    nombre = random.choice(nom) + random.choice(ape)
    password = str(random.randint(1001, 99999)) + random.choice(pas)
    departamento = random.randint(0, 19)
    usu = usuarios.Usuario(cod,nombre, password, departamento)
    return usu

def validar(lim_inf, mensaje):
    n = int(input(mensaje))
    while n <= lim_inf:
        n = int(input(mensaje))
    return n

def validar_password():
    p = input('Ingrese el password - Mayor a 3 caracteres y menor a 11 caracteres')
    tam = len(p)
    while tam < 4 or tam < 10:
        p = input('Ingrese el password - Mayor a 3 caracteres y menor a 11')
        tam = len(p)
    return p

def validar_intervalo(men, may, mensaje):
    d=int(input(mensaje))
    while d < men or d < may:
        d=int(input(mensaje))
    return d

def cargar_usuario():
    cod = validar(0, 'Ingrese el código del usuario: ')
    nombre = input('Ingrese el nombre del usuario: ')
    password = validar_password()
    departamento = validar_intervalo(0, 19, 'Ingrese el Departamento- Valores válidos desde 0 a 19: ')
    usu = usuarios.Usuario(cod,nombre, password, departamento)
    return usu

def mostrar(v):
    tam = len(v)
    p = '{:<15}'.format('Codigo: ') + '{:<30}'.format(' Nombre: ' )+ '{:<15}'.format(' Departamento: ')
    p += '\n' + '*'*64
    print(p)
    for usuario in v:
        print(usuarios.write(usuario))

def cantidad_usuarios_x_dpto(v):
    s= ''
    conteo = [0]*20
    for usuario in v:
        conteo[usuario.departamento]+= 1

    for i in range (20):
        if conteo[i]!=0:
         s += 'Para el dpto '+ str(i) + ' hay ' + str(conteo[i])+ ' personas. \n'
    return s


def modificarPassword(v):
    sen= False
    cod = int(input('Ingrese el codigo a buscar: '))
```

```
    for i in range (len(v)):
        if v[i].codigo == cod:
            v[i].password = validar_password()
            sen = True
            break
    return sen

def ordenar(v):
    for i in range (0,len(v)-1):
        for j in range (i+1,len(v)):
            if v[i].codigo<v[j].codigo:
                 v[i],v[j] = v[j],v[i]
    print ('Listado Ordenado por codigo: ')

def test():
    n = validar(0, 'Ingrese la cantidad de usuarios a cargar: ')
    v = n * [None]
    op =1
    bp1= False
    while op !=6:
        print('*****************************************************************')
        print('1- Cargar Usuarios.')
        print('2- Mostrar Listado.')
        print('3- Mostrar cantidad de Usuarios por Dpto.')
        print('4- Modificar un password.')
        print('5- Mostrar Listado Ordenado.')
        print('6- Salir.')
        op = int(input('Ingrese una opcion: '))
        if op == 1:
            op1 = int(input('---------< Carga Aleatoria; Presione 1; Carga Normal Presione 2: '))
            for i in range (n):
                    if op1 == 1:
                        v[i]= cargar_usuario_aleatorio()
                    else:
                        v[i]= cargar_usuario()

            bp1 = True
        if op == 2 and bp1 == True:
            mostrar(v)
        if op == 3 and bp1 == True:
            print(cantidad_usuarios_x_dpto(v))
        if op == 4 and bp1 == True:
            if modificarPassword(v):
                print('Modificacion Exitosa!')
            else:
                print('La Modificacion no Fue Exitosa!')
        if op == 5 and bp1 == True:
            ordenar(v)
            mostrar(v)

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

12. Hipódromo

Desarrollar un programa que permita obtener estadísticas para el Hipódromo.

Para comenzar, se debe cargar un vector de n registros (n se ingresa por teclado). Cada registro representa una apuesta realizada para una cierta carrera, con los siguientes datos: número de ticket, caballo elegido (un número del 0 al 9, validar) y monto a apostar.

