Parciales anteriores

Ejercicios resueltos

1) Un sitio web de ventas online necesita un programa para procesar las ventas a diferentes partes del mundo. De cada venta se tiene la siguiente información: número de operación (único), nombre del comprador, país de destino, monto en $, código del artículo comprado. Estos datos deben almacenarse en un diccionario, permitiendo acceder de forma directa a las compras de cada comprador.

Por otra parte, en otra estructura se deben almacenar los artículos disponibles a la venta, de los cuales se tiene: el código de cada artículo y su descripción. Generar un diccionario con esta información.

a) Permitir la carga de las ventas realizadas y de artículos disponibles. Elegir las condiciones de corte y hacer las funciones necesarias.

b) Hacer una función que reciba el código de un artículo y el contenedor o estructura necesaria como parámetro e imprima:

b-1) todas las ventas de ese artículo, respetando el siguiente formato:

OPERACIÓN: 92301
COMPRADOR: John Edwards
PAÍS: Inglaterra
MONTO: $405.78

b-2) promedio de ventas del artículo indicado.

c) Imprimir las descripciones de todos los artículos que fueron vendidos al menos una vez (evitando imprimir descripciones repetidas).

RESOLUCIÓN:

```
#funciones.py

def cargarArticulos(articulos):
    #Carga artículos en el diccionario pasado por parámetro y luego lo retorna
    codigo=int(input("Código de artículo (-1 para cortar): "))
    while codigo!=-1:
        descripcion=input("Descripción del artículo: ")
        articulos[codigo]=descripcion
        codigo=int(input("Código de artículo: "))
    return articulos


def cargarVentas(ventas):
    #Carga ventas en el diccionario pasado por parámetro y luego lo retorna. La clave es el
número de operación y el valor es una lista con el resto de los datos
```

```
    operacion=int(input("Número de operación (0 para cortar): "))
    while operacion!=0:
        comprador=input("Comprador: ")
        pais=input("País: ")
        monto=float(input("Monto de la compra: "))
        cod_articulo=int(input("Código de artículo: "))
        ventas[operacion]=[comprador, pais, monto, cod_articulo]
        operacion=int(input("Número de operación (0 para cortar): "))
    return ventas


def ventasPorPaises(ventas):
    #Retorna un diccionario cuyas claves son los nombres de países y los valores representan la cantidad de ventas de cada país
    paises={}
    for venta in ventas.values():
        if venta[1] in paises.keys():
            paises[venta[1]]+=1
        else:
            paises[venta[1]]=1
    return paises


def mayoresVentas(paises):
    #Retorna el nombre del país con la mayor cantidad de ventas, de acuerdo a los contadores de ventas por país contenidos en el diccionario pasado por parámetro
    cantidad_mayor=0
    pais_mayor=""
    for pais, ventas in paises.items():
        if ventas > cantidad_mayor:
            cantidad_mayor=ventas
            pais_mayor=pais
    return pais_mayor


def imprimirVentas(ventas,articulo):
    #Imprime los datos de las ventas del artículo pasado por parámetro
    for operacion, info in ventas.items():
        if info[3]==articulo:
            print("OPERACIÓN: ", operacion)
            print("COMPRADOR: ", info[0])
            print("PAÍS: ", info[1])
            print("MONTO: $", info[2])
```

```
def articulosVendidos(ventas,articulos):
    #Retorna un conjunto con las descripciones de los artículos vendidos al menos una vez
    descripciones=set()
    for lista in ventas.values():
        descripcion=articulos[lista[3]]  #lista[3] contiene un código de artículo
        descripciones.add(descripcion)
    return descripciones


#programa.py

#inciso A
articulos={}
print("***Carga de los artículos***")
articulos=cargarArticulos(articulos)
ventas_realizadas={}
print("***Carga de las ventas***")
ventas_realizadas=cargarVentas(ventas_realizadas)
#inciso B
cantidad_por_pais=ventasPorPaises(ventas_realizadas)
print("País al que se hicieron más ventas: ", mayoresVentas(cantidad_por_pais))
#inciso C
codigo=int(input("Código de artículo a buscar: "))
imprimirVentas(ventas_realizadas,codigo)
#inciso D
print("Artículos vendidos en total: ")
for descripcion in articulosVendidos(ventas_realizadas,articulos):
    print(descripcion)
```

2) Las patrullas del CPC (Comando de Policías Comunitaria) de Junín, por cada intervención del 911 donde participan, registran en una Planilla de Comando: el código de evento (numérico), la fecha (numérico formato AAAAMMDD), hora (numérico formato HHMM) y dirección donde ocurre el evento. Al final del turno la planilla se entrega a un data entry que registra los datos en el sistema, finalizando cuando ingresa el código de evento -1.

