+++ date = '2025-02-21T10:18:14-08:00' draft = false title = 'Practica 2' +++

Primer código: biblioteca.py

```
'''Módulo para definir las clases de la biblioteca y sus operaciones'''

import json
from memory_management import memory_management


class Genre:
    '''Clase para definir los generos de los libros'''
    FICTION = "Ficcion"
    NON_FICTION = "No Ficcion"
    SCIENCE = "Ciencia"
    HISTORY = "Historia"
    FANTASY = "Fantasia"
    BIOGRAPHY = "Biografia"
    OTHER = "Otro"

    @classmethod
    def all_genres(cls):
        return [cls.FICTION, cls.NON_FICTION, cls.SCIENCE, cls.HISTORY,
cls.FANTASY, cls.BIOGRAPHY, cls.OTHER]


class Book:
    '''Clase para definir los libros de la biblioteca'''
    def __init__(self, book_id, title, author, publication_year, genre, quantity):
        self.id = book_id
        self.title = title
        self.author = author
        self.publication_year = publication_year
        self.genre = genre
        self.quantity = quantity
        memory_management.increment_heap_allocations(1)

    def __del__(self):
        memory_management.increment_heap_deallocations(1)

    def to_dict(self):
        '''Método para convertir los datos del libro en un diccionario'''
        return {
            "id": self.id,
            "title": self.title,
            "author": self.author,
            "publication_year": self.publication_year,
            "genre": self.genre,
            "quantity": self.quantity
        }
```

```
    @staticmethod
    def from_dict(data):
        '''Método para crear un objeto libro a partir de un diccionario'''
        return Book(
            int(data["id"]),
            data["title"],
            data["author"],
            int(data["publication_year"]),
            data["genre"],
            int(data["quantity"])
        )


class DigitalBook(Book):
    '''Clase para definir los libros digitales de la biblioteca'''
    def __init__(self, book_id, title, author, publication_year, genre, quantity,
file_format):
        '''Constructor de la clase DigitalBook'''
        super().__init__(book_id, title, author, publication_year, genre,
quantity)
        self.file_format = file_format

    def to_dict(self):
        '''Método para convertir los datos del libro digital en un diccionario'''
        data = super().to_dict()
        data["file_format"] = self.file_format
        return data

    @staticmethod
    def from_dict(data):
        '''Método para crear un objeto libro digital a partir de un diccionario'''
        return DigitalBook(
            int(data["id"]),
            data["title"],
            data["author"],
            int(data["publication_year"]),
            data["genre"],
            int(data["quantity"]),
            data["file_format"]
        )


class Member:
    '''Clase para definir los miembros de la biblioteca'''
    def __init__(self, member_id, name):
        '''Constructor de la clase Member'''
        self.id = member_id
        self.name = name
        self.issued_books = []
        memory_management.increment_heap_allocations(1)

    def __del__(self):
        memory_management.increment_heap_deallocations(1)
```

```
    def to_dict(self):
        '''Método para convertir los datos del miembro en un diccionario'''
        return {
            "id": self.id,
            "name": self.name,
            "issued_books": self.issued_books
        }

    @staticmethod
    def from_dict(data):
        '''Método para crear un objeto miembro a partir de un diccionario'''
        member = Member(int(data["id"]), data["name"])
        member.issued_books = data["issued_books"]
        return member


class Library:
    '''Clase para definir la biblioteca'''
    def __init__(self):
        '''Constructor de la clase Library'''
        self.books = []
        self.members = []
        memory_management.increment_heap_allocations(1)

    def __del__(self):
        memory_management.increment_heap_deallocations(1)

    def add_book(self, book):
        '''Método para agregar un libro a la biblioteca'''
        self.books.append(book)
        print("\nEl libro fue agregado exitosamente!\n")
        memory_management.display_memory_usage()

    def find_book_by_id(self, book_id):
        '''Método para buscar un libro por su ID'''
        for book in self.books:
            if book.id == book_id:
                return book
        return None

    def display_books(self):
        '''Método para mostrar los libros disponibles en la biblioteca'''
        if not self.books:
            print("\nNo hay libros disponibles.\n")
            return
        print("\nLibros disponibles en biblioteca:\n")
        for book in self.books:
            print(f"ID libro: {book.id}\nTitulo: {book.title}\nAutor:
{book.author}\nAno de publicacion: {book.publication_year}\nGenero:
{book.genre}\nCantidad: {book.quantity}\n")
            if isinstance(book, DigitalBook):
                print(f"Formato de archivo: {book.file_format}\n")
        memory_management.display_memory_usage()
```

```
    def add_member(self, member):
        '''Método para agregar un miembro a la biblioteca'''
        self.members.append(member)
        print("\nMiembro agregado exitosamente!\n")
