DAY 22: MORNING ASSESSMENT

1. Modify the project so that all data (books, members) is auto-saved every time an operation

happens, without requiring manual save().

from functools import wraps

def autosave(method):

@wraps(method)

def wrapper(self, \*args, \*\*kwargs):

result = method(self, \*args, \*\*kwargs)

self.\_save()

return result

return wrapper

Use it on all methods that mutate state.

Example:  
class LibrarySystem:

@autosave

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)

logging.info(f"Book added: {book\_id} - {title}")

we add this method to all the other methods like remove\_book(),register\_user(),issue\_book() etc.

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

def search\_books(self, query: str):

q = query.strip().lower()

return [

b for b in self.books.values()

if q in b.title.lower() or q in b.author.lower()

]

Then we can add this as a choice in the CLI:  
elif choice == "9":

q = input("Search (title/author): ")

for b in lib.search\_books(q):

print(f"[{b.book\_id}] {b.title} / {b.author} / {b.isbn}")

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

key functions.

def sorted\_books(self, by: str = "title", reverse: bool = False):

key\_funcs = {

"title": lambda b: (b.title.lower(), b.book\_id),

"author": lambda b: (b.author.lower(), b.book\_id),

"available": lambda b: (not b.available, b.title.lower()),

}

key = key\_funcs.get(by, key\_funcs["title"])

return sorted(self.books.values(), key=key, reverse=reverse)

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

def borrowed\_books(self):

return [b for b in self.books.values() if not b.available]

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

import csv

from datetime import datetime

def export\_csv(self, base\_name: str = "library\_export"):

stamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

books\_file = f"{base\_name}\_books\_{stamp}.csv"

users\_file = f"{base\_name}\_users\_{stamp}.csv"

tx\_file = f"{base\_name}\_transactions\_{stamp}.csv"

with open(books\_file, "w", newline="", encoding="utf-8") as f:

w = csv.writer(f)

w.writerow(["book\_id", "title", "author", "isbn", "available"])

for b in self.books.values():

w.writerow([b.book\_id, b.title, b.author, b.isbn, b.available])

with open(users\_file, "w", newline="", encoding="utf-8") as f:

w = csv.writer(f)

w.writerow(["user\_id", "name"])

for u in self.users.values():

w.writerow([u.user\_id, u.name])

with open(tx\_file, "w", newline="", encoding="utf-8") as f:

w = csv.writer(f)

w.writerow(["book\_id", "user\_id", "issue\_date", "due\_date", "return\_date"])

for t in self.transactions:

w.writerow([t["book\_id"], t["user\_id"], t["issue\_date"], t["due\_date"], t["return\_date"]])

logging.info(f"Exported CSVs: {books\_file}, {users\_file}, {tx\_file}")

return books\_file, users\_file, tx\_file

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

def as\_dict(self): # keep for JSON

return {

"book\_id": self.book\_id,

"title": self.title,

"author": self.author,

"isbn": self.isbn,

"available": self.available,

}

@staticmethod

def from\_dict(d: dict) -> "Book":

return Book(d["book\_id"], d["title"], d["author"], d["isbn"], d.get("available", True))

@dataclass

class User:

user\_id: str

name: str

def as\_dict(self):

return {"user\_id": self.user\_id, "name": self.name}

@staticmethod

def from\_dict(d: dict) -> "User":

return User(d["user\_id"], d["name"])

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

def current\_loans\_by\_user(self):

# user\_id -> [Book]

loans = {}

active = [t for t in self.transactions if t["return\_date"] is None]

for t in active:

loans.setdefault(t["user\_id"], []).append(self.books.get(t["book\_id"]))

return loans

def report\_user\_with\_books(self):

loans = self.current\_loans\_by\_user()

users = list(self.users.values())

borrowed = [", ".join([b.title for b in loans.get(u.user\_id, []) if b]) for u in users]

for u, titles in zip(users, borrowed):

print(f"{u.user\_id} - {u.name}: {titles or 'No active loans'}")

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

import re

ISBN10\_RE = re.compile(r"^(?:\d[\- ]?){9}[\dXx]$")

ISBN13\_RE = re.compile(r"^(?:\d[\- ]?){13}$")

def is\_valid\_isbn(isbn: str) -> bool:

s = isbn.replace("-", "").replace(" ", "")

if len(s) == 10 and ISBN10\_RE.match(isbn):

total = sum((10 - i) \* (10 if x in "Xx" else int(x)) for i, x in enumerate(s))

return total % 11 == 0

if len(s) == 13 and ISBN13\_RE.match(isbn):

total = sum((1 if i % 2 == 0 else 3) \* int(d) for i, d in enumerate(s))

return total % 10 == 0

return False

if not is\_valid\_isbn(isbn):

raise ValueError("Invalid ISBN")

9. Introduce a StaffMember subclass with permission to remove books, while normal members

cannot.

Add a role concept; only StaffMember can remove:

@dataclass

class Member(User):

pass

@dataclass

class StaffMember(User):

pass

Update your user creation logic to instantiate the right subclass.example is  
@autosave

def remove\_book(self, book\_id: str, actor\_id: str = None):

# permission check

if actor\_id:

actor = self.users.get(actor\_id)

if not isinstance(actor, StaffMember):

raise PermissionError("Only staff can remove books")

b = self.books.get(book\_id)

if not b:

raise LookupError("Book not found")

if not b.available:

raise RuntimeError("Book is currently issued")

del self.books[book\_id]

logging.info(f"Book removed by {actor\_id or 'SYSTEM'}: {book\_id}")

10. Implement operator overloading (\_\_eq\_\_, \_\_lt\_\_) so two books can be compared by ISBN.

from functools import total\_ordering

@total\_ordering

@dataclass

class Book:

book\_id: str

title: str

author: str

isbn: str

available: bool = True

def \_\_eq\_\_(self, other):

if not isinstance(other, Book):

return NotImplemented

return self.isbn == other.isbn

def \_\_lt\_\_(self, other):

if not isinstance(other, Book):

return NotImplemented

return self.isbn < other.isbn