Los posibles eventos con sus códigos son:

(1) Accidentes	(2) Consumación de ilícito	(3) Verificación de alarmas
(4) Actitud sospechosa	(5) Consumo alcohol/droga	(6) Incendio y otros estragos
(7) Amenaza de bomba	(8) Conflicto familiar/vecinal	(9) Disturbios

Se solicita:

a) Cargar el registro de los camiones, se deben almacenar en una estructura respetando el orden de carga. Se debe validar que el código de evento se encuentra entre los definidos.

b) Imprimir por pantalla los códigos de eventos sin repetir que ocurrieron en horario nocturno de [19:00hs a 7:00hs) y los códigos de eventos sin repetir que ocurrieron en horario diurno de [7:00hs a 19:00hs).

c) Imprimir por pantalla la cantidad de ocurrencias de cada uno de los eventos ocurridos. Se debe visualizar el código de evento y la cantidad.

d) Informar por pantalla el código de evento que más ocurrencias tuvo, si hay más de un código con igual cantidad, deben informarse todos.

RESOLUCIÓN:

```
#funciones.py

def cargar_eventos():
	#Carga cada evento en una lista y lo ingresa a otra lista de eventos. Finaliza con el evento de código -1.
	eventos = []
	print('Ingrese los datos del evento: ')
	codigo_evento = int(input('Código de evento: '))
	while codigo_evento != -1:
		while True:  #Validación de código de evento
			if (codigo_evento < 1) or (codigo_evento > 9):
				codigo_evento = int(input('Código inválido. Reingrese: '))
			else:
				break

		fecha = int(input('Fecha (aaaammdd): '))
		hora = int(input('Hora (hhmm): '))
		direccion = input('Dirección: ')
		un_evento = [codigo_evento, fecha, hora, direccion]
		eventos.append(un_evento)
		print('Ingrese los datos del evento: ')
		codigo_evento = int(input('Código de evento: '))
	return eventos


def clasificar_eventos(eventos):
	#Dada una lista de eventos retorna una lista compuesta de 2 conjuntos con los eventos clasificados por turno
	diurno = set()
	nocturno = set()
	for e in eventos:
		hora = e[2] // 100
		if hora >= 7 and hora < 19:  #diurno
			diurno.add(e[0])
		else: #nocturno
			nocturno.add(e[0])
	return [diurno, nocturno]
```

```
def imprimir(lista_cadenas):
	#Imprime en pantalla una lista de cadenas
	for c in lista_cadenas:
		print(c)
	return None


def eventos_por_turno(eventos):
	#Dada una lista de eventos, la clasifica por turno y la imprime. Turno diurno = [7:00hs a 19:00hs) y nocturno = [19:00hs a 7:00hs)
	eventos_x_turno = clasificar_eventos(eventos)
	print('Códigos de eventos diurnos:')
	imprimir(eventos_x_turno[0])
	print('Códigos de eventos nocturnos:')
	imprimir(eventos_x_turno[1])
	return None


def imprimir_dic(diccionario):
	#Imprime en pantalla cada clave y valor
	for codigo, cantidad in diccionario.items():
		print(codigo, ": ", cantidad)
	return None


def eventos_x_cantidad(eventos):
	#Dada una lista de eventos, retorna un diccionario con la asociación [código de evento]: cantidad
	eventos_cantidad = {}
	for e in eventos:
		if e[0] in eventos_cantidad:  #si la clave existe en el diccionario, se le suma 1. Si no, se inicia en 1.
			eventos_cantidad[e[0]]+=1
		else:
			eventos_cantidad[e[0]]=1
	return eventos_cantidad


def eventos_mayor_cantidad(eventos_cantidad):
	#Retorna una lista con los códigos de eventos que más ocurrieron
	mayor = -1
	mayores_eventos = []
	for codigo in eventos_cantidad:
		if mayor < eventos_cantidad[codigo]:
```

```
                mayor = eventos_cantidad[codigo]
                mayores_eventos.clear()
                mayores_eventos.append(codigo)
            elif mayor == eventos_cantidad[codigo]:
                mayores_eventos.append(codigo)
    return mayores_eventos


def informar_eventos_mayor_cantidad(eventos_cantidad):
    #Informa en pantalla los códigos de eventos que más ocurrieron
    mayores_eventos = eventos_mayor_cantidad(eventos_cantidad);
    print('Eventos que más ocurrieron:')
    imprimir(mayores_eventos)
    return None


#programa.py

#Inciso A
eventos = cargar_eventos()
#Inciso B
eventos_por_turno(eventos)
#Inciso C
print("Cantidad de eventos por código:")
eventos_cantidad = eventos_x_cantidad(eventos)
imprimir_dic(eventos_cantidad)
#Inciso D
informar_eventos_mayor_cantidad(eventos_cantidad)
```

3) Una red social necesita almacenar los nombres de sus usuarios en una estructura que permita acceder a ellos mediante un código identificador numérico. Es decir, de cada usuario se tiene: código identificador y nombre (por ejemplo, el código 3145 corresponde al usuario “laureano_98”).

