Day-23 Morning Assessment

21. import threading

  class Library:

     def \_\_init\_\_(self):

         self.books = {"B001": 1}

         self.\_lock = threading.Lock()

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

         with self.\_lock:

             if self.books.get(book\_id, 0) > 0:

                 self.books[book\_id] -= 1

                 print(f"{member} borrowed {book\_id}")

             else:

                 print(f"{member} could not borrow {book\_id} (unavailable)")

lib = Library()

def tryBorrow(member):

lib.borrow(member, "B001")

threads = [threading.Thread(target=tryBorrow, args=(m,)) for m in ["Alice","Bob"]]

[t.start() for t in threads]

[t.join() for t in threads]

22. class Library:

     def \_\_init\_\_(self):

         self.books = {}

         self.members = {}

     def registerMember(self, member\_id):

         if not member\_id.isalnum():

             raise ValueError("Member ID must be alphanumeric")

         self.members[member\_id] = []

     def addBook(self, book\_id, title):

         if book\_id in self.books:

             raise ValueError("Book ID must be unique")

         self.books[book\_id] = title

lib = Library()

lib.registerMember("A123")

lib.addBook("B001","Clean Code")

23. class MaxBorrowError(Exception): pass

  class Library:

     def borrow\_books(self, member, book\_ids):

         if len(book\_ids) > 5:

             raise MaxBorrowError("Cannot borrow more than 5 books at once")

         print(f"{member} borrowed {book\_ids}")

lib = Library()

lib.borrow\_books("Alice", ["B1","B2","B3","B4","B5"])

lib.borrow\_books("Alice", ["B1","B2","B3","B4","B5","B6"])

24. import time, json

def saveRetry(filename, data, retries=5):

     for attempt in range(retries):

         try:

             with open(filename, "w") as f:

                 json.dump(data, f)

             print("Saved successfully")

             return

         except PermissionError:

             wait = 0.1 \* (2\*\*attempt)

             print(f"File locked. Retrying in {wait}s...")

             time.sleep(wait)

     raise Exception("Failed to save after retries")

 saveRetry("library.json", {"books": ["B1","B2"]})

25. import shutil, json

from datetime import datetime

from pathlib import Path

def saveBackup(path, data):

     path = Path(path)

     if path.exists():

ts = [datetime.now](http://datetime.now)().strftime("%Y%m%d-%H%M%S")

         backup = path.with\_name(f"{path.stem}.{ts}.bak.json")

         shutil.copy2(path, backup)

     with open(path, "w") as f:

         json.dump(data, f, indent=2)

saveBackup("library.json", {"books":["B1"]})

26. import json, datetime

  def logAction(action, details):

     entry = {"time": datetime.datetime.utcnow().isoformat(),

              "action": action,

              "details": details}

     with open("actions.log","a") as f:

         f.write(json.dumps(entry) + "\n")

logAction("add\_book", {"book\_id":"B001"})

logAction("borrow", {"member":"Alice","book":"B001"})

27. import json, csv

def export\_txt(books):

     with open("books.txt","w") as f:

         for bid,title in books.items():

             f.write(f"{bid}|{title}\n")

def export\_csv(books):

     with open("books.csv","w",newline="") as f:

         w=csv.writer(f); w.writerow(["id","title"])

         for bid,title in books.items(): w.writerow([bid,title])

def import\_json(fname):

     with open(fname) as f: return json.load(f)

books = {"B001":"Clean Code","B002":"Pragmatic Programmer"}

export\_txt(books); export\_csv(books)

28. import json, datetime

books = {

     "B001": {"title":"Clean Code","last\_modified":datetime.datetime.utcnow().isoformat()}

}

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

     json.dump(books, f, indent=2)

29. import pickle

libraryState = {"books":{"B1":"Clean Code"}, "members":["Alice","Bob"]}

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

     pickle.dump(libraryState,f)

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

     loaded = pickle.load(f)

  print(loaded)

30. def calculate\_fine(days\_late):

     if days\_late <= 0: return 0

     if days\_late <= 7: return days\_late \* 5

     if days\_late <= 30: return (7\*5) + (days\_late-7)\*10

     return (7\*5) + (23\*10) + (days\_late-30)\*20 + 100

print(calculate\_fine(3))

print(calculate\_fine(10))

print(calculate\_fine(40))