# 🧪 Práctica Evaluativa –Parcial I

**Asignatura:** Programación I

**Tema:** Estructuras Repetitivas con Listas

**Caso 2 — Clínica — Turnos por especialidad (cupos del día)**

**Caso 2 – Clínica – Turnos por especialidad (cupos del día)**

**Enunciado / Descripción**

Una clínica desea administrar eficientemente las diferentes especialidades médicas que ofrece y la disponibilidad de cupos (espacios para citas) diarios para cada una. Se pide implementar un programa que utilice listas paralelas: una lista especialidades[] para almacenar los nombres de las especialidades (e.g., "Cardiología", "Dermatología") y otra lista cupos[] para almacenar el número de cupos disponibles para cada especialidad en un día específico. Ambas listas deben compartir el mismo índice, de manera que el cupo en cupos[i] corresponda a la especialidad en especialidades[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:**

* especialidades[] = ["Cardiología", "Dermatología", "Pediatría"]
* cupos[] = [5, 3, 10]

En este ejemplo:

* Cardiología tiene 5 cupos disponibles.
* Dermatología tiene 3 cupos disponibles.
* Pediatría tiene 10 cupos disponibles.

**Desafío:** Implementar el sistema de gestión utilizando las listas paralelas y el menú interactivo.

**Menú:**

1. **Ingresar lista de especialidades:** (Añadir especialidades a la lista)
   * Permite al usuario ingresar los nombres de las especialidades que ofrece la clínica.
   * Ejemplo: El usuario ingresa "Neurología", "Oftalmología".
2. **Ingresar lista de cupos disponibles por especialidad:** (Definir la cantidad de cupos para cada especialidad)
   * Permite al usuario ingresar el número de cupos disponibles para cada especialidad. Debe corresponder al orden de las especialidades ingresadas previamente.
   * Ejemplo: Si las especialidades son "Neurología" y "Oftalmología", el usuario podría ingresar "8" para Neurología y "6" para Oftalmología.
3. **Mostrar agenda:** (Mostrar la disponibilidad de cupos por especialidad)
   * Muestra una lista de todas las especialidades y el número de cupos disponibles para cada una.
   * Ejemplo de salida:
     + "Cardiología: 5 cupos"
     + "Dermatología: 3 cupos"
     + "Pediatría: 10 cupos"
4. **Consultar cupos de una especialidad:** (Verificar la disponibilidad de una especialidad específica)
   * Permite al usuario ingresar el nombre de una especialidad y ver cuántos cupos están disponibles.
   * Ejemplo: El usuario ingresa "Dermatología" y el programa muestra "3 cupos disponibles".
5. **Listar especialidades sin cupo:** (Identificar las especialidades que están llenas)
   * Muestra una lista de todas las especialidades que tienen 0 cupos disponibles.
6. **Agregar especialidad:** (Sumar una nueva especialidad a la lista)
   * Permite al usuario agregar una nueva especialidad a la lista, incluyendo su número inicial de cupos.
7. **Ver sin cupo:** (Mostrar solo las especialidades sin disponibilidad)
   * Muestra una lista reducida que solo incluye las especialidades que no tienen cupos disponibles (cupo = 0).
8. **Actualizar cupos (reservar/cancelar):** (Modificar la disponibilidad al reservar o cancelar citas)
   * Permite al usuario actualizar el número de cupos disponibles al reservar una cita (disminuir el cupo) o cancelar una cita (aumentar el cupo).
   * Ejemplo: Si un paciente reserva una cita en Cardiología, el usuario puede actualizar el cupo de 5 a 4.
9. **Ver agenda:** (Mostrar la agenda completa)
   * Muestra una lista detallada de todas las especialidades y sus cupos correspondientes. Similar a la opción 3, pero puede incluir detalles adicionales como la hora de inicio y fin de la jornada.
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 Reservas de Especialidades Médicas**