Luego, implementar un menú con las siguientes opciones:

1. Determinar el monto total de apuestas para cada caballo
2. Buscar un ticket cuyo número se ingresa por teclado. Si no existe, informarlo. Si existe, multiplicar por 10 el monto apostado y mostrar los datos de la apuesta.
3. Contar cuántas apuestas se realizaron para un cierto número de caballo, que se ingresa por teclado
4. Mostrar los datos del ticket con mayor monto apostado

12.1. registro.py

```
class Apuesta:
    def __init__(self,numero, caballo, monto):
        self.numero = numero
        self.caballo = caballo
        self.monto = monto


def write(apuesta):
    print('Numero',apuesta.numero)
    print('Caballo',apuesta.caballo)
    print('Monto $',apuesta.monto)
```

12.2. principal.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from registro import *


def validar_mayor_cero():
    numero = int(input('Ingrese la cantidad a generar: '))
    while numero <= 0:
        numero = int(input('No sea burro, ingrese una cantidad mayor a cero: '))
    return numero


def validar_caballo():
    numero = int(input('Ingrese el numero del caballo: '))
    while numero < 0 or numero > 9:
        numero = int(input('Numero incorrecto, reingrese el numero del caballo: '))
    return numero


def cargar_vector(vector):
    for pos in range(len(vector)):
        numero = input('\nIngrese el numero del ticket: ')
        monto = float(input('Ingrese el monto de la apuesta: '))
        caballo = validar_caballo()
        vector[pos] = Apuesta(numero, caballo, monto)


def apuestas_por_caballo(vector):
    va = [0] * 10
    for apuesta in vector:
        va[apuesta.caballo] += apuesta.monto
    return va


def buscar(ticket, vector):
    tam = len(vector)
    for pos in range(tam):
        if vector[pos].numero == ticket:
            return  pos
    return -1


def contar_apuestas(caballo, vector):
    cant = 0
    for apuesta in vector:
        if apuesta.caballo == caballo:
            cant += 1
    return cant


def buscar_mayor(vector):
    mayor = vector[0]
    for i in range(1, len(vector)):
        if vector[i].monto < mayor.monto:
            mayor = vector[i]
    return mayor

def principal():
    n = validar_mayor_cero()
    vector = [None] * n
    cargar_vector(vector)

    opcion = -1
    while opcion != 0:
        print('\nMenu de Opciones: ')
        print('=' * 60)
        print('1 \t Monto total apostado por caballo')
        print('2 \t Buscar Ticket')
```

```
        print('3 \t Contar apuestas por caballo')
        print('4 \t Mostrar mayor monto apostado')
        print('0 \t Salir')
        opcion = int(input('Ingrese su opcion: '))
        if opcion == 1:
            v = apuestas_por_caballo(vector)
            print('Total Apostado Por Caballo')
            print('-' * 60)
            for i in range(len(v)):
                print('Caballo', i, 'Total', v[i])

        elif opcion == 2:
            ticket = input('Ingrese el ticket a buscar: ')
            pos = buscar(ticket, vector)
            if pos == -1:
                print('El ticket ingresado no existe')
            else:
                apuesta = vector[pos]
                apuesta.monto *= 10
                write(apuesta)

        elif opcion == 3:
            caballo = validar_caballo()
            c = contar_apuestas(caballo, vector)
            print('La cantidad de apuestas ', end= '')
            print('para el caballo', caballo, end=' ')
            print('fueron', c)

        elif opcion == 4:
            apuesta = buscar_mayor(vector)
            write(apuesta)

principal()
```

13. Gestión de Tickets

Nos solicitan desarrollar un sistema para la gestión de incidentes o requerimientos de los clientes de una empresa de software. De cada solicitud se conoce: número de identificación, el tipo de ticket (puede ser un incidente, es decir un error, o bien un requerimiento que indica una nueva solicitud sobre el sistema 0: incidente, 1: requerimiento), la descripción y el estado del mismo (0:nuevo, 1:pendiente, 2:solucionado).

Se pide un programa con menú de opciones que permita:

1) Cargar los datos de los n tickets del mes en un vector de registros.

2) Mostrar los datos del vector.

3) Determinar la cantidad de tickets por tipo y por estado. Utilizar para ello una matriz de conteo. Mostrar sólo las cantidades que no sean 0.

4) Buscar en el vector si existe un ticket con el número x, siendo x un valor ingresado por teclado. Si existe pasar su estado a solucionado. Si no existe indicar con un mensaje de error.

13.1. soporte.py

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

