# 🧪 Práctica Evaluativa –Parcial I

**Asignatura:** Programación I

**Tema:** Estructuras Repetitivas con Listas

**Caso 3 — SUBE — Tarjetas y saldos**

**Enunciado / Descripción**

Se requiere desarrollar un sistema para gestionar tarjetas SUBE (Sistema Único de Boleto Electrónico) y sus respectivos saldos. El programa debe implementar un menú interactivo basado en el uso de listas paralelas: una lista tarjetas[] para almacenar los números de identificación de las tarjetas SUBE y otra lista saldos[] para almacenar el saldo correspondiente a cada tarjeta. La relación entre las listas se define por el índice: el saldo en saldos[i] corresponde a la tarjeta con número tarjetas[i]. El programa debe usar un bucle while para presentar un menú al usuario, permitiéndole realizar diversas operaciones hasta que elija la opción "Salir".

**Ejemplo:**

* tarjetas[] = ["1234567890123456", "9876543210987654", "5555666677778888"]
* saldos[] = [150.50, 25.00, -10.00]

En este ejemplo:

* La tarjeta "1234567890123456" tiene un saldo de $150.50.
* La tarjeta "9876543210987654" tiene un saldo de $25.00.
* La tarjeta "5555666677778888" tiene un saldo de -$10.00 (saldo negativo).

**Desafío:** Implementar el sistema de gestión de tarjetas SUBE y saldos utilizando listas paralelas y un menú interactivo.

**Menú:**

1. **Ingresar números de tarjeta:** (Registrar las tarjetas SUBE)
   * Permite al usuario ingresar los números de identificación de las tarjetas SUBE. Debe validarse que el número ingresado tenga un formato válido (e.g., 16 dígitos).
   * Ejemplo: El usuario ingresa "1111222233334444", "6666777788889999".
2. **Ingresar saldos correspondientes:** (Asignar saldos iniciales a las tarjetas)
   * Permite al usuario ingresar el saldo inicial para cada tarjeta SUBE. Debe corresponder al orden de las tarjetas ingresadas previamente.
   * Ejemplo: Si las tarjetas son "1111222233334444" y "6666777788889999", el usuario podría ingresar "50.00" para la primera tarjeta y "100.00" para la segunda.
3. **Mostrar todas las tarjetas y saldos:** (Listar todas las tarjetas registradas y sus saldos)
   * Muestra una lista de todos los números de tarjeta SUBE registrados y el saldo correspondiente a cada una.
   * Ejemplo de salida:
     + "Tarjeta: 1234567890123456 - Saldo: $150.50"
     + "Tarjeta: 9876543210987654 - Saldo: $25.00"
     + "Tarjeta: 5555666677778888 - Saldo: -$10.00"
4. **Consultar saldo por número:** (Verificar el saldo de una tarjeta específica)
   * Permite al usuario ingresar un número de tarjeta SUBE y ver su saldo actual. Si la tarjeta no está registrada, se debe mostrar un mensaje de error.
   * Ejemplo: El usuario ingresa "9876543210987654" y el programa muestra "Saldo: $25.00".
5. **Listar saldos en negativo o cero:** (Identificar tarjetas con saldo insuficiente)
   * Muestra una lista de todos los números de tarjeta SUBE que tienen un saldo negativo o cero.
6. **Agregar tarjeta:** (Sumar una nueva tarjeta al sistema)
   * Permite al usuario agregar un nuevo número de tarjeta SUBE al sistema, incluyendo su saldo inicial.
7. **Ver saldos ≤ 0:** (Mostrar las tarjetas con saldos negativos o cero)
   * Muestra una lista de las tarjetas cuyo saldo es menor o igual a cero.
8. **Cargar/debitar saldo:** (Aumentar o disminuir el saldo de una tarjeta)
   * Permite al usuario cargar (agregar) o debitar (restar) saldo de una tarjeta SUBE. Debe solicitarse el monto a cargar o debitar, y realizar las validaciones necesarias (e.g., no permitir débitos que resulten en saldos menores a un valor mínimo).
   * Ejemplo: El usuario elige "Cargar" para la tarjeta "1234567890123456" e ingresa un monto de "50.00". El saldo de la tarjeta se actualiza a $200.50.
9. **Ver todas:** (Mostrar todas las tarjetas y saldos)
   * Muestra una lista completa de todas las tarjetas registradas y sus saldos, similar a la opción 3.
10. **Salir:** (Terminar el programa)
    * Termina la ejecución del programa.Principio del formulario

📤 **Entregables**

El estudiante deberá **subir el archivo del programa en lenguaje Python** a la plataforma institucional. NO SUBIR UN REPOSITORIO DE GITHUB. **SOLO SUBIR EL ARCHIVO.PY**

El código debe cumplir con:

* + Todas las funcionalidades solicitadas reflejadas en el menú.
  + Buena ejecución sin errores.
  + Nomenclatura clara en el nombre de las variables.
  + Legibilidad general y buenas prácticas de codificación.