| **Código** | **Criterio** | **Peso** | **Descripción Detallada** |
| --- | --- | --- | --- |
| **C1** | **Correctitud Funcional** | **50%** | **Qué evaluar específicamente en este caso: La correcta implementación de las funciones principales del sistema.**   * **Agregar especialidad: Permite agregar nuevas especialidades (ej., Cardiología, Dermatología) al sistema, validando que el nombre de la especialidad sea único.** * **Ver sin cupo: Permite listar las especialidades que no tienen cupos disponibles (cupo = 0). Esto indicaría que están completas o no se ofrecen en ese momento.** * **Reservar/Cancelar: Permite reservar un turno para una especialidad (disminuyendo el cupo) y cancelar una reserva (aumentando el cupo).** * **Ver agenda: Permite mostrar una "agenda" de las especialidades disponibles, con sus cupos actuales.** * **Consulta específica: Permite consultar la información de una especialidad en particular (nombre y cupos disponibles).** * **Casos borde: cupo = 0, especialidad inexistente: Maneja correctamente la reserva cuando el cupo ya es cero, así como la consulta o reserva de una especialidad que no ha sido agregada.** |
| **C2** | **Cumplimiento de Restricciones** | **20%** | **Qué evaluar específicamente en este caso: El respeto a las restricciones de diseño.**   * **Listas paralelas: Se deben utilizar listas paralelas para almacenar los nombres de las especialidades y sus cupos disponibles. No se permite el uso de otras estructuras de datos como diccionarios o clases.** * **Conservar especialidades con cupo = 0: El sistema debe mantener el registro de una especialidad aunque no tenga cupos disponibles. Esto permite que la especialidad se reactive si se liberan cupos o se amplía la capacidad.** * **Sin estructuras prohibidas: El uso de estructuras de datos no permitidas (ej., diccionarios, clases) está prohibido y resultará en una penalización.** * **Sincronía entre especialidades[] y cupos[]: El programa debe asegurar que la sincronización entre las listas se mantenga en todo momento. El cupo en la posición i de la lista cupos[] debe corresponder a la especialidad en la posición i de la lista especialidades[].** |
| **C3** | **Interacción y Validación** | **10%** | **Qué evaluar específicamente en este caso: La calidad de la interacción con el usuario y la validez de los datos ingresados.**   * **Nombre no vacío: El programa debe validar que el nombre de la especialidad ingresada por el usuario no esté vacío.** * **Duplicado al agregar: El programa debe verificar que el nombre de la especialidad 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.** * **Reservar ≤ cupos: Al reservar un turno, el programa debe verificar que haya cupos disponibles en la especialidad.** * **Cancelar ≥ 1: Al cancelar un turno, el programa debe verificar que la cantidad a cancelar (que normalmente será 1) sea un valor válido (positivo).** * **Especialidad inexistente: Al intentar reservar o cancelar un turno, o consultar información, el programa debe verificar que la especialidad 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) mostrando mensajes de error informativos.** |
| **C4** | **Estructura y Legibilidad** | **10%** | **Qué evaluar específicamente en este caso: La calidad y claridad del código.**   * **Variables descriptivas: Los nombres de las variables deben ser descriptivos y reflejar el propósito de la variable.** * **Estructura del menú clara: La organización de las opciones del menú debe ser lógica e intuitiva para el usuario.** * **Mensajes coherentes: Los mensajes mostrados al usuario deben ser claros, concisos y fáciles de entender.** * **Evitar duplicaciones: 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%** | **Qué evaluar específicamente en este caso: La cobertura de diferentes escenarios y situaciones límite al probar el programa.**   * **Especialidad con cupos = 0: Probar el sistema con una especialidad que no tiene cupos disponibles.** * **Reservar más que lo disponible: Intentar reservar más turnos de los que hay cupos disponibles en una especialidad.** * **Cancelar: Probar la funcionalidad de cancelar una reserva.** * **Consulta inexistente: Intentar consultar información de una especialidad que no existe en el sistema.** * **Ver sin cupo: Probar la funcionalidad de listar las especialidades sin cupo.** |
| **C6** | **Gestión de Casos Borde** | **5%** | **Qué evaluar específicamente en este caso: El manejo adecuado de situaciones límite o casos especiales que pueden causar errores.**   * **Reservar cuando cupos = 0: Intentar reservar un turno en una especialidad que ya no tiene cupos.** * **Nombre vacío: Intentar agregar una especialidad con un nombre vacío.** * **Especialidad no agregada: Intentar realizar una operación (reserva, consulta) sobre una especialidad que no existe en el sistema.** * **Cancelaciones inválidas: Intentar cancelar una reserva que no existe (ej., si no se lleva un registro individual de las reservas, esto podría ser intentar cancelar "más" turnos de los que "deberían" estar reservados).** |