Además, esta red social permite que los usuarios creen páginas temáticas donde pueden hacer publicaciones; la información de estas páginas debe almacenarse en otra estructura. De cada página se tienen los siguientes datos: **título**, **tema**, **código** del usuario que creó la página, **visitas** que tuvo la página.

Suponer la existencia de dos funciones de carga de los datos, que ya están codificadas y listas para utilizar, llamadas: **cargarUsuarios(usuarios)** y **cargarPaginas(paginas)**, que se encargan, la primera de almacenar los datos de usuarios en el diccionario recibido por parámetro (cuya clave es el identificador de usuario y el valor es el nombre) y, la segunda, de almacenar los datos de páginas en la lista pasada por parámetro (esta lista contiene, a su vez, listas con los datos de cada página, en el orden en que se indican en el párrafo anterior).

a) Según el tipo de cada estructura necesaria, crearlas adecuadamente, invocando a las funciones cargarUsuarios(usuarios) y cargarPaginas(paginas) para cargar los datos en ambas estructuras.

b) Solicitar el ingreso por teclado de un código identificador de usuario y listar todas las páginas del usuario cuyo código coincide con el ingresado, con el siguiente formato:
Título: [título de la página]
Grupo temático: [tema de la página]
Visitas: [visitas de la página]

c) Imprimir el título y tema de la página que tuvo mayor cantidad de visitas.

d) Mostrar un listado de los nombres de los usuarios que no tienen ninguna página.

Modularizar adecuadamente donde sea necesario.

RESOLUCIÓN:

```
#funciones.py

#LAS DOS PRIMERAS FUNCIONES SE ASUMEN YA HECHAS Y NO DEBEN RESOLVERSE EN EL EXAMEN

def cargarUsuarios(usuarios):
	#carga en el diccionario algunos usuarios de ejemplo. La clave es el código identificador y el valor es el nombre de usuario.
	usuarios[3145]="laureano_98"
	usuarios[1123]="marinaperez"
	usuarios[1423]="russo1234"
	usuarios[2235]="snowYWinter"
	usuarios[2631]="flanders.981"
	usuarios[3676]="pinky9bear"
	return usuarios

def cargarPaginas(paginas):
	#carga en la lista algunas páginas de ejemplo. Cada página se representa por una lista cuyos datos son: título, tema, código identificador del usuario que la creó, cantidad de visitas
	paginas.append(["Niños creativos","Juguetes artesanales",3145,192])
	paginas.append(["Mistify","Comercios locales",3676,1344])
	paginas.append(["Bull Rock","Bebidas",2631,233])
	paginas.append(["Animal freedom","ONGs",3676,98112])
	paginas.append(["Asociacion de estilistas argentinos","ONGs",3145,895])
	paginas.append(["La voz","Programas de TV",3145,87])
	return paginas


#A PARTIR DE ACÁ, LA RESOLUCIÓN DEL EJERCICIO:

def listarPaginasDeUsuario(usuario,paginas):
	#imprime, con el formato pedido, los datos de páginas que correspondan al código identificador de usuario pasado por parámetro
```

```
	for pagina in paginas:
		if pagina[2]==usuario:
			print("Título: ",pagina[0])
			print("Grupo temático: ",pagina[1])
			print("Visitas: ",pagina[3])
	return None


def mayorCantidadVisitas(paginas):
	#calcula la página que tuvo más visitas y retorna una lista con los datos de esa página
	maxVisitas=0
	paginaMaxVisitas=[]
	for pagina in paginas:
		if pagina[3]>maxVisitas:
			maxVisitas=pagina[3]
			paginaMaxVisitas=pagina
	return paginaMaxVisitas


def usuariosConPaginas(usuarios,paginas):
	#retorna un conjunto con los códigos identificadores de usuarios que tienen alguna página
	conPaginas=set()
	for pagina in paginas:
		conPaginas.add(pagina[2])
	return conPaginas


def listarUsuariosSinPaginas(usuarios,paginas):
	#muestra los usuarios que no tienen páginas
	conPaginas=usuariosConPaginas(usuarios,paginas)
	for codigo,nombre in usuarios.items():
		if codigo not in conPaginas:
			print(nombre)
	return None


#programa.py

usuarios=dict()
paginas=list()
cargarUsuarios(usuarios)
cargarPaginas(paginas)
user=int(input("Usuario para listar sus páginas: "))
listarPaginasDeUsuario(user,paginas)
pagina_mayor_visitas=mayorCantidadVisitas(paginas)
```

```
print("Página con mayor cantidad de visitas")
print("Título: ", pagina_mayor_visitas[0], " - Tema: ", pagina_mayor_visitas[1])
print("Usuarios que no tienen páginas:")
listarUsuariosSinPaginas(usuarios,paginas)
```

