מבחן: בלאקג'ק (Blackjack)

מטרה

המשחק משוחק עם 2 שחקנים - השחקן והדילר. בכל תור אפשר לבחור אם לעצור או לשלוף עוד קלף.
המטרה היא לשלוף קלף אחד כל פעם ולהוסיף אותו ליד, כל עוד הערך של כלל הקלפים ביד לא עובר את 21.
אם הערך עובר את 21 - נפסלים. אם שני השחקנים בוחרים לעצור לפני שהם נפסלו - משווים את ערכי הקלפים ביד, והערך שיותר קרוב ל21 מנצח. אם הערכים שווים - תיקו.

ערכי קלפים

- 2–10: הערך המספרי.
- J / Q / K: הערך 10.
- A (אס): הערך **1 בלבד (ניתן להרחיב כמשימת בונוס, ראו בהמשך)**.

FLOW של המשחק

1. **בניית חפיסה** של 52 קלפים (4 סוגים × 13 דרגות).
2. **ערבוב** החפיסה לפי אלגוריתם ה־suit (מתואר בהמשך).
3. **חלוקה ראשונית**: לכל אחד (שחקן, דילר) שני קלפים מראש החפיסה. מחשבים ומדפיסים את ערכי הידיים.
4. **תור שחקן**:
   - שואלים מה השחקן מעוניין לעשות - H או S.
   - H = HIT, להמשיך, לשלוף קלף נוסף ולהוסיף אותו ליד.
   - S = STAND, לעצור, להעביר את התור לדילר.

   - אם השחקן בחר H:
     i. שולפים קלף מראש החפיסה ומוסיפים אותו ליד של השחקן
     ii. מחשבים את ערך היד
       1. אם הערך קטן מ21: חוזרים לשלב 4 (שואלים את השחקן מה הוא רוצה לעשות - H או S)
       2. אם הערך גדול מ21 - השחקן הפסיד והמשחק נגמר.
   - אם השחקן בחר S: מתחיל התור של הדילר, שנמשך עד שהוא נפסל או עד שהוא בוחר S, ואז המשחק נגמר.
5. **תור דילר**: הדילר מושך קלפים עד שערך ידו ≤ 17. אם עבר 21 → הפסד מיידי. ברגע שערך היד עובר את 17 ואין פסילה - הדילר אוטומטית עוצר והמשחק עובר לחישוב הכרעה.

The dealer can not stop after he stated - no decisions??

6. **הכרעה**: אם שני הצדדים עצרו ולא נפסלו – משווים את הסכומים של הידיים של הדילר והשחקן. הקרוב ביותר ל־21 מנצח. שוויון → תיקו.

7. הדפסת תוצאה סופית.

השחקן והדילר ממומשים כDICT שיש בתוכו HAND, שמאותחל לרשימה ריקה (ראו בהמשך איפה בדיוק האיתחול הזה קורה):

```
player = {"hand": [ ] }

dealer = {"hand": [ ] }
```

מבנה תיקיות (חובה)

```
core/
  deck.py        ← בניית חפיסה ואלגוריתם הערבוב
  game_logic.py  ← מהלך הסיבוב, חישובי יד, דילר, הכרעה
  player_io.py   ← איסוף החלטת שחקן
main.py          ← נקודת כניסה והרצה
readme.md
```

הרצה:

```
python main.py
```

פורמט README

קובץ readme.md חייב להתחיל כך (כל פריט בשורה נפרדת) באנגלית:

שם פרטי

שם משפחה

כיתה

ת"ז

אלגוריתם הערבוב

1. מבצעים 5000 החלפות.
2. בכל איטרציה:
   - בוחרים אינדקס ראשון i אקראי.
   - בודקים איזה קלף נמצא בחפיסה במיקום i.
   - את האינדקס השני j בוחרים באקראי אבל מלבד החובה שהוא לא יהיה שווה ל i , הוא חייב לעמוד בתנאי נוסף, לפי סוג הקלף שנמצא באינדקס i :
     - H: האינדקס j חייב להתחלק ב־5.
     - C: אינדקס j חייב להתחלק ב־3.
     - D: אינדקס j חייב להיות זוגי .
     - S: אינדקס j חייב להתחלק ב7.
     - אם i == j או ש j לא עומד בתנאי של סוג הקלף - יש להגריל שוב עד שיוצא j מתאים. **לא עוברים לאיטרציה הבאה לפני שיש i וגם j תקינים ומתבצעת החלפה בין שני קלפים!**

הערבוב מתבצע על אותה רשימה (in-place). אין ליצור חפיסה חדשה.

פונקציות חובה (7)

1 core/deck.py — build_standard_deck

```
def build_standard_deck() -> list[dict]:
```

- בונה חפיסה של 52 קלפים (ללא כפילויות).
- מחזירה רשימת קלפים, כל קלף במבנה: {rank": str, “suite”:str"}

---

2 core/deck.py — shuffle_by_suit

```
def shuffle_by_suit(deck: list[dict], swaps: int = 5000) -> list[dict]:
```

- מערבבת את החפיסה לפי הכללים שלמעלה.
- הערבוב מתבצע על אותה רשימה (in-place).
- מחזירה את אותה רשימה לאחר ערבוב.

