# Guía de Ejercicios Prácticos - Ficha 15

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos - Ficha 15
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:25

# Descripción

Esta guía contiene enunciados de algunos ejercicios para aplicar los conceptos de programación en ***Python*** que se analizan en la ***Ficha 15***. Los alumnos no deben subir nada al aula virtual: la guía se propone como fuente de ejercicios generales. Se sugiere intentar resolver cada uno de estos problemas ya sea trabajando solos o en grupos de estudio, y cuando las soluciones se publiquen, controlar lo hecho con las sugerencias propuestas por sus docentes. Utilice el foro del curso para plantear dudas y consultas, que cualquier alumno puede intentar responder.

# Tabla de contenidos

# 1. Tren de la Costa

Desarrollar un programa que represente el recorrido de ida y vuelta del Tren de la Costa.

El recorrido del tren abarca las siguientes estaciones: Maipú, Borges, Libertador, Anchorena, Barrancas, San Isidro R, Punta Chica, Marina Nueva, San Fernando R, Canal, Delta.

Se pide cargar un vector que contenga el nombre de las estaciones y otros dos que representen el viaje de ida y de vuelta del tren, respectivamente. En los dos últimos, se cargará la cantidad de pasajeros que ascendieron al tren en la estación correspondiente.

Con la información cargada, plantear el siguiente menú de opciones:

1. Mostrar los datos cargados
2. Cuántos pasajeros en total subieron en el viaje de ida, y cuántos en el viaje de vuelta
3. En qué estación subió la mayor cantidad de pasajeros, durante el viaje de ida
4. En cuántas estaciones del viaje de vuelta no subieron pasajeros, y qué porcentaje representan sobre el total de estaciones
5. Mostrar las estaciones la cantidad de pasajeros del viaje de ida fue mayor a la del viaje de vuelta

1.1. tren

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

import random
from vectores import *


def cargar_estaciones():
    e = ['Maipú', 'Borges', 'Libertador', 'Anchorena', 'Barrancas', 'San Isidro R', 'Punta Chica', 'Marina Nueva',
         'San Fernando R', 'Canal', 'Delta']
    return e


def cargar_pasajeros(estaciones):
    n = len(estaciones)
    ida = [0] * n
    vuelta = [0] * n
    for i in range(n):
        ida[i] = random.randint(0, 5)
        vuelta[i] = random.randint(0, 5)
    return ida, vuelta


def mostrar_datos(estaciones, ida, vuelta):
    print('Estacion', 'Ida', 'Vuelta', sep=' | ')
    for i in range(len(estaciones)):
        print(estaciones[i], ida[i], vuelta[i], sep=' | ')


def comparar_estaciones(ida, vuelta, estaciones):
    rta = list()
    for i in range(len(ida)):
        if ida[i] > vuelta[i]:
            rta.append(estaciones[i])
    return rta


def principal():
    estaciones = cargar_estaciones()
    ida, vuelta = cargar_pasajeros(estaciones)
    opcion = -1
    while (opcion != 0):
        print('=' * 50)
        print('TREN DE LA COSTA')
        print('1. Mostrar los datos')
        print('2. Total de pasajeros')
        print('3. Estacion con mayor cantidad de pasajeros (ida)')
        print('4. Estaciones sin pasajeros')
        print('5. Estaciones con mas pasajeros a la ida')
        print('0. Salir')
        opcion = int(input('Ingrese su opción: '))
        if opcion == 1:
            mostrar_datos(estaciones, ida, vuelta)
        elif opcion == 2:
            tot_ida = sumar_vector(ida)
            tot_vuelta = sumar_vector(vuelta)
            print('Total de pasajeros:', tot_ida, '(ida) y', tot_vuelta, '(vuelta)')
        elif opcion == 3:
            pos = buscar_mayor(ida)
            print('Estacion con mayor cantidad de pasajeros a la ida:', estaciones[pos], '(', ida[pos], 'pasajeros)')
        elif opcion == 4:
            cant = contar_valores(vuelta, 0)
            porc = cant * 100 / len(estaciones)
            print('Estaciones sin pasajeros:', cant, '. Representa un', round(porc, 2), '% del total')
        elif opcion == 5:
            rta = comparar_estaciones(ida, vuelta, estaciones)
            print(rta)
        elif opcion == 0:
```

```
            print('Hasta pronto!')
        else:
            print('Opción invalida')
        print('=' * 50)


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

## 1.2. vectores

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


def sumar_vector(v):
    suma = 0
    for i in range(len(v)):
        suma += v[i]
    return suma


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


def contar_valores(v, x):
    cant = 0
    for i in range(len(v)):
        if v[i] == x:
            cant += 1
    return cant
```

# 2. Empresa de transportes

Una empresa dedicada al transporte de mercadería solicita un programa para manejar estadísticas de sus envios.
Para ello solicita un programa, controlado por menú con opciones para lo siguiente:

- Ingresar los datos de los transportes realizados en el mes, de cada uno se conoce:
  - dia del mes (1-31)
  - Descripción
  - Monto del transporte

- Determinar el monto promedio obtenido en el mes
- Genere un listado de los envíos realizados, ordenado por importe en forma decreciente
- Determine que día del mes tuvo mayor cantidad de transportes realizados

## 2.1. Solución

```
def menu():
    print("Menu de opciones: ")
    print("1_ Cargar datos")
    print("2_ Determinar monto promedio")
    print("3_ Generar listado")
    print("4_ Dia mayor cantidad")
    print("5_ Salir")
    return int(input("Ingrese opción: "))


def cargar_datos():
    cantidad = int(input("Ingrese la cantidad de transportes realizados: "))
    dias = [0] * cantidad
    descripciones = [''] * cantidad
    montos = [0.0] * cantidad

    for i in range(cantidad):
        dias[i] = int(input("Ingrese dia: "))
        descripciones[i] = input("Ingrese descripción: ")
        montos[i] = float(input("Ingrese monto: "))

    return dias, descripciones, montos


def calcular_promedio(montos):
    suma = 0
    for valor in montos:
        suma += valor
    return suma / len(montos)


def listar(dias, descripciones, montos):
    for i in range(len(dias)):
        print("Dia:", dias[i], "Monto:", montos[i],
              "Descripcion:", descripciones[i])


def ordenar(dias, descripciones, montos):
    n = len(montos)
    for i in range(n - 1):
        for j in range(i+1, n):
            if montos[i] < montos[j]:
                montos[i], montos[j] = montos[j], montos[i]
                dias[i], dias[j] = dias[j], dias[i]
                descripciones[i], descripciones[j] = descripciones[j], descripciones[i]


def calcular_transportes_dia(dias):
    cant = [0] * 31
    for x in dias:
        cant[x-1] += 1
    return cant


def dia_mas_transportes(dias):
    x_dia = calcular_transportes_dia(dias)
    mayor = mayor_dia = 0
    for i in range(len(x_dia)):
        if x_dia[i] > mayor:
            mayor = x_dia[i]
            mayor_dia = i
    return mayor_dia + 1, mayor


def main():
    op = 0
    while op != 5:
```

```
        op = menu()
        if op == 1:
            dias, descripciones, montos = cargar_datos()
        elif op == 2:
            promedio = calcular_promedio(montos)
            print("El monto promedio es:", promedio)
        elif op == 3:
            ordenar(dias, descripciones, montos)
            listar(dias, descripciones, montos)
        elif op == 4:
            mayor, cantidad = dia_mas_transportes(dias)
            print("El dia de mayor cantidad de transportes fue:", mayor)
            print("Con un total de", cantidad, "transportes realizados")


