Jupyter Notebook Execution Report

Student:	Ganesh Kumbhar
Roll Number:	21f1234567
Course:	IITM BS Degree - Data Science
Assignment:	Part A: City Library Management System
Date:	November 15, 2025

Cell 1: ■ Markdown

Part A- City Library (IITM Mini Proj)

Cell 2: ■ Code

```
# Basic book List to hold Book data
# Each book is a dict with keys: 'id', 'title', 'author', 'genre', 'available'
from typing import List, Dict, Optional
```

```
books: List[Dict] = [] # in-memory list holding all book records
```

Cell 3: ■ Code

```
# Functions to create, add, list, find, and update books
#Each book has: Book ID, Title, Author, Genre, Availability (Available/Issued).
#The system should be broken into reusable parts (for example: adding a book,
issuing
#a book, searching, etc.).
```

```
def create_book(book_id, title, author, genre, available=True) -&gt: Dict:
"""Create a simple book dict. All values are converted to basic types like
str,bool.
```

```
"""
return {
'id': str(book_id),
'title': str(title),
'author': str(author),
'genre': str(genre),
'available': bool(available)
}
#The system should allow adding new books and updating availability when
# issued/returned.
def add_book(book: Dict) -&gt: None:
"""Add a book dict to the global books list.
"""
```

```
books.append(book)
```

```
def list_books() -&gt: List[Dict]:
"""Return all stored book dicts."""
return books
```

```
def find_book_by_id(book_id) -&gt: Optional[Dict]:
"""Find a book by id and return it, or None if not found."""
for b in books:
if b['id'] == str(book_id):
return b
return None
#The system should allow adding new books and updating availability when
# issued/returned.
def set_availability(book_id, available: bool) -&gt: bool:
"""Set a book's availability. Return True if available, False if not available."""
b = find_book_by_id(book_id)
if b is None:
return False
b['available'] = bool(available)
return True
```

```
# Show all available books by author or title search.
def search_books(term: str) -&gt: List[Dict]:
"""Return books whose title or author contains the text."""
t = term.lower()
return [b for b in books if t in b['title'].lower() or t in b['author'].lower()]
#Show all available books in a given genre.
def available_by_genre(genre: str) -&gt: List[Dict]:
"""Return available books for a given genre."""
g = genre.lower()
return [b for b in books if b['available'] and b['genre'].lower() == g]
```

Cell 4: ■ Code

```
# Member records
```

```
from typing import List, Dict, Optional
```

```
members: List[Dict] = []
#Each member has: Member ID, Name, Age, Contact Info
def make_member(member_id, name, age, contact_info) -&gt: Dict:
"""Create a simple member dict. Converts fields to basic types."""
return {
'member_id': str(member_id),
'name': str(name),
'age': int(age),
'contact_info': str(contact_info)
}
```

```
def add_member(member: Dict) -&gt: None:
"""Add a member dict to the members list."""
```

```
members.append(member)
```

```
def list_members() -&gt: List[Dict]:
return members
```

```
def find_member_by_id(member_id) -&gt: Optional[Dict]:
for m in members:
if m['member_id'] == str(member_id):
return m
return None
```

Cell 5: ■ Code

```
# Add sample members M1, M2, M3
add_member(make_member('M1', 'M1member', 30, 'member1@citylibrary.com'))
add_member(make_member('M2', 'M2member', 25, 'member2@citylibrary.com'))
add_member(make_member('M3', 'M3member', 60, 'member3@citylibrary.com'))
```

```
# Show members using pandas if available
import pandas as pd
```

```
display(pd.DataFrame(list_members()))
```

Output:

member_id	name	age	contact_info
M1	M1member	30	member1@citylibrary.com
M2	M2member	25	member2@citylibrary.com
M3	M3member	60	member3@citylibrary.com

Cell 6: ■ Markdown

Now we add a short demo cell that creates and adds a few example books and prints them.

If you have pandas installed, the output will be formatted as a DataFrame; otherwise it will print a simple table.

Cell 7: ■ Code

```
# Demo: add a few books and show output (run this cell to see results)
# Add sample books
#We can use this add_book to add books
add_book(create_book('B1', 'B1 Title', 'B1 Author', 'Fantasy',True))
add_book(create_book('B2', 'B2 Title', 'B2 Author', 'Science',True))
add_book(create_book('B3', 'B3 Title', 'B3 Author', 'Friction',True))
```