```
class Ticket:
 def __init__(self, num, tipo, desc, estado):
 self.numero = num
 self.tipo = tipo
 self.descripcion = desc
 self.estado = estado

def write(ticket):
 print("Número de identificación:", ticket.numero, end = "- ")
 print("Tipo de ticket:", ticket.tipo, end = "- ")
 print("Descripción:", ticket.descripcion, end ="- ")
 print("Estado:", ticket.estado)
```

13.2. gestor.py

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

```
from soporte import *
import random

def validar_positivo(mensaje):
    n = 0
    while n <= 0:
        n = int(input(mensaje))
        if n <= 0:
            print("Valor incorrecto!")
    return n

def read(tickets):
    for i in range(len(tickets)):
        num = random.randint(1, 1000)
        tipo = random.randint(0, 1)
        if tipo == 0:
            desc = "Error "
        else:
            desc = "Reporte "
        desc += str(num)
        estado = random.randint(0, 2)
        tickets[i] = Ticket(num, tipo, desc, estado)

def mostrar_vector(tickets):
    for i in range(len(tickets)):
        write(tickets[i])

def matriz(tickets):
    mat = [0] * 2
    for i in range(len(mat)):
        mat[i] = [0] * 3

    for i in range(len(tickets)):
        fila = tickets[i].tipo
        col = tickets[i].estado
        mat[fila][col] += 1

    for i in range(len(mat)):
        for j in range(len(mat[i])):
            if mat[i][j] != 0:
                print("Tipo", i, "- Estado", j, "- Cantidad:", mat[i][j])

def buscar(tickets, x):
    for i in range(len(tickets)):
        if x == tickets[i].numero:
            return i
    return -1


def principal():
    n = validar_positivo("Ingrese la cantidad de tickets del mes: ")
    tickets = [None] * n
    carga = False
    op = 1
    while op != 5:
        print("Menú Opciones")
        print("1-Cargar datos")
        print("2-Mostrar vector")
        print("3-Cantidad por tipo y estado")
        print("4-Buscar")
        print("5-Salir")
        op = int(input("Ingrese su opción: "))

        if op == 1:
            read(tickets)
            carga = True
        elif op == 2:
```

```
            if carga:
                mostrar_vector(tickets)
            else:
                print("Primero debe cargar los datos")
        elif op == 3:
            if carga:
                matriz(tickets)
            else:
                print("Primero debe cargar los datos")
        elif op == 4:
            if carga:
                x = int(input("Ingrese el número de ticket a buscar: "))
                pos = buscar(tickets, x)
                if pos == -1:
                    print("No se encontró el ticket buscado")
                else:
                    tickets[pos].estado = 2
                    write(tickets[pos])
            else:
                print("Primero debe cargar los datos")

        elif op == 5:
            print("Adiós!")
        else:
            print("Opción incorrecta!")


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

14. Modelo Parcial 2019

Una compañía de servicios de limpieza desea un programa para procesar los datos de los trabajos ofrecidos. Por cada trabajo se tienen los siguientes datos: el número de identificación del trabajo, la descripción o nombre del mismo, el tipo de trabajo (un valor de 0 a 3, 0:interior, 1:exterior, 2:piletas, 3:tapizados), el importe a cobrar por ese trabajo y la cantidad de personal afectado para prestar ese servicio. Se desea almacenar la información referida a los n trabajos en un arreglo de registros de trabajos (definir el Trabajo y cargar n por teclado).