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

# 3. Multas de tránsito

Se necesita desarrollar un script que permita procesar las n multas de tránsito labradas el último fin de semana en las rutas de la provincia. Para ello se pide cargar un arreglo con los n códigos de infracción, valores comprendidos en [1:20], junto con los importes en pesos de los cincos tipos de infracciones, tal como se indica en la siguiente tabla:

Código Tipo de infracción
[1,6, 11,16] 1
[2,7, 12,17] 2
[3,8, 13,18] 3
[4,9, 14,19] 4
[5,10,15,20] 0

Con los datos cargados en los arreglos se pide:
-Generar un tercer vector con los Importes totales facturados por tipo de infracción
-Determinar el código de infracción que más apareció en las multas y la cantidad de multas labradas para dicho código.
-Informar el importe total facturado durante el fin de semana.

## 3.1. Solución

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

def validate(inf, msg='Ingrese un valor.')
    n=inf
    while n<=inf:
        n = int(input(msg + ' Valor superior a ' + str(inf) + ', por favor: '))
        if n<=inf:
            print('Error. Valor Incorrecto!')
    return n

def validate_range(inf,sup, msg='Ingrese un valor.'):
    n=inf-1
    while n<inf or n>sup:
        n = int(input(msg + ' Comprendido en [' + str(inf) + ':' + str(sup) + '], por favor: '))
        if n<inf or (n>sup):
            print('Error. Valor Incorrecto!')
    return n

def read(v,inf, sup, msg):
    for i in range(len(v)):
        v[i]= validate_range(inf,sup,msg)

def pto_1(codigos,importes):
    aux = [0]*5
    for i in range(len(codigos)):
        indi = codigos[i]%5
        aux[indi]+=importes[indi]
    return aux

def pto_2(codigos):
    aux = [0]*20
    for i in range(len(codigos)):
        aux[codigos[i]-1]+=1
    may = aux[0]
    may_cod = 1
    for i in range(1,20):
        if aux[i]>may:
            may, may_cod = aux[i], i+1
    return may,may_cod

def test():
    n = validate(0, 'Ingrese cantidad de multas.')
    infracciones=[None] * n
    importes=[None]*5

    #Cargar los n códigos de infracción labrados
    read(infracciones, 1, 20, 'Ingrese un código de infracción.')

    #Cargar los 5 importes correspondientes a los tipos de infracciones
    read(importes, 1, 50000, 'Ingrese un importe en pesos.')

    #Generar un tercer vector con los Importes totales facturados por tipo de infracción
    totales = pto_1(infracciones, importes)
    print('Totales facturados por tipo:')
    for i in range(5):
        if totales[i]!=0:
            print('Tipo ' + str(i) + ': ' + str(totales[i]), ' pesos.')