Cell 8: ■ Code

```
import pandas as pd
```

```
all_books = list_books()
print('Initial List of Books')
display(pd.DataFrame(all_books))
```

```
# Example: issue a book (mark as unavailable) and show lookup
print()
print('Issuing B2...')
set_availability('B2', False)
print('\nLookup B2:')
print(find_book_by_id('B2'))
```

```
print('List after Issuing B2')
display(pd.DataFrame(all_books))
```

```
# Show available Fantasy books using helper
print()
print('Available Fantasy books:')
display(pd.DataFrame(available_by_genre('Fantasy')))
```

```
# Returning a book (mark as available) and show lookup
print('Returning book B2...')
set_availability('B2', True)
print('\nLookup B2:')
print(find_book_by_id('B2'))
```

```
#List after returning
print('List after Returning B2')
display(pd.DataFrame(all_books))
```

Output:

```
Initial List of Books
```

id	title	author	genre	available
B1	B1 Title	B1 Author	Fantasy	True
B2	B2 Title	B2 Author	Science	True
B3	B3 Title	B3 Author	Friction	True

```
Issuing B2...
Lookup B2:
{'id': 'B2', 'title': 'B2 Title', 'author': 'B2 Author', 'genre': 'Science', 'available': False}
List after Issuing B2
```

id	title	author	genre	available
B1	B1 Title	B1 Author	Fantasy	True

id	title	author	genre	available
B2	B2 Title	B2 Author	Science	False
B3	B3 Title	B3 Author	Friction	True

```
Available Fantasy books:
```

id	title	author	genre	available
B1	B1 Title	B1 Author	Fantasy	True

```
Returning book B2...
Lookup B2:
{'id': 'B2', 'title': 'B2 Title', 'author': 'B2 Author', 'genre': 'Science', 'available': True}
List after Returning B2
```

id	title	author	genre	available
B1	B1 Title	B1 Author	Fantasy	True
B2	B2 Title	B2 Author	Science	True
B3	B3 Title	B3 Author	Friction	True

Cell 9: 🟦 Code

```
# Borrow logic
```

```
borrow_log = [] # list of dicts {member_id, book_id, action, timestamp}
from datetime import datetime
```

```
member_holdings = {} # member_id -&gt; list of book ids
```

```
#A member can borrow multiple books, but only if the book is available.
#Include clear conditions (e.g., prevent issuing a book if already borrowed).
def borrow_book(member_id, book_id):
"""Attempt to borrow a book for a member.
Checks availability first; if available, marks book unavailable and records the
borrow.
Returns True if borrow succeeded, False otherwise.
"""
m = find_member_by_id(member_id)
if m is None:
print(f"Member {member_id} not found")
return False
b = find_book_by_id(book_id)
if b is None:
print(f"Book {book_id} not found")
return False
if not b['available']:
print(f"Book {book_id} is not available")
return False
# Record the transaction in a borrow log.
```

```
set_availability(book_id, False)
member_holdings.setdefault(member_id, []).append(book_id)
borrow_log.append({
'member_id': member_id,
'book_id': book_id,
'action': 'borrow',
'time': datetime.now().isoformat()
})
print(f"{member_id} successfully borrowed {book_id}")
return True
```

```
#Update book availability again when returning
#Record the transaction in a borrow log.
```

```
def return_book(member_id, book_id):
m = find_member_by_id(member_id)
if m is None:
print(f"Member {member_id} not found")
return False
if book_id not in member_holdings.get(member_id, []):
print(f"Member {member_id} does not hold book {book_id}")
return False
set_availability(book_id, True)
member_holdings[member_id].remove(book_id)
borrow_log.append({
'member_id': member_id,
'book_id': book_id,
'action': 'return',
'time': datetime.now().isoformat()
})
print(f"{member_id} returned {book_id}")
return True
```

```
# Borrow B1 and B2 for M1, checking availability
borrow_book('M1', 'B1')
borrow_book('M1', 'B2')
```

```
# Show available books
print('\nAvailable books:')
```

```
import pandas as pd
```

```
avail = [b for b in list_books() if b['available']]
display(pd.DataFrame(avail))
```

```
# Show members holding books
print('\nMembers holding books:')
for m_id, holdings in member_holdings.items():
print(m_id, holdings)
```

```
# Show borrow log
print('\nBorrow log:')
```

```
for entry in borrow_log:
print(entry)
```

Output:

```
M1 successfully borrowed B1
M1 successfully borrowed B2
Available books:
```

id	title	author	genre	available
B3	B3 Title	B3 Author	Friction	True

```
Members holding books:
M1 ['B1', 'B2']
Borrow log:
{'member_id': 'M1', 'book_id': 'B1', 'action': 'borrow', 'time': '2025-11-15T00:50:16.780675'}
{'member_id': 'M1', 'book_id': 'B2', 'action': 'borrow', 'time': '2025-11-15T00:50:16.780685'}
```

