Ingeniería Biomédica

UTEC
Universidad Tecnológica

Sistema de matriculaciones para Ingeniería Biomédica v2.0

Trabajo Práctico del Módulo 3,4 y 5

```python
import json

class PlanEstudio():...


class UC():...


class GestorExpediente:
    """...

    def __init__(self, nombre: str, ci: int, añoIngreso: int):
        """...
        self.datos = {"Nombre": nombre,...

        nombre_archivo = nombre.replace(" ", "_")
        self.archivo = f"{nombre_archivo}_{ci}.json"

    def crearArchivo(self, indentacion: int = 4):
        """...
        with open(self.archivo, 'w', encoding='utf-8') as file:
            json.dump(self.datos, file, ensure_ascii=False, indent=indentacion)
```

Docentes: Acosta Juan, Sanchez Joaquin y Sosa Florencia

Docentes: MSc. Bioing. Baldezzari Lucas e Ing. ALVAREZ Emiliano

Unidad Curricular: Programación Digital Avanzada

Fecha de entrega: 05/05/2025

Ingeniería Biomédica

Instituto Tecnológico Regional Sur-Oeste

Universidad Tecnológica del Uruguay

Índice

1. INTRODUCCIÓN

Los sistemas de matriculaciones en instituciones de educación superior requiere herramientas que garanticen eficiencia, organización y cumplimiento de los planes de estudio. Uno de los procesos más sensibles es la inscripción a cursos y exámenes, especialmente cuando estos dependen de requisitos previos (tener unidades curriculares aprobadas). En el caso de la carrera de Ingeniería Biomédica de la Universidad Tecnológica del Uruguay (UTEC), el sistema actual no verifica automáticamente dichas condiciones, lo que genera errores frecuentes y sobrecarga administrativa.

El objetivo de este trabajo práctico es abordar esta problemática, se desarrolló un Sistema de Matriculaciones (SdM) básico, aplicando principios de Programación Orientada a Objetos (POO). El sistema permite gestionar la trayectoria académica del estudiante, controlando la inscripciones a unidades curriculares, inscripción a exámenes, siempre en función de las previas definidas por el plan de estudios.

Este informe detalla el diseño, implementación y funcionamiento del sistema, así como las decisiones técnicas adoptadas durante su desarrollo. Además, se analizó la estructura del código, el uso de diagramas UML para visualizar las relaciones entre clases y se realizó un modo de uso para poder utilizar el código.

2. MARCO TEÓRICO

Los procesos administrativos en instituciones de educación superior han adquirido una importancia creciente en los últimos años, tanto para optimizar la gestión interna como para mejorar la experiencia del estudiante. En particular, la matriculación y la inscripción a exámenes son actividades fundamentales que, en muchos casos, requieren cumplir requisitos previos definidos por el plan de estudio de cada carrera.

En la actualidad, dentro del sistema Universita XXI utilizado por la Universidad Tecnológica del Uruguay (UTEC), los y las estudiantes tienen la posibilidad de matricularse libremente a cualquier unidad curricular (UC) o examen final (EF), sin que el sistema verifique automáticamente si se cumplen las previaturas exigidas para cada unidad curricular. Esta falta de control automatizado traslada la responsabilidad de validación al propio estudiante, en segunda instancia, al personal de secretaría.

El resultado es una carga administrativa significativa y una posible pérdida de tiempo tanto para los estudiantes como para los docentes y el personal técnico administrativo. Es frecuente que las correcciones a inscripciones erróneas se realicen varias semanas después de comenzado el curso, lo que afecta el normal desarrollo de la actividad académica.

Frente a esta problemática, surge la necesidad de desarrollar un Sistema de Matriculación (SdM) básico que contemple automáticamente los requisitos del plan de estudios de la carrera de Ingeniería Biomédica (IBIO). Dicho sistema debe ser capaz de determinar, en base al historial académico del estudiante, si cumple o no con las condiciones necesarias para cursar o presentarse a examen en cada UC. Esto incluye la verificación de unidades curriculares previamente aprobadas o rendidas en modalidad de examen, de modo que se pueda evitar la inscripción indebida desde el inicio y garantizar un proceso más eficiente y justo.

3. OBJETIVOS

3.1 Objetivo General

- Implementar un Sistema de Matriculaciones -básico- a Unidades Curriculares y Exámenes para la carrera de Ingeniería Biomédica (plan 2021) utilizando Python y los conceptos de Programación Orientada a Objetos (POO) vistos hasta el momento.