    #Determinar el código de infracción que más apareció en las multas y la cantidad de multas labradas para dicho código.
    cod, cant = pto_2(infracciones)
    print('Código de infracción más frecuente: ' + str(cod) + ', con ' + str(cant), ' multas.')
    #Informar el importe total facturado durante el fin de semana.
    acu = 0
    for i in range(5):
        acu+=totales[i]
    print('Importe total facturado durante el fin de semana: ' + str(acu) +  ' pesos.')
```

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

# 4. Partido de tenis

**Introducción**

Piense en los resultados de los partidos de tenis, donde se disputan 3 o 5 Sets.

| Set | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Roger Federer | 6 | 3 | 4 | 6 | 7 |
| Rafael Nadal | 4 | 6 | 6 | 2 | 6 |

Cada *set* se juega hasta que uno de los jugadores gana 6 *games*, teniendo una diferencia mínima de dos *games* sobre el otro jugador. En la figura, se puede ver que Roger Federer ganó 6 *games* contra 4 que ganó Rafael Nadal, permitiéndole ganar el primer *set*.

Como puede ocurrir que, en un set, se dé un resultado parcial 6-6, y para ganar un *set* hace falta diferencia de 2 *games*, existe un desempate denominado *tie-break*, lo que permite un resultado de 7-6, como en el último set del ejemplo. También es válido, el resultado 7-5, que le permite a un jugador obtener ventaja de 2 *sets* que necesita, a pesar de haber ganado ya 6 *games*

El partido completo lo gana el jugador que ganó la mayor cantidad de *sets*, en un partido a 3 *sets*, el jugador que gane 2 vencerá, en un partido a 5 *sets*, como el del ejemplo, el que logre 3 *sets* (Roger Federer, en nuestro caso). Pero claro, si un jugador hubiese ganado la mayoría antes del total de los sets (si, por ejemplo Federer hubiese ganado los 3 primeros), ya no se jugarían los restantes *sets*, porque un jugador ya habría ganado.

**Se Pide**

Teniendo en cuenta solamente lo expuesto, se solicita un programa que permita:

1. Que el usuario ingrese por teclado la cantidad de sets (3 o 5) e ingrese el resultado de un partido finalizado (una tabla como la del ejemplo)
2. Validar que los resultados cargados sean válidos de acuerdo a lo expuesto previamente, por ejemplo:
    1. Que no haya sets con cantidad de games incorrectos (8 a 6, 1 a 1, etc.)
    2. Que todos los games hayan sido ganados por diferencia de 2 o más, exceptuando las situaciones *tie-break*
    3. Que no se hayan jugado sets de más (ej: que un jugador haya ganado los 3 primeros *sets* de un partido de 5 y que se hayan jugado los sets restantes)
    4. Otras validaciones que se desprendan de la descripción
3. Determinar quién ganó (y en qué set lo hizo)
4. Informar la cantidad de *sets* ganados por *tie-break*
5. Mostrar un *array* que contenga la diferencia de resultado para cada *set* (en el ejemplo, el primer set terminó con una diferencia de 2 *games*)
6. Mostrar los *sets* ordenados de acuerdo a la diferencia entre jugadores (de mayor a menor)

## 4.1. Solución

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

def validate_values(mensaje, valores_permitidos, mensaje_error):
    val = int(input(mensaje))
    while val not in valores_permitidos:
        print(mensaje_error)
        val = int(input(mensaje))
    return val


def validate_range(mensaje, inf, sup):
    n = inf - 1
    while n < inf or n > sup:
        n = int(input(mensaje + " (entre "+ str(inf) + " y " + str(sup) + "): "))
        if n < inf or n > sup:
            print("Valor incorrecto!")
    return n


def cargar_datos(nombre1, nombre2):
    # se solicita la cantidad de sets a la que se juega el partido, 3 o 5
    cantidad_sets = validate_values('Cantidad de sets:',[3,5],'Valor Incorrecto, reintente')
    # Hay una cantidad mínima de sets que debe ser jugado para que uno pueda ganar
    cantidad_minima_sets = cantidad_sets // 2 + 1
    # Se solicita la cantidad de sets que se desean cargar como resultado
    cantidad_sets_jugados = validate_range('Cantidad de sets jugados',cantidad_minima_sets,cantidad_sets)
    # Se incializan los vectores que almacenarán los puntajes de cada jugador
    puntos1 = [0] * cantidad_sets_jugados
    puntos2 = [0] * cantidad_sets_jugados
    # Para cada set que se va a cargar, se solicitan los games de cada jugador
    for i in range(cantidad_sets_jugados):
        puntos1[i] = validate_range(nombre1 + ', Set ' + str(i + 1), 0, 7)
        puntos2[i] = validate_range(nombre2 + ', Set ' + str(i + 1), 0, 7)
    return cantidad_sets, puntos1, puntos2


def mostrar(nombre1, nombre2, puntos1, puntos2):
    print('Resultados cargados:')
    print(nombre1, ':', puntos1)
    print(nombre2, ':', puntos2)


def determinar_ganador(cantidad_sets, puntos1, puntos2):
    # Se acumulan los sets ganados por cada jugador
    sets_ganados1 = 0
    sets_ganados2 = 0
    # El ganador debe haber ganado más de la mitad de los sets
    cantidad_minima_sets = cantidad_sets // 2 + 1
    # Para cada set cargado como resultado
    for i in range(len(puntos1)):
        if puntos1[i] > puntos2[i]:
            sets_ganados1 += 1
        else:
            sets_ganados2 += 1

        # Si uno de los jugadores alcanzó la cantidad mínima de games
        # para ganar, ganó
        if sets_ganados1 == cantidad_minima_sets:
            return True, 1, (i + 1)

        if sets_ganados2 == cantidad_minima_sets:
            return True, 2, (i + 1)

    return False, 0, 0


def validar_cantidad_games(set, punto1, punto2):
```

```
    # Para ganar un set, se deben ganar, al menos, 6 games
    if punto1 < 6 and punto2 < 6:
        print('Se deben ganar al menos 6 games en un set', '(Set=' + str(set) + ')')
        return False

    return True


def validar_diferencias(set, punto1, punto2):
    # Debe haber diferencia de 2 games, salvo tiebreak (7-6)
    if abs(punto1 - punto2) < 2:
        # Si terminó 7-6 es válido, en otro caso, no
        if punto1 + punto2 != 13:
            # No se alcanzó diferencia de al menos 2 sets y no se trata de
            # un tie-break
            print('No hay diferencia de 2 games', '(Set=' + str(set) + ')')
            return False
    return True


def validar_game_extra(set, punto1, punto2):
    # Si se llegó a 7 games es un tiebreak o el resultado es 7-5
    if punto1 == 7 or punto2 == 7:
        suma_puntos = punto1 + punto2
        # Si el resultado no es 7-5 o 7-6, no es válido
        if suma_puntos != 12 and suma_puntos != 13:
            print('Si uno de los jugadores llegó a 7 games, o es 7-6 (tie-break) o 7-5', '(Set=' + str(set) + ')')
            return False

    return True


def validar_cantidad_sets(cantidad_sets, puntos1, puntos2):

    # En los resultados, debió ganar un jugador, y luego de ganar, no se debieron jugar más sets
    hubo_ganador, jugador, set = determinar_ganador(cantidad_sets, puntos1, puntos2)

    # Si los datos son incorrectos y no hubo ganador...
    if not hubo_ganador:
        print('Ningún jugador alcanzó la cantidad mínima de sets para ganar');
        return False

    if set < len(puntos1):
        # Significa que se registraron más sets que los necesarios
        print('Se jugaron más sets que los necesarios para un ganador')
        return False


def validar(cantidad_sets, puntos1, puntos2):
    # Se recorren los resultados cargados y se valida que los puntos estén bien
    for i in range(len(puntos1)):

        if not validar_cantidad_games(i+1, puntos1[i], puntos2[i]):
            return False

        if not validar_diferencias(i+1, puntos1[i], puntos2[i]):
            return False

        if not validar_game_extra(i+1, puntos1[i], puntos2[i]):
            return False

    if not validar_cantidad_sets(cantidad_sets, puntos1, puntos2):
        return False

    return True
```

```
def contar_tiebreak(puntos1, puntos2):
    # se acumula la cantidad de tie-breaks (resultados 7-6)
    cant_tiebreaks = 0
    # se recorren los resultados
    for i in range(len(puntos1)):
        # Si alguno de los dos terminó con 7 games
        if puntos1[i] == 7 or puntos2[i] == 7:
            # Si suman 13, es porque salieron 7-6
            if (puntos1[i] + puntos2[i]) == 13:
                cant_tiebreaks += 1

    # se devuelve la cantidad de tie-breaks
    return cant_tiebreaks


def calcular_diferencias(puntos1, puntos2):
    # se inicializa el vector con las diferencias
    diferencias = [0] * len(puntos1)
    # se recorren los resultados
    for i in range(len(puntos1)):
        # simplemente se asigna la diferencia en el elemento correcto del array
        # se toma el valor absoluto para no asumir que un jugador ganó más games que otro
        diferencias[i] = abs(puntos1[i] - puntos2[i])

    # se devuelven los valores
    return diferencias


def ordenar(diferencias):
    # cantidad de elementos del vector de diferencias
    n = len(diferencias)
    # vector con los números de sets
    sets = list(range(1, n+1))
    # Ordenamiento
    for i in range(n - 1):
        for j in range(i + 1, n):
            if diferencias[i] < diferencias[j]:
