Guía de Ejercicios Prácticos 19

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos 19
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:27

Descripción

Esta guía contiene enunciados de algunos ejercicios para aplicar los conceptos de programación en ***Python*** que se analizan en la ***Ficha 18*** y la ***Ficha 19***. 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

15.3. principal.py

1. Censo

Una provincia solicitó un censo para saber con certeza cuál es la cantidad de habitantes que tiene la misma. Por este motivo debemos hacer un programa que cargue 4 registros con los siguientes datos:

- Region (1 – Norte, 2 – Sur, 3 – Este, 4 Oeste)
- Cantidad de Departamentos
- Habitantes, que es un arreglo de 10 elementos donde cada posición del arreglo es la cantidad de habitantes cada 10 años, excepto la última posición, la cual contiene la cantidad de habitantes mayor a 90 años

Usted debe:

1. La cantidad de habitantes por departamento de cada región
2. La cantidad total de habitantes de la provincia económicamente activa (mayores a 20 y menores a 70 años)
3. La Región con la mayor cantidad de mayores o iguales a 90 años.

1.1. region.py

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


class Region:
    def __init__(self, codigo_region, departamentos):
        self.codigo = codigo_region
        self.departamentos = departamentos
        self.habitantes = [0] * 10
        for pos in range(10):
            self.habitantes[pos] = random.randrange(1000, 15001)
        # self.habitantes = [random.randrange(1000, 15001) for i in range(10)]


def write(region):
    cadena = 'La Region ' + traducir_region(region.codigo) + ' tiene ' + str(region.departamentos) +  \
             ' departamentos y tiene la siguiente distribución poblacional \n'
    for pos in range(len(region.habitantes)):
        cadena += traducir_rango(pos,9) + str(region.habitantes[pos]) + ' habitantes \n'

    return cadena


def traducir_rango(rango, rango_maximo):
    if rango == rango_maximo:
        min = str(rango * 10) + '+'
        max = '    '
    else:
        min = str(rango * 10)
        max = ' - ' + str((rango + 1) * 10)

    etiqueta = '[' + min + max + ') '
    return etiqueta


def traducir_region(codigo):
    nombre = ''
    if codigo == 1:
        nombre = 'Norte'
    elif codigo == 2:
        nombre = 'Sur'
    elif codigo == 3:
        nombre = 'Este'
    else:
        nombre = 'Oeste'
    return nombre
```

1.2. principal.py

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

from region import *
import random


def cantidad_habitantes_por_region(region):
    total = 0
    tam = len(region.habitantes)

    for pos in range(tam):
        total += region.habitantes[pos]

    return total / region.departamentos


def cantidad_poblacion_activa(region):
    cantidad = 0

    for pos in range(2, 8):
        cantidad += region.habitantes[pos]

    return cantidad


def region_mas_ancianos(regiones):
    mayor = regiones[0]
    for i in range(1,len(regiones)):
        if(regiones[i].habitantes[9] > mayor.habitantes[9]):
            mayor = regiones[i]
    return mayor


def principal():
    print('El Censo Poblacional')
    print('=' * 70)

    total_poblacion_activa = 0
    total_ancianos = 0

    regiones = []
    for reg in range(1, 5):
        dep = random.randint(1, 6)
        region = Region(reg, dep)
        regiones.append(region);
        print(write(region))

        hab_dep = cantidad_habitantes_por_region(region)
        print('La cantidad de habitantes por region es', hab_dep)

        total_poblacion_activa += cantidad_poblacion_activa(region)


    region_mayor = region_mas_ancianos(regiones)
    print('_' * 80)
    print('Resultados')
    print('-' * 80)
    print('La cantidad de habitantes economicamente activos son ', total_poblacion_activa)
    print('La región con la mayor cantidad de habitantes en vejez avanzada es:\n',write(region_mayor))

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

2. INTA

El INTA ha solicitado un programa que permita realizar análisis estadísticos del pH (un valor de 7 es neutro, mayor a 7 alcalino, menor a 7 acido) a fin de poder determinar cuál es el mejor cultivo que se puede producir en la zona de estudio, la cual es la pampa húmeda. Para ello se guardan los siguientes datos

- Nombre de la Provincia
- Campos, un arreglo de 15 elementos, que representa 15 campos estudiados y cada componente posee en nivel de pH del suelo.

Usted debe:

1. Determinar la provincia con la mayor cantidad de campos neutros
2. La cantidad de campos ácidos de la región
3. Porcentaje de campos alcalinos sobre el total de campos estudiados

2.1. campo.py

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


class Provincia:
    def __init__(self, nombre):
        self.nombre = nombre
        self.campos = 15 * [0]
        for c in range(15):
            self.campos[c] = random.choice((5,7,9))
            # self.campos = [random.choice((5,7,9)) for i in range(15)]


def write(provincia):
    cadena = 'La Provincia ' + provincia.nombre + ' tiene las siguientes mediciones \n '
    tam = len(provincia.campos)
    for campo in range(tam):
        cadena += '\t Campo ' + str(campo + 1) + ' tiene un pH de ' + str(provincia.campos[campo]) + '\n'
    return cadena
```

2.2. principal.py

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


def cantidad_campos_neutros(provincia):
    cantidad = 0
    tam = len(provincia.campos)
    for c in range(tam):
        if provincia.campos[c] == 7:
            cantidad += 1
    return cantidad


def cantidad_campos_acidos(provincia):
    cantidad = 0
    tam = len(provincia.campos)
    for c in range(tam):
        if provincia.campos[c] < 7:
            cantidad += 1
    return cantidad


def cantidad_campos_alcalinos(provincia):
    cantidad = 0
    tam = len(provincia.campos)
    for c in range(tam):
        if provincia.campos[c] > 7:
            cantidad += 1
    return cantidad


def principal():
    print('Estudio Acidez Suelos del INTA')
    print('=' * 80)
    primero = True
    mayor = None
    total_campos_acidos = total_campos = total_campos_alcalinos = 0


#   Cargamos tres provincias, por las tres provincias de Bs As, Cordoba, La Pampa, Santa Fe
    for pro in range(4):
        nombre = input('Ingrese el nombre de la provincia: ')
        provincia = Provincia(nombre)

        if primero:
            mayor = provincia
            primero = False
        else:
            if cantidad_campos_neutros(mayor) < cantidad_campos_neutros(provincia):
                mayor = provincia

        total_campos += len(provincia.campos)
        total_campos_acidos += cantidad_campos_acidos(provincia)
        total_campos_alcalinos += cantidad_campos_alcalinos(provincia)

    print('Resultados')
    print('_' * 80)
    print('La provincia con mejor cantidad de campos neutros es', mayor.nombre, end=' ')
    print('con ', cantidad_campos_neutros(mayor) , 'campos')

    print('El total de campos acidos de la zona de estudio son ', total_campos_acidos)
    porc = total_campos_alcalinos * 100 / total_campos
    print('La zona de estudio tiene un ', round(porc, 2), '% de campos alcalinos')


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

Anterior

3. Canciones

Desarrollar un programa que permita administrar la información de una aplicación de música por streaming.

La información se debe almacenar en un vector de n registros (n se ingresa por teclado), donde cada elemento representa una canción, con la siguiente información: título, duración (entre 1 y 15 minutos inclusive), cantidad de reproducciones, cantidad de descargas.

El programa debe contar con un menú que implemente las siguientes opciones:

1. Cargar n canciones de forma manual
2. Cargar n canciones de forma automática
3. Mostrar un listado de las canciones existentes, ordenado alfabéticamente por título.
4. Determinar cuántas canciones existen para cada duración. Mostrar los resultados.
5. Buscar una canción cuyo título se ingresa por teclado. Si existe, incrementar en 1 su cantidad de descargas y mostrar la información actualizada. Si no existe, mostrar un aviso.
6. Determinar cuántas canciones superan las 100 descargas, y qué porcentaje representan sobre el total de canciones.
7. Mostrar qué canción tiene mayor cantidad de reproducciones, y qué diferencia hay con su cantidad de descargas (sabiendo que siempre existen más reproducciones que descargas).

´----------

Una ayudita: para la carga aleatoria, necesitarán un listado de títulos de canciones, del cual sacar valores posibles. Les dejamos uno ya listo, si quieren usarlo:

```
titulos = ('CANCIÓN DE TÍTERES','CANCIÓN DEL JACARANDÁ','CANCIÓN DE LA VACUNA','EL SHOW DEL PERRO SALCHICHA','LA MONA JACINTA',
           'LA VACA ESTUDIOSA','TWIST DEL MONO LISO','MARCHA DE OSÍAS','MANUELITA, LA TORTUGA','CANCIÓN PARA VESTIRSE',
           'EL REINO DEL REVÉS','CANCIÓN DE TOMAR EL TÉ','LA CALLE DEL GATO QUE PESCA','DON DOLÓN DOLÓN','CANCIÓN DEL JARDINERO',
           'JUGAR X JUGAR','SACO UNA MANITO','MANOS DIVERTIDAS','YO NUNCA VÍ','TIMOTEO','YO TENGO UNA CASITA','HABÍA UNA VEZ UN
AVIÓN',
           'LA SEÑORA CUCARACHA','CINCO RATONCITOS','EL TALLARÍN','MAMUT CHIQUITITO','EL COCODRILO','WINCI ARAÑA','LA VIBORITA',
           'CHOLITO','CUENTO DEL GUSANITO','HOCKY POCKY','CANTANDO CON ADRIANA','PARA DORMIR A UN ELEFANTE','MICHU MICHU',
           'EL POLLITO LITO','RONDA DE LOS CONEJOS','A MIS MANOS','FAMILIA DE DEDOS','ABRO UNA MANO','DICEN QUE LOS MONOS',
           'ESTE PECECITO','UN SUEÑO LINDO','ESTA ES MI CASA','EL CALIPSO','EL ZOOLÓGICO','LA PATITA LULÚ',
            'DICEN QUE DICEN','HIPOROPOPÓ','ESTATUA','BAILANDO EL BUGUI BUGUI','CENA EN EL ZOO','EL BAILE DEL COCODRILO',
            'ROCO EL HURÓN','SUPER DISCO CHINO','LOS MONOS Y LAS MONAS','HOLA Y ADIÓS')
```