4) El sistema de molinetes de un estado de fútbol almacena, por cada ingreso al estadio, información de los socios en una estructura. De cada socio del club se registra el número de socio y datos personales, como DNI, apellido, nombre y fecha de afiliación (en formato aaaammdd) y edad. La carga finaliza con el socio cuyo número de socio es 999 que debe ser ingresado y procesado.

Se solicita:

a) Durante la carga verificar que no haya sido ingresado un socio con el mismo número de socio, en dicho caso, se debe informar la situación y continuar con el próximo.

b) Procesar la estructura e informar cuántos socios menores (menor a 8 años), activos (entre 8 y 60 años, ambos inclusive) y mayores (mayor a 60) ingresaron.

c) Solicitar el ingreso de un número de socio, en caso de que haya ingresado mostrar los datos personales, si no se encuentra informar tal situación.

d) Procesar la estructura e informar los años de afiliación de los socios ingresados. No se deben informar años duplicados.

RESOLUCIÓN:

```
#funciones.py

def cargar_ingreso_socios():
      #retorna un diccionario con los datos de socios, donde la clave es el número y el valor es una lista con los demás datos
      socios = {}
      while True:
            print('Ingrese los datos del socio:')
            nro_socio = int(input("Nro de socio: "))
            dni = input("DNI: ")
            apellido = input("Apellido: ")
            nombre = input("Nombre: ")
            fecha = input("Fecha de afiliación: ")
            edad = int(input("Edad: "))
            if nro_socio in socios.keys():
                  print("El socio ", nro_socio, " ya fue ingresado.")
            else:
                  socios[nro_socio] = [dni, apellido, nombre, fecha, edad]
            if nro_socio == 999:
```

```
                break
        return socios


def clasificar_ingreso_socios(socios):
        #clasifica los socios del diccionario según su edad y retorna una lista con la cantidad de socios menores de 8 años, cantidad de activos y cantidad de mayores de 60.
        menor = 0
        activo = 0
        mayor = 0
        for datos in socios.values():
                if datos[4] < 8:
                        menor += 1
                elif datos[4] > 60:
                        mayor += 1
                else:
                        activo += 1;
        return [menor, activo, mayor]

def informar_socio(numero, socio):
        #Recibe el número de un socio y una lista con sus datos e imprime la información del socio
        print("Nro de socio: ", numero)
        print("DNI: ", socio[0])
        print("Apellido: ", socio[1])
        print("Nombre: ", socio[2])
        print("Fecha de afiliación: ", socio[3][6:8] + '/' + socio[3][4:6] + '/' + socio[3][0:4])
        print("Edad: ", socio[4])


def buscar_socio(socios, numero):
        #busca en el diccionario de socios al socio cuyo número corresponde al pasado por parámetro. Si lo encuentra, informa sus datos. Si no, informa un mensaje de error.
        if numero not in socios.keys():
                print("Error: no se encuentra socio con número ", numero)
        else:
                informar_socio(numero, socios[numero])


def calcular_anios(socios):
        #retorna un conjunto con todos los años de las fechas que están en las listas del diccionario socios
        fechas=set()
        for datos in socios.values():
                fechas.add(datos[3][0:4])
```

```
    return fechas


def imprimir_fechas(socios):
    #Imprime los años de las fechas que están en las listas del diccionario socios
    print('Las fechas de los socios que ingresaron a la cancha son:')
    for fecha in calcular_anios(socios):
        print(fecha)


#programa.py

#Inciso A
listadoSocios = cargar_ingreso_socios();
#Inciso B
clasif = clasificar_ingreso_socios(listadoSocios)
print("Ingresaron ", clasif[0], " menores, ", clasif[1], " activos y ", clasif[2], "
mayores.")
#Inciso C
nro_socio_buscado = int(input("Ingrese el Nro. de Socio a buscar: "))
buscar_socio(listadoSocios, nro_socio_buscado)
#Inciso D
imprimir_fechas(listadoSocios)
```

5) Realizar un programa que permita al usuario ingresar strings por teclado, finalizando el ingreso al leerse un string que contenga ";" (punto y coma), el cual también debe procesarse. Los strings no deben almacenarse sino que se procesan a medida que se ingresan.