                diferencias[i], diferencias[j] = diferencias[j], diferencias[i]
                sets[i], sets[j] = sets[j], sets[i]
    return sets, diferencias


def menu():
    print('')
    print('1) Cargar datos')
    print('2) Validar datos')
    print('3) Determinar ganador')
    print('4) Sets ganados por tie-break')
    print('5) Mostrar diferencias')
    print('6) Mostrar Sets con mayor diferencia')
    print('')
    print('7) - Salir')
    print('')
    return validate_range('Ingrese opción: ', 1, 7)
    print('')


def main():
    nombre1 = ''
    nombre2 = ''
    puntos1 = []
    puntos2 = []
    sets = 0
    validos = False
    op = menu()
    while op != 7:
```

```
        if op == 1:
            # Se solicitan los nombres de los jugadores
            nombre1 = input('Ingrese el nombre del jugador 1: ')
            nombre2 = input('Ingrese el nombre del jugador 2: ')
            # Se cargan los resultados del partido
            sets, puntos1, puntos2 = cargar_datos(nombre1, nombre2)
            # Se muestran los resultados
            mostrar(nombre1, nombre2, puntos1, puntos2)
            # Se marcan como inválidos los datos, hasta que se validen
            validos = False
        elif op == 2:
            validos = validar(sets, puntos1, puntos2)
            if not validos:
                print('Validación fallida!!!')
            else:
                print('Datos válidos')
        elif op == 3:
            if not validos:
                print('Debe cargar datos válidos')
            else:
                mostrar(nombre1, nombre2, puntos1, puntos2)
                hubo_ganador, jugador, set = determinar_ganador(sets, puntos1, puntos2)
                if jugador == 1:
                    print('Ganó el jugador ' + nombre1 + ', en el Set ', str(set))
                else:
                    print('Ganó el jugador ' + nombre2 + ', en el Set ', str(set))
        elif op == 4:
            if not validos:
                print('Debe cargar datos válidos')
            else:
                mostrar(nombre1, nombre2, puntos1, puntos2)
                print('La cantidad de sets ganados por tie-break es: ' + str(contar_tiebreak(puntos1,puntos2)))
        elif op == 5:
            if not validos:
                print('Debe cargar datos válidos')
            else:
                mostrar(nombre1, nombre2, puntos1, puntos2)
                print('Las diferencias fueron: ',calcular_diferencias(puntos1, puntos2))
        elif op == 6:
            if not validos:
                print('Debe cargar datos válidos')
            else:
                mostrar(nombre1, nombre2, puntos1, puntos2)
                sets_ordenados, diferencias = ordenar(calcular_diferencias(puntos1, puntos2))
                print('Sets (ordenados por diferencia): ', sets_ordenados)
                print('Diferencias                    : ', diferencias)
        op = menu()


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

# 5. Encuesta de fábrica de zapatillas

Una fábrica de zapatillas realiza una encuesta para determinar cuál es el calzado que tiene mayor demanda en los jóvenes entre 10 y 25 años. La encuesta determinaba:

- Número que calza
- Color de preferencia (blanco, negro, azul)
- Edad
- Qué prefiere: cuero o tela

Hacer un programa que permita cargar la información en 4 vectores de las 100 encuestas realizadas y determinar las siguientes estadísticas:

- Cuál es el número de zapatillas promedio.
- Los jóvenes entre 10 y 18 años que color prefieren mas.
- Cuál es la preferencia de los jóvenes entre 19 y 25 años, con respecto al material de las zapatillas (cuero o tela).
- Si el numero de calzado va desde 35 a 40, generar un vector que contenga cual es la demanda de cada número de calzado (un vector contador de 6 elementos) y determinar cuál es el número de mayor y menor demanda.

## 5.1. Solución

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

import random


# funcion que carga en forma aleatoria los valores de los 4 vectores a analizar
def carga_datos_aleatorios(cant):
    tupla_col = "Blanco", "Negro", "Azul"
    tupla_tipo = "Cuero", "Tela"
    n = list()
    c = list()
    e = list()
    t = list()
    for i in range(0,cant):
        n.append(random.randint(35,40))
        c.append(random.choice(tupla_col))
        e.append(random.randint(10,25))
        t.append(random.choice(tupla_tipo))
    return n, c, e, t


# Funcion que muestra las opciones  del menu
def menu():
    print("\n ESTADISTICAS ")
    print("1) Número de zapatillas promedio")
    print("2) Color de Preferencia entre 10 y 18 años")
    print("3) Material de  preferencia de los jóvenes entre 19 y 25 años")
    print("4) Demanda de cada número de calzado")
    print("5) El número de mayor y menor demanda")
    print("0) Salir del menu")


# Funcion que permite calcular el promedio de nro. de zapatillas
def calcular_Promedio(v_nro):
    p = s = 0
    cant = len(v_nro)
    for i in range(cant):
        s += v_nro[i]
    if cant != 0:
        p = s / cant
    return p


# Funcion que determina cual color prefieren mas
def mayor(a, n, b):
    if a > n and a > b:
        c_may = a
        color_m = "Azul"
    elif n > a and n > b:
        c_may = n
        color_m= "Negro"
    else:
        c_may = b
        color_m = "Blanco"
    return c_may, color_m


#Funcion que cuenta cuantos prefieren cada color de zapatillas
def determinar_pref(v_edad, v_color):
    cant = len(v_color)
    cont_b, cont_n, cont_a = 0, 0, 0
    for i in range(cant):
        if v_edad[i] >= 10 and v_edad[i] < 19:
            if v_color[i] == "Blanco":
                cont_b += 1
            elif v_color[i] == "Negro":
                cont_n += 1
```

```
            else:
                cont_a += 1
    may = mayor(cont_a,cont_n,cont_b)
    return may


# funcion que determina qué tipo de material prefieren los jovenes
def determinar_pref_tipo(v_edad, v_tipo):
    cant = len(v_tipo)
    c_tela = c_cuero = 0
    for i in range(cant):
        if v_edad[i] > 19:
            if v_tipo[i] == "Tela":
                c_tela += 1
            else:
                c_cuero += 1
    if c_cuero > c_tela:
        pref = "Cuero"
    else:
        pref = "Tela"
    return pref


# Funcion que genera un vector que contiene
# la demanda de cada número de zapatillas
def determinar_demanda(v_nro):
    v = [0] * 6
    for i in range (len(v_nro)):
        v[v_nro[i] - 35] += 1
    return v


#Funcion que determina el nro. de mayor y menor demanda de zapatillas
def buscar_May_men(v_demanda):
    may = 0
    men = 1000
    for i in range(len(v_demanda)):
        if v_demanda[i]> may:
            may = v_demanda[i]

        if v_demanda[i] < men:
            men = v_demanda[i]

    return may, men


# Función principal
def main():
    v_nro = []
    v_color = []
    v_edad = []
    v_tipo = []
    v_demanda = None
    print("ESTADISTICAS\n")
    cant = int(input("\nIngrese cantidad de encuestas a cargar :"))
    v_nro,v_color, v_edad, v_tipo = carga_datos_aleatorios(cant)
    print("Datos cargados :\n Nro. de Calzados: ",v_nro)
    print("\n Colores elegidos",v_color, "\n Edades: ",
          v_edad, "\n tipo de material:", v_tipo)
    opcion = 1
    while opcion != 0:
        menu()
        opcion = int(input("Ingrese opcion :"))
        if opcion == 1:
            print("EL VALOR PROMEDIO DE NRO DE CALZADOS ES: ",
                  calcular_Promedio(v_nro))
```

```
        elif opcion == 2:
            color_pref = determinar_pref(v_edad, v_color)
            print("el color de mayor demanda es :", color_pref[1],
                  ", con un total de :", color_pref[0])

        elif opcion == 3:
            material_pref = determinar_pref_tipo(v_edad, v_tipo)
            print("El material que prefieren los jóvenes mayores "
                  "de 19 años es el : ", material_pref)

        elif opcion == 4:
            v_demanda = determinar_demanda(v_nro)
            n= 35
            print ("\nLa demanda de cada numero de zapatillas es :")
            for i in range(len(v_demanda)):
                print("nro. ",(n+i), ": ", v_demanda[i])

        elif opcion == 5:
            # controlo que el vector v_demanda este cargado, sino lo cargo
            if v_demanda == None:
                v_demanda = determinar_demanda(v_nro)
            mayor, menor = buscar_May_men(v_demanda)
            # para controlar si hay mas de un nro. con mayor o menor demanda,
            # recorro el vector  de demandas y muestro los de igual demanda
            for i in range(len(v_demanda)):
                if v_demanda[i]== mayor:
                    print("El nro. de calzado de mayor demanda es : ", i + 35,
                          " con ", v_demanda[i], "")
            for i in range(len(v_demanda)):
                if v_demanda[i] == menor:
                    print("\n y el de menor demanda es: ", i + 35,
                          " con ", v_demanda[i])

        elif opcion ==0:
            print("FIN DEL PROGRAMA")

        else:
            print("ERROR, DEBE INGRESAR UN VALOR ENTRE 0 Y 5 ")

main()
```