3.2 Objetivos Específicos

- Reforzar los conceptos de Programación Orientada a Objetos (POO), incluyendo abstracción, encapsulamiento y los distintos tipos de relaciones entre clases, como asociación simple, agregación y composición.
- Reforzar los conceptos de Clase y Objeto mediante la implementación del sistema de matriculaciones ya que deberá definir clases, objetos e interacciones entre estos.
- Reforzar los conceptos de diagramas UML.
- Implementar conceptos de Manejo de Excepciones y Manejo de Archivos en Python

4. MATERIALES

- Python 3.9 (o superior).
- Visual Studio (recomendado).
- Un repositorio en Github.com.
- Plan de estudios de la carrera de Ingeniería Biomédica.

5. PROCEDIMIENTO

A Continuación, se detalla el procedimiento que se tuvo en cuenta para crear el Sistema de Matriculación:

1. Se comenzó con la implementación de la clase PlanEstudio, cuya función principal es cargar y estructurar el plan de estudios a partir de un archivo JSON. Esta clase permite acceder fácilmente a la información de cada unidad curricular (como su nombre, créditos y previas), y constituye la base sobre la que se validan todas las operaciones académicas posteriores.
2. Luego se definió la clase UC, que representa a cada unidad curricular del plan. A través de esta clase es posible consultar los datos de una materia específica, permitiendo validar si un estudiante puede o no acceder a ella, ya sea para cursarla o rendir examen.
3. Una vez establecida la estructura del plan, se desarrolló la clase GestorExpediente, encargada de gestionar el archivo JSON individual de cada estudiante. Esta clase permite registrar materias aprobadas, inscripciones a cursos y exámenes, y mantener actualizado el historial académico. Además, administra el archivo global Registros.json, que contiene un resumen general de las inscripciones del sistema.
4. Con esa base se diseñó la clase Estudiante, que permite interactuar con el expediente desde la perspectiva del alumno. A través de ella, se pueden realizar operaciones como matricularse, inscribirse a exámenes o consultar el estado académico, todo sujeto a las restricciones del plan de estudios.
5. Finalmente, se creó la clase Secretaria, orientada a cubrir el rol administrativo. Esta clase permite cargar un expediente y realizar acciones similares a las del estudiante, pero con mayor nivel de control, como agregar manualmente materias aprobadas o consultar listas de inscriptos a nivel global.
6. Cada una de estas clases se articuló respetando los principios de la Programación Orientada a Objetos, asegurando una estructura modular, reutilizable y coherente con la lógica académica real.

7. RESULTADOS Y DISCUSIÓN

El sistema desarrollado permite gestionar la inscripción, matrícula y aprobación de las Unidades Curriculares (UCs) de la carrera de Ingeniería Biomédica. Incluye la validación de la aprobación de las previas (otras UCs que deben ser aprobadas previamente), así como el control de créditos académicos y el registro de datos personales y académicos de los estudiantes.

6.1 Clases y Roles del Código

A continuación se describen las clases implementadas en el sistema y los roles que cumple cada una dentro de la estructura del código:

Clase PlanEstudio

- Rol: Cargar el plan de estudio desde un archivo JSON.
- Constructor: def __init__(self, archivo="PlanEstudio2021.json")
  Inicializa el objeto PlanEstudio con el nombre del archivo de PlanEstudio2021. Permite establecer un archivo por defecto o personalizarlo al instanciar la clase.
- Método:
  - def cargarPlanEstudio(self):
    Abre y carga el archivo JSON utilizando codificación UTF-8. Devuelve un diccionario con la información del plan, organizada por semestres.
- Interacción: Esta clase es utilizada por UC para acceder a la información de las materias.

UC (Unidad Curricular)

- Rol: Representa una materia específica del plan.
- Constructor: def __init__(self, plan: PlanEstudio, semestre: str, codigo: str):

Recibe un objeto PlanEstudio, un semestre (clave del diccionario) y el código de la materia. Carga internamente el diccionario del plan usando cargarPlanEstudio().