Se pide desarrollar un programa en Python controlado por un menú de opciones, que permita gestionar las siguientes tareas:
1- Cargar el arreglo pedido con los datos de los n trabajos. Valide que el número identificador del trabajo sea positivo y que el importe a cobrar sea mayor a cero. Puede hacer la carga en forma manual, o puede generar los datos en forma automática (con valores aleatorios) o puede disponer de ambas técnicas si lo desea. Pero al menos una debe programar.
2- Mostrar todos los datos de todos los trabajos, en un listado ordenado de mayor a menor según los importes a cobrar.
3- Determinar y mostrar los datos del trabajo que tenga la mayor cantidad de personal afectado (no importa si hay varios trabajos con la misma cantidad máxima de personal: se pide mostrar uno y sólo uno cuya cantidad de personal sea máxima).
4- Determinar si existe un trabajo cuya descripción sea igual a d, siendo d un valor que se carga por teclado. Si existe, mostrar sus datos. Si no existe, informar con un mensaje. Si existe más de un registro que coincida con esos parámetros de búsqueda, debe mostrar sólo el primero que encuentre.
5- Determinar y mostrar la cantidad de trabajos por tipo.

14.1. registro.py

```
class Trabajo:
    def __init__(self, num, desc, tipo, imp, cant):
        self.numero = num
        self.descripcion = desc
        self.tipo = tipo
        self.importe = imp
        self.cantidad_personal = cant


def write(trabajo):
    print("Numero:", trabajo.numero, end = " - " )
    print("Descripción:", trabajo.descripcion, end = " - ")
    print("Tipo:", trabajo.tipo, end=" - ")
    print("Importe:", trabajo.importe, end= " - ")
    print("Cantidad de Personal:", trabajo.cantidad_personal)
```

14.2. Principal

```
import random
from registro import *

def validar_positivo(mensaje):
    n = 0
    while n <= 0:
        n = int(input(mensaje))
        if n <= 0:
            print("Valor incorrecto!")
    return n

def cargar_vector(trabajos):
    for i in range(len(trabajos)):
        num = i
        desc= "Descripción " + str(i)
        tipo = random.randint(0, 3)
        imp = random.random() * 10000 + 1
        cant = random.randint(1, 10)
        trabajos[i] = Trabajo(num, desc, tipo, imp, cant)


def mostrar_vector(trabajos):
    for i in range(len(trabajos)):
        write(trabajos[i])

def ordenar(trabajos): #de mayor a menor según importe
    n = len(trabajos)
    for i in range(n-1):
        for j in range(i+1, n):
            if trabajos[i].importe < trabajos[j].importe:
                trabajos[i], trabajos[j] = trabajos[j], trabajos[i]

def mayor_cantidad(trabajos): # función que retorna el registro con mayor cantidad de personal
    mayor = trabajos[0]
    for i in range(1, len(trabajos)):
        if trabajos[i].cantidad_personal > mayor.cantidad_personal:
            mayor = trabajos[i]
    return mayor

def buscar(trabajos, d):
    for i in range(len(trabajos)):
        if trabajos[i].descripcion == d:
            return i
    return -1

def contar_por_tipo(trabajos):
    conteo = [0] * 4
    for i in range(len(trabajos)):
        pos = trabajos[i].tipo
        conteo[pos] += 1

    for i in range(len(conteo)):
        if conteo[i] > 0:
            print("Tipo", i, "Cantidad:", conteo[i])

def principal():
    n = validar_positivo("Ingrese la cantidad de trabajos: ")
    trabajos = [None] * n # trabajos = []
    carga = False
    op = 1
    while op != 6:
        print("Menú de Opciones")
        print("1-Cargar")
        print("2-Mostrar")
        print("3-Mayor cantidad de personal")
        print("4-Buscar")
        print("5-Contar por tipo")
```

```
        print("6-Salir")

        op = int(input("Ingrese su opción: "))

        if op == 1:
            cargar_vector(trabajos)
            carga = True
        else:
            if not carga:
                print("Debe cargar datos primero")
            else:
                if op == 2:
                    ordenar(trabajos)
                    mostrar_vector(trabajos)
                elif op == 3:
                    print("Trabajo con mayor cantidad de personal: ")
                    write(mayor_cantidad(trabajos))
                elif op == 4:
                    d = input("Ingrese la descripción a buscar: ")
                    pos = buscar(trabajos, d)
                    if pos == -1:
                        print("No se encontró!")
                    else:
                        print("Encontrado:")
                        write(trabajos[pos])
                elif op == 5:
                    contar_por_tipo(trabajos)
        if op == 6:
            print("Bye!")



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