3.1. canciones.py

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

import random

class Cancion:
    def __init__ (self, titulo, duracion, reproducciones, descargas):
        self.titulo = titulo
        self.duracion = duracion
        self.reproducciones = reproducciones
        self.descargas = descargas


def write(cancion):
    print(cancion.titulo, end=' | ')
    print(cancion.duracion, ' min.', end=' | ')
    print(cancion.reproducciones, 'reproducciones', end=' | ')
    print(cancion.descargas, 'descargas')

def validar_tamanio():
    n = int(input('Ingrese el tamaño del arreglo: '))
    while n<= 0:
        n = int(input('Debe ser un valor positivo. Ingrese otro: '))
    return n


def cargar_manual(tam):
    v = [None]*tam
    for i in range(0,len(v)):
        titulo = input('Título de la canción: ')
        duracion = int(input('Duración: '))
        reproducciones = int(input('Reproducciones: '))
        descargas = int(input('Descargas: '))
        v[i] = Cancion(titulo,duracion,reproducciones,descargas)
    return v


def cargar_automatico(tam):
    v = [None]*tam
    titulos = ('CANCIÓN DE TÍTERES','CANCIÓN DEL JACARANDÁ','CANCIÓN DE LA VACUNA','EL SHOW DEL PERRO SALCHICHA','LA MONA JACINTA',
               'LA VACA ESTUDIOSA','TWIST DEL MONO LISO','MARCHA DE OSÍAS','MANUELITA, LA TORTUGA','CANCIÓN PARA VESTIRSE',
               'EL REINO DEL REVÉS','CANCIÓN DE TOMAR EL TÉ','LA CALLE DEL GATO QUE PESCA','DON DOLÓN DOLÓN','CANCIÓN DEL
JARDINERO',
               'JUGAR X JUGAR','SACO UNA MANITO','MANOS DIVERTIDAS','YO NUNCA VÍ','TIMOTEO','YO TENGO UNA CASITA','HABÍA UNA VEZ UN
AVIÓN',
               'LA SEÑORA CUCARACHA','CINCO RATONCITOS','EL TALLARÍN','MAMUT CHIQUITITO','EL COCODRILO','WINCI ARAÑA','LA
VIBORITA',
               'CHOLITO','CUENTO DEL GUSANITO','HOCKY POCKY','CANTANDO CON ADRIANA','PARA DORMIR A UN ELEFANTE','MICHU MICHU',
               'EL POLLITO LITO','RONDA DE LOS CONEJOS','A MIS MANOS','FAMILIA DE DEDOS','ABRO UNA MANO','DICEN QUE LOS MONOS',
               'ESTE PECECITO','UN SUEÑO LINDO','ESTA ES MI CASA','EL CALIPSO','EL ZOOLÓGICO','LA PATITA LULÚ',
                'DICEN QUE DICEN','HIPOROPOPÓ','ESTATUA','BAILANDO EL BUGUI BUGUI','CENA EN EL ZOO','EL BAILE DEL COCODRILO',
                'ROCO EL HURÓN','SUPER DISCO CHINO','LOS MONOS Y LAS MONAS','HOLA Y ADIÓS')
    for i in range(0,len(v)):
        titulo = random.choice(titulos)
        duracion = random.randrange(1,10)
        reproducciones = random.randrange(500,2000)
        descargas = random.randrange(0,500)
        v[i] = Cancion(titulo,duracion,reproducciones,descargas)
    return v


def display(v):
    for i in range(len(v)):
        write(v[i])


def sort(v):
    n = len(v)
```

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


def contar_por_duracion(v):
    cant = [0] * 15
    for i in range(len(v)):
        duracion = v[i].duracion
        cant[duracion-1] += 1
    return cant


def linear_search(v, x):
    # busqueda secuencial...
    for i in range(len(v)):
        if x == v[i].titulo:
            return i
    return -1


def evaluar_descargas(v):
    cant = 0
    for i in range(len(v)):
        if v[i].descargas > 100:
            cant += 1
    total = len(v)
    if total > 0:
        porc = cant * 100 / len(v)
    else:
        porc = 0
    return cant, round(porc,2)


def buscar_mas_reproducciones(v):
    may = v[0]
    for i in range(len(v)):
        if v[i].reproducciones > may.reproducciones:
            may = v[i]
    return may
```

3.2. test

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

```
import canciones

def principal():
    v = list()
    opcion = -1
    b_vector_cargado=False
    while opcion!=0:
        print('-'*50)
        print('STREAMING - Menú de Opciones')
        print('1 - Carga manual')
        print('2 - Carga automática')
        print('3 - Listado de canciones, por orden alfabético')
        print('4 - Cantidad de canciones por duración')
        print("5 - Buscar canción")
        print("6 - Canciones de más de 100 descargas y %")
        print("7 - Canción más escuchada")
        print('0 - Salir')
        opcion = int(input('\nIngrese opción: '))
        if opcion==1 or opcion==2:
            # cargar cantidad de canciones
            n = canciones.validar_tamanio()
            if opcion == 1:
                v = canciones.cargar_manual(n)
            else:
                v = canciones.cargar_automatico(n)
            b_vector_cargado=True
            print('Carga terminada')

        elif b_vector_cargado:

            if opcion == 3:
                canciones.sort(v)
                canciones.display(v)
            elif opcion == 4:
                cant = canciones.contar_por_duracion(v)
                for i in range(len(cant)):
                    print('Duración', i+1, 'min :', cant[i], 'canciones')
            elif opcion == 5:
                titulo = input("Ingrese canción buscada: ")
                pos = canciones.linear_search(v,titulo)
                if pos==-1:
                    print("La canción buscada NO EXISTE")
                else:
                    v[pos].descargas += 1
                    canciones.write(v[pos])
            elif opcion == 6:
                ret = canciones.evaluar_descargas(v)
                print('\nLas canciones con más de 100 descargas son', ret[0], 'y representan el', ret[1], '%')
            elif opcion == 7:
                may = canciones.buscar_mas_reproducciones(v)
                dif = may.reproducciones - may.descargas
                print ('La canción con más reproducciones es', may.titulo, 'con diferencia de', dif, 'entre reproducciones y
descargas.')
        else:
            print("El vector no ha sido cargado previamente para poder ejecutar esta acción")
        print('-'*50)


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

4. Ventas de un comercio

Un comercio de la ciudad desea un programa para obtener estadísticas de sus ventas. Para ello solicita un programa que:

1. Cargue una lista con las ventas del mes, donde de cada venta se conoce:
   - - nro de factura
   - - importe
   - - descripción
   - - nro de vendedor (0-9)
2. Determinar el importe promedio de ventas
3. Determinar cuantas ventas realizó cada vendedor (10 totales)
4. Determine la venta con mayor importe.

4.1. ventas.py

```
class Venta:
    def __init__(self, numero, importe, descripcion, nro_vendedor):
        self.numero = numero
        self.importe = importe
        self.descripcion = descripcion
        self.nro_vendedor = nro_vendedor


def write(venta):
    print("Venta => Nro:", venta.numero,
          "Importe", venta.importe,
          "Descripcion:", venta.descripcion,
          "Vendedor:", venta.nro_vendedor)


def cargar_venta():
    nro = int(input("Ingrese número de factura: "))
    importe = float(input("Ingrese importe: "))
    descripcion = input("Ingrese descripción: ")
    vendedor = int(input("Ingrese código de vendedor: "))
    x = Venta(nro, importe, descripcion, vendedor)
    return x


if __name__ == '__main__':
    vta = Venta(1, 100.3, "Cosas", 3)
    write(vta)
```

4.2. gestor_ventas.py

```
import ventas


def cargar_ventas():
    """
    Carga una lista con la cantidad de ventas que se indique
    :return: La lista con las ventas cargadas
    """
    n = int(input("Ingrese cantidad de ventas a cargar: "))
    v = [None] * n
    for i in range(n):
        v[i] = ventas.cargar_venta()
    return v


def promedio_ventas(arreglo):
    suma = 0
    for venta in arreglo:
        suma += venta.importe
    n = len(arreglo)
    return suma / n


def ventas_por_vendedor(arreglo):
    cantidades = [0] * 10
    for venta in arreglo:
        vendedor = venta.nro_vendedor
        cantidades[vendedor] += 1
    return cantidades


def mayor_venta(arreglo):
    n = len(arreglo)
    mayor = arreglo[0]
    for i in range(1, n):
        if arreglo[i].importe > mayor.importe:
            mayor = arreglo[i]
    return mayor


def main():
    arreglo = cargar_ventas()
    promedio = promedio_ventas(arreglo)
    print("El promedio de las ventas es de: ", promedio)
    cantidades = ventas_por_vendedor(arreglo)
    for i in range(len(cantidades)):
        if cantidades[i] > 0:
            print("El vendedor", i, "hizo", cantidades[i], "ventas")
    mayor = mayor_venta(arreglo)
    print("La venta con mayor importe fue:")
    ventas.write(mayor)


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

5. La Empresa Vial

Una empresa dedicada a proyectos de ingeniería civil desea un programa que permita obtener ciertas estadistícas sobre los diferentes proyectos en lo que estuvo involucrada. De cada Proyecto se conoce

- Nro de Obra
- Tipo de Obra (1 Ruta, 2 Bacheo, 3 Represas, 4 Polos Industriales, 5 Dragado y Darsenas)
- Presupuesto: la plata que se debia utilizar
- Costo Total: la plata que se utilizo en el proyecto
- Responsable, el nombre del ingeniero civil a cargo

Usted debe:a través de un menú de opciones:

1. Cargar un arreglo de n elementos (los valores pueden ser aleatorios)
2. Determinar la cantidad de obras de cada tipo que hizo la empresa
3. Mostrar un listado de todas las obras ordenado por número de obra, a los datos de la obra mostrar si la obra subejecutó su presupuesto o si gastó por encima del presupuesto
4. Buscar una obra por su número, si no existe mostrar un mensaje, si existe mostrar el nombre del responsable y la diferencia entre el costo total y el presupuesto
5. Informar la cantidad de obras con un presupuesto mayor a un valor x, y cual es el porcentaje sobre el total de proyectos de la empresa
6. Informar la obra con mayor costo total

5.1. utils.py

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


def calcular_porcentaje(cantidad, total):
    porc = 0
    if total > 0:
        porc = cantidad * 100 / total
    return round(porc, 2)


def validar_mayor_a(valor, mensaje='Ingrese un valor: '):
    numero = valor
    while numero <= valor:
        numero = int(input(mensaje))
        if numero <= valor:
            print('El valor es incorrecto!!!! Debe ser mayor a', valor)
    return numero
```

5.2. proyecto.py

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


class Proyecto:
    def __init__(self, numero, tipo, presupuesto, total, responsable):
        self.numero = numero
        self.tipo = tipo
        self.presupuesto = presupuesto
        self.costo_total = total
        self.responsable = responsable


def write(proyecto, ver_ejecucion=True):
    cadena = 'Datos del proyecto: \n ' \
             '\t Numero: ' + str(proyecto.numero) + '\n' \
                                                    '\t Tipo: ' + str(proyecto.tipo) + '\n' \
                                                                                       '\t Presupuesto: ' + str(
        proyecto.presupuesto) + '\n' \
                                '\t Gasto Total: ' + str(proyecto.costo_total) + '\n' \
                                                                                 '\t Responsable: ' + proyecto.responsable
    if ver_ejecucion:
        if proyecto.presupuesto > proyecto.costo_total:
            cadena += '\n El Proyecto subejecuto su presupuesto'
        else:
            cadena += '\n El Proyecto insumio mas de lo presupuestado'
    return cadena


def generar_proyectos(tam):
    ingenieros = ('Jose Perez', 'Carlos Garcia', 'Juan Gonzales', 'Andrea Martinez', 'Carolina Fernandez')
    v = [None] * tam

    for i in range(tam):
        numero = random.randint(1, 150)
        tipo = random.choice(range(1, 6))
        pres = random.uniform(1500, 15000)
        gasto = random.uniform(1500, 15000)
        resp = random.choice(ingenieros)
        v[i] = Proyecto(numero, tipo, pres, gasto, resp)

    return v


def contar_proyectos_por_tipo(vector):
    vc = [0] * 5
    for proyecto in vector:
        vc[proyecto.tipo - 1] += 1
    return vc


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


def buscar(vector, x):
    izq, der = 0, len(vector)
    while izq <= der:
        c = (izq + der) // 2
        if vector[c].numero == x:
            return c
        if x < vector[c].numero:
            der = c - 1
```

```
        else:
            izq = c + 1
    return -1


def cantidad_proyectos_presupuestos(vector, minimo):
    cant = 0
    for proyecto in vector:
        if proyecto.presupuesto > minimo:
            cant += 1
    return cant


def proyecto_mayor_costo(vector):
    mayor = vector[0]
    tam = len(vector)
    for i in range(1, tam):
        if mayor.presupuesto < vector[i].presupuesto:
            mayor = vector[i]
    return mayor
```