Cada vez que el usuario ingrese un string de longitud 1 que contenga sólo una barra "/" se considera que termina una línea. Se debe informar cuántas líneas en total se ingresaron (incluyendo la última, que termina con ";" aunque no haya una "/"). Además, por cada línea, informar cuántos dígitos numéricos (del 0 al 9) aparecieron en total (en todos los strings que componen en esa línea). Por ejemplo, si los strings ingresados por el usuario son:

"Roberto", "1233", "registrado en 3er lugar", "/", "Juan Martínez", "cinco", "68", "mayor", "1er premio", "/", "exceptuado", "9001", "pagado", "/", "visto", "finalizar;"

Se deberá informar:
Aparecen 5 dígitos en la línea.
Aparecen 3 dígitos en la línea.
Aparecen 4 dígitos en la línea.
Aparecen 0 dígitos en la línea.
Se leyeron 4 líneas.

RESOLUCIÓN:

```
lineas=0
digitos="0123456789"
cantidadDigitos=0
while True:
      cadena=input("Cadena: ")
      for caracter in cadena:
            if caracter in digitos:
                  cantidadDigitos+=1
      if cadena=="/" or ";" in cadena:
            lineas+=1
            print("Aparecen ", cantidadDigitos, " digitos en la línea")
            cantidadDigitos=0
      if ";" in cadena:
            break
print("Se leyeron ",lineas," lineas")
```

6) Implementar una función que cumpla con lo siguiente:

Dado un número natural **n** que se recibe por parámetro, la función retorna una lista conformada por dos listas: una con los dígitos pares y otra con los dígitos impares. Ambas no contienen dígitos repetidos. En caso de no tener algún tipo de dígitos, las listas se retornan vacías. Además, si **n** no es natural, se debe retornar un mensaje de error.

RESOLUCIÓN:

```
def clasificacion_digitos(n):
      if n < 0:
            return 'ERROR: el número no es natural'
      pares = set()
      impares = set()
      while (n!=0):
            resto = n % 10
            if resto % 2 == 0:
                  pares.add(resto)
            else:
                  impares.add(resto)
            n = n // 10
      return [list(pares), list(impares)]
```

7) Escriba un programa que solicite ingresar una cantidad de números a procesar para luego ir ingresando, uno a uno,

la cantidad pedida de números enteros. El programa debe crear una lista con cada número ingresado de la siguiente manera: si el número es mayor que cero la lista debe estar compuesta por los números pares desde el cero hasta el ingresado incluido (por ejemplo, si el número ingresado es el 6 la lista será [0, 2, 4, 6]), pero si el número es menor a cero la lista estará formada por todos los números desde el ingresado hasta el cero (por ejemplo, si el número ingresado es el -5 la lista será [-5, -4,-3, -2, -1, 0]).

La creación de cada lista debe delegarse en dos funciones diferentes.

Además en la lectura de cada uno de los números el programa principal debe delegar en una tercera función la impresión de la lista creada para ese número, pero no lo hará con la sentencia print(lista) sino que la función debe informar algo como lo siguiente:

El elemento en el lugar 0 es 0
El elemento en el lugar 1 es 2
El elemento en el lugar 2 es 4
El elemento en el lugar 3 es 6

RESOLUCIÓN:

```
#funciones.py

def crearListaMayor(numero):
	#Retorna una lista con los números pares desde el 0 hasta el pasado por parámetro
	lista=list()
	for i in range(0, numero +1, 2):
		lista.append(i)
	return lista


def crearListaMenor(numero):
	#Retorna una lista con los números desde el pasado por parámetro hasta el 0
	lista=list()
	for i in range(numero, 1):
		lista.append(i)
	return lista


def imprimirLista(lista):
	#Imprime la lista pasada por parámetro con el formato pedido
	for i in range(len(lista)):
		print("El elemento en el lugar ", i, " es ", lista[i])


#programa.py

cantidad=int(input("Ingrese la cantidad de números a procesar: "))
listaActual=[]
```

```
for i in range(cantidad):
	número=int(input("Ingrese el número a procesar: "))
	if número > 0:
		listaActual=crearListaMayor(número)
	else:
		listaActual=crearListaMenor(número)
	imprimirLista(listaActual)
```

8) El jardín maternal "Tallitos" cumple 10 años desde su creación. Federica, su directora, quiere hacer una fiesta y para eso necesita organizar algunos datos de manera de poder invitar a todos.