        memory_management.display_memory_usage()

    def issue_book(self, book_id, member_id):
        '''Método para prestar un libro a un miembro'''
        book = self.find_book_by_id(book_id)
        member = self.find_member_by_id(member_id)
        print(f"\nDebug##book[{book}] member[{member}] \n")
        if book and member and book.quantity > 0:
            book.quantity -= 1
            member.issued_books.append(book_id)
            print("\nLibro prestado satisfactoriamente!\n")
        else:
            print("\nLibro o miembro no encontrados.\n")
        memory_management.display_memory_usage()

    def return_book(self, book_id, member_id):
        '''Método para devolver un libro prestado por un miembro'''
        book = self.find_book_by_id(book_id)
        member = self.find_member_by_id(member_id)
        if book and member and book_id in member.issued_books:
            book.quantity += 1
            member.issued_books.remove(book_id)
            print("\nLibro devuelto satisfactoriamente!\n")
        else:
            print("\nLibro o miembro no encontrados.\n")
        memory_management.display_memory_usage()

    def find_member_by_id(self, member_id):
        '''Método para buscar un miembro por su ID'''
        for member in self.members:
            if member.id == member_id:
                return member
        return None

    def display_members(self):
        '''Método para mostrar los miembros disponibles en la biblioteca'''
        if not self.members:
            print("\nNo hay miembros disponibles.\n")
            return
        print("\nMiembros disponibles en biblioteca:\n")
        for member in self.members:
            print(f"ID miembro: {member.id}\nNombre: {member.name}\nCantidad de
libros prestados: {len(member.issued_books)}\n")
            for book_id in member.issued_books:
                book = self.find_book_by_id(book_id)
                if book:
                    print(f"  Libro ID: {book.id}\n  Titulo: {book.title}\n
Autor: {book.author}\n")
        memory_management.display_memory_usage()
```

```
    def search_member(self, member_id):
        '''Método para buscar un miembro por su ID y mostrar los libros
prestados'''
        member = self.find_member_by_id(member_id)
        if member:
            print(f"ID miembro: {member.id}\nNombre: {member.name}\nCantidad de
libros prestados: {len(member.issued_books)}\n")
            for book_id in member.issued_books:
                book = self.find_book_by_id(book_id)
                if book:
                    print(f"  Libro ID: {book.id}\n  Titulo: {book.title}\n
Autor: {book.author}\n")
        else:
            print("\nMiembro no encontrado.\n")
        memory_management.display_memory_usage()

    def save_library_to_file(self, filename):
        '''Método para guardar la biblioteca en un archivo JSON'''
        with open(filename, 'w', encoding='UTF-8') as file:
            json.dump([book.to_dict() for book in self.books], file)
        print(f"Biblioteca guardada exitosamente en {filename}\n")
        memory_management.display_memory_usage()

    def load_library_from_file(self, filename):
        '''Método para cargar la biblioteca desde un archivo JSON'''
        try:
            with open(filename, 'r', encoding='UTF-8') as file:
                books_data = json.load(file)
                self.books = [DigitalBook.from_dict(data) if "file_format" in data
else Book.from_dict(data) for data in books_data]
            print(f"Biblioteca cargada exitosamente desde {filename}\n")
        except FileNotFoundError:
            print("Error al abrir el archivo para cargar la biblioteca.\n")
        memory_management.display_memory_usage()

    def save_members_to_file(self, filename):
        '''Método para guardar los miembros en un archivo JSON'''
        with open(filename, 'w', encoding='UTF-8') as file:
            json.dump([member.to_dict() for member in self.members], file)
        print(f"Miembros guardados exitosamente en {filename}\n")
        memory_management.display_memory_usage()

    def load_members_from_file(self, filename):
        '''Método para cargar los miembros desde un archivo JSON'''
        try:
            with open(filename, 'r', encoding='UTF-8') as file:
                members_data = json.load(file)
                self.members = [Member.from_dict(data) for data in members_data]
            print(f"Miembros cargados exitosamente desde {filename}\n")
        except FileNotFoundError:
            print("Error al abrir el archivo para cargar los miembros.\n")
        memory_management.display_memory_usage()
```

```
def main():
    '''Función principal para ejecutar el programa'''
    library = Library()
    library.load_library_from_file("library.json")
    library.load_members_from_file("members.json")

    while True:
        print("\nMenu de sistema de manejo de biblioteca\n")
        print("\t1. Agregar un libro")
        print("\t2. Mostrar libros disponibles")
        print("\t3. Agregar un miembro")
        print("\t4. Prestar libro")
        print("\t5. Devolver libro")
        print("\t6. Mostrar miembros disponibles")
        print("\t7. Buscar miembro")
        print("\t8. Guardar y salir")