- Métodos:
    - def infoUC(self):
      Extrae la información asociada a la UC como nombre, previas y créditos. Lanza un error si las credenciales no son válidas.
    - def nombreMateria(self):
      Retorna el nombre de la materia si infoUC() fue ejecutado correctamente.
    - def nombrePrevias(self):
      Devuelve la lista de previas.
    - def getCreditos(self):
      Devuelve la cantidad de créditos de la UC.
- Interacción: Es utilizada por las clases GestorExpediente, Estudiante y Secretaria como fuente de información académica.

Clase GestorExpediente

- Rol: Gestiona los archivos JSON por estudiante. Permite crear, cargar y modificar archivos JSON individuales para cada estudiante, facilitando el almacenamiento y la administración de su información personal y académica
- Constructor: def __init__(self, nombre: str, ci: int, añoIngreso: int):
  Inicializa los datos básicos del estudiante y prepara el nombre del archivo JSON personalizado por estudiante (usa nombre y cédula).

- Métodos:

    - def crearArchivo(self, indentacion: int = 4):
      Crea el archivo JSON con los datos iniciales del estudiante.

    - def crear_registros(self, indentacion: int = 4):
      Crea un archivo JSON para guardar las inscripciones de Matrículas y Exámenes de todos los estudiantes.

    - def cargarDatos(self):
      Carga el contenido de un archivo JSON y lo almacena internamente en el atributo self.data. Permite al sistema recuperar la información académica y personal de un estudiante desde su archivo JSON (por ejemplo, materias aprobadas, matriculadas, inscripciones a examen, créditos, etc.)

    - def actualizar_archivo(self, nombre_archivo, info):
      Se encarga de actualizar (sobrescribir) el contenido de un archivo JSON con nueva información proporcionada. Guarda los cambios realizados en los datos del estudiante (como nuevas inscripciones, aprobaciones o cambios de estado) dentro de su archivo JSON.

    - def _inscribir_examen(self, uc :UC):
      Gestiona la inscripción a exámenes, permitiendo agregar una Unidad Curricular (UC) de la lista de Inscripción a Examen. Recibe como argumento un objeto UC correspondiente a la materia deseada. Este método valida si el estudiante cumple con los requisitos antes de inscribirlo. Registra la inscripción en su archivo individual y también en el archivo global Registros.json (a través de _registrar_inscripcion). Devuelve True si la inscripción fue exitosa, o False si ya estaba inscripto o no cumple con las previas

- def _quitar_examen(self, materia:UC):
  Se encarga de eliminar una inscripción a un examen de un estudiante para una materia determinada. Permite anular la inscripción a un examen en caso de que el estudiante se haya equivocado o decida no presentarse, y actualiza el archivo individual del estudiante y el archivo de registros.json.

- def _matricular_uc(self, uc :UC):
  Permite matricular al estudiante en una unidad curricular, siempre que cumpla con las previas o no esté ya matriculado o tenga la unidad curricular aprobada. Recibe como parámetro un objeto UC que representa la unidad curricular que se desea matricular. Guarda los cambios en el archivo individual del estudiante y actualiza el archivo registro.json.

- def _desmatricular_uc(self, materia :UC):
  Permite eliminar una unidad curricular de la lista de materias matriculadas del archivo individual del estudiante y del archivo registros.json. Recibe como parámetro un objeto UC que representa la materia que se desea desmatricular.

- def _agregar_uc_aprobada(self, materia :UC):
  Se encarga de registrar una unidad curricular como aprobada para un estudiante, y sumar los créditos correspondientes a su total acumulado. Permite registrar una uc como superada por el estudiante, lo cual también implica aumentar el conteo de créditos obtenidos. Es útil para que la Secretaría actualice el avance académico del estudiante.

- def _quitar_uc_aprobada(self, materia :UC):
  Se encarga de eliminar una unidad curricular previamente aprobada del expediente del estudiante y restar sus créditos del total acumulado. Permite corregir errores en el expediente, como por ejemplo una unidad curricular aprobada que se registró por equivocación. También es útil si se detecta que se sumaron créditos erróneamente.