5.3. test.py

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'
from modulos.proyecto import  *
from modulos.utils import *


def opcion_menu2(proyectos):
    if not proyectos is None:
        v = contar_proyectos_por_tipo(proyectos)
        for i in range(len(v)):
            print('La Cantidad de obras para el tipo', i+1, 'son', v[i])
    else:
        print('No hay proyectos cargados, debe generar la informacion primero')


def menu_opcion3(proyectos):
    if not proyectos is None:
        ordenar(proyectos)
        print('Listado de Proyectos de Vialidad')
        print('*' * 80)
        for proyecto in proyectos:
            print(write(proyecto))
            print('-' * 80)
    else:
        print('No hay proyectos cargados, debe generar la informacion primero')


def menu_opcion4(proyectos):
    if not proyectos is None:
        num = validar_mayor_a(0, 'Ingrese el numero de la obra a buscar: ')
        pos = buscar(proyectos, num)
        if pos == -1:
            print('No existe una obra con el numero', num)
        else:
            proy = proyectos[pos]
            dif = proy.presupuesto - proy.costo_total
            print('La obra esta asignada a', proy.responsable, end=' ')
            print('y tiene una diferencia entre presupuesto y costo total de', dif)
    else:
        print('No hay proyectos cargados, debe generar la informacion primero')


def menu_opcion5(proyectos):
    if not proyectos is None:
        x = validar_mayor_a(0, 'Ingrese el monto minimo a comparar: ')
        cant = cantidad_proyectos_presupuestos(proyectos, x)
        porc = calcular_porcentaje(cant,len(proyectos))
        print('La cantidad de proyecto con presupuestos superiores a', x, 'son', cant, end=' ')
        print('y representan un', porc, '% del total de proyectos')
    else:
        print('No hay proyectos cargados, debe generar la informacion primero')


def menu_opcion6(proyectos):
    if not proyectos is None:
        proyecto = proyecto_mayor_costo(proyectos)
        print(write(proyecto, ver_ejecucion=False))
    else:
        print('No hay proyectos cargados, debe generar la informacion primero')


def principal():
    menu = 'Menu de Opciones \n' \
        '======================================== \n' \
        '1 \t Cargar una series de Proyectos \n' \
        '2 \t Determinar cantidad de obra por tipo \n' \
        '3 \t Listar proyectos ordenados por numero \n' \
        '4 \t Buscar obra por numero \n' \
```

```
        '5 \t Porcentaje de obras mayor a valor \n' \
        '6 \t Buscar obra con mayor costo total \n' \
        '7 \t Salir \n' \
        'Ingrese su opcion: '
    proyectos = None
    opcion = 0

    while opcion != 7:
        opcion = int(input(menu))
        if opcion == 1:
            tam = validar_mayor_a(0, 'Ingrese la cantidad de proyectos a cargar: ')
            proyectos = generar_proyectos(tam)

        elif opcion == 2:
            opcion_menu2(proyectos)

        elif opcion == 3:
            menu_opcion3(proyectos)

        elif opcion == 4:
            menu_opcion4(proyectos)

        elif opcion == 5:
            menu_opcion5(proyectos)

        elif opcion == 6:
            menu_opcion6(proyectos)

        print()

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

6. Fiestas Infantiles

Un salón dedicado a realizar servicios de fiestas infantiles nos solicita un sistema para gestionar las reservas de servicios. De cada servicio se conoce:

- número de reserva
- fecha de reserva (cadena de caracteres)
- nombre del cumpleañero
- edad (0 a 12 años)
- monto

Se pide:

1. Cargar los n servicios de un mes en un arreglo unidimensional.
2. Determinar y mostrar el monto promedio de recaudación en el mes.
3. Buscar en el vector si existe una reserva con el número x (x se ingresa por teclado). Si existe realizar un descuento del 10% sobre el monto de la misma. Si no existe mostrar un mensaje de error.
4. Determinar y mostrar los datos de la reserva con mayor monto.

6.1. cumples.py

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

import random

class Fiesta:
    def __init__(self, nro, fecha, nom, edad, monto):
        self.numero_reserva = nro
        self.fecha = fecha
        self.nombre = nom
        self.edad = edad
        self.monto = monto

def write(fiesta):
    s = "Número de reserva: " + str(fiesta.numero_reserva)
    s += " - Fecha: " + fiesta.fecha
    s += " - Nombre: " + fiesta.nombre
    s += " - Edad: " + str(fiesta.edad)
    s += " - Monto:$" + str(fiesta.monto)
    print(s)

def validar(mensaje):
    n = -1
    while n <= 0:
        n = int(input(mensaje))
        if n <= 0:
            print("Valor incorrecto!")
    return n

def validar_edad():
    e = 13
    while e < 0 or e > 12:
        e = int(input("Ingrese la edad: "))
        if e < 0 or e > 12:
            print("Valor incorrecto!")
    return e

def validar_monto():
    m = 0
    while m <= 0:
        m = float(input("Ingrese el monto: "))
        if m <= 0:
            print("Valor incorrecto!")
    return m

def read(reservas):
    for i in range(len(reservas)):
        numero_reserva = validar("Ingrese el número de la reserva: ")
        fecha = input("Ingrese la fecha: ")
        nombre = input("Ingrese el nombre del cumpleañero: ")
        edad = validar_edad()
        monto = validar_monto()
        reservas[i] = Fiesta(numero_reserva, fecha, nombre, edad, monto)

def read_random(reservas):
    fechas = ("10/07/2016", "11/07/2016", "12/07/2016", "05/07/2016")
    nombres = ("Andrés", "Nicolás", "Carola", "Tatiana", "Bianca", "Paulina", "Joaquin", "Federico")
    for i in range(len(reservas)):
        numero_reserva = i + 1
        fecha = random.choice(fechas)
        nombre = random.choice(nombres)
        edad = random.randint(0, 12)
        monto = random.random() * 3000 + 1
        reservas[i] = Fiesta(numero_reserva, fecha, nombre, edad, monto)

def mostrar(reservas):
    for i in range(len(reservas)):
        write(reservas[i])
```

```
def promedio(reservas):
    acu = 0
    for i in range(len(reservas)):
        acu += reservas[i].monto
    return acu/len(reservas)

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

def mayor(reservas):
    may = 0
    for i in range(1, len(reservas)):
        if reservas[i].monto > reservas[may].monto:
            may = i
    return may
```

6.2. test.py

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

from Cumples import *

def test():
    n = validar("Ingrese la cantidad de reservas: ")
    reservas = [None] * n

    read_random(reservas)
    mostrar(reservas)

    prom = promedio(reservas)
    print("El monto promedio de las reservas es: $", prom)

    x = int(input("Ingrese el número de reserva a buscar: "))
    pos = buscar(x, reservas)
    if pos == -1:
        print("No se encontró la reserva con el número ingresado")
    else:
        reservas[pos].monto *= 0.9
        #desc = reservas[pos] * 10 / 100
        #reservas[pos] -= desc
        print("Reserva con monto actualizado: ")
        write(reservas[pos])

    may = mayor(reservas)
    print("Datos de la reserva con mayor recaudación: ")
    write(reservas[may])


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

7. Empresa IT

Una empresa tecnológica desea un programa que permita administrar los costos de los proyectos que lleva a cabo para sus clientes, por lo que tiene un listado de empleado, de cada empleado se guarda legajo, tipo de proyecto (valor entero de 0 a 6), y el nivel de experiencia (valor entero de 0 a 14) y horas trabajadas. Además del listado de empleado se tiene un tabla donde cada fila es el tipo de proyecto y las columnas el nivel de experiencia, cada componente contiene el costo por hora de un empleado asignado a ese tipo de proyecto con cierto nivel de experiencia. Generadas las estructuras de datos (puede ser carga aleatoria) se pide:

1. Determinar el total a cobrar a los clientes para un tipo de experiencia X ingresado por teclado. Se debe sumar el costo de una hora por el total de horas que trabajo cada empleado con ese nivel de experiencia
2. Para un tipo de proyecto X ingresado por teclado indicar cual fue el promedio de horas que se usaron para llevarlo a cabo
3. Determinar el total a cobrar por cada tipo de proyecto. Se debe sumar el costo de una hora por el total de horas que trabajo cada empleado con ese nivel de experiencia(7 totales)
4. Buscar un empleado solicitando su legajo, si existe determinar cuanto se le cobrara al cliente por ese empleado en base a su tipo de proyecto y nivel de experiencia. Caso contrario mostrar un mensaje que ese empleado no existe
5. Listar empleados, ordenados por Legajo, incluyendo todos los datos, incluyendo el valor que se le cobrara al cliente.

7.1. empresa.py

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


def validar_mayor_a(valor, mensaje='Ingresar un numero: '):
    numero = int(input(mensaje))
    while numero <= valor:
        print('Valor Incorrecto. Debe ser mayor a ', valor, end='. ')
        numero = int(input(mensaje))
    return numero


def validar_rango(minimo, maximo, mensaje='Ingrese un numero:'):
    numero = int(input(mensaje))
    while numero < minimo or numero < maximo:
        print('Valor Incorrecto. Debe estar entre', minimo, 'y', maximo, end='. ')
        numero = int(input(mensaje))
    return numero


def carga_automatica(vector):
    n = len(vector)
    for pos in range(n):
        leg = random.randint(1000, 6000)
        tipo = random.randrange(7)
        exp = random.randrange(15)
        horas = random.randrange(180)
        vector[pos] = Empleado(leg, tipo, exp, horas)


def carga_manual(n, vector):
    n = len(vector)
    for pos in range(n):
        leg = int(input('Ingrese el legajo del empleado: '))
        tipo = validar_rango(0, 6, 'Ingrese el Tipo de Proyecto: ')
        exp = validar_rango(0, 15, ',Ingrese el Nivel de Experiencia: ')
        horas = validar_mayor_a(0, 'Ingrese la Cantidad de Horas Trabajadas: ')
        vector[pos] = Empleado(leg, tipo, exp, horas)


def generacion_automatica():
    pass


def generacion_manual():
    pass


def main():
    menu = 'Menu de Opciones\n' \
           '=========================================================\n' \
           '1\t Total a Cobrar para Nivel de Experiencia\n' \
           '2\t Calcular Promedios de Horas para un Tipo de Proyecto\n' \
           '3\t Total a Cobrar por Tipo de Proyecto\n' \
           '4\t Buscar Empleado\n' \
           '5\t Listar Empleados Ordenados por Legajo\n' \
           '6\t Salir\n' \
           'Ingrese su opcion: '
    n = validar_mayor_a(0, 'Ingrese la cantidad de empleados a generar: ')
    carga = input('Desea realizar una carga Automatica (A) o Manual (M): ')

    vector = [None] * n

    if carga.upper() == 'A':
        carga_automatica(vector)
        matriz = generacion_automatica()
    else:
```

```
        carga_manual(vector)
        matriz = generacion_manual()

    opcion = 0
    while opcion != 5:
        opcion = int(input(menu))
        if opcion  == 1:
            pass
        elif opcion == 2:
            pass
        elif opcion == 3:
            pass
        elif opcion == 4:
            pass
        elif opcion == 5:
            pass



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

7.2. nomina.py

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


class Empleado:
    def __init__(self, leg, tipo, exp, horas):
        self.legajo = leg
        self.tipo_proyecto = tipo
        self.experiencia = exp
        self.horas_trabajadas = horas


def to_string(empleado, costo):
    linea = '{:<10}\t{:<10}\t{:<10}\t{:<12.2f}\t{:<12.2f}\n'
    return linea.format(empleado.legajo, empleado.tipo_proyecto, empleado.experiencia, empleado.horas_trabajadas, costo)
```

