31. Add a holiday calendar so due dates skip weekends/holidays.

import datetime

HOLIDAYS = {

datetime.date(2025, 1, 1),

datetime.date(2025, 8, 15),

}

def calculate\_due\_date(days: int) -> datetime.date:

due = datetime.date.today()

added = 0

while added < days:

due += datetime.timedelta(days=1)

if due.weekday() < 5 and due not in HOLIDAYS:

added += 1

return due

Date is moving forward by 1 day at a time using datetime.timedelta(days=1).

It counts the day if it is a weekday (Mon-Fri → weekday()<5) and not in the HOLIDAYS set and we have to stop when we have counted enough valid days.

When we are borrowing a book-we have to add-

due\_date = calculate\_due\_date(14)

print(f"Due date: {due\_date}")

32. Allow books to be reserved: if a book is borrowed, the next member can queue for it.

from collections import defaultdict, deque

class Library:

def \_\_init\_\_(self):

self.books: Dict[str, Book] = {}

self.members: Dict[str, Member] = {}

self.reservations = defaultdict(deque)

self.load()

When we are borrowing- we have to add-

if not b.available:

print("Book already issued.added to queue")

self.reservations[book\_id].append(member\_id)

return

When we are returning,we have to add:

b.available = True

m.borrowed.remove(book\_id)

self.save()

print(f"'{b.title}' returned by {m.name}.")

if self.reservations[book\_id]:

next\_member\_id = self.reservations[book\_id].popleft()

print(f"Book reserved for {self.members[next\_member\_id].name}.")

Here if a book is already issued, you let other members join a waiting list.

We use collections.deque as a queue, so the first person who asks gets the book first

33. Implement a renewal system where members can extend due dates only once.

Here every renewal is logged in self.renewalsand If they try to renew again, you reject it. We calculate a new due date the same way as borrowing.

self.renewals = {}

def renew\_book(self, member\_id: str, book\_id: str):

key = (member\_id, book\_id)

if key in self.renewals and self.renewals[key]:

print("You have already renewed this book once.")

return

if book\_id not in self.members[member\_id].borrowed:

print("You don't have this book.")

return

due\_date = calculate\_due\_date(7)

self.renewals[key] = True

print(f"Book renewed. New due date: {due\_date}")

34. Track and print a monthly report of top borrowed books.

from collections import Counter

self.borrow\_counter = Counter()

in borrow\_book():

self.borrow\_counter[book\_id] += 1

def monthly\_report(self):

print("Borrowed Books This Month are:")

for book\_id, count in self.borrow\_counter.most\_common(5):

b = self.books[book\_id]

print(f"{b.title} — borrowed {count} times")

F. Performance & Optimization

35. Use generators to lazily iterate through all books instead of storing them in memory.

Instead of storing books in a list when listing them, we return them one by one using yield.This saves memory when dealing with many books.

def iter\_books(self):

for book in self.books.values():

yield book

for b in lib.iter\_books():

print(b)

36. Profile the system using cProfile and identify bottlenecks.

->python -m cProfile -s time your\_script.py

cProfile is used to record how long each function takes.

-s time sorts output by time consumed.

this is a tool for analyzing runtime performance

37. Cache frequently accessed books using functools.lru\_cache.

If the same book is accessed repeatedly we have to store it in memory to avoid re-reading or re-computing and use functools.lru\_cache.

from functools import lru\_cache

@lru\_cache(maxsize=128)

def get\_book(self, book\_id: str):

return self.books.get(book\_id)

in borrow:

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

38. Write a function that uses multiprocessing to simulate 100 members borrowing simultaneously.

import multiprocessing

def borrow(lib, member\_id, book\_id):

lib.borrow\_book(member\_id, book\_id)

def many\_borrows(lib):

with multiprocessing.Pool(processes=10) as pool:

pool.starmap(simulate\_borrow, [(lib, f"M{i}", "A1") for i in range(1,101)])

39. Replace normal dictionaries with collections.defaultdict or OrderedDict where applicable.

defaultdict -gives a default value automatically when a key is missing.

OrderedDict- preserves insertion order

from collections import OrderedDict

self.books = OrderedDict()

self.members = OrderedDict()

40. Benchmark file vs JSON persistence performance with large data (10,000+ books).

import json, time

def benchmark():

books = [Book(str(i), f"Title{i}", "AuthorX", "ISBNX") for i in range(10000)]

start = time.time()

with open("books.txt", "w") as f:

for b in books:

f.write(b.to\_line())

print("Text save time:", time.time()-start)

start = time.time()

with open("books.json", "w") as f:

json.dump([b.\_\_dict\_\_ for b in books], f)

print("JSON save time:", time.time()-start)