        choice = int(input("Indica tu opcion: "))

        if choice == 1:
            book_id = int(input("Ingresa ID del libro: "))
            title = input("Ingresa titulo del libro: ")
            author = input("Ingresa nombre del autor: ")
            publication_year = int(input("Ingresa el ano de publicacion: "))
            genre = input("Ingresa el genero del libro: ")
            quantity = int(input("Ingresa la cantidad de libros: "))
            is_digital = input("Es un libro digital? (s/n): ").lower() == 's'
            if is_digital:
                file_format = input("Ingresa el formato del archivo: ")
                book = DigitalBook(book_id, title, author, publication_year,
genre, quantity, file_format)
            else:
                book = Book(book_id, title, author, publication_year, genre,
quantity)
            library.add_book(book)
        elif choice == 2:
            library.display_books()
        elif choice == 3:
            member_id = int(input("Ingresa el ID del miembro: "))
            name = input("Ingresa el nombre del miembro: ")
            member = Member(member_id, name)
            library.add_member(member)
        elif choice == 4:
            member_id = int(input("Ingresa el ID del miembro: "))
            book_id = int(input("Ingresa el ID del libro: "))
            library.issue_book(book_id, member_id)
        elif choice == 5:
            member_id = int(input("Ingresa el ID del miembro: "))
            book_id = int(input("Ingresa el ID del libro: "))
            library.return_book(book_id, member_id)
        elif choice == 6:
            library.display_members()
        elif choice == 7:
            member_id = int(input("Ingresa el ID del miembro: "))
            library.search_member(member_id)
```

```
        elif choice == 8:
            library.save_library_to_file("library.json")
            library.save_members_to_file("members.json")
            print("Saliendo del programa\n")
            break
        else:
            print("Esta no es una opcion valida!!!\n")


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

1. **Nombres**:
    - Se usan nombres significativos para clases (`Book`, `Library`, `Member`, `DigitalBook`, etc.), atributos (`title`, `author`, `quantity`) y métodos (`add_book`, `find_member_by_id`, `issue_book`), siguiendo buenas prácticas de legibilidad.
2. **Objetos**:
    - Los objetos principales del sistema son instancias de las clases `Book`, `DigitalBook`, `Member`, y `Library`.
    - También hay objetos intermedios como diccionarios al serializar (`to_dict`) y listas (`books`, `members`, `issued_books`).
3. **Entornos**:
    - El entorno de ejecución considera interacciones entre el usuario (entrada/salida por consola) y la memoria dinámica a través del módulo `memory_management`.
    - El entorno de persistencia usa archivos `library.json` y `members.json`.
4. **Bloques**:
    - Las definiciones de métodos (`def ...`) y condicionales (`if`, `else`) son bloques de código bien estructurados.
    - El ciclo `while True` en `main()` es un bloque principal de interacción.
5. **Alcance**:
    - Los atributos de instancia tienen alcance dentro del objeto.
    - Las funciones como `add_book()` tienen acceso a atributos de instancia mediante `self`.
    - Las clases y funciones están definidas en el ámbito global del módulo.
6. **Administración de memoria**:
    - Se simula usando `memory_management`, que registra asignaciones y liberaciones al crear y destruir objetos (`__init__`, `__del__`).
    - No hay manejo explícito de punteros como en C, pero el monitoreo emula gestión dinámica.
7. **Expresiones**:

- Se usan expresiones aritméticas y lógicas como `book.quantity > 0`, `choice == 1`, y `is_digital = input(...) == 's'`.
- También hay expresiones de conversión de tipo como `int(input(...))`.

8. **Comandos**:
   - Las llamadas a métodos (`library.add_book(book)`, `member.issued_books.append(...)`) son comandos que cambian el estado del sistema.

9. **Secuencia**:
   - El flujo de ejecución del menú sigue una secuencia controlada por un ciclo infinito y decisiones del usuario.
   - Dentro de cada opción del menú, hay una secuencia lógica de operaciones (leer entrada, crear objetos, llamar métodos, etc.).

10. **Selección**:
    - Uso de estructuras condicionales `if`, `elif`, `else` para dirigir el flujo según la opción elegida o validaciones (`if book and member`).
    - También se usa `try/except` para manejo de errores al cargar archivos.

11. **Iteración**:
    - Se emplean bucles `for` para iterar sobre listas de libros o miembros (por ejemplo, al mostrar información con `for book in self.books`).

12. **Recursión**:
    - No hay funciones recursivas en este sistema.

13. **Subprogramas**:
    - Se definen múltiples subprogramas: constructores, métodos de instancia, funciones auxiliares (`main()`).
    - Cada subprograma tiene responsabilidades bien delimitadas.

14. **Tipos de datos**:
    - Tipos primitivos: `int`, `str`, `bool`, `list`, `dict`.
    - Tipos definidos por el usuario: clases `Book`, `DigitalBook`, `Member`, `Library`.
    - También se usan tipos compuestos al cargar/guardar archivos JSON.

Segundo código: biblioteca_web.py