# 📊 Rúbrica de Evaluación

**Rúbrica de Evaluación Detallada para Gestión de Tarjetas de Crédito/Débito (Simplificado)**

| **Código** | **Criterio** | **Peso** | **Descripción Detallada** |
| --- | --- | --- | --- |
| **C1** | **Correctitud Funcional** | **50%** | **Este criterio evalúa si el programa cumple con las funciones principales del sistema de gestión de tarjetas. Esto incluye:**   * **Agregar tarjeta: Permite agregar nuevas tarjetas al sistema, validando que el identificador (número de tarjeta) sea único.** * **Ver saldos ≤ 0: Permite listar las tarjetas que tienen un saldo menor o igual a 0.** * **Cargar/Debitar: Permite cargar (aumentar el saldo) y debitar (disminuir el saldo) de una tarjeta.** * **Consulta por número: Permite consultar el saldo de una tarjeta específica a partir de su número.** * **Manejar casos borde sin errores: El programa debe manejar correctamente situaciones como intentar debitar más del saldo disponible, consultar una tarjeta inexistente, etc., mostrando mensajes de error apropiados y evitando comportamientos inesperados.** |
| **C2** | **Cumplimiento de Restricciones** | **20%** | **Este criterio evalúa el cumplimiento de las restricciones impuestas en el diseño del programa. Esto incluye:**   * **Listas paralelas: Se deben utilizar listas paralelas para almacenar los identificadores de las tarjetas y sus saldos correspondientes. No se permite el uso de otras estructuras de datos como clases, diccionarios, etc.** * **Conservar tarjetas aunque saldo = 0: El sistema debe mantener el registro de una tarjeta incluso si su saldo es cero. No se deben eliminar tarjetas del sistema.** * **Sin clases ni diccionarios: El uso de clases o diccionarios está prohibido y resultará en una penalización. Se busca evaluar la capacidad de resolver el problema utilizando listas.** * **Sincronía entre tarjetas[] y saldos[]: El programa debe asegurar que la sincronización entre las listas se mantenga en todo momento. El saldo en la posición i de la lista saldos[] debe corresponder a la tarjeta en la posición i de la lista tarjetas[].** |
| **C3** | **Interacción y Validación** | **10%** | **Este criterio evalúa la calidad de la interacción con el usuario y la robustez de las validaciones implementadas. Esto incluye:**   * **Identificador no vacío: El programa debe validar que el identificador de la tarjeta (ej., número de tarjeta) ingresado por el usuario no esté vacío.** * **Saldo inicial ≥ 0: El programa debe validar que el saldo inicial ingresado por el usuario sea un número no negativo.** * **Débito ≥ 1 y ≤ saldo actual: Al debitar de una tarjeta, el programa debe verificar que la cantidad a debitar sea mayor o igual a 1 (evitando débitos nulos o negativos) y que no exceda el saldo actual de la tarjeta.** * **Carga ≥ 1: Al cargar saldo en una tarjeta, el programa debe verificar que la cantidad a cargar sea mayor o igual a 1 (o un valor positivo).** * **Tarjeta inexistente: Al intentar cargar o debitar, o consultar el saldo, el programa debe verificar que la tarjeta exista en el sistema.** * **Menú persistente: El menú principal del programa debe permanecer visible y funcional hasta que el usuario seleccione la opción de salir.** * **Opciones inválidas: El programa debe manejar las opciones inválidas del menú (ej., ingresar letras en lugar de números) con gracia, mostrando un mensaje claro y comprensible al usuario.** |
| **C4** | **Estructura y Legibilidad** | **10%** | **Este criterio evalúa la calidad del código en términos de estructura, legibilidad y estilo. Esto incluye:**   * **Variables descriptivas: Los nombres de las variables deben ser descriptivos y reflejar el propósito de la variable.** * **Flujo claro: El código debe seguir un flujo lógico, fácil de entender y mantener.** * **Mensajes consistentes: Los mensajes mostrados al usuario deben ser claros, concisos y coherentes.** * **Evitar redundancia: El código debe evitar la duplicación innecesaria de código. Se deben utilizar funciones para reutilizar la lógica común.** |
| **C5** | **Casos de Prueba / Datos de Ejemplo** | **5%** | **Este criterio evalúa la variedad y cobertura de los casos de prueba utilizados para verificar la funcionalidad del programa. Esto incluye:**   * **Saldo = 0: Probar el sistema con una tarjeta que tiene un saldo de 0.** * **Intentar debitar más del saldo: Intentar debitar una cantidad mayor al saldo disponible en la tarjeta.** * **Tarjeta inexistente: Intentar cargar o debitar, o consultar el saldo, de una tarjeta que no existe en el sistema.** * **Ver saldos ≤ 0: Probar la funcionalidad de ver las tarjetas con saldo menor o igual a 0.** |
| **C6** | **Gestión de Casos Borde** | **5%** | **Este criterio evalúa la capacidad del programa para manejar situaciones límite o casos especiales que pueden surgir. Esto incluye:**   * **Débito con saldo 0: Intentar debitar de una tarjeta que ya tiene un saldo de 0.** * **Débito mayor al saldo: Intentar debitar una cantidad mayor al saldo disponible en la tarjeta.** * **Identificador vacío: Intentar agregar una tarjeta con un identificador (número) vacío.** * **Tarjeta no registrada: Intentar realizar una operación (carga, débito, consulta) sobre una tarjeta que no existe en el sistema.** |