- def _registrar_inscripcion(self, materia: UC, instancia:str):
  Registra al estudiante en el archivo global registros.json cuando se matricula o se inscribe a un examen. Mantiene actualizado el archivo global de inscripciones (Registros.json) con los nombres de los estudiantes matriculados o inscriptos a examen para cada unidad curricular. Es llamado por:_matricular_uc() y _inscribir_examen()

- def _eliminar_inscripcion(self, materia: UC, instancia:str):
  Se encarga de remover al estudiante del archivo registros.json cuando se desinscribe de un examen o se desmatricula de una unidad curricular. Cuando un estudiante se desmatricula o cancela una inscripción a examen, el archivo registros.json refleja correctamente que ya no está inscripto en esa uc. Es el complemento del método _registrar_inscripcion()

- def _inscriptos_examen(self):
  Devuelve un diccionario con todos los estudiantes inscriptos a exámenes, agrupados por materia. Permite consultar el estado actual de inscripción a exámenes en todo el sistema. Es utilizado por la Secretaría.

- def _inscriptos_matriculas(self):
  Devuelve un diccionario que indica, para cada unidad curricular, quiénes son los estudiantes actualmente matriculados. Permite consultar el estado actual de los matriculados en todo el sistema. Es utilizado por la Secretaría.

- Interacción: Es el núcleo de operaciones usado por Estudiante y Secretaria. Se apoya en UC para validar requisitos.

Clase Estudiante

- Rol: Modela al estudiante y permite consultar su información y realizar acciones académicas.

- Constructor: def __init__(self, usuario: GestorExpediente):
  Toma un objeto GestorExpediente, lo utiliza para cargar los datos desde archivo JSON, y expone esos datos como atributos.

- Métodos:

    - def infoEstudiante(self):
      Devuelve el nombre y la cédula de identidad del estudiante en forma de tupla, lo cual permite identificarlo de manera única dentro del sistema académico.

    - def ucs_aprobadas(self):
      Devuelve el atributo self.aprobadas, que es una lista previamente cargada en el constructor de la clase Estudiante. Esa lista viene directamente del archivo JSON del estudiante, y contiene los nombres (como strings) de las unidades curriculares aprobadas.

    - def inscribir_examen(self, materia: UC):
      Permite que un estudiante se inscriba a un examen. Recibe como parámetro un objeto UC, que representa una unidad curricular. Llama al método _inscribir_examen() del objeto self.usuario, que es GestorExpediente. El gestor es quien tiene acceso y permiso para modificar el archivo del estudiante y registrar la inscripción en Registros.json.

    - def quitar_examen(self, materia: UC):
      Elimina la inscripción del estudiante a un examen para la materia especificada. Llama internamente al método privado _quitar_examen() del objeto GestorExpediente, que es quien realmente modifica el archivo del estudiante y actualiza registros.json.

- def matricular_uc(self, materia: UC)
  Matrícula al estudiante en una Unidad Curricular (UC), si cumple los requisitos previos. Utiliza el método _matricular_uc() del GestorExpediente para actualizar el archivo del estudiante y registrar la matrícula en registros.json.

- def desmatricular_uc(self, materia: UC):
  Elimina la matrícula del estudiante de una determinada unidad curricular. Utiliza el método _desmatricular_uc() del GestorExpediente, que borra la uc de la lista del archivo del estudiante y la remueve del archivo de registros.

- def _str_(self):
  Devuelve una representación en formato de cadena (string) del estudiante, incluyendo información básica como su nombre y cédula, para facilitar su identificación dentro del sistema

- Interacción: Utiliza los métodos del GestorExpediente para realizar operaciones, siempre con objetos UC

Clase Secretaria

- Rol: Simula el rol administrativo de secretaria o coordinadora.

- Constructor: def __init__(self, usuario: GestorExpediente):
  Inicializa con nombre y prepara la clase para operar con expedientes.

- Métodos:

    - def cargar_estudiante(self, expediente: GestorExpediente):
      Asocia a un estudiante al contexto de trabajo de la secretaría, cargando su expediente académico para futuras operaciones (como inscripciones o consultas). Permite todos los métodos posteriores de la clase Secretaria (como inscribir_examen, matricular_uc, etc.) trabajen sobre ese estudiante sin tener que pasarlo cada vez.

    - def inscribir_examen(self, materia: UC):
      Utiliza el expediente cargado previamente (self.expediente) para inscribir al estudiante a un examen, si cumple con las materias previas.Llama al método interno _inscribir_examen() del GestorExpediente.

    - def quitar_examen(self, materia: UC):
      Elimina la inscripción del estudiante a un examen. Llama al método interno _quitar_examen() del GestorExpediente. También se actualiza el archivo Registros.json para que refleje la baja.

