A. Core Python & Data Handling

1. Modify the project so that all data (books, members) is auto-saved every time an operation happens, without requiring manual save().

To make sure no data is lost, we call _save() automatically after each operation (like add, remove, issue, return).

Example:

```
def add_book(self, book_id: str, title: str, author: str, isbn: str):

    if book_id in self.books:

        raise ValueError("Book ID exists")

    self.books[book_id] = Book(book_id, title, author, isbn)

    self._save()

    logging.info(f"Book added: {book_id} - {title}")
```

2. Add a feature to search books by partial title or author name (case-insensitive).

First we convert both search keyword and book details to lowercase.

```
def search_books(self, keyword: str):

  keyword = keyword.lower()

  return [book for book in self.books.values()

        if keyword in book.title.lower() or keyword in book.author.lower()]
```

3. Implement sorting of books by title, author, or availability using Python’s sorted() and custom key functions.

```
def sort_books(self, by: str = "title"):
  if by == "title":
    return sorted(self.books.values(), key=lambda b: b.title)
  elif by == "author":
    return sorted(self.books.values(), key=lambda b: b.author)
  elif by == "availability":
    return sorted(self.books.values(), key=lambda b: b.available)
  else:
    raise ValueError("Invalid sort key")
```

4. Use list comprehensions to fetch all currently borrowed books.

```
borrowed = [b for b in self.books.values() if not b.available]

for b in borrowed:

    print("Borrowed:", b.title)
```

5. Add a feature to export all library data to a CSV file.

```
import csv

def export_to_csv(self, filename="library_data.csv"):

    with open(filename, "w", newline="") as f:

        writer = csv.writer(f)

        writer.writerow(["BookID", "Title", "Author", "ISBN", "Available"])

        for b in self.books.values():

            writer.writerow([b.book_id, b.title, b.author, b.isbn, b.available])
```

6. Convert the book and member collections into dictionaries of dataclasses instead of normal classes.

```
from dataclasses import dataclass

@dataclass

class Book:

    book_id: str

    title: str

    author: str

    isbn: str

    available: bool = True

@dataclass

class User:

    user_id: str

    name: str
```

7. Use zip() to pair members with the books they borrowed for a custom report.

```
def report(self):
    for u in self.users.values():
        books = [t["book_id"] for t in self.transactions
                 if t["user_id"] == u.user_id and t["return_date"] is None]
        pairs = list(zip([u.name]*len(books), books))
        print(pairs)
```

8. Write a function that uses regular expressions to validate ISBN numbers.

```
import re
def validate_isbn(self, isbn: str) -> bool:
    pattern = r"^(?:\d{10}|\d{13})$"
    return bool(re.match(pattern, isbn))
```

B. Advanced OOP Concepts

9. Introduce a StaffMember subclass with permission to remove books, while normal members cannot.

```
class StaffMember(User):
    def remove_book(self, library, book_id):
        if book_id not in library.books:
            raise LookupError("Book not found")
        if not library.books[book_id].available:
            raise RuntimeError("Book is currently issued")
        del library.books[book_id]
        library._save()
```

10. Implement operator overloading (__eq__, __lt__) so two books can be compared by ISBN.

```
@dataclass
class Book:
    book_id: str
    title: str
    author: str
    isbn: str
    available: bool = True
    def __eq__(self, other):
        return isinstance(other, Book) and self.isbn == other.isbn
    def __lt__(self, other):
        return isinstance(other, Book) and self.isbn < other.isbn
```