**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 (Sin Funciones, Excepciones, Diccionarios ni Estructuras Avanzadas):**

Esta rúbrica asume que el código del estudiante consistirá en un script principal lineal, sin funciones ni estructuras de datos complejas. La evaluación se centrará en observar el estado de las variables especialidades[] y cupos[] y analizar la salida del programa.

* **C1 Correctitud Funcional (50%):**
  + **Agregar especialidad:**
    - **Lista especialidades[] y cupos[]:** Después de la sección de código para agregar una especialidad:
      * Verificar que la longitud de la lista especialidades[] haya aumentado en 1.
      * Verificar que la longitud de la lista cupos[] haya aumentado en 1.
      * Verificar que el nombre de la especialidad (simulado) esté presente en la lista especialidades[].
      * Verificar que el cupo inicial (simulado) esté presente en la lista cupos[].
    - **Evitar especialidades duplicadas:**
      * Ejecutar el código para agregar una especialidad duplicada.
      * Verificar que la longitud de la lista especialidades[] **no** haya cambiado.
      * Verificar que se imprima un mensaje (si se espera) indicando el error.
  + **Consultar disponibilidad:**
    - **Búsqueda de la especialidad:**
      * Ejecutar la sección de código que busca una especialidad específica.
      * Capturar la salida estándar del programa.
      * Verificar que la salida estándar contenga la cantidad correcta de cupos disponibles.
    - **Especialidad inexistente:**
      * Ejecutar la sección de código que busca una especialidad que no existe.
      * Capturar la salida estándar del programa.
      * Verificar que la salida estándar contenga un mensaje indicando que la especialidad no se encuentra.
  + **Ver sin cupo:**
    - **Lista de especialidades sin cupo:**
      * Ejecutar la sección de código que muestra la lista de especialidades sin cupo.
      * Capturar la salida estándar del programa.
      * Verificar que la salida estándar contenga la lista correcta de especialidades con cupos[i] == 0, en el formato esperado.
  + **Reservar/Cancelar:**
    - **Reserva:** Simular la reserva de un turno. Capturar la salida. Verificar que el cupo en la lista cupos[] se haya disminuido correctamente.
    - **Cancelar:** Simular la cancelación de un turno. Capturar la salida. Verificar que el cupo en la lista cupos[] se haya aumentado correctamente.
* **C2 Cumplimiento de Restricciones (20%):**
  + **Sólo listas paralelas:** Analizar el código fuente (manualmente o con expresiones regulares muy simples) para confirmar la ausencia de diccionarios, clases, etc.
  + **Sincronía entre listas:** Después de cada operación, verificar que las listas especialidades[] y cupos[] tengan la misma longitud.
* **C3 Interacción y Validación (10%):**
  + **Nombre no vacío:** Simular la entrada de un nombre vacío. Capturar la salida. Verificar el mensaje de error. Verificar que no se haya agregado la especialidad.
  + **Cupos iniciales ≥ 0:** Simular la entrada de un valor no numérico o negativo. Capturar la salida. Verificar el mensaje de error. Verificar que no se haya agregado la especialidad.
  + **Reservar ≤ cupos:** Intentar reservar más turnos de los disponibles. Capturar la salida. Verificar el mensaje de error. Verificar que el cupo no haya cambiado.
* **C4 Estructura y Legibilidad (10%):** No automatizable directamente. Requeriría análisis visual o IA.
* **C5 Casos de Prueba / Datos de Ejemplo (5%):** Seleccionar datos de prueba representativos y evaluar la salida para esos casos.
* **C6 Gestión de Casos Borde (5%):** Diseñar pruebas específicas para situaciones límite y evaluar la salida del programa.

# 