Cell 10: ■ Code

```
# M2 tries to borrow B1 and B3
print("M2 tries to borrow B1:")
borrow_book('M2', 'B1') # Should show not available if B1 already borrowed
print("M2 tries to borrow B3:")
borrow_book('M2', 'B3') # Should succeed if B3 is available
```

```
# Final member holdings and available books
print('\nFinal members holding books:')
for m_id, holdings in member_holdings.items():
print(f"Member {m_id}: {holdings}")
```

```
print('\nAvailable books:')
import pandas as pd
```

```
avail = [b for b in list_books() if b['available']]
display(pd.DataFrame(avail))
```

```
#List of members who have borrowed books.
print('\nFinal borrow log:')
for entry in borrow_log:
print(entry)
```

Output:

```
M2 tries to borrow B1:
Book B1 is not available
M2 tries to borrow B3:
M2 successfully borrowed B3
Final members holding books:
Member M1: ['B1', 'B2']
Member M2: ['B3']
Available books:
```

```
0__

Final borrow log:
{'member_id': 'M1', 'book_id': 'B1', 'action': 'borrow', 'time': '2025-11-15T00:50:16.780675'}
{'member_id': 'M1', 'book_id': 'B2', 'action': 'borrow', 'time': '2025-11-15T00:50:16.780685'}
{'member_id': 'M2', 'book_id': 'B3', 'action': 'borrow', 'time': '2025-11-15T00:50:16.781162'}
```

Cell 11: ■ Code

```
# Search by Title function
def search_by_title(title_term: str):
t = title_term.lower()
return [b for b in list_books() if t in b['title'].lower()]
```

```
# Demo: search by title
print("Search results for title term 'B1':")
results = search_by_title('B1')
```

```
import pandas as pd
```

```
display(pd.DataFrame(results))
```

Output:

```
Search results for title term 'B1':
```

id	title	author	genre	available
B1	B1 Title	B1 Author	Fantasy	False

Cell 12: ■ Code

```
# Search by Author
def search_by_author(author_term: str):
t = author_term.lower()
return [b for b in list_books() if t in b['author'].lower()]
```

```
# Demo: search by author
print("Search results for author term 'Author':")
results = search_by_author('B3 Author')
```

```
import pandas as pd
```

```
display(pd.DataFrame(results))
```

Output:

```
Search results for author term 'Author':
```

id	title	author	genre	available
B3	B3 Title	B3 Author	Friction	False

Cell 13: ■ Code

```
# Add B4 (genre: Friction), let M3 borrow it, so we have multiple books in Friction
genre and as
# M3 is borrowing it we will have list of popular genres
# Add new book B4
#Display the most popular genre (based on issued books).
add_book(create_book('B4', 'B4 Title', 'B4 Author', 'Friction', True))
print("Added B4. Current books:")
import pandas as pd
```

```
display(pd.DataFrame(list_books()))
```

```
# Let M3 borrow B4
print('\nM3 borrowing B4:')
borrow_book('M3', 'B4')
```

```
# calculate genre counts based on borrow log
from collections import Counter
borrow_genre_counts = Counter()
for entry in borrow_log:
if entry.get('action') == 'borrow':
bid = entry.get('book_id')
book = find_book_by_id(bid)
if book:
borrow_genre_counts[book['genre']] += 1
```

```
# Also calculate currently issued books per genre (books with available == False)
issued_genre_counts = Counter()
for b in list_books():
if not b['available']:
issued_genre_counts[b['genre']] += 1
```

```
print('\nBorrow counts by genre (historical):')
for g, c in borrow_genre_counts.items():
print(f"{g}: {c}")
```

```
print('\nCurrently issued books by genre:')
for g, c in issued_genre_counts.items():
print(f"{g}: {c}")
```

```
# Show most popular genre by historical borrow count
if borrow_genre_counts:
most_popular = borrow_genre_counts.most_common(1)[0]
print(f"\nMost popular genre (by borrows): {most_popular[0]} with {most_popular[1]}
borrows")
else:
print('\nNo borrow records yet.')
```

```
df_hist = pd.DataFrame(list(borrow_genre_counts.items()),
columns=['genre','borrow_count'])
df_issued = pd.DataFrame(list(issued_genre_counts.items()),
columns=['genre','issued_count'])
print('\nHistorical borrow counts:')
display(df_hist)
print('\nCurrently issued counts:')
```

```
display(df_issued)
```

Output:

```
Added B4. Current books:
```

id	title	author	genre	available
B1	B1 Title	B1 Author	Fantasy	False
B2	B2 Title	B2 Author	Science	False
B3	B3 Title	B3 Author	Friction	False
B4	B4 Title	B4 Author	Friction	True

```
M3 borrowing B4:
M3 successfully borrowed B4
Borrow counts by genre (historical):
Fantasy: 1
Science: 1
Friction: 2
Currently issued books by genre:
Fantasy: 1
Science: 1
Friction: 2
Most popular genre (by borrows): Friction with 2 borrows
```