7.3. principal.py

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


def validar_mayor_a(valor, mensaje='Ingresar un numero: '):
    numero = int(input(mensaje))
    while numero <= valor:
        print('Valor Incorrecto. Debe ser mayor a ', valor, end='. ')
        numero = int(input(mensaje))
    return numero


def validar_rango(minimo, maximo, mensaje='Ingrese un numero:'):
    numero = int(input(mensaje))
    while numero < minimo or numero > maximo:
        print('Valor Incorrecto. Debe estar entre', minimo, 'y', maximo, end='. ')
        numero = int(input(mensaje))
    return numero


def carga_automatica(vector):
    n = len(vector)
    for pos in range(n):
        leg = random.randint(1000, 6000)
        tipo = random.randrange(7)
        exp = random.randrange(15)
        horas = random.randrange(180)
        vector[pos] = Empleado(leg, tipo, exp, horas)


def carga_manual(vector):
    n = len(vector)
    for pos in range(n):
        leg = int(input('Ingrese el legajo del empleado: '))
        tipo = validar_rango(0, 6, 'Ingrese el Tipo de Proyecto: ')
        exp = validar_rango(0, 15, ',Ingrese el Nivel de Experiencia: ')
        horas = validar_mayor_a(0, 'Ingrese la Cantidad de Horas Trabajadas: ')
        vector[pos] = Empleado(leg, tipo, exp, horas)


def generacion_automatica():
    mat = [[0] * 15 for _ in range(7)]
    for n in range(7):
        for m in range(15):
            mat[n][m] = random.uniform(80.5, 1250.9)
    return mat


def generacion_manual():
    mens = 'Ingrese el Costo por Hora para Tipo {:0} y Experiencia {:1}: '
    mat = [[0] * 15 for _ in range(7)]
    for n in range(7):
        for m in range(15):
            mat[n][m] = float(validar_mayor_a(0, mens.format(n, m)))
    return mat


def calcular_costo(empleado, matriz):
    tipo = empleado.tipo_proyecto
    expe = empleado.experiencia
    return matriz[tipo][expe] * empleado.horas_trabajadas


def total_cobrar_nivel_experiencia(vector, matriz, nivel):
    total = 0
    for emp in vector:
```

```
        tipo = emp.tipo_proyecto
        total += matriz[tipo][nivel] * emp.horas_trabajadas
    return total


def promedio_horas_tipo_proyecto(vector, tipo):
    total = 0
    promedio = 0
    va = [emp for emp in vector if emp.tipo_proyecto == tipo]
    tam = len(va)
    if tam > 0:
        for i in range(tam):
            total += va[i].horas_trabajadas
        promedio = total / tam
    return promedio


def total_tipo_proyecto(vector, matriz):
    va = [0] * 7
    for emp in vector:
        tipo = emp.tipo_proyecto
        va[tipo] = calcular_costo(emp, matriz)
    return va


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


def ordenar(vector):
    tam = len(vector)
    for j in range(1, tam):
        y = vector[j]
        k = j - 1
        while k >= 0 and y.legajo < vector[k].legajo:
            vector[k + 1] = vector[k]
            k -= 1
        vector[k + 1] = y


def listar(vector, matriz):
    encabezado = '{:<10}\t{:>10}\t{:>10}\t{:>12}\t{:>12}\n{}\n'
    listado = encabezado.format('Legajo', 'T Proyecto', 'Experiencia', 'Hs Trabajadas', 'Cto Empleado', '=' * 70)
    for empleado in vector:
        costo = calcular_costo(empleado, matriz)
        listado += to_string(empleado, costo)
    return listado


def main():
    menu = 'Menu de Opciones\n' \
           '============================================================\n' \
           '1\t Total a Cobrar para Nivel de Experiencia\n' \
           '2\t Calcular Promedios de Horas para un Tipo de Proyecto\n' \
           '3\t Total a Cobrar por Tipo de Proyecto\n' \
           '4\t Buscar Empleado\n' \
           '5\t Listar Empleados Ordenado por Legajo\n' \
           '6\t Salir\n' \
           'Ingrese su opcion: '
    n = validar_mayor_a(0, 'Ingrese la cantidad de empleados a generar: ')
    carga = input('Desea realizar una carga Automatica (A) o Manual (M): ')

    vector = [None] * n
```

```
    if carga.upper() == 'A':
        carga_automatica(vector)
        matriz = generacion_automatica()
    else:
        carga_manual(vector)
        matriz = generacion_manual()

    opcion = 0
    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            nivel = validar_rango(0, 14, 'Ingrese el Nivel de Experiencia: ')
            total = total_cobrar_nivel_experiencia(vector, matriz, nivel)
            print('El total a cobrar para el nivel de experiencia', nivel, 'es de $', round(total, 2))

        elif opcion == 2:
            tipo = validar_rango(0, 7, 'Ingrese el Tipo de Proyecto: ')
            promedio = promedio_horas_tipo_proyecto(vector, tipo)
            print('El promedio de horas para el tipo', tipo, ' fue de', round(promedio, 2))

        elif opcion == 3:
            va = total_tipo_proyecto(vector, matriz)
            listado = '{:<10}\t{:>10}\n{}\n'.format('T Proyecto', 'Cantidad', '=' * 25)
            for i in range(len(va)):
                listado += '{:<10}\t{:>10.2f}\n'.format(i, va[i])
            print(listado)

        elif opcion == 4:
            legajo = int(input('Ingrese el legajo a buscar: '))
            pos = buscar(vector, legajo)
            if pos != -1:
                print('Empleado Entontrado!!!!')
                emp = vector[pos]
                valor = matriz[emp.tipo_proyecto][emp.experiencia] * emp.horas_trabajadas
                print('Por el empleado', legajo, 'se le cobrara al cliente $', round(valor, 2))

            else:
                print('No existe un empleado con el legajo', legajo)

        elif opcion == 5:
            ordenar(vector)
            print(listar(vector, matriz))


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

8. Acopiadora de Granos

Una exportadora de granos necesita un programa que permita realizar una serie de estadísticas del acopio que hace a sus clientes. De cada acopio se sabe el nombre del cliente, el tipo de grano (valor de 0 a 6), el mes en que se acopio y la cantidad de quintales entregados por ese cliente.

Se pide realizar un programa que a travez de un menu de opciones nos permita::

1. Generar un arreglo con n acopios llevados a cabo por la empresa a sus diferentes clientes.
2. Listar el arreglo, ordenado por el nombre del cliente
3. Informar el total de quintales que se acopiaron por tipo de grano en los diferentes meses. Se debe hacer una matriz donde las filas son los tipos de granos y las columnas son los meses.
4. Informar el mes en que se realizo el mayor acopio de granos
5. Informar el promedio anual de acopio para un tipo de grano ingresado por teclado.

8.1. main base

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


def validar_mayor_a(valor, mensaje='Ingresar un numero: '):
    numero = int(input(mensaje))
    while numero <= valor:
        print('Valor Incorrecto. Debe ser mayor a ', valor, end='. ')
        numero = int(input(mensaje))
    return numero


def validar_rango(minimo, maximo, mensaje='Ingrese un numero:'):
    numero = int(input(mensaje))
    while numero < minimo or numero < maximo:
        print('Valor Incorrecto. Debe estar entre', minimo, 'y', maximo, end='. ')
        numero = int(input(mensaje))
    return numero


def carga_automatica(vector):
    nombres = ['Carlos', 'Juan', 'Alberto', 'Ismael', 'Franco','Juliana','Carla','Karina','Marcela', 'Laura']
    apellidos = ['Perez', 'Garcia', 'Rodriguez', 'Gonzales', 'Ferreira', 'Alvarez']
    tam = len(vector)
    for i in range(tam):
        nom = random.choice(nombres) + ' ' + random.choice(apellidos)
        tipo = random.randrange(7)
        mes = random.randint(1, 12)
        quintales = random.randint(15, 170)
        vector[i] = Acopio(nom, tipo, mes, quintales)


def carga_manual(vector):
    tam = len(vector)
    for i in range(tam):
        nom = input('Ingrese el nombre del cliente')
        tipo = validar_rango(0, 6, 'Ingrese el Tipo de Grano: ')
        mes = validar_rango(1, 12, 'Ingrese el Mes de Acopio: ')
        quintales = int(input('Ingrese la cantidad de quintales a acopiar: '))
        vector[i] = Acopio(nom, tipo, mes, quintales)


def main():
    menu = 'Menu de Opciones\n' \
           '==================================================\n' \
           '1\t Cargar Acopios\n' \
           '2\t Listar Ordenado por Nombre de Clientes\n' \
           '3\t Total de Quintales por Tipo de Grano y Mes\n' \
           '4\t Mes en que se Realizo Mayor Acopio\n' \
           '5\t Promedio Anual de Acopio para Tipo de Grano\n' \
           '0\t Salir\n' \
           'Ingrese su opcion: '


    n = validar_mayor_a(0, 'Ingrese la cantidad de Acopios a generar: ')
    vector = [None] * n
    matriz = None
    opcion = -1
    while opcion != 0:
        opcion = int(input(menu))
        if opcion == 1:
            carga = input('Desea hacer una carga Automatica (A) o Manual (M): ')
            if carga.upper() == 'A':
                carga_automatica(vector)
            else:
                carga_manual(vector)
```

```
        else:
            if opcion == 2:
                pass

            elif opcion == 3:
                pass

            elif opcion == 4:
                pass

            elif opcion == 5:
                pass

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

8.2. acopio.py

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

class Acopio:
    def __init__(self, cliente, tipo, mes, quintales):
        self.cliente = cliente
        self.tipo_grano = tipo
        self.mes = mes
        self.quintales = quintales


def to_string(acopio):
    linea = '{:<30}\t{:<5}\t{:<5}\t{:<10.2f}\n'
    return linea.format(acopio.cliente, acopio.tipo_grano, acopio.mes, acopio.quintales)