# 6. Búsquedas en Vectores

Se solicita un programa que solicite un número entero positivo n por teclado y luego gener un vector de n números aleatorios entre n y (n*n). Luego con el vector será necesario realizar:

1) Mostrar el vector

2) Generar y mostrar un segundo vector de m elementos aleatorios a buscar, m se carga por teclado, los elementos aleatorios también deben ser generados entre n y (n*n).

3) Determinar cuántos elementos del segundo vector están en el vector original, es decir para cada elemento del segundo vector, buscarlo en el vector original, si está contarlo y sino solo continuar con el siguiente.

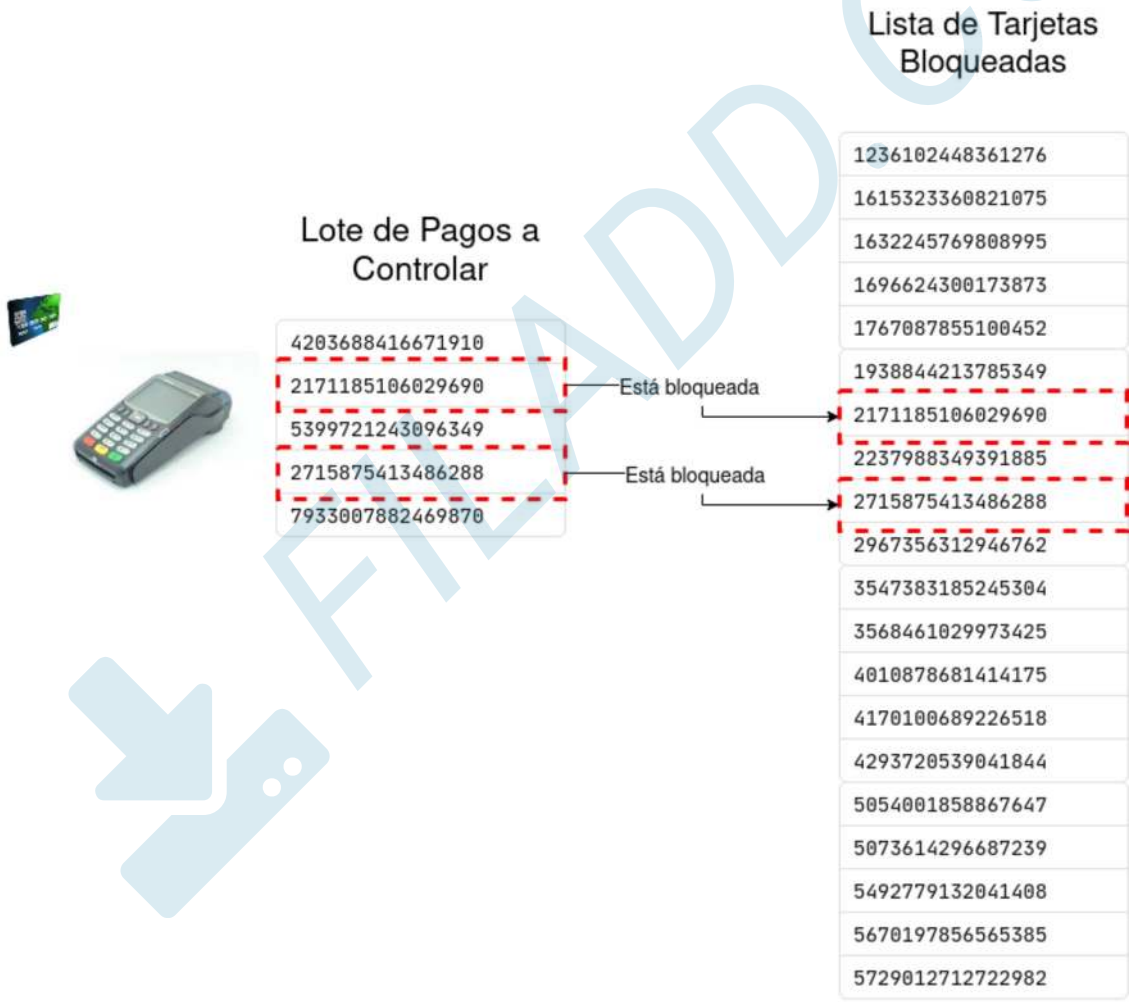
4) Realizar el mismo conteo del punto anterior pero con búsqueda binaria (para esto debe ordenar el vector primero, pero debe realizarlo sin alterar el vector original).