```
'''Este archivo contiene la implementación de la aplicación web de la biblioteca.
Se utiliza Flask para crear una API REST que permite realizar operaciones CRUD
sobre los libros y miembros de la biblioteca.'''

from flask import Flask, request, jsonify, render_template
from memory_management import memory_management
```

```
from biblioteca import Book, DigitalBook, Member, Library, Genre

app = Flask(__name__)
library = Library()

# Cargar datos al iniciar la aplicación
library.load_library_from_file("library.json")
library.load_members_from_file("members.json")


@app.route('/')
def index():
    '''Ruta principal de la aplicación web.'''
    return render_template('index.html')


@app.route('/books', methods=['GET'])
def get_books():
    '''Obtener todos los libros de la biblioteca.'''
    books = [book.to_dict() for book in library.books]
    return jsonify(books)


@app.route('/books', methods=['POST'])
def add_book():
    '''Agregar un libro a la biblioteca'''
    data = request.json
    is_digital = data.get('is_digital', False)
    if is_digital:
        book = DigitalBook(
            int(data['id']), data['title'], data['author'],
data['publication_year'],
            data['genre'], data['quantity'], data['file_format']
        )
    else:
        book = Book(
            int(data['id']), data['title'], data['author'],
int(data['publication_year']),
            data['genre'], int(data['quantity'])
        )
    library.add_book(book)
    # Guardar datos después de agregar un libro
    library.save_library_to_file("library.json")
    memory_management.display_memory_usage()
    return jsonify(book.to_dict()), 201