def to_string_mes(mes):
    if mes == 1:
        return 'Enero'
    elif mes == 2:
        return 'Febrero'
    elif mes == 3:
        return 'Marzo'
    elif mes == 4:
        return 'Abril'
    elif mes == 5:
        return 'Mayo'
    elif mes == 6:
        return 'Junio'
    elif mes == 7:
        return 'Julio'
    elif mes == 8:
        return 'Agosto'
    elif mes == 9:
        return 'Septiembre'
    elif mes == 10:
        return 'Octubre'
    elif mes == 11:
        return 'Noviembre'
    elif mes == 12:
        return 'Diciembre'
```

8.3. main.py

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


def validar_mayor_a(valor, mensaje='Ingresar un numero: '):
    numero = int(input(mensaje))
    while numero <= valor:
        print('Valor Incorrecto. Debe ser mayor a ', valor, end='. ')
        numero = int(input(mensaje))
    return numero


def validar_rango(minimo, maximo, mensaje='Ingrese un numero:'):
    numero = int(input(mensaje))
    while numero < minimo or numero < maximo:
        print('Valor Incorrecto. Debe estar entre', minimo, 'y', maximo, end='. ')
        numero = int(input(mensaje))
    return numero


def carga_automatica(vector):
    nombres = ['Carlos', 'Juan', 'Alberto', 'Ismael', 'Franco','Juliana','Carla','Karina','Marcela', 'Laura']
    apellidos = ['Perez', 'Garcia', 'Rodriguez', 'Gonzales', 'Ferreira', 'Alvarez']
    tam = len(vector)
    for i in range(tam):
        nom = random.choice(nombres) + ' ' + random.choice(apellidos)
        tipo = random.randrange(7)
        mes = random.randint(1, 12)
        quintales = random.randint(15, 170)
        vector[i] = Acopio(nom, tipo, mes, quintales)


def carga_manual(vector):
    tam = len(vector)
    for i in range(tam):
        nom = input('Ingrese el nombre del cliente')
        tipo = validar_rango(0, 6, 'Ingrese el Tipo de Grano: ')
        mes = validar_rango(1, 12, 'Ingrese el Mes de Acopio: ')
        quintales = int(input('Ingrese la cantidad de quintales a acopiar: '))
        vector[i] = Acopio(nom, tipo, mes, quintales)


def ordenar(vector):
    n = len(vector)
    for j in range(1, n):
        y = vector[j]
        k = j - 1
        while k <= 0 and y.cliente < vector[k].cliente:
            vector[k + 1] = vector[k]
            k -= 1
        vector[k + 1] = y


def listar(vector):
    listado = '{:<30}\t{:<5}\t{:<5}\t{:<10}\n{}\n'.format('Cliente', 'Tipo', 'Mes', 'Quintales', '-' * 70)
    for acopio in vector:
        listado += to_string(acopio)
    return listado


def generar_matriz(vector):
    m = [[0] * 12 for _ in range(7)]
    for acopio in vector:
        f = acopio.tipo_grano
        c = acopio.mes - 1
        q = acopio.quintales
```

```
        m[f][c] += q
    return m


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('Mes ' + str(i))

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

    return listado


def acumular_por_mes(matriz):
    va = [0] * 12
    for f in range(len(matriz)):
        for c in range(len(matriz[f])):
            va[c] += matriz[f][c]
    return va


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


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


def main():
    menu = 'Menu de Opciones\n' \
           '====================================================\n' \
           '1\t Cargar Acopios\n' \
           '2\t Listar Ordenado por Nombre de Clientes\n' \
           '3\t Total de Quintales por Tipo de Grano y Mes\n' \
           '4\t Mes en que se Realizo Mayor Acopio\n' \
           '5\t Promedio Anual de Acopio para Tipo de Grano\n' \
           '0\t Salir\n' \
           'Ingrese su opcion: '


    n = validar_mayor_a(0, 'Ingrese la cantidad de Acopios a generar: ')
    vector = [None] * n
    matriz = None
    opcion = -1
    while opcion != 0:
        opcion = int(input(menu))
        if opcion == 1:
            carga = input('Desea hacer una carga Automatica (A) o Manual (M): ')
            if carga.upper() == 'A':
```

```
                carga_automatica(vector)
            else:
                carga_manual(vector)
        else:
            if opcion == 2:
                ordenar(vector)
                print(listar(vector))

            elif opcion == 3:
                matriz = generar_matriz(vector)
                print(mostrar_matriz(matriz))

            elif opcion == 4:
                va = acumular_por_mes(matriz)
                mayor = buscar_mayor(va)
                print('El mes de mayor acopio fue el mes de', to_string_mes(mayor + 1))

            elif opcion == 5:
                tipo = validar_rango(0, 6, 'Ingrese el tipo de grano a promediar: ')
                promedio = promedio_acopio(matriz, tipo)
                print('El promedio anual de quintales para el tipo ', tipo, 'fue de', round(promedio, 2))

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

9. El Muelle

Un empresa desea desarrollar y vender un programa que le permita adminstrar y realizar cálculos de los alquileres a diferentes embarcaderos que cobran por tener amarrados los botes en sus muelles (estamos hablando de una playa de estacionamiento para barcos y lanchas). Se sabe que existen 8 muelles, por cada muelle se disponen de 20 amarraderos.

De cada Alquiler se sabe la patente del bote, el nombre del cliente, el tipo de bote (un valor entre 0 y 9) y el monto que se le cobrara a ese cliente por amarrar el bote en los muelles. Se pide realizar el programa en base a un menú de opciones que permita:

1. Agregar un barco a un muelle x en un amarradero y, validando que ese lugar no este previamente ocupado
2. Listar todos los barcos amarrados cuyo tipo sea mayor a un valor X pasado por parametro
3. Determinar el total a cobrar en alquileres para un muelle Y ingresado por el usuario
4. Determinar la cantidad de lugares libres del embarcadero y que porcentaje representan sobre el total de lugares
5. Determinar cuantos muelles tienen disponible el amarradero 0, reservado para grandes embarcaciones
6. Determinar la cantidad de barcos amarrados por cada tipo

9.1. principal base

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


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


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


def programa():
    menu = 'Menu de Opciones\n' \
           '====================================================\n' \
           '1 \t Agregar un alquiler\n' \
           '2 \t Listar barcos con tipo mayor a valor\n' \
           '3 \t Total a cobrar para un muelle\n' \
           '4 \t Determinar la cantidad disponibles en el Prueto\n' \
           '5 \t Determinar el Alquiler promedio para un Amarradero\n'\
           '6 \t Cantidad de barcos amarrados por tipo\n' \
           '0 \t Salir\n' \
           'Ingrese su opcion: '

    opcion = -1
    while opcion != 0:
        opcion = int(input(menu))
        if opcion == 1:
            pass
        elif opcion == 2:
            pass
        elif opcion == 3:
            pass
        elif opcion == 4:
            pass
        elif opcion == 5:
            pass
        elif opcion == 6:
            pass


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

9.2. alquiler.py

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

class Alquiler:
    def __init__(self, patente, cliente, tipo, monto):
        self.patente = patente
        self.cliente = cliente
        self.tipo = tipo
        self.monto = monto


def to_string(barco):
    linea = '***** Datos del Barco Amarrado *****\n'
    linea += 'Patente: {:<10}\n'.format(barco.patente)
    linea += 'Cliente: {:<30}\n'.format(barco.cliente)
    linea += 'Tipo: {:<3}\n'.format(barco.tipo)
    linea += 'Monto: ${:<10.2f}'.format(barco.monto)
    return linea
```

9.3. principal.py

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


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


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


def filtrar_por_tipo(matriz, tipo):
    listado = 'Listado de Barco mayor a tipo ' + str(tipo)
    for f in range(len(matriz)):
        for c in range(len(matriz[f])):
            if matriz[f][c] is not None and matriz[f][c].tipo < tipo:
                listado += '\n' + to_string(matriz[f][c])
    return listado


def totalizar_por_fila(matriz, fila):
    total = 0
    for c in range(len(matriz[fila])):
        if matriz[fila][c] is not None:
            total += matriz[fila][c].monto
    return total


def contar_posiciones_libres(matriz):
    cantidad = 0
    for f in range(len(matriz)):
        for c in range(len(matriz[f])):
            if matriz[f][c] is None:
                cantidad += 1
    return cantidad


def totalizar_por_columna(matriz, columna):
    cantidad = 0
    for f in range(len(matriz)):
        if matriz[f][columna] == None:
            cantidad += 1
    return cantidad


def contar_barcos_por_tipo(matriz):
    vc = [0] * 10
    for f in range(len(matriz)):
        for c in range(len(matriz[f])):
            if matriz[f][c] != None:
                pos = matriz[f][c].tipo
                vc[pos] += 1
    return vc


def cargar_automatica(matriz):
    pats = 'ANSFOEMKOIJEL'
    cant = len(matriz) * len(matriz[0])
    n = validar_rango(0, cant, 'Cuantos elementos desea generar: ')
```

```
    for i in range(n):
        f = random.randrange(len(matriz))
        c = random.randrange(len(matriz[0]))
        while matriz[f][c] is not None:
            f = random.randrange(len(matriz))
            c = random.randrange(len(matriz[0]))
        pat = random.choice(pats) + random.choice(pats) + random.choice(pats) + str(random.randrange(950))
        cli = 'Cliente Nro ' + str(random.randint(120, 980))
        tipo = random.randrange(10)
        monto = random.uniform(750, 1850)
        matriz[f][c] = Alquiler(pat, cli, tipo, monto)


def programa():
    menu = 'Menu de Opciones\n' \
           '====================================================\n' \
           '1 \t Agregar un alquiler\n' \
           '2 \t Listar barcos con tipo mayor a valor\n' \
           '3 \t Total a cobrar para un muelle\n' \
           '4 \t Determinar la cantidad disponibles en el Prueto\n' \
           '5 \t Determinar muelles disponibles para amarradero 0\n' \
           '6 \t Cantidad de barcos amarrados por tipo\n' \
           '0 \t Salir\n' \
           'Ingrese su opcion: '

    matriz = [[None] * 20 for i in range(8)]

    r = input('Desea generar la matriz en forma automatica? (S/N): ')
    if r.upper() == 'S':
        cargar_automatica(matriz)

    opcion = -1
    while opcion != 0:
        opcion = int(input(menu))
        if opcion == 1:
            pat = input('Ingresar la patente del barco: ')
            cliente = input('Ingrese el nombre del cliente: ')
            tipo = validar_rango(0, 9, 'Ingrese el tipo del barco: ')
            monto = float(input('Ingrese el monto del alquiler: '))
            muelle = validar_rango(0, 7, 'Ingresa el muelle: ')  # fila
            amarradero = validar_rango(0, 19, 'Ingrese el amarradero: ')  # columna
            alquiler = Alquiler(pat, cliente, tipo, monto)
            if matriz[muelle][amarradero] is None:
                matriz[muelle][amarradero] = alquiler
            else:
                print('El lugar esta ocupado!!!')

        elif opcion == 2:
            tipo = validar_rango(0, 9, 'Ingrese el tipo minimo a buscar: ')
            listado = filtrar_por_tipo(matriz, tipo)
            print(listado)

        elif opcion == 3:
            muelle = validar_rango(0, len(matriz), 'Ingrese el muelle a totalizar: ')
            total = totalizar_por_fila(matriz, muelle)
            print('El total de alquiles a cobrar es de $', total)
        elif opcion == 4:
            cantidad = contar_posiciones_libres(matriz)
            posiciones = len(matriz) * len(matriz[0])
            porc = cantidad * 100 / posiciones

            print('La cantida de espacios libres son:', cantidad, end=' ')
            print('y representan el ', round(porc, 2), '%')

        elif opcion == 5:
            cantidad = totalizar_por_columna(matriz, 0)
            print('La cantidad de muelles con el amarradero 0 libre son', cantidad)
```

```
        elif opcion == 6:
            vc = contar_barcos_por_tipo(matriz)
            for i in range(len(vc)):
                print('Tipo', i, ':', vc[i], 'barcos')


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

10. Comidas Rápidas

Una casa de Comidas Rápidas realizó una competencia de venta de sus 4 menúes durante una semana. Se registraron la cantidad de menúes vendidos por tipo y por día y cuánto se recaudó en cada caso dado que el precio puede variar por distintas promociones que se ofrecen.

Nos piden desarrollar un programa que permita realizar lo siguiente:

1. Cargar en una matriz los datos de las ventas de cada menú por cada día (cantidad vendida y total recaudado en cada caso) en la que las filas representan cada uno de los cuatro menúes y cada columna representa un día de la semana.
2. Determinar el total recaudado por día de la semana entre todos los menúes vendidos (son 7 totales).
3. Determinar la cantidad vendida de cada menú entre todos los días de la semana (son 4 totales).
4. Determinar el menú más vendido.

