# 🧪 Práctica Evaluativa –Parcial I

**Asignatura:** Programación I

**Tema:** Estructuras Repetitivas con Listas

**Caso 4 — Deporte — Inscripción a clases (cupos)**

**Enunciado / Descripción**

Un gimnasio necesita un sistema para administrar las clases que ofrece y el número de cupos (espacios disponibles) para cada clase. Se pide construir un programa que emplee listas paralelas: una lista clases[] para almacenar los nombres de las clases (e.g., "Yoga", "Funcional", "Zumba") y otra lista cupos[] para almacenar el número de cupos disponibles para cada clase. Ambas listas deben estar ligadas por el índice, de manera que el cupo en cupos[i] corresponda a la clase en clases[i]. El programa debe presentar un menú al usuario y utilizar un bucle while para permitirle realizar diferentes operaciones hasta que elija la opción "Salir".

**Ejemplo:**

* clases[] = ["Yoga", "Funcional", "Zumba"]
* cupos[] = [10, 15, 20]

En este ejemplo:

* La clase de Yoga tiene 10 cupos disponibles.
* La clase de Funcional tiene 15 cupos disponibles.
* La clase de Zumba tiene 20 cupos disponibles.

**Desafío:** Implementar el sistema de gestión de clases y cupos utilizando listas paralelas y un menú interactivo.

**Menú:**

1. **Ingresar lista de clases (Yoga, Funcional...):** (Registrar las clases ofrecidas)
   * Permite al usuario ingresar los nombres de las clases que ofrece el gimnasio.
   * Ejemplo: El usuario ingresa "Pilates", "Spinning".
2. **Ingresar cupos por clase:** (Definir la capacidad de cada clase)
   * Permite al usuario ingresar el número de cupos disponibles para cada clase. Debe corresponder al orden de las clases ingresadas previamente.
   * Ejemplo: Si las clases son "Pilates" y "Spinning", el usuario podría ingresar "8" para Pilates y "12" para Spinning.
3. **Mostrar clases con cupos:** (Mostrar la disponibilidad de cada clase)
   * Muestra una lista de todas las clases y el número de cupos disponibles para cada una.
   * Ejemplo de salida:
     + "Yoga: 10 cupos"
     + "Funcional: 15 cupos"
     + "Zumba: 20 cupos"
     + "Pilates: 8 cupos"
     + "Spinning: 12 cupos"
4. **Consultar cupo de una clase:** (Verificar la disponibilidad de una clase específica)
   * Permite al usuario ingresar el nombre de una clase y ver cuántos cupos están disponibles.
   * Ejemplo: El usuario ingresa "Funcional" y el programa muestra "15 cupos disponibles".
5. **Listar clases sin cupo:** (Identificar las clases que están llenas)
   * Muestra una lista de todas las clases que tienen 0 cupos disponibles.
6. **Agregar clase:** (Sumar una nueva clase a la lista)
   * Permite al usuario agregar una nueva clase a la lista, incluyendo su número inicial de cupos.
7. **Ver sin cupo:** (Mostrar las clases sin disponibilidad)
   * Muestra una lista reducida que solo incluye las clases que no tienen cupos disponibles (cupo = 0).
8. **Actualizar cupos (inscribir/baja):** (Gestionar inscripciones y bajas de alumnos)
   * Permite al usuario actualizar el número de cupos disponibles al inscribir un alumno a una clase (disminuir el cupo) o dar de baja a un alumno de una clase (aumentar el cupo). Debe validarse que no se pueda inscribir más alumnos de los cupos disponibles, ni dar de baja a alumnos que no están inscriptos.
   * Ejemplo: Si un alumno se inscribe en la clase de Yoga, el usuario puede actualizar el cupo de 10 a 9. Si un alumno se da de baja, el cupo se actualiza de 9 a 10.
9. **Ver todas clases:** (Mostrar todas las clases y sus cupos)
   * Muestra una lista completa de todas las clases y sus cupos correspondientes, similar a la opción 3.
10. **Salir:** (Terminar el programa)
    * Termina la ejecución del programa.

