Day-23 Evening Assessment

from datetime import date, timedelta

HOLIDAYS = {date(2025, 1, 1), date(2025, 8, 15)}

def next\_business\_day(start: date, days: int, holidays=HOLIDAYS) -> date:

     """Advance 'days' business days from 'start', skipping weekends & holidays."""

     d = start

     added = 0

     while added < days:

         d += timedelta(days=1)

         if d.weekday() < 5 and d not in holidays:  # 0-4 = Mon-Fri

             added += 1

     return d

32. from collections import deque, defaultdict

class ReservableLibrary:

     def \_\_init\_\_(self):

         self.available = {"B001": 1}

         self.queue = defaultdict(deque)

[self.loans](http://self.loans) = defaultdict(list)

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

         if self.available.get(book\_id, 0) > 0 and not self.queue[book\_id]:

             self.available[book\_id] -= 1

[self.loans](http://self.loans)[member\_id].append(book\_id)

             return f"{member\_id} borrowed {book\_id} immediately"

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

         return f"{member\_id} queued for {book\_id}"

     def return\_book(self, member\_id, book\_id):

[self.loans](http://self.loans)[member\_id].remove(book\_id)

         if self.queue[book\_id]:

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

[self.loans](http://self.loans)[next\_member].append(book\_id)

             return f"{book\_id} assigned to next in queue: {next\_member}"

         self.available[book\_id] = self.available.get(book\_id, 0) + 1

         return f"{book\_id} returned; copies now {self.available[book\_id]}"

33. from datetime import date, timedelta

class RenewalPolicy:

     def \_\_init\_\_(self):

         self.due\_dates = {}

         self.renewals = {}

    def set\_due(self, member, book, due):

         self.due\_dates[(member, book)] = due

         self.renewals[(member, book)] = 0

     def renew\_once(self, member, book, extension\_days=14):

         key = (member, book)

         if self.renewals.get(key, 0) >= 1:

             raise RuntimeError("Renewal limit reached")

         self.due\_dates[key] = self.due\_dates[key] + timedelta(days=extension\_days)

         self.renewals[key] += 1

         return self.due\_dates[key]

34. from collections import Counter, defaultdict

from datetime import datetime

class Stats:

     def \_\_init\_\_(self):

         self.borrow\_log = []  # list of (book\_id, timestamp)

     def log\_borrow(self, book\_id, ts=None):

         self.borrow\_log.append((book\_id, ts or datetime.utcnow()))

     def top\_books\_by\_month(self, year: int, month: int, top\_n=5):

         c = Counter(

             book\_id for book\_id, ts in self.borrow\_log

             if ts.year == year and ts.month == month

         )

         return c.most\_common(top\_n)

     def print\_monthly\_report(self, year: int, month: int, top\_n=5):

         print(f"Top borrowed books for {year}-{month:02d}")

         for rank, (bid, count) in enumerate(self.top\_books\_by\_month(year, month, top\_n), 1):

             print(f"{rank}. {bid} — {count} borrows")

35. import json

from typing import Iterator

def iter\_books\_jsonlines(path: str) -> Iterator[dict]:

     """Each line in file is a JSON object for one book (JSONL).

     This yields one book at a time without loading entire file."""

     with open(path, "r", encoding="utf-8") as f:

         for line in f:

             if line.strip():

                 yield json.loads(line)

36. import cProfile, pstats, io

def heavy\_work():

     s = 0

     for i in range(1\_000\_00):

         s += (i \* i) % 97

     return s

def profile\_func(func):

     pr = cProfile.Profile()

     pr.enable()

     func()

     pr.disable()

     s = io.StringIO()

     pstats.Stats(pr, stream=s).sort\_stats("cumtime").print\_stats(20)

     print(s.getvalue())

37. from functools import lru\_cache

BOOK\_DB = {

     "B001": {"title": "Clean Code"},

     "B002": {"title": "Pragmatic Programmer"},

}

@lru\_cache(maxsize=1024)

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

return BOOK\_DB.get(book\_id)

38. from multiprocessing import Process, Manager, Lock

def borrow\_worker(member\_id, book\_id, available, lock, results):

     with lock:

         if available[book\_id] > 0:

             available[book\_id] -= 1

             results[member\_id] = "success"

         else:

             results[member\_id] = "unavailable"

if \_\_name\_\_ == "\_\_main\_\_":

     manager = Manager()

     available = manager.dict({"B001": 10})  # 10 copies

     results = manager.dict()

     lock = Lock()

procs = []

     for i in range(100):

         p = Process(target=borrow\_worker, args=(f"M{i:03d}", "B001", available, lock, results))

         procs.append(p); p.start()

     for p in procs: p.join()

39. from collections import defaultdict, OrderedDict

loans\_by\_member = defaultdict(list)

loans\_by\_member["A123"].append("B001")

loans\_by\_member["A123"].append("B002")

catalog = OrderedDict()

catalog["B001"] = "Clean Code"

catalog["B002"] = "Pragmatic Programmer"

40. import json, pickle, random, string, time

from pathlib import Path

def fake\_book(i):

     title = "".join(random.choices(string.ascii\_letters, k=24))

     return {"id": f"B{i:05d}", "title": title, "author": "Anon", "copies": 1}

N = 12\_000

books = [fake\_book(i) for i in range(N)]

t0 = time.perf\_counter()

Path("books.json").write\_text(json.dumps(books), encoding="utf-8")

json\_write = time.perf\_counter() - t0

t1 = time.perf\_counter()

\_ = json.loads(Path("books.json").read\_text(encoding="utf-8"))

json\_read = time.perf\_counter() - t1

t2 = time.perf\_counter()

with open("books.pkl","wb") as f: pickle.dump(books, f, protocol=pickle.HIGHEST\_PROTOCOL)

pkl\_write = time.perf\_counter() - t2

t3 = time.perf\_counter()

with open("books.pkl","rb") as f: \_ = pickle.load(f)

pkl\_read = time.perf\_counter() - t3

print({

     "json\_write\_s": round(json\_write, 4),

     "json\_read\_s": round(json\_read, 4),

     "pickle\_write\_s": round(pkl\_write, 4),

     "pickle\_read\_s": round(pkl\_read, 4),

})