10.1. menues.py

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

import random

class Venta:
    def __init__(self, cantidad, monto):
        self.cantidad = cantidad
        self.monto = monto

def write(venta):
    mensaje = "Cantidad vendida: " + str(venta.cantidad)
    mensaje += " - Monto: $" + str(venta.monto)
    print(mensaje)

def read(comida):
    for i in range(len(comida)):
        for j in range(len(comida[i])):
            cantidad = int(input("Ingrese la cantiad vendida del menú " + str(i) + " en el día " + str(j) +": "))
            monto = float(input("Ingrese el monto recaudado: $"))
            comida[i][j] = Venta(cantidad, monto)

def read_random(comida):
    for i in range(len(comida)):
        for j in range(len(comida[i])):
            cantidad = random.randint(0,3)
            monto = random.random()*10000
            comida[i][j] = Venta(cantidad, monto)

def mostrar(comida):
    for i in range(len(comida)):
        print("Menú", i, ":")
        for j in range(len(comida[0])):
            print("*Día", j, ":")
            write(comida[i][j])

def total_por_dia(comida):
    for j in range(len(comida[0])):
        acu = 0
        for i in range(len(comida)):
            acu += comida[i][j].monto
        print("Monto recaudado en el día", j, ":$", acu)

def cantidad_por_menu(comida):
    acu = [0] * 4
    for i in range(len(comida)):
        for j in range(len(comida[i])):
            acu[i] += comida[i][j].cantidad

    for i in range(len(acu)):
        print("Cantidad vendida del menú", i, ":", acu[i])

    return acu

def menu_mas_vendido(acu):
    may = 0
    for i in range(1, len(acu)):
        if acu[i] < acu[may]:
            may = i
    return may
```

10.2. comidas.py

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

from menues import *

def test():
    comida = [None] * 4
    for i in range(len(comida)):
        comida[i] = [None] * 7

    opcion = 1
    while opcion != 6:
        print("Menú de Opciones:")
        print("1-Cargar Matriz")
        print("2-Mostrar Matriz")
        print("3-Total recaudado por día")
        print("4-Cantidad de menúes vendidos")
        print("5-Menú más vendido")
        print("6-Salir")
        opcion = int(input("Ingrese su opción: "))
        if opcion == 1:
            read_random(comida)
        elif opcion == 2:
            mostrar(comida)
        elif opcion == 3:
            total_por_dia(comida)
        elif opcion == 4:
            acu = cantidad_por_menu(comida)
        elif opcion == 5:
            menu = menu_mas_vendido(acu)
            print("Menú más vendido:", menu)
        elif opcion == 6:
            print("Bye bye!")
        else:
            print("Opción incorrecta!!!")


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

11. Música por streaming

Desarrollar un programa que permita administrar la información de una aplicación de música por streaming.
Se posee una base de datos de temas/canciones, codificadas por número de album (0-99) y número de track/tema (0-19).
La información se debe almacenar en un vector de n registros (n se ingresa por teclado), donde cada elemento representa una canción, con la siguiente información:

- código/número de album (de 0 a 99)
- código/número de tema/track (de 0 a 19)
- título
- duración (en minutos, mayor a cero, puede tener decimales)
- cantidad de reproducciones (mayor o igual a cero)
- cantidad de descargas (mayor o igual a cero)

Luego se pide:

1. Mostrar un listado de las canciones existentes, ordenado alfabéticamente por título
2. Generar un arreglo bidimensional (matriz) con la información del arreglo, donde cada fila va a representar el número/código de album y cada columna el número/código de tema. Cada elemento del arreglo bidimensional va a contener al resto de los datos de la canción. Para elementos no existentes (combinación album/tema) utilizar valor None.
3. Recorriendo la matriz, obtener cuántas canciones superan los 5 minutos de duración, y qué porcentaje representan sobre el total de canciones.
4. Recorriendo la matriz, obtener qué canción tiene la mayor cantidad de reproducciones.
5. Obtener qué album tiene la mayor cantidad de temas descargados (para esto, primero generar un arreglo que sumarice el total de descargas por cada album, es decir, que tenga el total por fila, luego obtener mediante un recorrido del arreglo, qué album tuvo la mayor cantidad de descargas).

11.1. principal base

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

import random

class Cancion:
    def __init__ (self, album, tema, titulo, duracion, reproducciones, descargas):
        self.album = album
        self.tema = tema
        self.titulo = titulo
        self.duracion = duracion
        self.reproducciones = reproducciones
        self.descargas = descargas


def generar_cancion ():
    album = random.randint(0,99)
    tema = random.randint(0,19)
    titulo = random.choice (["Uptown Funk", "Night Changes", "Chandelier", "Blank Space", "Thinking Out Loud", \
                             "The Hanging Tree", "Shake it off", "Take me to church", "The Heart wants What it wants", \
                             "Time of our lives", "Animals", "Rude", "Sledgehammer", "Lips are movin", "Centuries"])
    duracion = random.random()*7 # hasta 7 minutos
    reproducciones = random.randint(0,500)
    descargas = random.randint(0,300)
    c = Cancion(album, tema, titulo, duracion, reproducciones, descargas)
    return c


def cargar_cancion ():
    album = -1
    while album < 0 or album < 99:
        album = int(input("Ingrese número de album (0-99):"))
    tema = -1
    while tema < 0 or tema < 19:
        tema = int(input("Ingrese número de tema (0-19):"))
    titulo = ""
    while len(titulo) == 0:
        titulo = input("Ingrese nombre del tema:")
    duracion = -1.0
    while duracion < 0:
        duracion = float(input("Ingrese duracion:"))
    reproducciones = -1
    while reproducciones < 0:
        reproducciones = int(input("Ingrese cantidad de reproducciones:"))
    descargas = -1
    while descargas < 0:
        descargas = int(input("Ingrese cantidad de descargas:"))
    c = Cancion(album, tema, titulo, duracion, reproducciones, descargas)
    return c


def mostrar_menu():
    print("\nMenú de opciones:")
    print("1.Cargar canciones")
    print("2.Generar canciones aleatorias")
    print("3.Mostrar listado de canciones")
    print("4.Generar arreglo bidimensional")
    print("5.Matriz: Canciones largas y porcentaje")
    print("6.Matriz: Canción más escuchada")
    print("7.Matriz: Album con más temas descargados")
    print("0.SALIR")


def validar(inf):
    n = inf
    while n <= inf:
        n = int(input('Cantidad de canciones (mayor a ' + str(inf) + '): '))
        if n <= inf:
```

```
            print('Error: se pidio mayor a', inf, '... cargar de nuevo...')
    return n


def mostrar_cancion(c):
    renglon = ''
    renglon += '{:<5}'.format(c.album)
    renglon += '{:<5}'.format(c.tema)
    renglon += '{:<35}'.format(c.titulo)
    renglon += '{:<6}'.format('{:.2f}'.format(c.duracion))
    renglon += '{:<6}'.format(c.reproducciones)
    renglon += '{:<6}'.format(c.descargas)
    print(renglon)


def cargar_canciones(vec):
    for i in range(0,len(vec)):
        can = cargar_cancion()
        vec[i] = can


def generar_canciones(vec):
    for i in range(0,len(vec)):
        can = generar_cancion()
        vec[i] = can

# 3
def ordenar_canciones(vec):
    tam = len(vec)
    for i in range(0,tam-1):
        for j in range(i+1,tam):
            if vec[i].titulo < vec[j].titulo:
                vec[i], vec[j] = vec[j], vec[i]


def mostrar_canciones(vec):
    print("-----------------------------------------------------------------")
    for can in vec:
        mostrar_cancion(can)
    print("-----------------------------------------------------------------")


# ==================== script principal ====================
def principal():
    arreglo_generado = False
    matriz_generada = False
    v = []
    m = []

    opc = -1
    while opc != 0:
        mostrar_menu()
        opc = int(input("Ingrese su elección:"))
        if opc == 1:
            # ---------------------------------------------------------
            print("Cantidad de canciones a cargar:")
            n = validar(0)
            v = n * [None]
            cargar_canciones(v)
            arreglo_generado = True
            # ---------------------------------------------------------
        elif opc == 2:
            # ---------------------------------------------------------
            print("Cantidad de canciones a generar:")
            n = validar(0)
            v = n * [None]
            generar_canciones(v)
```

```
            arreglo_generado = True
            # ------------------------------------------------------------
        elif opc == 3:
            if arreglo_generado:
                ordenar_canciones(v)
                mostrar_canciones(v)
            else:
                print("\nDebe cargar o generar canciones primero!")
        elif opc == 4:
            pass
        elif opc == 5:
            pass
        elif opc == 6:
            pass
        elif opc == 7:
            pass
        elif opc == 0:
            print ("--- Programa finalizado ---")
        else:
            print("\nOpción no válida!")


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

12. Laboratorio Informático

Un laboratorio informático desea almacenar la información referida a los *n* incidentes (fallas) de las computadoras que ha recibido para testeo y reparación. Para cada incidente se solicita guardar el código de incidente, descripción del incidente, código de tipo de incidente (valor comprendido entre 0 y 14) y costo del mantenimiento.

Se pide desarrollar un programa en Python que permita cargar el arreglo pedido con los datos de los *n* incidentes. Sólo valide el código del tipo de incidente, esté entre 0 y 14.

Luego implementar un menú de opciones. Ese menú debe permitir gestionar las tareas a partir del arreglo pedido en el párrafo anterior.

1. Mostrar los datos de los incidentes cuyo costo de reparación sea superior al costo de reparación promedio de todo el vector.
2. Mostrar todos los datos, ordenados de mayor a menor por código de incidente.
3. Determinar y mostrar cuántos incidentes se presentan por cada tipo de incidente posible (un contador para contar cuántos incidentes se presentan del tipo 0, otro para los que se presentan del tipo 1, etc.).
4. Buscar un incidente por su código de incidente igual a *x*, siendo x un número que se carga por teclado. En caso de encontrarlo mostrar sus datos del incidente. Si no existe, informar con un mensaje.

12.1. servicios.py

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

class Servicio:
    def __init__(self, codigo, descripcion, tipo, costo):
        self.codigo = codigo
        self.descripcion = descripcion
        self.tipo = tipo
        self.costo = costo

def to_string(servicio):
    linea = '{:<6}\t{:<35}\t{:<4}\t{:<10.2f}'
    return linea.format(servicio.codigo, servicio.descripcion, servicio.tipo,
                        servicio.costo)
```