    - def matricular_uc(self, materia: UC):
      Matrícula al estudiante en una unidad curricular si cumple con los requisitos previos (aprobadas). Llama al método interno _matricular_uc() del GestorExpediente. Se registra en el archivo del estudiante y en registros.json.

    - def desmatricular_uc(self, materia: UC):
      Elimina la materia del listado de cursadas del estudiante.También borra el registro en el archivo global si corresponde. Llama al método interno _desmatricular_uc() del GestorExpediente.

- def agregar_uc_aprobada(self, aprobada: UC):
  Añade una materia al listado de aprobadas del estudiante. Suma sus créditos al total acumulado. Llama al método interno _agregar_uc_aprobada() del GestorExpediente.

- def quitar_uc_aprobada(self, materia: UC):
  Elimina una materia previamente aprobada y descuenta sus créditos. Llama al método interno _quitar_uc_aprobada() del GestorExpediente.

- def ver_inscriptos_examen(self):
  Retorna un diccionario con todas las materias y los estudiantes que se anotaron a sus exámenes. Llama al método interno _inscriptos_examen() del GestorExpediente.

- def ver_inscriptos_examen(self):
  Similar al anterior, pero devuelve los estudiantes cursando materias. Llama al método interno _inscriptos_matriculas() del GestorExpediente.

- Interacción: Accede al GestorExpediente como el estudiante, pero con más permisos.

6.2 Relaciones entre Clases

El siguiente diagrama UML representa gráficamente la estructura del sistema de matriculación, mostrando las clases definidas, sus atributos, métodos y relaciones. Este recurso facilita la comprensión del diseño orientado a objetos y la interacción entre componentes del sistema:

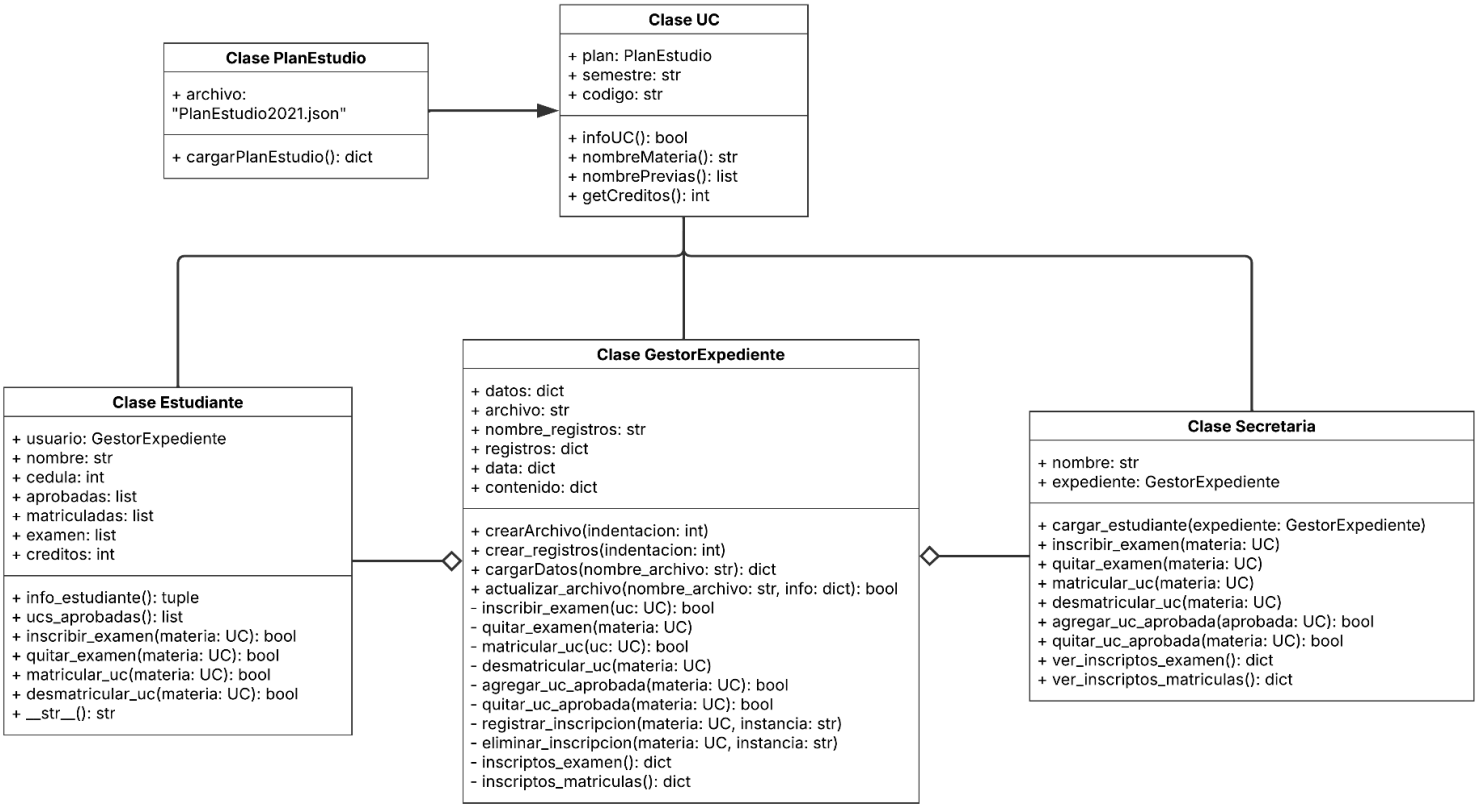

Figura 1. Diagrama UML

Se identifican distintos tipos de relaciones entre clases, como composición, agregación y asociaciones simples, representando diferentes grados de dependencia entre los objetos:

- PlanEstudio y UC mantienen una relación de composición. Cada objeto UC necesita obligatoriamente una instancia de PlanEstudio, ya que obtiene toda su información (nombre, créditos, previas) directamente desde ese plan. Si no existe un plan cargado, la UC no puede operar.

- La clase UC se encuentra en una asociación simple con las clases Estudiante, GestorExpediente y Secretaria. Estas clases no la contienen como atributo fijo, pero interactúan constantemente con objetos UC como parámetros de inscripción, aprobación o validación.
- La clase GestorExpediente tiene una composición fuerte con el archivo JSON del estudiante: crea, edita y guarda directamente el archivo con sus datos académicos. También mantiene y actualiza el archivo global Registros.json.
- La clase Estudiante tiene una relación por agregación con GestorExpediente: depende de él para operar sobre sus datos, pero no se responsabiliza por su creación o estructura interna.
- Finalmente, la clase Secretaria establece una relación de agregación con GestorExpediente. Utiliza una instancia externa que le es provista mediante el método cargar_estudiante, y opera sobre ella para realizar acciones administrativas. Aunque depende funcionalmente del expediente para ejecutar sus funciones, accede a los metodos del GestorExpediente.

6.3 Modo de Uso

En esta sección se detalla el procedimiento necesario para el correcto uso del Sistema de Matriculación desarrollado. Se presentan, paso a paso, las acciones requeridas para inicializar el entorno, registrar estudiantes y ejecutar operaciones comunes, como la inscripción a materias o exámenes, ya sea por parte del estudiante o la secretaría.

Para su puesta en marcha, es imprescindible contar con el archivo PlanEstudio2021.json, el cual contiene la estructura oficial del plan de estudios 2021 de la carrera y permite instanciar correctamente las unidades curriculares.

A continuación, se presentan los pasos necesarios para utilizar el sistema:

1. Accedemos a una jupyternetbook e importar el sistema de matriculación desde el archivo "SistemaMatriculaciones.py", junto con las clases necesarias como se muestra a continuación:

```
from SistemaMatriculaciones import UC, PlanEstudio, Estudiante,
Secretaria, GestorExpediente
```

2. A continuación, se carga el archivo PlanEstudio2021.json, que contiene la información necesaria para instanciar las unidades curriculares del plan de estudios:

```
planIBIO = PlanEstudio("PlanEstudio2021.json")
planIBIO.cargarPlanEstudio()
```

3. Se instancian una o más unidades curriculares, las cuales serán utilizadas para operar el Sistema de Matriculación:

```
matematica = UC(planIBIO, "Semestre 1", "S1UC1", )
print(matematica.infoUC())
print(matematica.nombreMateria())
print(matematica.nombrePrevias())

print(matematica.getCreditos())
```

4. Se crea un objeto con los datos personales y académicos iniciales del estudiante, lo que genera un archivo JSON que representa su expediente académico vacío. Además, se crea el archivo Registros.json, encargado de almacenar todas las matrículas e inscripciones a exámenes registradas en el sistema:

```
expediente = GestorExpediente("Agustín Acosta", 56692489, 2022)
expediente.crearArchivo()
expediente.crear_registros()
```

5. Se instancia un objeto de la clase Estudiante a partir del expediente previamente creado, lo que permite acceder y manipular la información académica del estudiante:

```
agustin = Estudiante(expediente)
print(agustin)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: []

Créditos: 0
```

Interacción con el Sistema de Matriculación desde la perspectiva del estudiante.