En un cuaderno tiene registrados los siguientes datos de todos los alumnos del jardín desde sus comienzos: DNI, nombre y apellido, fecha de ingreso (formato AAAAMMDD), fecha de egreso (formato AAAAMMDD), dirección.

La carga de información finaliza al ingresar DNI con valor 0 (cero).

a) Almacenar la información en una estructura adecuada, donde se acceda por DNI.

b) Informar el nombre y apellido de aquellos alumnos que egresaron en el año 2008. Delegar este punto en una función.

c) A partir de la estructura obtenida en el punto **a**, armar una lista que contenga los alumnos que ingresaron en el año 2006. La lista debe contener listas con DNI y nombre y apellido. Realizar el inciso usando una función que reciba la estructura original y retorne la lista.

Es el programa principal quien deberá informar la lista obtenida.

d) Implementar una función que informe cuál fue el año en que egresaron más alumnos y su cantidad respectiva.

RESOLUCIÓN:

```
#funciones.py

def obtenerAnio(fecha):
	#Dada una fecha en formato AAAAMMDD, retorna el año
	return fecha//10000


def egresadosAnio(alumnos, anio):
	#Imprime los alumnos egresados en el año pasado por parámetro
	for valor in alumnos.values():
		if obtenerAnio(valor[2])==anio:
			print ("El alumno ", valor[0], " egresó en el año 2008 ")


def ingresantesAnio(alumnos, anio):
	#Retorna una lista que contenga los alumnos que ingresaron en el año pasado por parámetro,
con dni y apellido contenidos en listas
	lista=[]
	for clave, valor in alumnos.items():
		if obtenerAnio(valor[1])==anio:
			alumno=[valor[0], clave]
			lista.append(alumno)
```

```
      return lista


def cantidadEgresadosPorAnio(alumnos):
      #Retorna un diccionario con los años de egreso y cantidad de egresados por cada uno
      egresados={}
      for datos in alumnos.values():
            anio=obtenerAnio(datos[2])
            if anio in egresados.keys():
                  egresados[anio]+=1
            else:
                  egresados[anio]=1


#programa.py

alumnos={}
#Inciso A
dni=int(input("Ingrese un número de DNI "))
while dni!=0:
      nombre=input("Ingrese el nombre y apellido del alumno ")
      fIngreso=int(input("Ingrese la fecha de ingreso con el formato AAAAMMDD "))
      fEgreso=int(input("Ingrese la fecha de egreso con el formato AAAAMMDD "))
      dirección=input("Ingrese la dirección del alumno ")
      alumnos[dni]=[nombre, fIngreso, fEgreso, dirección]
      dni=int(input("Ingrese un número de DNI "))

#Inciso B
if len(alumnos)!=0:
      egresadosAnio(alumnos, 2008)
#Inciso C
print("Alumnos que ingresaron en el año 2006 ", ingresantesAnio(alumnos, 2006))
#Inciso D
egresados=cantidadEgresadosPorAnio(alumnos)
maximoEgresos=0
for clave, valor in egresados.items():
      if valor > maximoEgresos:
            maximoEgresos=valor
            maximoanio=clave
print ("El año en que egresaron más alumnos fue :", maximoanio, " y su cantidad
respectiva fue: ", maximoEgresos)
```

9) La Martina es una empresa que posee sucursales en puntos estratégicos del país. La casa central procesa las ventas realizadas por todas las sucursales en períodos regulares de tiempo.

Las ventas deben ser almacenadas en un contenedor finalizando la carga con el producto cuyo código de sucursal es -1, que no debe ser almacenado (seleccionar el contenedor más apropiado según la problemática a resolver). De cada venta se cuenta con la siguiente información: el código de sucursal, código de artículo, la descripción, la cantidad vendida y el precio unitario.

Se solicita:

a) Hacer el módulo de carga de datos.
b) Hacer un módulo en el cual, dado un código de sucursal y un porcentaje de incremento en el precio unitario, actualice el nuevo precio para todos los artículos de dicha sucursal. Se debe informar, para cada artículo actualizado, todos sus datos, con el monto total (cantidad * precio unitario) anterior y posterior a la modificación.
c) Hacer un módulo que, dado un código de artículo, informe ventas totales del mismo.
d) Hacer un módulo que retorne el código de los artículos que en alguna oportunidad hayan tenido una venta donde su cantidad exceda el promedio de la cantidad de artículos vendidos en total. Imprimir los códigos.