@app.route('/members', methods=['GET'])
def get_members():
    '''Obtener todos los miembros de la biblioteca.'''
    members = [member.to_dict() for member in library.members]
    return jsonify(members)
```

```python
@app.route('/members', methods=['POST'])
def add_member():
    '''Agregar un miembro a la biblioteca.'''
    data = request.json
    member_id = int(data['id'])
    member = Member(member_id, data['name'])
    library.add_member(member)
    # Guardar datos después de agregar un miembro
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify(member.to_dict()), 201


@app.route('/issue_book', methods=['POST'])
def issue_book():
    '''Prestar un libro a un miembro de la biblioteca.'''
    data = request.json
    book_id = int(data['book_id'])
    member_id = int(data['member_id'])
    library.issue_book(book_id, member_id)
    # Guardar datos después de prestar un libro
    library.save_library_to_file("library.json")
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Libro prestado satisfactoriamente!"})


@app.route('/return_book', methods=['POST'])
def return_book():
    '''Devolver un libro a la biblioteca.'''
    data = request.json
    book_id = int(data['book_id'])
    member_id = int(data['member_id'])
    library.return_book(book_id, member_id)
    # Guardar datos después de devolver un libro
    library.save_library_to_file("library.json")
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Libro devuelto satisfactoriamente!"})


@app.route('/save', methods=['POST'])
def save():
    '''Guardar los datos de la biblioteca en un archivo.'''
    library.save_library_to_file("library.json")
    library.save_members_to_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Datos guardados exitosamente!"})


@app.route('/load', methods=['POST'])
def load():
    '''Cargar los datos de la biblioteca desde un archivo.'''
    library.load_library_from_file("library.json")
```

```
    library.load_members_from_file("members.json")
    memory_management.display_memory_usage()
    return jsonify({"message": "Datos cargados exitosamente!"})


@app.route('/genres', methods=['GET'])
def get_genres():
    '''Obtener todos los géneros de la biblioteca.'''
    genres = Genre.all_genres()
    return jsonify(genres)


if __name__ == '__main__':
    app.run(debug=True)
```

1. **Nombres**:
    - Identificadores significativos como `app`, `library`, `get_books`, `add_book`, `issue_book` hacen que el código sea legible.
    - Las clases importadas como `Book`, `DigitalBook`, `Library`, `Member`, `Genre` siguen un estándar claro.
2. **Objetos**:
    - Los objetos principales en ejecución incluyen `app` (instancia de Flask) y `library` (instancia de Library).
    - Se crean objetos de tipo `Book`, `DigitalBook`, y `Member` con los datos recibidos desde peticiones JSON.
3. **Entornos**:
    - El entorno es un servidor web ejecutado con Flask, que responde a peticiones HTTP.
    - El entorno de persistencia involucra archivos `library.json` y `members.json`.
4. **Bloques**:
    - Cada ruta Flask decorada con `@app.route(...)` constituye un bloque de código funcional.
    - También hay bloques para condicionales (`if`) y conversión de datos.
5. **Alcance**:
    - Variables como `library` son globales dentro del contexto de la aplicación.
    - Los datos JSON se manejan dentro del ámbito local de las funciones.
6. **Administración de memoria**:
    - Usando `memory_management`, se registran asignaciones y liberaciones de memoria asociadas a objetos creados.
    - Se llama `display_memory_usage()` al finalizar operaciones relevantes.
7. **Expresiones**:

- Comparaciones lógicas (`data.get('is_digital', False)`), conversiones de tipos (`int(data['id'])`) y estructuras de datos como listas y diccionarios son expresiones comunes.

8. **Comandos**:
   - Llamadas a métodos como `library.add_book(...)`, `library.save_members_to_file(...)` modifican el estado del sistema.
9. **Secuencia**:
   - El orden de ejecución es secuencial para cada solicitud HTTP: recibir datos, procesarlos, modificar el estado, y retornar respuesta.
10. **Selección**:
    - Uso de `if` y `else` para tomar decisiones, por ejemplo, para diferenciar si un libro es digital o no (`if is_digital`).
11. **Iteración**:
    - Se utiliza `for` para recorrer listas de libros y miembros al responder solicitudes `GET`.
12. **Recursión**:
    - No hay evidencia de recursión en este módulo.
13. **Subprogramas**:
    - Cada ruta (`@app.route`) define un subprograma que responde a un endpoint específico.
    - Subprogramas auxiliares también se ejecutan como `library.load_members_from_file(...)`.
14. **Tipos de datos**:
    - Tipos primitivos: `int`, `str`, `bool`.
    - Estructuras: `list`, `dict`.
    - Tipos definidos por usuario: `Book`, `DigitalBook`, `Member`, `Library`.

Tercer código: biblioteca_web.py