# 7. Procesamiento de Tarjetas

La empresa Esfera S.A. (nuestro cliente) se dedica al procesamiento de transacciones de tarjetas de crédito de distintas "marcas". Los pagos se realizan un equipo del comercio y éstas son informadas (enviadas), en el mismo momento, al sistema de autorizaciones de nuestro cliente donde se decide si el pago se puede hacer o no. Para nuestro ejemplo, esto significa que únicamente se envía el número de tarjeta.

Existen casos excepcionales (algunos vuelos, transporte marítimo, etc.) donde estos pagos no pueden ser enviados en el mismo momento en el que se generan (compré unas papas fritas en un avión y el equipo donde presenté la tarjeta no tiene conexión con el sistema de autorizaciones). Ante estos casos Esfera S.A. acepta que, en algún momento del día, se descargue el listado de los pagos realizados en un equipo particular (por ejemplo, el equipo del avión donde compré las papas fritas) y se carguen todas juntas en su sistema central. Esta lista contiene únicamente el número de tarjeta que realizó el pago.

Pero hay una complicación adicional: al no envierse estos pagos en el mismo momento en el que el usuario presentó la tarjeta, podría haberse realizado un pago con una tarjeta que estaba bloqueada (por ejemplo, se informó el robo de la misma, o se trataba de alguna falsificación sofisticada, etc.). Por este motivo, el sistema de autorizaciones, mantiene una lista **ordenada** a la que vamos a llamar "Lista de Tarjetas Inválidas" donde tiene el número de cada tarjeta que no es válida, en un momento dado, para realizar pagos. Esta lista se actualiza una única vez al día y se utiliza para controlar todas las transacciones realizadas. Si un pago se realizó con una tarjeta que está en esta lista, ese pago se rechaza.




Como detalle adicional, el primer dígito del número de tarjeta (que no puede ser cero) indica su "marca".
A saber: (1=Vasa, 2=American Slow, 3=MaterCard, 4=Tarjeta Pomelo, 5=Launcher's Club, 6=JNoCB, 7=AsiaPay, 8=Undiscovered Card, 9=AED Card).

Por ejemplo, la tarjeta cuyo número es: 9987015911049952, corresponde a una "AED Card"

Nuestro cliente nos pide desarrollar el sistema de autorizaciones, para los pagos que no pudieron ser informados en el momento de compra. Para eso se debe desarrollar un programa Python que permita:

1. Cargar por única vez, al inicio del programa, la Lista de Tarjetas Inválidas
2. Cargar lotes de transacciones para su procesamiento. Recordar que estos lotes de transacciones son, simplemente, listas de números de tarjetas que realizaron pagos en un equipo.
3. Informar el porcentaje de pagos que fueron rechazados porque la tarjeta estaba en la Lista de Tarjetas Inválidas
4. Informar cuántos pagos fueron rechazados, pero discriminados por "marca" de la tarjeta. (Cuántos pagos se rechazaron con Vasa, cuántos con American Slow, etc.)
5. Informar cuál es la marca de tarjeta con mayores pagos rechazados

## 7.1. utilidades.py

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


def calcular_porcentaje(muestra, total):
    """
    Calcula el porcentaje que representan las muestras
    sobre el total
    :param muestra: Muestra sobre el total
    :param total: Total de casos
    :return: Porcentaje que de muestra sobre total
    """
    porcentaje = 0  # Por defecto, vale cero
    # Si el total es válido...
    if total != 0:
        porcentaje = (muestra * 100) / total

    # Retorno del porcentaje
    return porcentaje
```

## 7.2. validaciones.py

```
def validar_entre(lim_inferior, lim_superior, mensaje='Ingrese un valor: '):
    """
    Permite la carga y validación de un valor entero que debe estar en el
    intervalo cerrado [lim_inferior, lim_superior]. El método no retorna
    hasta que no se ingrese un valor correcto
    :param lim_inferior: El límite inferior a validar
    :param lim_superior: El límite superior a validar
    :param mensaje: Mensaje que se le presenta al usuario
    :return: Un valor en el rango validado
    """
    # Se carga un valor
    n = int(input(mensaje))
    # Mientras no sea correcto...
    while n < lim_inferior or n > lim_superior:
        # Se muestra un mensaje de error
        print('El valor debe estar entre', lim_inferior, 'y', lim_superior)
        # Se solicita otro valor
        n = int(input(mensaje))

    # Retorno del valor
    return n


def validar_mayor_igual(limite, mensaje='Ingrese un valor: '):
    """
    Permite la carga y validación de un valor entero que debe ser
    Mayor o igual a limite. El método no retorna hasta que no se
    ingrese un valor correcto
    :param limite: El límite a validar
    :param mensaje: Mensaje que se le presenta al usuario
    :return: Un valor en el rango validado
    """
    # Se carga un valor
    n = int(input(mensaje))
    # Mientras no sea correcto...
    while n < limite:
        # Se muestra un mensaje de error
        print('El valor debe ser mayor o igual a', limite)
        # Se solicita otro valor
        n = int(input(mensaje))
    # Retorno del valor
    return n