1. Se matricula al estudiante en una unidad curricular específica mediante el método matricular_uc, siempre que cumpla con los requisitos previos definidos en el plan de estudios y se imprime por pantalla la información académica actual del estudiante:

```
agustin.matricular_uc(matematica)
agustin=Estudiante(expediente)
print(agustin)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: ['Álgebra, Análisis y Geometría Analítica']
Inscripcion Examenes: []
Créditos: 0
```

2. Se desmatricula al estudiante de la unidad curricular especificada mediante el método desmatricular_uc, lo que actualiza tanto su expediente académico como el registro global del sistema. Luego, se imprime su información para verificar que la desmatriculación se haya realizado correctamente:

```
agustin.matricular_uc(matematica)
agustin=Estudiante(expediente)
print(agustin)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: ['Álgebra, Análisis y Geometría Analítica']
Inscripcion Examenes: []
Créditos: 0
```

3. Se desmatricula al estudiante de la unidad curricular especificada mediante el método desmatricular_uc, lo que actualiza tanto su expediente académico como el registro global del sistema. Luego, se imprime su información para verificar que la desmatriculación se haya realizado correctamente.

```
agustin.desmatricular_uc(matematica)
agustin=Estudiante(expediente)
print(agustin)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: []
Créditos: 0
```

4. Se inscribe al estudiante en el examen de una unidad curricular determinada, siempre que cumpla con los requisitos previos. Finalmente, se imprime la información académica:

```
agustin.inscribir_examen(matematica)
agustin=Estudiante(expediente)
print(agustin)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: ['Álgebra, Análisis y Geometría Analítica']
Créditos: 0
```

5. Se elimina su inscripción al examen de la unidad curricular especificada mediante el método quitar_examen. Luego, se imprime su información académica para verificar que la acción se haya completado correctamente

```
agustin.quitar_examen(matematica)
agustin=Estudiante(expediente)
print(agustin)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: []
Créditos: 0
```

Si el estudiante intenta matricularse o inscribirse a un examen de una unidad curricular sin haber aprobado las materias previas requeridas, el sistema no registrará la inscripción y, por lo tanto, la unidad curricular no se verá reflejada en su archivo JSON.

Interacción con el Sistema de Matriculación desde la perspectiva de la Secretaria

1. Se instancia un objeto de la clase Secretaria con el nombre "Jessica", lo que permite simular acciones administrativas sobre los expedientes académicos de los estudiantes, tales como inscripciones, matrículas y gestión de unidades curriculares aprobadas.

```
jessica = Secretaria("Jessica")
```

2. Se carga el expediente académico del estudiante en el objeto Secretaria mediante el método cargar_estudiante, lo que le permite a la secretaria realizar operaciones administrativas sobre dicho expediente, como matriculación, inscripción a exámenes o gestión de materias aprobadas

```
jessica.cargar_estudiante(expediente)
```

3. Matricula al estudiante en la unidad curricular "Física" utilizando el método matricular_uc, verificando que se cumplan los requisitos académicos establecidos en el plan de estudios antes de registrar la inscripción en el expediente y en el archivo de registros

```
jessica.matricular_uc(fisica)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: ['Mecánica, Ondas y Calor']
Inscripcion Examenes: []
Créditos: 0
```

4. Desmatricula al estudiante de la unidad curricular "Física" mediante el método desmatricular_uc, lo que actualiza tanto el expediente académico como el archivo de registros

```
jessica.desmatricular_uc(fisica)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: []
Créditos: 0
```

5. Iinscribe al estudiante al examen de la unidad curricular "Física" utilizando el método inscribir_examen. Esta acción se registra en su expediente académico y en el archivo de registro seccion exámenes, siempre que se cumplan las materias previas.

```
jessica.inscribir_examen(fisica)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: ['Mecánica, Ondas y Calor']
Créditos: 0
```

