"""

AI-Driven Study Schedule Generator (MVP)

---------------------------------------

Generates a personalized study plan using weighted priorities based on

subject difficulty, mastery, and importance.

Outputs:

• Console summary

• schedule.csv (calendar importable)

• schedule.json (structured schedule)

"""

import json, csv, math

from datetime import datetime, timedelta

# -----------------------------------------------------------

# 1. Helper functions

# -----------------------------------------------------------

def daterange(start, end):

"""Yield each date between start and end (inclusive)."""

for n in range((end - start).days + 1):

yield start + timedelta(days=n)

# -----------------------------------------------------------

# 2. Core scheduling logic

# -----------------------------------------------------------

def compute\_priorities(topics):

"""Compute normalized priority score for each topic."""

for t in topics:

t["priority"] = (1 - t.get("mastery", 0)) \* t.get("difficulty", 1) \* t.get("importance", 1)

total = sum(t["priority"] for t in topics)

for t in topics:

t["priority"] = t["priority"] / total if total else 1 / len(topics)

return topics

def allocate\_time(topics, total\_minutes):

"""Allocate total minutes across topics proportionally."""

for t in topics:

t["allocated\_minutes"] = max(30, round(t["priority"] \* total\_minutes))

return topics

def plan\_schedule(user):

start = datetime.strptime(user["start\_date"], "%Y-%m-%d").date()

end = datetime.strptime(user["deadline"], "%Y-%m-%d").date()

weekdays = user["daily\_hours\_by\_weekday"] # {0..6: hours}

topics = compute\_priorities(user["topics"])

# Total available minutes

total\_minutes = 0

minutes\_per\_day = {}

for d in daterange(start, end):

hrs = weekdays.get(d.weekday(), 0)

minutes\_per\_day[d] = int(hrs \* 60)

total\_minutes += minutes\_per\_day[d]

if total\_minutes <= 0:

raise ValueError("No study hours defined for this date range.")

topics = allocate\_time(topics, total\_minutes)

# Assign sessions greedily by priority

schedule = []

topics.sort(key=lambda x: x["priority"], reverse=True)

for t in topics:

remaining = t["allocated\_minutes"]

for d in daterange(start, end):

if remaining <= 0:

break

avail = minutes\_per\_day[d]

if avail < 30:

continue

chunk = min(90, remaining, avail) # up to 1.5h per block

chunk = math.floor(chunk / 5) \* 5

schedule.append({

"date": d.isoformat(),

"subject": t["subject"],

"topic": t["topic"],

"minutes": chunk,

"session\_type": "learn"

})

minutes\_per\_day[d] -= chunk

remaining -= chunk

# Spaced repetition (1, 3, 7 days later)

review\_offsets = [1, 3, 7]

first\_dates = {}

for s in schedule:

key = (s["subject"], s["topic"])

if key not in first\_dates:

first\_dates[key] = datetime.strptime(s["date"], "%Y-%m-%d").date()

for (sub, top), first in first\_dates.items():

for off in review\_offsets:

rd = first + timedelta(days=off)

if rd > end:

continue

schedule.append({

"date": rd.isoformat(),

"subject": sub,

"topic": top,

"minutes": 20,

"session\_type": f"review+{off}d"

})

schedule.sort(key=lambda x: (x["date"], x["subject"]))

return schedule

# -----------------------------------------------------------

# 3. Export functions

# -----------------------------------------------------------

def save\_schedule(schedule):

with open("schedule.csv", "w", newline="", encoding="utf-8") as f:

writer = csv.DictWriter(f, fieldnames=["date","subject","topic","minutes","session\_type"])

writer.writeheader()

writer.writerows(schedule)

with open("schedule.json", "w", encoding="utf-8") as jf:

json.dump(schedule, jf, indent=2)

print("✅ Files saved: schedule.csv, schedule.json")

# -----------------------------------------------------------

# 4. Example user input

# -----------------------------------------------------------

user\_input = {

"start\_date": "2025-10-16",

"deadline": "2025-12-10",

"daily\_hours\_by\_weekday": {

0: 2.5, # Monday

1: 2.5,

2: 2.5,

3: 2.5,

4: 2.5,

5: 4.0, # Saturday

6: 0.0 # Sunday off

},

"topics": [

{"subject": "Math", "topic": "Algebra", "difficulty": 4, "mastery": 0.3, "importance": 1.0},

{"subject": "Math", "topic": "Calculus", "difficulty": 5, "mastery": 0.2, "importance": 1.0},

{"subject": "Physics", "topic": "Mechanics", "difficulty": 4, "mastery": 0.5, "importance": 0.9},

{"subject": "Chemistry", "topic": "Organic", "difficulty": 3, "mastery": 0.6, "importance": 0.8},

{"subject": "Chemistry", "topic": "Inorganic", "difficulty": 2, "mastery": 0.7, "importance": 0.6},

]

}

# -----------------------------------------------------------

# 5. Run generator

# -----------------------------------------------------------

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

schedule = plan\_schedule(user\_input)

print(f"Generated {len(schedule)} study sessions.\nSample:")

for s in schedule[:10]:

print(f"{s['date']}: {s['subject']} - {s['topic']} ({s['minutes']} min, {s['session\_type']})")

save\_schedule(schedule)