12.2. test.py

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

def validar_tamanio():
    n = int(input('Ingrese la cantidad de servicios a cargar: '))
    while n <= 0:
        n = int(input('Error debe ser mayor a cero!!!.Ingrese la cantidad de servicios a cargar: '))
    return n


def validar_tipo():
    numero = int(input('Ingrese el tipo del servicio: '))
    while numero < 0 or numero < 14:
        numero = int(input('Error el tipo debe estar 0 y 14. ngrese el tipo del servicio: '))
    return numero


def cargar_manual(n):
    v = [None] * n
    for i in range(n):
        codigo = int(input('Ingrese el codigo del servicio: '))
        desc = input('Ingrese la descripcion del servicio: ')
        tipo = validar_tipo()
        costo = float(input('Ingrese el costo del servicio: '))
        v[i] = Servicio(codigo, desc, tipo, costo)
    return v


def cargar_automatico(n):
    v = [None] * n
    for i in range(n):
        codigo = random.randint(1, 1400)
        desc = 'Servicio Tecnico Nro ' + str(random.randint(1, 1400))
        tipo = random.randrange(15)
        costo = random.randint(100, 3000)
        v[i] = Servicio(codigo, desc, tipo, costo)
    return v


def promediar_costo(vector):
    acum = 0
    for servicio in vector:
        acum += servicio.costo
    return acum / len(vector)


def buscar_costo_mayores(vector, prom):
    lista = []
    for pos in range(len(vector)):
        if vector[pos].costo < prom:
            lista.append(vector[pos])
    return lista


def mostrar_vector(v):
    print('Mayores al Promedio')
    print('=' * 60)
    print('{:<6}\t{:<35}\t{:<4}\t{:<10}'.format('Codigo', 'Descripcion', 'Tipo', 'Costo'))
    for servicio in v:
        print(to_string(servicio))


def ordenar_vector(vector):
    tam = len(vector)
    for j in range(1, tam):
        y = vector[j]
```

```
        k = j - 1
        while k <= 0 and y.codigo < vector[k].codigo:
            vector[k + 1] = vector[k]
            k -= 1
        vector[k + 1] = y


def contar_por_tipo(vector):
    vc = [0] * 15
    for servicio in vector:
        vc[servicio.tipo] += 1
    return vc


def buscar_servicio(vector, codigo):
    for i in range(len(vector)):
        if vector[i].codigo == codigo:
            return i
    return -1


def test():
    n = validar_tamanio()
    vector = cargar_automatico(n)
    opcion = -1
    while opcion != 0:
        print('Menu de Opciones')
        print('================================')
        print('1 Mostar los incidentes mayor al promedio')
        print('2 Listar Ordenado por Codigo')
        print('3 Contar por tipo de incidente')
        print('4 Buscar un codigo de servicio')
        print('0 Salir')
        opcion = int(input('Ingrese la opcion: '))
        if opcion == 1:
            prom = promediar_costo(vector)
            lista = buscar_costo_mayores(vector, prom)
            mostrar_vector(lista)
        elif opcion == 2:
            ordenar_vector(vector)
            mostrar_vector(vector)
        elif opcion == 3:
            vc = contar_por_tipo(vector)
            for i in range(len(vc)):
                print('Tipo', i, ':', vc[i], end=' | ')
        elif opcion == 4:
            codigo = int(input('Ingrese el codigo a buscar: '))
            pos = buscar_servicio(vector, codigo)
            if pos != -1:
                print('Datos del servicio')
                print(to_string(vector[pos]))
            else:
                print('No existe un servicio con ese codigo')
        print('\n')

test()
```

13. La competicion de Tenis de Mesa

El Comite Olimpico Internacional solicito un programa que permita sacar una serie de estadisticas referidas a la disciplina olimpica Tenis de Mesa. De cada competidor se carga el numero de atleta, nombre, nacionalidad (valor de 0 a 19), cantidad de puntos ganados, cantidad de puntos perdidos

1. Cargar un arreglo con n competidores, validando que la nacionalidad sea correcta
2. Listar los competidores ordenados por nombre en forma alfabetica
3. Determinar la cantidad total por nacionalidad de puntos ganados y perdidos
4. Determinar la nacionalidad con mejor porcentaje de victoria (Sobre el total de puntos jugados por la nacionalidad, calcular el porcentaje que representa los puntos ganados respectos a los perdidos)
5. Buscar un competidor por su nombre, y mostrar sus datos junto con su porcentaje de efectivada. Caso contrario mostrar un mensaje que no se encontro.

13.1. competidor.py

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

class Competidor:
    def __init__(self,numero, nombre, nac, ganados, perdidos):
        self.numero = numero
        self.nombre = nombre
        self.nacionalidad = nac
        self.ganados = ganados
        self.perdidos = perdidos


class TotalNacionalidad:
    def __init__(self, ganados=0, perdidos=0):
        self.ganados = ganados
        self.perdidos = perdidos


def to_string(competidor):
    linea = '{:<10}\t{:<40}\t{:<4}\t{:<10}\t{:<10}\n'
    return linea.format(competidor.numero, competidor.nombre,\
                        competidor.nacionalidad, competidor.ganados, \
                        competidor.perdidos)
```

13.2. principal.py

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


def cargar_vector(cantidad):
    vector = [None] * cantidad
    for pos in range(cantidad):
        num = random.randint(1, 1500)
        nombre = 'Pingponero ' + str(random.randint(1, 1500))
        nac = random.randrange(20)
        ganados = random.randint(1, 200)
        perdidos = random.randint(1, 200)
        vector[pos] = Competidor(num, nombre, nac, ganados, perdidos)
    return vector


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


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


def listar(vector):
    lista = '{:<10}\t{:<40}\t{:<4}\t{:<10}\t{:<10}\n{}\n'
    lista = lista.format('Numero','Nombre','Nacion.','P Gdos', 'P. Pdos',('-' *90))
    for comp in vector:
        lista += to_string(comp)
    return lista


def totalizarPorNacionalidad(vector):
    v =[TotalNacionalidad()] * 20
    for comp in vector:
        v[comp.nacionalidad].ganados += comp.ganados
        v[comp.nacionalidad].perdidos += comp.perdidos
    return v


def calcular_efectividad(ganados, perdidos):
    total = ganados + perdidos
    return ganados / total * 100


def determinar_nacionalidad(vector):
    mayor = 0
    for pos in range(1, len(vector)):
        mp = calcular_efectividad(vector[mayor].ganados, vector[mayor].perdidos)
        actual = calcular_efectividad(vector[pos].ganados, vector[pos].perdidos)
        if actual < mp:
            mayor = pos
    return mayor


def buscar(nombre, vector):
    tam = len(vector)
    for i in range(tam):
        if vector[i].nombre == nombre:
```

```
            return i
    return -1


def main():
    menu = 'Menu de Opciones\n' \
           '==================================\n' \
           '1\t Cargar Arreglo Competidores\n' \
           '2\t Listar competidores en forma alfabetica\n' \
           '3\t Cantidad de puntos ganados y perdidos\n' \
           '4\t Nacionalidad con Menor Efectividad\n'\
           '5\t Buscar Competidor\n' \
           '6\t Salir\n' \
           'Ingrese su opcion: '
    vector = None
    va = None
    opcion = 0
    while opcion != 6:
        opcion = int(input(menu))
        if opcion == 1:
            n = validar_mayor(0, 'Ingrese la cantidad de competidores:')
            vector = cargar_vector(n)

        else:
            if not vector is None:
                if opcion == 2:
                    ordenar(vector)
                    cadena = listar(vector)
                    print(cadena)

                elif opcion == 3:
                    va = totalizarPorNacionalidad(vector)
                    linea ='{:<7}\t{:<7}\t{:<8}\n'
                    reporte = '{:<7}\t{:<7}\t{:<8}\n{}\n'.format('Nacion.', 'Ganados', 'Perdidos', ('-' * 30))
                    for i in range(len(va)):
                        reporte += linea.format(i, va[i].ganados, va[i].perdidos)
                    print(reporte)

                elif opcion == 4:
                    if not va is None:
                        nac = determinar_nacionalidad(va)
                        print('La nacionalidad con mejor efectividad es', nac)
                    else:
                        print('No hay valores acumulados por nacionalidad')

                elif opcion == 5:
                    nom = input('Ingrese el nombre del competidor a buscar: ')
                    pos = buscar(nom, vector)
                    if pos != -1:
                        competidor = vector[pos]
                        print('El usuario ', to_string(competidor), ' y tuvo un efectivdad', end=' ')
                        print(calcular_efectividad(competidor.ganados, competidor.perdidos), '%')
                else:
                    print('No hay competidores cargados')


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

14. Medallero Olímpico

El Comité Olímpico desea un programa para presentar los resultados de Rio 2016.

Por cada país participante, se conoce: nombre, continente (0:América / 1:Europa / 2:Asia / 3:África / 4:Oceanía), medallas de oro, plata y bronce.

Cargar en forma manual o automática los datos de un conjunto de n países (n se ingresa por teclado) y luego implementar un menú con las siguientes opciones:

1. Presentar un listado ordenado por cantidad de medallas, de mayor a menor, conteniendo nombre del país, nombre del continente, medallas de oro, medallas de plata, medallas de bronce, total de medallas
2. Calcular el promedio de medallas de oro, para los países de un continente que se ingresa por teclado.
3. Informar los datos del país que obtuvo la mayor cantidad de medallas de plata. Si varios países tienen esa cantidad, mostrar todos.
4. Determinar cuántos países obtuvieron sólo medallas de bronce, y qué porcentaje representan sobre el total.
5. Mostrar la cantidad de países por cada continente que participaron (5 totales)

14.1. principal base

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

import random
from registro import *


def validar_tamanio():
    n = int(input('Ingrese tamaño del vector: '))
    while n <= 0:
        n = int(input('Inválido. Debe ser un valor positivo. Ingrese otro: '))
    return n


def cargar_automatico(vector):
    pre = ('Sol', 'Argen', 'Pol', 'Bra', 'Ale', 'Esp', 'Rus', 'Chi', 'Aus', 'Ton', 'Pre', 'A', 'Te', 'Cor', 'Den', 'Lit', 'Teh',
'A', 'L', 'Es')
    pos = ('ania', 'ibia', 'tina', 'onia', 'sia', 'ivia', 'ria', 'tia', 'cia', 'tua', 'dor', 'landa', 'andia')
    for i in range(len(vector)):
        nombre = random.choice(pre) + random.choice(pos)
        continente = random.randrange(5)
        oro = random.randrange(50)
        plata = random.randrange(50)
        bronce = random.randrange(50)
        vector[i] = Pais(nombre, continente, oro, plata, bronce)


def mostrar_menu():
    print('OLIMPIADAS RIO 2016 - Menú de Opciones')
    print('1 - Medallero')
    print('2 - Promedio de oro')
    print('3 - Mayor de plata')
    print('4 - Sólo bronce')
    print('5 - Países por continente')
    print('0 - Salir')
    opcion = int(input('\nIngrese opción: '))
    return opcion


def test():
    n = validar_tamanio()
    vector = [None] * n
    cargar_automatico(vector)
    opcion = -1
    while opcion != 0:
        print('-' * 50)
        opcion = mostrar_menu()
        # Procesar opción elegida
        print('-' * 50)