6. Elimina la inscripción del estudiante al examen de la unidad curricular "Física" mediante el método quitar_examen. Esta acción actualiza tanto el expediente académico del estudiante como el archivo de registros.

```
jessica.quitar_examen(fisica)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: []
Inscripcion Examenes: []
Créditos: 0
```

7. Agrega la unidad curricular "Matemática" como materia aprobada en el expediente del estudiante mediante el método agregar_uc_aprobada. Esta acción actualiza la lista de materias aprobadas y suma los créditos correspondientes en el archivo JSON del estudiante.

```
jessica.agregar_uc_aprobada(matematica)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: ['Álgebra, Análisis y Geometría Analítica']
Materias Matriculadas: []
Inscripcion Examenes: []
Créditos: 8
```

8. Elimina la unidad curricular "Matemática" de la lista de materias aprobadas del estudiante mediante el método quitar_uc_aprobada. Esta acción actualiza el expediente académico, removiendo la materia y descontando los créditos correspondientes en el archivo JSON del estudiante.

```
jessica.quitar_uc_aprobada(matematica)
```

```
Información del estudiante:
Nombre: Agustín Acosta
C.I: 56692489
Materias aprobadas: []
Materias Matriculadas: ['Óptica y Radiaciones']
Inscripcion Examenes: ['Óptica y Radiaciones']
Créditos: 0
```

9. La secretaria consulta los registros del sistema utilizando los métodos ver_inscriptos_examen y ver_inscriptos_matriculas, los cuales devuelven diccionarios que asocian cada unidad curricular con la lista de estudiantes inscriptos a exámenes o matriculados, respectivamente. Actualmente, el sistema cuenta con un solo estudiante, pero está diseñado para manejar múltiples registros. Si dos estudiantes se inscriben o se matriculan en la misma unidad curricular, el sistema mostrará una única entrada con el nombre de la UC y una lista que incluye los nombres de ambos estudiantes. En cambio, si cada estudiante se registra en unidades diferentes, cada unidad aparecerá por separado con los nombres correspondientes. Esta lógica de agrupamiento se aplica tanto a las inscripciones a exámenes como a las matrículas.

```
jessica.ver_inscriptos_examen()
```

```
{'Álgebra, Análisis y Geometría Analítica': ['Florencia Sosa'],
 'Taller Inicial de Tecnologías': ['Florencia Sosa']}
```

10. Dado que el sistema valida automáticamente los requisitos previos, ni el estudiante ni la secretaria pueden matricular o inscribir al estudiante en una unidad curricular si no se han aprobado las materias previas correspondientes. Sin embargo, una vez que el estudiante cumple con dichas previas, ya sea por aprobación directa o mediante la intervención de la secretaria, podrá ser matriculado o inscripto a examen sin restricciones.

```
jessica.ver_inscriptos_matriculas()
```

```
{'Taller Inicial de Tecnologías': ['Florencia Sosa', 'Agustín
Acosta'],
 'Álgebra, Análisis y Geometría Analítica': ['Florencia Sosa']}
```

CONCLUSIONES

El sistema desarrollado cumple con los objetivos planteados al inicio del trabajo. Se logró implementar una herramienta funcional que permite gestionar la inscripción a materias y exámenes de la carrera de Ingeniería Biomédica, respetando las previaturas definidas por el plan de estudios.

Durante su desarrollo se aplicaron los principales conceptos de Programación Orientada a Objetos, como clases, métodos, atributos, encapsulamiento y relaciones entre objetos. También se trabajó con archivos en formato JSON, lo que permitió almacenar y consultar tanto el historial académico individual de los estudiantes como los registros globales de inscripciones.

El sistema está preparado para operar desde dos roles distintos: el del estudiante y el de la secretaría. En ambos casos, se valida automáticamente si se cumplen las previas antes de permitir una inscripción, lo que evita errores comunes y garantiza que solo se registren acciones válidas.

Además, el diseño del sistema se organizó de forma modular y clara, dividiendo las clases según su función. Las clases estructurales, como PlanEstudio y UC, representan la base académica; las clases de gestión, como GestorExpediente, administran los datos y las operaciones internas; y las clases que simulan los distintos roles de usuario, como Estudiante y Secretaria, permiten interactuar con el sistema de acuerdo a los permisos correspondientes.