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

```
Holidays = ["2025-01-01", "2025-08-15"]
def get_due_date(self, borrow_date):
    due = borrow_date + timedelta(days=14)
    while due.weekday() >= 5 or due.strftime("%Y-%m-%d") in Holidays:
        due += timedelta(days=1)
    return due
```

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

```
from collections import deque
class Book:
    def __init__(self, book_id, title, author, isbn, available=True):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available = available
        self.reservation_queue = deque()
def reserve_book(self, book_id, user_id):
    book = self.books.get(book_id)
    if not book.available:
        book.reservation_queue.append(user_id)
        return f"Book reserved for {user_id}"
    return "Book is available"
```

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

```
def renew_book(self, book_id, user_id):
    for t in self.transactions:
        if t["book"] == book_id and t["user"] == user_id and "renewed" not in t:
            t["due_date"] = (datetime.fromisoformat(t["due_date"]) + timedelta(days=7)).isoformat()
            t["renewed"] = True
            self._save()
            return "Renewed successfully"
    return "Already renewed or no record"
```

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

```
from collections import Counter
def monthly_report(self, year, month):
    books_borrowed = [
        t["book"] for t in self.transactions
        if datetime.fromisoformat(t["date"]).year == year and
           datetime.fromisoformat(t["date"]).month == month
    ]
    counter = Counter(books_borrowed)
    return counter.most_common(5)
```

F. Performance & Optimization

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

```
def iter_books(self):
    for book in self.books.values():
        yield book
for b in library.iter_books():
    print(b.title)
```

36. Profile the system using cProfile and identify bottlenecks.

```
import cProfile
import library_system
def run_profile():
    import library_system
    cProfile.run("library_system.main()", sort="cumulative")
```

37. Cache frequently accessed books using functools.lru_cache.

```
from functools import lru_cache
@lru_cache(maxsize=50)
def get_book(self, book_id):
    return self.books.get(book_id)
```

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

```
from multiprocessing import Pool
def simulate_borrow(user_id):
    library.issue_book("B1", user_id)
def simulate():
    with Pool(10) as p:
        p.map(simulate_borrow, [f"U{i}" for i in range(100)])
```

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

```
from collections import defaultdict, OrderedDict
self.transactions = defaultdict(list)
self.books = OrderedDict()
```

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

- JSON is generally faster for bulk writes because the entire list of books is dumped at once using json.dump, whereas CSV/file storage writes each row line by line, making it a bit slower.
- JSON usually provides quicker reads since json.load can reconstruct the entire Python list in one go. CSV requires parsing line by line and type conversion (e.g., strings back to integers/booleans), which adds overhead.
- CSV/plain text often produces smaller files since it only stores values separated by commas. JSON files are slightly larger because of extra syntax like braces, quotes, and field names repeated for every object.
- JSON is more convenient when dealing with nested or structured data (objects, arrays, booleans). CSV is simpler and human-readable but works best for flat tabular data only.
- For 10,000+ records, JSON handles bulk load and save efficiently, but CSV can be better if you need streaming access (reading only a few rows without loading everything into memory).