RESOLUCIÓN:

```
#funciones.py

def cargar_ventas(ventas):
    #Carga las ventas en la lista pasada por parámetro y la retorna
    cod_sucursal = int(input("Código de sucursal. Finalizar con -1: "))
    while cod_sucursal != -1:
        venta = list()
        venta.append(cod_sucursal)
        venta.append(int(input("Código de artículo: ")))
        venta.append(input("Descripción del artículo: "))
        venta.append(int(input("Cantidad vendida: ")))
        venta.append(float(input("Precio unitario del artículo: $")))
        ventas.append(venta)
        cod_sucursal = int(input("Código de sucursal (finalizar con -1): "))
    return ventas


def imprimir_venta(venta):
    #Muestra datos de una venta
    print("Sucursal ", venta[0])
    print("Código ", venta[1])
    print("Descripción ", venta[2])
    print("Cantidad vendida ", venta[3])
    print("Costo unitario ", venta[4])
```

```
def filtrarSucursal(ventas,sucursal):
    #Retorna una lista con las ventas de la sucursal pasada por parámetro
    filtrada=[]
    for venta in ventas:
        if venta[0]==sucursal:
            filtrada.append(venta)
    return filtrada


def incrementar_costo_suc(suc, incremento, ventas):
    #Por cada venta de la lista, si corresponde a la sucursal suc, incrementa su precio
    for venta in ventas:
        if venta[0] == suc:
            venta[4] += venta[4] * incremento / 100
    return ventas


def ventas_articulo(cod_art, ventas):
    #Retorna cantidad de ventas de un artículo
    cantidad = 0
    for venta in ventas:
        if cod_art == venta[1]:
            cantidad += venta[3]
    return cantidad


def promedio_ventas(ventas):
    #Retorna promedio de ventas
    suma = 0
    for venta in ventas:
        suma += venta[3]
    return suma / len(ventas)


def codigos_encima_promedio(ventas):
    #Retorna un conjunto con códigos de artículos con cantidad vendida mayor al promedio
    promedio = promedio_ventas(ventas)
    codigos = set()
    for venta in ventas:
        if venta[3] >= promedio:
            codigos.add(venta[1])
    return codigos


#programa.py
```

```
#Inciso A
ventas = list()
print("Carga de ventas")
ventas=cargar_ventas(ventas)
#Inciso B
sucursal=int(input("Ingrese la sucursal donde incrementar el costo unitario de los
artículos: "))
print("Montos anteriores")
for venta in filtrarSucursal(ventas, sucursal):
    print("Artículo:",venta[1], "- Monto: $",venta[3] * venta[4])
incremento = float(input("Ingrese el porcentaje de incremento (%): "))
ventas=incrementar_costo_suc(sucursal, incremento, ventas)
print("Montos actualizados")
for venta in filtrarSucursal(ventas, sucursal):
    print("Artículo:",venta[1], "- Monto: $",venta[3] * venta[4])
#Inciso C
cod_art = int(input("Ingrese el código de artículo para informar ventas totales: "))
print("La cantidad vendida del artículo",cod_art,"es:",ventas_articulo(cod_art,ventas))
#Inciso D
print("Códigos de artículos cuya cantidad vendida estuvo por encima de la media:")
for codigo in codigos_encima_promedio(ventas):
    print(codigo)
```

10) Se lee una secuencia de números que termina cuando uno de ellos se encuentra en el intervalo [-30,-20] (comprendido entre los números -30 y -20).

Se solicita informar:

a) Cantidad de números impares y pares procesados.
b) Para cada número la cantidad de dígitos del mismo.
c) Cantidad de cincos y ceros procesados.
d) Si al terminar el proceso se encontraron todos los dígitos (0..9).

Nota: se debe modularizar de forma adecuada.

RESOLUCIÓN:

```
#funciones.py

def cant_digitos(num):
      #Retorna la cantidad de dígitos del número pasado por parámetro, sin importar si es
positivo o negativo
      num = abs(num)
      cantidad = 0
      while num != 0:
            cantidad += 1
```

```
            num = num // 10
      return cantidad


def incrementar_numero(num, num_coincidente):
      #Retorna 1 ó 0, según si el número pasado como primer parámetro coincide con el segundo parámetro o no.
      if num == num_coincidente:
            return 1
      else:
            return 0


#programa.py

num_imp = 0
num_par = 0
cant_cinco = 0
cant_cero = 0
digitos = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
num = int(input("Ingrese número (finalizar con número en rango -30 a -20): "))
while num not in range(-30,-21):
      #Inciso A
      if num % 2 == 0:
            num_par += 1
      else:
            num_imp += 1
      #Inciso B
      print("La cantidad de dígitos del número", num, "es:", cant_digitos(num))
      #Inciso C
      cant_cero += incrementar_numero(num, 0)
      cant_cinco += incrementar_numero(num, 5)
      #Inciso D
      if num in digitos:
            digitos.remove(num)
      num = int(input("Ingrese número (finalizar con número en rango -30 a -20): "))

#Inciso A
print("La cantidad de números pares es ", num_par)
print("La cantidad de números impares es ", num_imp)
#Inciso C
print("La cantidad de números cero es ", cant_cero)
print("La cantidad de números cinco es ", cant_cinco)
#Inciso D
if len(digitos) == 0:
      print("Todos los dígitos del 0 al 9 se encuentran en la secuencia procesada")
```

```
else:
      print("No todos los dígitos del 0 al 9 se encuentran en la secuencia procesada")
```