def validar_tarjeta(mensaje='Ingrese un número de tarjeta: '):
    """
    Permite la carga y validación de un string que representa el
    número de una tarjeta de crédito. Debe tener 16 dígitos y no
    empezar por 0
    :param mensaje: Mensaje a presentar al usuario
    :return: Un número de tarjeta validado
    """
    nro_tarjeta = input(mensaje)
    # Mientras esté mal
    while len(nro_tarjeta) != 16 or nro_tarjeta[0] < '1' or nro_tarjeta[0] > '9':
        # Se muestra un mensaje de error
        print('Número de tarjeta inválido')
        # Se solicita un nuevo número de tarjeta
        nro_tarjeta = input(mensaje)
    # Tarjeta ya validada
    return nro_tarjeta
```

## 7.3. arreglos.py

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


def busqueda_binaria(vec, x):
    """
    Realiza una búsqueda binaria de x en vec

    :param vec: El vector sobre el que se realiza la búsqueda
    :param x: El valor a buscar
    :return: El índice donde se encuentra x, si se encuentra,
    -1 en caso contrario
    """
    # Inicialización de los índices
    izq, der = 0, len(vec) - 1
    # Mientras no se crucen los índices
    while izq <= der:
        c = (izq + der) // 2

        if vec[c] == x:
            # El elemento se encontró
            return c
        elif vec[c] < x:
            # El elemento podría estar a la derecha de c
            izq = c + 1
        else:
            # El elemento podría estar a la izquierda de c
            der = c - 1

    # No lo encontramos
    return -1


def existe_binaria(vec, x):
    """
    Determina si el valor x existe en vec con búsqueda binaria

    :param vec: El vector sobre el cuál se realiza la búsqueda
    :param x: El valor a buscar
    :return: True si x se encuentra en vec, False en caso contrario
    """
    return busqueda_binaria(vec, x) != -1


def buscar_mayor(vec):
    """
    Busca el elemento de mayor valor dentro de vec

    :param vec: El vector sobre el cuál se realiza la búsqueda
    :return: El índice donde se encuentra el mayor valor y el mayor valor
    """
    n = len(vec)            # Longitud del vector
    # Precaución por si el vector viene vacío
    if n == 0:
        return -1, None

    # Búsqueda del mayor
    indice_mayor = 0            # En principio el primer elemento
    mayor = vec[0]              # En principio el valor mayor es el primero
    # Se recorre el vector, evitando el primer elemento que ya se visitó
    for i in range(1, n):
        # Si es mayor que el mayor hasta ahora
        if vec[i] > mayor:
            # Se actualizan los valores
            indice_mayor = i
            mayor = vec[i]

    # Retorno de los valores
```

```
    return indice_mayor, mayor


def ordenar_seleccion_directa(vec):
    """
    Ordena, de menor a mayor, un vector con el algoritmo
    de selección directa.

    :param vec: El vector a ordenar
    :return: None
    """
    n = len(vec)
    # i no llega hasta el último elemento
    for i in range(n -1):
        # j inicia a la derecha de i
        for j in range(i+1, n):
            # Si hay que dar vuelta los valores...
            if vec[i] > vec[j]:
                vec[i], vec[j] = vec[j], vec[i]
```

