**Concurrency & Validation**

**21. How can we force the system to handle simultaneous borrow attempts (simulate concurrency with threading)?**  
You can use Python’s threading module. Each borrow attempt runs in a separate thread. To prevent data corruption, protect shared resources (like the books list or file writes) with threading.Lock.

**22. How do we implement input validation so that member ID must be alphanumeric, and book ID must be unique?**

Use str.isalnum() to check member IDs.

Before adding a book, check if its ID already exists in the dictionary or file. If it does, raise a ValueError.

**23. What happens if a member tries to borrow more than 5 books at once?**  
Raise a **custom exception** (e.g., BorrowLimitExceededError). Before borrowing, count how many books the member already has. If ≥ 5, block the action.

**24. How can we implement retry logic if the file is locked when saving?**  
Use a loop with try/except and time.sleep(). For example:

import time

for attempt in range(3):

try:

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

f.write(data)

break

except PermissionError:

time.sleep(1)

else:

raise Exception("File save failed after retries")

**Persistence & File/JSON**

**25. How do we add versioning to the JSON file so each save creates a backup copy?**  
Before saving, copy the old file as library\_backup\_TIMESTAMP.json. Use shutil.copy().

**26. How do we maintain an append-only log file for all actions?**  
Open a file with with open("log.txt", "a") and write actions like:

log.write(f"{timestamp} | Member X borrowed Book Y\n")

**27. How do we implement import/export between JSON, TXT, and CSV?**

* **JSON → TXT/CSV:** Parse JSON and write rows using csv.writer or plain text.
* **TXT/CSV → JSON:** Read line by line, then dump to JSON using json.dump().

**28. How do we store the last modified timestamp of each book inside JSON?**  
When updating book data (borrow/return), add book["last\_modified"] = datetime.now().isoformat().

**29. How do we use pickle for faster serialization of the entire library state?**

import pickle

with open("library.pkl", "wb") as f:

pickle.dump(library\_state, f)

with open("library.pkl", "rb") as f:

library\_state = pickle.load(f)

**Date & Time / Business Logic**

**30. How do we implement a fine calculator with sliding scale based on late return days?**  
consider ,

1–5 days late → ₹5/day

6–10 days late → ₹10/day

10 days late → ₹20/day

def calculate\_fine(days\_late):

if days\_late <= 5:

return days\_late \* 5

elif days\_late <= 10:

return 25 + (days\_late - 5) \* 10

else:

return 75 + (days\_late - 10) \* 20