11) Las docentes de IPI desean hacer un relevamiento de todos los alumnos que han cursado la materia desde el año 2000 hasta la actualidad. Para tal fin desean:

a) almacenar en un diccionario la siguiente información: DNI, Nombre, Apellido, Fecha en que la cursó (formato ddmmaaaa), aprobado (s/n). La carga de la información finaliza al ingresar un DNI igual a cero (0).
b) Informar el nombre y apellido de aquellos alumnos que aprobaron en el año 2015. Delegar este punto en una función.
c) A partir de la estructura obtenida en el punto **a**, armar una lista que contenga los alumnos que cursaron la materia en el año 2010 (sin importar si aprobaron o no). La lista debe contener lista con DNI, nombre y apellido. Realizar el inciso usando una función que reciba la estructura original y retorne la lista.

RESOLUCIÓN:

```
#funciones.py

def cargarAlumnos(alumnos):
      #Carga datos de los alumnos en un diccionario que tiene el dni como clave y el resto de
los datos en una lista como valor: nombre, apellido, fecha cursada, estado de aprobación.
      dni=int(input("Ingrese un número de DNI "))
      while dni!=0:
            nombre=input("Nombre: ")
            apellido=input("Apellido: ")
            fCursada=int(input("Fecha de ingreso (formato DDMMAAAA): "))
            aprobo=input("¿Aprobó? (s/n):")
            aprobado=(aprobo.lower()=="s")
            alumnos[dni]=[nombre, apellido, fCursada, aprobado]
            dni=int(input("Ingrese un número de DNI "))
      return (alumnos)


def aprobadosAnio(alumnos, anio):
      #Imprime los datos de los alumnos que hayan cursado en el año pasado por parámetro y que
hayan aprobado
      for datos in alumnos.values():
            if (datos[2]%10000==anio and datos[3]):
                  print ("El alumno", datos[0], datos[1], "aprobó en el", anio)


def filtrarAlumnos(alumnos, anio):
      #Retorna una lista con listas conteniendo dni, nombre y apellido de los alumnos que
cursaron en el año pasado por parámetro
      lista=[]
      for dni, datos in alumnos.items():
```

```
            if datos[2]%10000==anio:
                  filtrado=[dni,datos[0],datos[1]]
                  lista.append(filtrado)
      return lista


#programa.py

alumnos={}
alumnos=cargarAlumnos(alumnos)
aprobadosAnio(alumnos, 2015)
for alumno in armarLista(alumnos, 2010):
      print("DNI:", alumno[0], "- Nombre:", alumno[1], alumno[2])
```

12) Escribir un programa que solicite al usuario el ingreso de 10 números enteros. Cada vez que se lea un número el programa debe verificar si el número es múltiplo de 4 e insertarlo en un contenedor; caso contrario, se insertará en otro contenedor.
En el primer contenedor no importa que los números estén repetidos pero debe haber un orden, y en el segundo contenedor no deben guardarse repetidos.
Por último se debe informar el producto de dos elementos sucesivos, guardados en el contenedor de los múltiplos de 4. Por ejemplo, si el contenedor tuviera los elementos: 56, 24, 4, 16, 8; debería informar: 1344, 96, 64, 128 (dado que 56 * 24 es 1344, 24 * 4 es 96 y así sucesivamente).

RESOLUCIÓN:

```
multiplosDeCuatro=list()
resto=set()
for i in range (10):
      numero=int(input("Número entero:"))
      if numero%4==0:
            multiplosDeCuatro.append(numero)
      else:
            resto.add(numero)
print ("El producto de dos elementos sucesivos, guardados en el contenedor de los
múltiplos de 4 es:")
for i in range(len(multiplosDeCuatro)-1):
      print(multiplosDeCuatro[i]*multiplosDeCuatro[i+1])
```