```
'''Module to manage memory usage'''


class MemoryManagement:
    '''Class to manage memory usage'''
    def __init__(self):
        self.heap_allocations = 0
        self.heap_deallocations = 0

    def increment_heap_allocations(self, size):
        '''Increment heap allocations'''
        self.heap_allocations += size
```

```
    def increment_heap_deallocations(self, size):
        '''Increment heap deallocations '''
        self.heap_deallocations += size

    def display_memory_usage(self):
        '''Display memory usage'''
        print(f"Heap allocations: {self.heap_allocations} bytes")
        print(f"Heap deallocations: {self.heap_deallocations} bytes")


memory_management = MemoryManagement()
```

1. **Nombres**:
    - Identificadores como `MemoryManagement`, `increment_heap_allocations`, `increment_heap_deallocations`, y `display_memory_usage` son autoexplicativos.
    - La instancia `memory_management` sigue la convención de nombrado en minúsculas.
2. **Objetos**:
    - Se define un objeto principal `memory_management`, instancia de la clase `MemoryManagement`, que almacena el estado de las asignaciones y liberaciones de memoria en el heap.
3. **Entornos**:
    - El código representa un entorno de monitoreo de uso de memoria (en este caso, de heap).
    - Se usa dentro de un entorno mayor que maneja operaciones de asignación de objetos (como libros o miembros).
4. **Bloques**:
    - Cada método dentro de la clase `MemoryManagement` es un bloque de código.
    - El constructor `__init__` inicializa los contadores.
5. **Alcance**:
    - Los atributos `heap_allocations` y `heap_deallocations` tienen alcance de instancia.
    - La variable `memory_management` es global si el módulo es importado.
6. **Administración de memoria**:
    - Se realiza un seguimiento manual de las asignaciones y liberaciones de memoria en el heap.
    - No hay gestión de stack, solo heap.
7. **Expresiones**:
    - Las operaciones de suma (`+=`) son expresiones utilizadas para modificar los contadores.
    - La función `print()` construye expresiones de salida.
8. **Comandos**:

- Los métodos `increment_heap_allocations` y `increment_heap_deallocations` actúan como comandos que alteran el estado de los contadores.
- `display_memory_usage` imprime el estado actual.

9. **Secuencia**:
    - Las llamadas a los métodos siguen una secuencia esperada: primero se incrementa y luego se puede mostrar el uso.
10. **Selección**:
    - No se utilizan estructuras de selección (`if`, `else`) en este módulo, pero podrían incorporarse para validaciones.
11. **Iteración**:
    - No se emplean bucles en este código.
12. **Recursión**:
    - No se utiliza recursión.
13. **Subprogramas**:
    - Se definen subprogramas/métodos: `__init__`, `increment_heap_allocations`, `increment_heap_deallocations`, `display_memory_usage`.
14. **Tipos de datos**:
    - Tipos primitivos usados: `int`.
    - Estructuras de control: clase `MemoryManagement`.
    - La instancia `memory_management` es un objeto de tipo definido por el usuario.

Link de la pagina: https://lalit0o.github.io/portafolio/