**Descriptores globales por nivel**  
Excelente (90–100): Cumple todos los criterios sin fallas; maneja casos borde exhaustivamente; interacción robusta; código claro y consistente.  
Muy Bueno (80–89): Cumple la mayoría; pequeños desajustes no críticos; validación adecuada; estilo mayormente claro.  
Aprobado (60–79): Cumple los básicos; algunos fallos funcionales menores o validación incompleta; legibilidad aceptable.  
Insuficiente (<60): Incumple criterios clave; errores que impiden el uso; omite restricciones solicitadas.

**Penalizaciones y observaciones**  
• −30% si utiliza estructuras prohibidas (diccionarios, clases, etc.).  
• −10% si no conserva ítems con cantidad 0 (elimina de una lista y no de la otra).  
• −10% por ausencia total de validación de entradas.  
• −5% por mensajes ininteligibles o menú que no persiste.

**Notas**  
• Asignar puntajes parciales por criterio y calcular la nota final ponderada.  
• Compartir la rúbrica con estudiantes antes de la evaluación.  
• Mantener consistencia entre enunciados, resultados de aprendizaje y criterios de evaluación.

**Rúbrica Adaptada para SUBE (Sin Funciones, Excepciones, Diccionarios):**

Esta rúbrica asume un código lineal, sin funciones, excepciones ni estructuras de datos complejas. La evaluación se centra en observar tarjetas[], saldos[] y la salida del programa.

* **C1: Correctitud Funcional (50%)**
  + **Agregar tarjeta:**
    - Longitud de tarjetas[] y saldos[] aumenta en 1.
    - El número de tarjeta (simulado) se encuentra en tarjetas[].
    - El saldo inicial (simulado) se encuentra en saldos[].
  + **Consultar saldo por número:**
    - Capturar la salida estándar.
    - Verificar que la salida contenga el saldo correcto para el número de tarjeta simulado.
  + **Ver saldos ≤ 0:**
    - Capturar la salida estándar.
    - Verificar que la salida contenga los números de tarjeta con saldos negativos o cero (en el formato esperado).
  + **Cargar/Debitar saldo:**
    - **Cargar:** Simular una carga. Capturar la salida. Verificar que el saldo en saldos[] haya aumentado correctamente.
    - **Debitar:** Simular un débito. Capturar la salida. Verificar que el saldo en saldos[] haya disminuido correctamente.
* **C2: Cumplimiento de Restricciones (20%)**
  + **Listas paralelas:** Analizar el código fuente para verificar la ausencia de diccionarios, clases, etc. (Este paso es manual o con expresiones regulares simples).
  + **Sincronía entre listas:** Verificar que tarjetas[] y saldos[] tengan la misma longitud después de cada operación.
* **C3: Interacción y Validación (10%)**
  + **Formato de número de tarjeta (16 dígitos):** Simular la entrada de un número de tarjeta inválido (ej., menos de 16 dígitos, con letras). Capturar la salida. Verificar el mensaje de error.
  + **Saldo inicial ≥ 0:** Simular la entrada de un saldo inicial negativo o no numérico. Capturar la salida. Verificar el mensaje de error.
  + **Débito ≥ 1 y ≤ saldo actual:** Simular un débito mayor al saldo actual. Capturar la salida. Verificar el mensaje de error.
* **C4: Estructura y Legibilidad (10%):** Evaluación visual/manual.
* **C5: Casos de Prueba / Datos de Ejemplo (5%):** Preparar datos de prueba específicos y evaluar la salida.
* **C6: Gestión de Casos Borde (5%):** Pruebas para situaciones límite.