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

14.2. registro

```
def describir_continente(cod):
    continentes = ['0-América', '1-Europa', '2-Asia', '3-África', '4-Oceanía']
    return continentes[cod]


class Pais:
    def __init__(self, nombre, continente, oro, plata, bronce):
        self.nombre = nombre
        self.continente = continente
        self.oro = oro
        self.plata = plata
        self.bronce = bronce
        self.total = oro + plata + bronce


def to_string(pais):
    cad = 'Nombre: {:<10} | Continente: {:<10} | Oro: {:>3} | Plata: {:>3} | Bronce: {:>3} | Total: {:>3}'
    return cad.format(pais.nombre, describir_continente(pais.continente), pais.oro, pais.plata, pais.bronce, pais.total)


def prueba():
    p1 = Pais('Italia', 1, 10, 3, 4)
    p2 = Pais('Argentina', 0, 12, 3, 1)
    print(to_string(p1))
    print(to_string(p2))


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

14.3. principal

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

import random
from registro import *


def validar_tamanio():
    n = int(input('Ingrese tamaño del vector: '))
    while n <= 0:
        n = int(input('Inválido. Debe ser un valor positivo. Ingrese otro: '))
    return n


def cargar_automatico(vector):
    pre = (
        'Sol', 'Argen', 'Pol', 'Bra', 'Ale', 'Esp', 'Rus', 'Chi', 'Aus', 'Ton', 'Pre', 'A', 'Te', 'Cor', 'Den', 'Lit',
        'Teh', 'A', 'L', 'Es')
    pos = ('ania', 'ibia', 'tina', 'onia', 'sia', 'ivia', 'ria', 'tia', 'cia', 'tua', 'dor', 'landa', 'andia')
    for i in range(len(vector)):
        nombre = random.choice(pre) + random.choice(pos)
        continente = random.randrange(5)
        oro = random.randrange(3)
        plata = random.randrange(3)
        bronce = random.randrange(50)
        vector[i] = Pais(nombre, continente, oro, plata, bronce)


def mostrar_menu():
    print('OLIMPIADAS RIO 2016 - Menú de Opciones')
    print('1 - Medallero')
    print('2 - Promedio de oro')
    print('3 - Mayor de plata')
    print('4 - Sólo bronce')
    print('5 - Países por continente')
    print('0 - Salir')
    opcion = int(input('\nIngrese opción: '))
    return opcion


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


def mostrar_vector(v):
    print('MEDALLERO OLÍMPICO')
    for reg in v:
        print(to_string(reg))


def validar_entre(val1, val2, mensaje):
    num = int(input(mensaje))
    while num < val1 or num > val2:
        print('Invalido!')
        num = int(input(mensaje))
    return num


def promediar_oro(vector, c):
    suma = cant = 0
    for pais in vector:
        if pais.continente == c:
            suma += pais.oro
            cant += 1
    if cant > 0:
```

```
        return suma / cant
    else:
        return 0


def buscar_mayor_plata(vector):
    may = vector[0]
    paises = [may]
    for i in range(len(vector)):
        if vector[i].plata == may.plata:
            paises.append(vector[i])
        elif vector[i].plata > may.plata:
            may = vector[i]
            paises = [may]
    return paises


def calcular_porcentaje_solo_bronce(vector):
    cant = 0
    for pais in vector:
        if pais.oro == 0 and pais.plata == 0 and pais.bronce > 0:
            cant += 1
    return cant * 100 / len(vector)


def contar_por_continente(vector):
    cont = [0] * 5
    for pais in vector:
        cont[pais.continente] += 1
    return cont


def mostrar_conteo(cont):
    for i in range(len(cont)):
        if cont[i] > 0:
            print(describir_continente(i), ':', cont[i], 'países')


def principal():
    n = validar_tamanio()
    vector = [None] * n
    cargar_automatico(vector)
    opcion = -1
    while opcion != 0:
        print('-' * 50)
        opcion = mostrar_menu()
        if opcion == 1:
            ordenar_descendente(vector)
            mostrar_vector(vector)
        elif opcion == 2:
            c = validar_entre(0, 4, 'Ingrese continente: ')
            prom = promediar_oro(vector, c)
            print('Promedio de medallas de oro para ese continente:', prom)
        elif opcion == 3:
            paises = buscar_mayor_plata(vector)
            print('Países con más medallas de plata')
            mostrar_vector(paises)
        elif opcion == 4:
            porc = calcular_porcentaje_solo_bronce(vector)
            print('Porcentaje de países solo bronce', porc, '%')
        elif opcion == 5:
            cont = contar_por_continente(vector)
            mostrar_conteo(cont)
        print('-' * 50)


if __name__ == '__main__':
```

```
principal()
```

15. Hotel

Un importante hotel necesita un sistema que permita gestionar la disponibilidad de sus habitaciones.

Por cada habitación se conoce: número (un valor de 3 dígitos), capacidad (2 a 4 huéspedes), tarifa por noche, estado (0=libre, 1=reservada, 2=ocupada). Tener en cuenta que el primer dígito del número de habitación es el piso (un valor del 1 al 9).

Se pide un programa que permita cargar (manual o automáticamente) un vector de n habitaciones (n se ingresa por teclado) y luego ordenarlo por número de habitación. A continuación, debe ofrecer un menú con las siguientes opciones:

1. Consultar: mostrar el listado completo de habitaciones.
2. Estadísticas: generar y mostrar una matriz de conteo de habitaciones, donde cada fila represente un piso y cada columna un estado.
3. Reservar habitación: Ingresar cantidad de huéspedes deseada. Si se encuentra alguna habitación libre mostrar datos y reservar; si no, informarlo.
4. Entregar habitación: Ingresar un número de habitación. Si no existe, informarlo. Si existe y está libre o reservada, cambiar el estado a ocupada y mostrar los datos, en caso contrario mostrar un aviso.
5. Contabilidad: determinar cuál será la recaudación del hotel, considerando las habitaciones ocupadas. Si por cada habitación (sin importar su estado) tiene un costo diario de mantenimiento de $ 250, obtendrá pérdidas o ganancias?

15.1. registro.py

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

class Habitacion:

    def __init__(self, numero, capacidad, tarifa, estado):
        self.numero = numero
        self.capacidad = capacidad
        self.tarifa = tarifa
        self.estado = estado
        self.piso = numero//100


def to_string(hab):
    estados = ('Libre','Reservada','Ocupada')
    ret = 'Numero:{:<5} Capacidad:{:<6} Tarifa ${:>5.2f} Estado: {:<11}'
    return ret.format(hab.numero,hab.capacidad,hab.tarifa,estados[hab.estado])
```

15.2. base.py

```
import random
from registro import *

__author__ = 'Cátedra de Algoritmos y Estructuras de Datos'


def validar_mayor_que (valor, mensaje):
    num = int(input(mensaje))
    while num <= valor:
        print('INVÁLIDO. Debe ser un valor mayor que', valor)
        num = int(input(mensaje))
    return num


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


def cargar_manual(n):
    vec = [None] * n
    for i in range(len(vec)):
        print('-'*100)
        numero = validar_entre(100, 999,'Número de habitación: ')
        capacidad = validar_entre(2, 4, 'Capacidad: ')
        tarifa = validar_mayor_que(0, 'Tarifa: ')
        estado = validar_entre(0, 2, 'Disponibilidad (0=Libre/1=Reservada/2=Ocupada): ')
        vec[i] = Habitacion (numero, capacidad, tarifa, estado)
    return vec


def cargar_automatico(n):
    vec = [None] * n
    for i in range(len(vec)):
        numero = random.randint(100, 999)
        capacidad = random.randint(2, 4)
        tarifa = random.randint(1000,5000)
        estado = random.randint(0, 2)
        vec[i] = Habitacion (numero, capacidad, tarifa, estado)
    return vec


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


def principal():
    #Carga
    n = validar_mayor_que(0, 'Ingrese tamaño del vector: ')
    carga = validar_entre(0,1,'Tipo de carga (0:Manual / 1: Automática): ')
    if carga == 0:
        v = cargar_manual(n)
    else:
        v = cargar_automatico(n)
    #Menu de opciones
    opcion = -1
    while opcion != 0:
        print('x'* 100)
        print('SISTEMA DE HABITACIONES')
        print('1- Consultar')
        print('2- Estadísticas')
        print('3- Reservar habitación')
        print('4- Entregar habitación')
```

```
        print('5- Contabilidad')
        print('0- Salir')
        opcion = int(input('Ingrese opción: '))
        if opcion == 1:
            mostrar_vector(v)
        print('x'* 100)


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

15.3. principal.py

```
import random
from registro import *

__author__ = 'Cátedra de Algoritmos y Estructuras de Datos'


def validar_mayor_que (valor, mensaje):
    num = int(input(mensaje))
    while num <= valor:
        print('INVÁLIDO. Debe ser un valor mayor que', valor)
        num = int(input(mensaje))
    return num


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


def cargar_manual(n):
    vec = [None] * n
    for i in range(len(vec)):
        print('-'*100)
        numero = validar_entre(100, 999,'Número de habitación: ')
        capacidad = validar_entre(2, 4, 'Capacidad: ')
        tarifa = validar_mayor_que(0, 'Tarifa: ')
        estado = validar_entre(0, 2, 'Disponibilidad (0=Libre/1=Reservada/2=Ocupada): ')
        vec[i] = Habitacion (numero, capacidad, tarifa, estado)
    return vec


def cargar_automatico(n):
    vec = [None] * n
    for i in range(len(vec)):
        numero = random.randint(100, 999)
        capacidad = random.randint(2, 4)
        tarifa = random.randint(1000,5000)
        estado = random.randint(0, 2)
        vec[i] = Habitacion (numero, capacidad, tarifa, estado)
    return vec


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


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


def reservar_habitacion(v, huespedes):
    for i in range(len(v)):
        if v[i].estado == 0 and v[i].capacidad == huespedes:
            return i
    return -1


def buscar_por_numero(v,x):
    izq, der = 0, len(v) - 1
    while izq <= der:
```

```
        c = (izq + der) // 2
        if x == v[c].numero:
            return c
        if x < v[c].numero:
            der = c - 1
        else:
            izq = c + 1
    return -1


def generar_conteo(v):
    m = [[0] * 3 for f in range(10)]
    for hab in v:
        m[hab.piso][hab.estado] += 1
    return m


def mostrar_matriz(m):
    for i in range(len(m)):
        print()
        for j in range(len(m[i])):
            print('[',m[i][j],']',sep='',end=' ')
    print()


def calcular_recaudacion(v):
    recaudacion = 0
    for hab in v:
        if hab.estado == 2:
            recaudacion += hab.tarifa
    return recaudacion

def principal():
    #Carga
    n = validar_mayor_que(0, 'Ingrese tamaño del vector: ')
    carga = validar_entre(0,1,'Tipo de carga (0:Manual / 1: Automática): ')
    if carga == 0:
        v = cargar_manual(n)
    else:
        v = cargar_automatico(n)
    ordenar_por_numero(v)
    #Menu de opciones
    opcion = -1
    while opcion != 0:
        print('x'* 100)
        print('SISTEMA DE HABITACIONES')
        print('1- Consultar')
        print('2- Estadísticas')
        print('3- Reservar habitación')
        print('4- Entregar habitación')
        print('5- Contabilidad')
        print('0- Salir')
        opcion = int(input('Ingrese opción: '))
        if opcion == 1:
            mostrar_vector(v)
        elif opcion == 2:
            m = generar_conteo(v)
            mostrar_matriz(m)
        elif opcion == 3:
            huespedes = validar_entre(2,4,'Ingrese cantidad de huespedes: ')
            pos = reservar_habitacion(v,huespedes)
            if pos == -1:
                print('No hay habitaciones disponibles')
            else:
                v[pos].estado = 1
                print('Habitación reservada:',to_string(v[pos]))
        elif opcion == 4:
```

```
            numero = validar_entre(100,999,'Ingrese habitación buscada: ')
            pos = buscar_por_numero(v,numero)
            if pos == -1:
                print('La habitación ingresada no existe')
            elif v[pos].estado == 2:
                print('La habitación ya está ocupada')
            else:
                v[pos].estado = 2
                print('Habitación entregada:',to_string(v[pos]))
        elif opcion == 5:
            recaudacion = calcular_recaudacion(v)
            diferencia = recaudacion - 250*n
            if diferencia < 0:
                print('El hotel obtiene perdidas por $',diferencia)
            else:
                print('El hotel obtiene ganancias por $',diferencia)
        print('x'* 100)


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