## 7.4. principal.py

```
import random
import arreglos
import validaciones
import utilidades


def convertir_codigo_a_marca(codigo):
    """
    Convierte el código de marca a la
    descripción de la misma

    :param codigo: El código de marca
    :return: La descripción de la marca
    """
    # Todas las descripciones de las marcas soportadas
    descripciones = (
        "Vusa", "American Slow", "MaterCard",
        "Tarjeta Pomelo", "Launcher's club", "JNoCB",
        "AsiaPay", "Undiscovered Card", "AED Card"
    )
    # Por defecto, la marca sería desconocida
    marca = "Desconocida"
    # Si está en el rango válido...
    if codigo >= 1 and codigo <= len(descripciones):
        return descripciones[codigo - 1]
    # Se devuelve la marca que corresponde al código
    return marca


def generar_tarjetas_bloqueadas(n):
    """
    Genera un vector con strings representando números de
    tarjeta aleatorios válidos (16 dígitos, no comienzan en 0).
    Y lo devuelveordenado.

    ********************** ATENCION ***************************
    NO es buena idea ordenar un vector para hacer unas pocas
    búsquedas. El costo de ordenamiento con una algoritmo de
    selección directa (como el que se implementa aquí) es
    altísimo.

    Para ESTE ejercicio en particular, donde se simula que el
    vector se ordena una única vez y se busca muchísimas veces
    sobre él, y donde el enunciado dice explícitamente que la
    lista se tiene ya ordenada, se realiza el ordenamiento luego
    de la creación
    ***********************************************************

    :param n: La cantidad de tarjetas que se quieren en la lista
    :return: Un vector ORDENADO de menor a mayor con n tarjetas
    """
    lista_bloqueadas = [None] * n
    # Se generan los valores
    for i in range(n):
        # No es válido que inicie con cero
        lista_bloqueadas[i] = str(random.randint(1000000000000000, 9999999999999999))
    # Se ordena el vector (leer el comentario)
    arreglos.ordenar_seleccion_directa(lista_bloqueadas)
    # Se devuelve el vector
    return lista_bloqueadas


def cargar_lote_pagos(n):
    """
    Crea un vector con n pagos realizados, solicitándole
    al usuario los números de tarjeta de los pagos
```

```
    :param n: La cantidad de pagos a cargar
    :return: un vector con n pagos cargados
    """
    vec = [None] * n
    # Ya se creó el vector con valores None, ahora se le carga un
    # valor a cada elemento
    for i in range(n):
        vec[i] = validaciones.validar_tarjeta("Ingrese Nro. Tarjeta: ")

    # Retorno del vector
    return vec


def calcular_porcentaje_rechazos(lote_pagos, tarjetas_bloqueadas):
    """
    Calcula el porcentaje de pagos rechazados dentro de lote_pagos.
    Esta función es muy similar (en su código) a calcular_rechazos_por_marca.
    ¿Pudieron ser una sola función y evitar recorrer dos veces el vector?. Sí,
    pero el código hubiera sido más difícil de entender, ya que se hubieran
    resuelto dos problemas en la misma función, se hubieran tenido que retornar
    dos valores en lugar de uno, etc.

    :param lote_pagos: Lista de números de tarjeta que pagaron
    :param tarjetas_bloqueadas: Lista de tarjetas bloqueadas
    :return: Porcentaje de rechazos sobre lote_pagos
    """
    cont_rechazos = 0
    # Recorrer el lote de pagos
    for i in range(len(lote_pagos)):  # pudo ser for un_pago_nro_tarjeta in lote_pagos:
        # Uno de los pagos a procear
        un_pago_nro_tarjeta = lote_pagos[i]
        # Revisar si está bloqueada
        if arreglos.existe_binaria(tarjetas_bloqueadas, un_pago_nro_tarjeta):
            print('Tarjeta:', un_pago_nro_tarjeta, 'BLOQUEADA')
            cont_rechazos += 1
        else:
            print('Tarjeta:', un_pago_nro_tarjeta, 'NO Bloqueada')

    return utilidades.calcular_porcentaje(cont_rechazos, len(lote_pagos))


def calcular_rechazos_por_marca(lote_pagos, tarjetas_bloqueadas):
    """
    Calcula la cantidad de pagos rechazados, discriminado por "marca"
    de tarjeta. La discriminación se realiza de acuerdo al primer
    dígito del número de tarjeta. Por ejemplo:
    La tarjeta 4203688416671910, tiene por primer dígito a 4, que
    corresponde a "Tarjeta Pomelo".
    Como las marcas de tarjeta van de 1 a 9, pero los índices del vector
    de 0 a 8, se resta 1 para colocarlo en el casillero correcto

    :param lote_pagos: El lote de pagos a verificar
    :param tarjetas_bloqueadas: El vector de tarjetas bloqueadas
    :return: Un vector contando los rechazos por cada marca.

    Por ejemplo:
    Si en el lote se rechazan
        * 3 tarjetas "Vasa (Cod. 1)",
        * 2 tarjetas "Tarjeta Pomelo (Cod. 4)" y
        * 5 tarjetas "Tarjeta AED (9)"
    El retorno será:
                ------------------------------------
    Valores:    | 3 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 5 |
                ------------------------------------
    Índices:      0   1   2   3   4   5   6   7   8

    """
```

```
    vec_rechazos_por_marca = [0] * 9
    # Recorrer el lote de pagos
    for i in range(len(lote_pagos)): # pudo ser for un_pago_nro_tarjeta in lote_pagos:
        # Uno de los pagos a procear
        un_pago_nro_tarjeta = lote_pagos[i]
        # Revisar si está bloqueada
        if arreglos.existe_binaria(tarjetas_bloqueadas, un_pago_nro_tarjeta):
            # Este print no es necesario, ni se pondrían en un
            # Software real, está puesto para ver el procesamiento
            print('Tarjeta:', un_pago_nro_tarjeta, 'BLOQUEADA')
            # Se obtiene el código de marca de ese número de tarjeta
            cod_marca = int(un_pago_nro_tarjeta[0])
            # Contamos un rechazo para la marca cod_marca
            vec_rechazos_por_marca[cod_marca - 1] += 1
        else:
            # Este print no es necesario, ni se pondrían en un
            # Software real, está puesto para ver el procesamiento
            print('Tarjeta:', un_pago_nro_tarjeta, 'NO Bloqueada')

    # Retorno del vector
    return vec_rechazos_por_marca


def mostrar_rechazos_por_marca(rechazos_marca):
    """
    Muestra el vector de rechazos por marca, pero
    convirtiendo el código a la descripción de la
    marca de la tarjeta

    :param rechazos_marca: El vector con los contadores de rechazo
    :return: None
    """
    # Se recorre el vector
    for i in range(len(rechazos_marca)):
        # El código de marca, es el índice + 1.
        # índice 0 -> código 1 -> Marca "Vasa"
        cod_marca = i + 1
        # Se muestran los rechazos
        print(convertir_codigo_a_marca(cod_marca), '\t\t\t\t-->', rechazos_marca[i])


def main():
    """
    Función principal del programa, maneja el menú e
    invoca a las funciones necesarias para resolver
    cada punto del problema.

    :return: None
    """
    menu = '========== MENÚ DE OPCIONES ==========\n' \
           '1) Cargar Lote de Pagos\n' \
           '2) Mostrar Lista de Bloqueadas\n' \
           '3) Mostrar lote de transacciones\n' \
           '4) Informe de porcentaje de rechazos\n' \
           '5) Informe de rechazos por marca\n' \
           '6) Informe de marca menos segura\n' \
           '0) Salir'

    # Esto se puede hacer de varias maneras. Una opción es con estas banderas que
    # agregué aquí. También podrían generarse los vectores en cuestión vaciós, o
    # Su referencia None.
    b_lote_cargado = False          # Indica si ya se cargó el lote de pagos
    b_rechazos_generados = False    # Indica si ya se calcularon los rechazos por marca

    # Carga de la lista de tarjetas bloqueadas (Para pruebas, 30 está OK)
    tarjetas_bloqueadas = generar_tarjetas_bloqueadas(30)
```

```
op = -1  # Opción para forzar el ingreso al menú
# Mientras la opción no sea salir...
while op != 0:
    # Se presenta el menú
    print(menu)
    # Se solicita el ingreso de la opción, no hace falta validarla con una función
    # porque el procesamiento de las opciones ya filtra las opciones inválidas
    op = int(input('Seleccione Opción: '))
    # Procesamiento de las opciones
    if op == 0:  # Salir
        print('Hasta Luego!')
    elif op == 1:  # Cargar Lote de transacciones
        # Pedir al usuario la cantidad de pagos a cargar
        n = validaciones.validar_mayor_igual(1, "Ingrese cantidad de pagos: ")
        # Generación del vector con los pagos
        lote_pagos = cargar_lote_pagos(n)
        # Se indica que ya se generó el lote
        b_lote_cargado = True
        # Se indica que no se generó todavía el vector de rechazos por marca
        b_rechazos_generados = False
    elif op == 2:  # Mostrar Lista de Bloqueadas
        # Se podría hacer una función para mostrar las tarjetas de manera
        # más prolija
        print('Tarjetas Bloqueadas: \n', tarjetas_bloqueadas)
    elif not b_lote_cargado:
        # Aún no se cargó un lote de transacciones, no se debe continuar
        # con las subsiguientes opciones
        print('Todavía no cargó el lote de transacciones')
    elif op == 3:  # Mostrar lote de transacciones
        # Se podría hacer una función para mostrar las tarjetas de manera
        # más prolija
        print('Lote de Pagos: \n', lote_pagos)
    elif op == 4:  # Informe de porcentaje de rechazos
        # Cálculo de los rechazos por marca
        porcentaje_rechazos = calcular_porcentaje_rechazos(lote_pagos, tarjetas_bloqueadas)
        # Se muestra el resultado
        print('Porc. rechazadas es', round(porcentaje_rechazos, 2), '%')
    elif op == 5:  # Informe de rechazos por marca
        # Cálculo de los rechazos por cada "marca" de tarjeta
        rechazos_por_marca = calcular_rechazos_por_marca(lote_pagos, tarjetas_bloqueadas)
        # Se indica que ya se generó el vector
        b_rechazos_generados = True
        # Se muestra el vector de rechazos
        print('Rechazos por marca: \n')
        mostrar_rechazos_por_marca(rechazos_por_marca)
    elif op == 6:  # Informe de marca menos segura
        if b_rechazos_generados:
            # Nos piden la marca de tarjeta menos segura. Esto sería, aquella que
            # tuvo mayor cantidad de rechazos (para nuestro ejemplo). Con lo cual
            # El problema se traduce en buscar, dentro del vector de rechazos por marca,
            # Aquel con el contador mayor
            indice, mayor = arreglos.buscar_mayor(rechazos_por_marca)
            cod_marca = indice + 1
            # Se muestra el resultado
            if mayor > 0:
                print('La marca menos segura es:', convertir_codigo_a_marca(cod_marca), 'con', mayor, 'rechazos')
            else:
                print('No hubo rechazos de ninguna marca')
            # Si recordamos, el índice + 1 equivale al código de la marca.
            # índice 1 -> código 2 -> Marca: American Slow
        else:
            # Se le indica al usuario que para poder buscar la marca menos segura.
            # ¿Es esto realmente necesario?. Se podría generar el vector aquí mismo
            print('Primero debe informar los rechazos por marca')
    else:
        # Si la opción no es ninguna de las válidas...
        print('Opción inválida!')
```

```
# Script principal
if __name__ == '__main__':
    main()
```