📤 **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 Clases**

| **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 clases. Esto incluye:**   * **Agregar clase: Permite agregar nuevas clases al sistema, validando que el nombre de la clase sea único.** * **Ver sin cupo: Permite listar las clases que no tienen cupos disponibles (cupo = 0).** * **Inscribir/Baja: Permite inscribir a un estudiante en una clase (si hay cupo disponible) y dar de baja a un estudiante de una clase.** * **Consulta de clase: Permite consultar la información de una clase específica (nombre y cupos disponibles).** * **Ver todas: Permite mostrar una lista completa de todas las clases y su información (nombre, cupos).** * **Incluir casos borde: El programa debe manejar correctamente situaciones como intentar inscribir en una clase llena, dar de baja a alguien que no está inscripto, etc., mostrando mensajes de error apropiados.** |
| **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 nombres de las clases y su cantidad de cupos disponibles. No se permite el uso de otras estructuras de datos como diccionarios o clases.** * **Conservar clases con cupo = 0: El sistema debe mantener el registro de una clase incluso si no tiene cupos disponibles. Esto permite que la clase se ofrezca más adelante si se agregan cupos.** * **Sin estructuras prohibidas: El uso de estructuras de datos no permitidas (ej., diccionarios, clases) está prohibido y resultará en una penalización. Se busca evaluar la capacidad de resolver el problema utilizando listas.** * **Sincronía entre listas: El programa debe asegurar que la sincronización entre las listas se mantenga en todo momento. El cupo en la posición i debe corresponder a la clase en la posición i de la otra lista.** |
| **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:**   * **Nombre no vacío: El programa debe verificar que el nombre de la clase ingresada por el usuario no esté vacío.** * **Duplicado al agregar: El programa debe verificar que el nombre de la clase que se intenta agregar no exista ya en el sistema.** * **Cupos iniciales ≥ 0: El programa debe validar que la cantidad inicial de cupos ingresada por el usuario sea un número no negativo.** * **Inscribir ≤ cupos: Al inscribir a un estudiante, el programa debe verificar que haya cupos disponibles en la clase.** * **Baja ≥ 1: Al dar de baja a un estudiante, el programa debe verificar que la cantidad sea al menos 1 (o un valor positivo) para evitar errores en la lógica (en caso de que se esté gestionando una lista de estudiantes inscritos por clase).** * **Clase inexistente: Al intentar inscribir o dar de baja, o consultar información, el programa debe verificar que la clase 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.** * **Estructura organizada: El código debe estar bien organizado, utilizando funciones para separar las diferentes funcionalidades. El flujo del programa debe ser claro y fácil de seguir.** * **Mensajes claros: Los mensajes mostrados al usuario deben ser claros, concisos y fáciles de entender.** * **Evitar repetición: 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:**   * **Cupo = 0: Probar el sistema con una clase que no tiene cupos disponibles.** * **Intentar inscribir más que lo disponible: Intentar inscribir a un estudiante en una clase que ya está llena.** * **Consulta inexistente: Intentar consultar información de una clase que no existe en el sistema.** * **Etc.: Se deben incluir otros casos de prueba que cubran diferentes escenarios y situaciones límite.** |
| **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:**   * **Cupo = 0: Intentar inscribir a un estudiante en una clase que no tiene cupos.** * **Clase no existente: Intentar inscribir o dar de baja a un estudiante de una clase que no existe en el sistema.** * **Nombre inválido: Intentar agregar una clase con un nombre vacío o que contenga caracteres no permitidos.** * **Baja inválida: Intentar dar de baja a un estudiante que no está inscripto en la clase (si se gestiona explícitamente la lista de inscriptos por clase).** |

**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:**

* **C1: Correctitud Funcional (50%)**
  + **Agregar clase:**
    - Verificar que clases[] y cupos[] aumenten en longitud.
    - Verificar que los valores simulados estén presentes en las listas.
  + **Consultar cupo:**
    - Capturar la salida estándar.
    - Verificar que la salida contenga el cupo correcto para la clase simulada.
  + **Listar clases sin cupo:**
    - Capturar la salida estándar.
    - Verificar que la salida contenga la lista correcta de clases con cupos = 0.
  + **Inscribir/Baja:**
    - Simular una inscripción/baja. Capturar la salida.
    - Verificar que el cupo en la lista cupos[] se haya actualizado correctamente.
* **C2: Cumplimiento de Restricciones (20%)**
  + **Listas paralelas:** Análisis manual (o con expresiones regulares básicas) del código fuente para verificar la ausencia de estructuras prohibidas.
  + **Sincronía entre listas:** Verificar que clases[] y cupos[] tengan la misma longitud después de cada operación.
* **C3: Interacción y Validación (10%)**
  + **Nombre no vacío:** Simular la entrada de un nombre vacío. Capturar la salida. Verificar el mensaje de error.
  + **Cupos iniciales ≥ 0:** Simular la entrada de un cupo no válido (ej., negativo, no numérico). Capturar la salida. Verificar el mensaje de error.
  + **Inscribir ≤ cupos:** Simular intentar inscribir a más alumnos de los disponibles. 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 representativos y evaluar la salida.
* **C6: Gestión de Casos Borde (5%):** Pruebas específicas para situaciones límite.

Principio del formulario