3️⃣ core/game_logic.py — calculate_hand_value

```
def calculate_hand_value(hand: list[dict]) -> int:
```

- מחשבת את ערך היד לפי הכללים:
  2–10 = הערך המספרי, J/Q/K = 10, A = 1.
- מחזירה מספר שלם.

בונוס (לא חובה):
הרחב כך ש־A יכול להיות 1 או 11 לפי הצורך, מבלי לעבור 21.

4️⃣ core/game_logic.py — deal_two_each

```
def deal_two_each(deck: list[dict], player: dict, dealer: dict) -> None:
```

- שולפת 2 קלפים לשחקן ו־2 לדילר מראש החפיסה.
- מוסיפה ל־hand ומסירה מהחפיסה.
- מדפיסה ערכי הידיים ההתחלתיים (כדאי להשתמש בcalculate_hand_value)

5️⃣ core/game_logic.py — dealer_play

```
def dealer_play(deck: list[dict], dealer: dict) -> bool:
```

- מושך קלפים עד שערך היד ≥ 17.
- אם ערך הקלפים עבר את 21 - הדפס שיש פסילה והחזר False
- אם הערך עבר את 17 ולא את 21 - הדילר מסיים לשחק, החזר True

6️⃣ core/player_io.py — ask_player_action

```
def ask_player_action() -> str:
```

- מבקש קלט מהמשתמש (H או S),
- בודק חוקיות הקלט, חוזר על הבקשה אם לא חוקי.
- מחזיר "H" או "S" בלבד. אותיות גדולות.

7️⃣ core/game_logic.py — run_full_game

```
def run_full_game(deck: list[dict], player: dict, dealer: dict) -> None:
```

מנהל את כל המשחק:

1. חלוקה ראשונית (deal_two_each).

2. תור השחקן: בקשת פעולה (ask_player_action)
   - כל עוד השחקן בוחר H:
     i. שליפת קלף מהחפיסה והוספתה ליד של השחקן
     ii. חישוב היד של השחקן (calculate_hand_value)
     iii. בדיקה האם הערך של היד של השחקן עבר את 21:
        1. אם עבר - פסילה. הדפסה בהתאם ועצירת המשחק.
        2. אם לא עבר - חזרה לסעיף 2.
   - אם השחקן בוחר S: הרצת תור הדילר (dealer_play)
   - הדילר משחק עד שהוא נפסל או עוצר
   - אם אין פסילה - חישוב ערך היד של השחקן ושל הדילר, והכרזה על המנצח, או על תיקו. הדפסת תוצאות

---

main.py

- בתוך התנאי:

```
if __name__ == "__main__":
```

  - יוצר חפיסה (build_standard_deck).
  - מערבב (shuffle_by_suit).
  - יוצר:

```
player = {"hand": [ ] }

dealer = {"hand": [ ] }
```

  - מריץ:

```
run_full_game(deck, player, dealer)
```

דגשים:

- האס = 1 (ברירת מחדל). בונוס: מימוש חישוב דינמי של אס כ־11 לפי הצורך.
- שמרו על חתימות הפונקציות בדיוק.
- ודאו שאין קריאות מיותרות בקוד. בשום קובץ מלבד MAIN לא צריכות להיות קריאות לפונקציה שאינן בתוך פונקציה אחרת.
- מותר להוסיף הדפסות ככל שתרצו - רק ודאו שההדפסות שנשארות בהגשה שלכם הן הדפסות בעלות משמעות, ולא הדפסות של בדיקות ששכחתם בקוד. הדפסות מיותרות מורידות ניקוד!
- הרצה: python main.py מתיקיית השורש של הREPO.
- מותר להוסיף פונקציות וקבצים ככל שתצטרכו, כל עוד אתם מוודאים שיש את פונקציות החובה ואת מבנה החובה של התיקיות ושאתם שומרים על סדר ועל עקרונות קוד נקי.

טיפים:

- תכננו את העבודה שלכם. צרו תרשים זרימה של כלל האפליקציה. על כל חלק שאתם עובדים - כתבו לכם בעברית או בפסאודו קוד איך האלגוריתם עובד. רק בסוף גשו לכתוב את הקוד.
- ודאו את עצמכם. כתבו מעט קוד והריצו בדיקות ורק אחרי שהבדיקות עוברות - עברו לכתוב עוד קוד. יותר קל למצוא בעיה בקוד אם כתבתם 5 שורות מאשר 50.
- עבדו על דבר אחד כל פעם. הגדירו לעצמכם בבירור על מה אתם עובדים. אל תניחו לעצמכם להיסחף לנושאים אחרים.
- תכננו ונהלו את הזמן שלכם. אל תגלו שהשקעתם שלוש שעות במשהו שהיה אמור לקחת 10 דקות - וכתוצאה מכך אתם מגישים מבחן ריק.