

מבחן משחק מלחמה

1. במבחן הזה נבנה משחק שמדמה את משחק הקלפים "מלחמה".
2. למי שלא מכיר, כך הולך המשחק:
 - a. משחקים עם חפיסת קלפים עם מספרים בין 10-2 ועם נסיך (J) מלכה (Q) מלך (K) ואס (A). זוהי "הדרגה" (rank) של הקלף.
 - b. מלבד הערך המספרי, לכל קלף יש גם "סוג" (suite) - לב (heart/h), תלתן (club/c), יהלום (diamond/d), עלה (spade/s). כך שיש 4 קלפים מכל דרגה בחפיסה, ו-52 קלפים בסך הכל.
 - c. לכל קלף יש ערך (value). ערכו של כל קלף מספרי שווה לדרגה שלו (2=2, 3=3 וכך הלאה), נסיך = 11, מלכה = 12, מלך = 13 ואס = 14.
 - d. מערבבים את החפיסה ומחלקים את כל הקלפים בין שני מתמודדים. כל הקלפים הפוכים ומסתירים את הערך שלהם. זוהי ערימת המשחק של השחקן - "היד" שלו (hand).
 - e. בכל סיבוב, כל שחקן שולף קלף אחד מראש הערימה האישית שלו ("היד"), והופך אותו - ומשווה לקלף שהיריב שלף.
 - f. השחקן עם הקלף הגבוה יותר - לוקח את שני הקלפים ששוחקו ושם אותם בערימה נפרדת - "ערימת הרווח" (won_pile).
 - g. אם הקלפים זהים, זהו מצב של "מלחמה", שבמבחן הזה נממש שאם הקלפים זהים - לא קורה כלום, הקלפים יוצאים מהחפיסה של השחקן ולא נכנסים לשום ערימת רווח, ומשחקים את הקלף הבא בתור.
 - אפשר לממש את הצורה האמיתית של המשחק כמשימת בונוס - ראו פירוט בסוף המבחן.
 - h. כשנגמרים הקלפים ב"יד" של השחקנים - משווים בין כמות הקלפים בערימות הרווח של כל שחקן. השחקן עם יותר קלפים בערימה שלו - מנצח. אם המספר שווה - תיקו (אפשר לממש מנגנון בדיקה נוסף, ראו משימות בונוס).
3. אתם נדרשים לכתוב פונקציות בקבצים שונים, עם ייבוא נכון ביניהן.
4. שימו לב:
 - a. מבנה התיקיות שמתואר כאן **חייב** להיות זהה.
 - b. שמות הפונקציות והמשתנים **חייבים** להיות זהים.
 - c. אתם חייבים להגיש את הקבצים **עם כל הפונקציות שנדרשות**. אם לא הספקתם לממש אותן - כתבו רק את החתימה ואת ערך ההחזרה שנדרש כשהוא ריק. **אין מצב של הגשה של קובץ ללא פונקציה שנדרשת**.
 - d. לפני ההגשה **הסירו כל הרצה** של פונקציה שאתם מריצים בעצמכם (חוץ ממה שכתוב במפורש שאתם נדרשים להריץ) והסירו כל הערה שעניינה DEBUGGING מהקוד, או קוד שנמצא בתוך הערות. הסירו כל PRINT שלא נכתב במקום שנכתב במפורש שמותר להדפיס.
5. אי עמידה באחד מהתנאים של סעיף 4 תוריד לכם נקודות עד כדי ציון 0! הקפידו על זה מאוד!

מבנה התיקיות הנדרש:

1. ההגשה תיעשה בלינק של ריפו בגיטהאב
2. בתוך הריפו לא יהיו שום קבצים שאינם קשורים למבחן מלבד הקבצים המתוארים להלן. **אי עמידה בסעיף זה עלולה להביא לפסילת המבחן.** ודאו כי אין שום תיקיות מיותרות או קבצים מיותרים ששכחתם.
3. אתם רשאים להוסיף קבצי `__init__` ככל שתצטרכו.
4. אתם רשאים להוסיף קבצים ותיקיות **נוספים** למה שנדרש בסעיף 5, בתנאי שהם קשורים למבחן ואתם מוודאים שהייבוא מהם לא מונע מהקוד לרוץ כמו שצריך. הקפידו על DRY KISS ועל קוד נקי וSINGLE RESPONSIBILITY בקבצים ובתיקיות שאתם יוצרים.
5. **מבנה התיקיות:**

- a. `game_logic/`
 - i. `game.py`
- b. `utils/`
 - i. `deck.py`
- c. `main.py`
- d. `readme.md` - קובץ שמכיל בשורה הראשונה שם פרטי שם משפחה כיתה - **אי הכללת ות"ז. עם רווחים ביניהם בעברית בלבד. ללא # בתחילת השורה. אי הכללת**
השורה הזו תפסול את המבחן

רשימת הפונקציות הנדרשת, פר קובץ (פירוט והסבר בהמשך, הסדר לא עקרוני בין הפונקציות באותו קובץ):

1. `game.py`:
 - a. `create_player(name:str) -> dict`
 - b. `init_game()->dict`
 - c. `play_round(p1:dict,p2:dict)`
2. `deck.py`:
 - a. `create_card(rank:str,suite:str) -> dict`
 - b. `compare_cards(p1_card:dict, p2_card:dict) -> str`
 - c. `create_deck() -> list[dict]`
 - d. `shuffle(deck:list[dict]) -> list[dict]`

3. בקובץ `main.py`:

- a. נדרש שתהיה הפעלת המשחק = הקריאה לפונקציות `init_game` ו `play_round` תיעשה **אך ורק** מתוך הקובץ הזה.
- b. הקריאה לפונקציות תעשה מתוך התנאי:
`if __name__ == "__main__":`
<הקריאה לפונקציות, וכל קוד אחר שלכם שאתם רוצים>

דגשים נוספים:

1. ודאו כי יש לכם את כל הקבצים הנדרשים ואת כל הפונקציות הנדרשות (גם אם הן ריקות ומחזירות רק ערך ריק שתואם לחתימת הפונקציה - כלומר, פוקנציה שדורשת להחזיר STR ולא הספקתם לממש, כתבו אותה ותחזירו STR ריק, וכך לערך החזרה של DICT או כל ערך אחר)!
2. הרצת המשחק תיעשה מתוך התיקיה הראשית, בעזרת הפקודה:
`python main.py`. ודאו כי אתם מצליחים להריץ את הקוד שלכם בצורה תקינה עם הפקודה הזו!
3. קראו את פירוט הפונקציות **לאט ובהקפדה**. ודאו שכל פונקציה שאתם כותבים **עומדת לחלוטין** בכל הדרישות ומיישמת **בדיוק** את מה שנדרש ממנה. כל דרישה שהתעלמתם ממנה, לא שמתם לב אליה או שכחתם ממנה תגרום להורדת ניקוד, עד כדי פסילת הפונקציה לחלוטין.
4. הדוגמאות של הקלט והפלט לכל פונקציה שמוצגות בפירוט הפונקציות (בהמשך) הן דוגמה עבורכם בלבד. שימו לב שאתם לא משאירים אותן בהגשה הסופית, ושימו לב שאתם לא מתעלמים בגללן מחתימת הפונקציה. חתימה הפונקציה היא המקור היחיד לשאלה מה אמור להתקבל כפרמטר ומה הפונקציה אמורה להחזיר. הדוגמאות תמיד נכונות ותואמות לחתימה, אבל הן להמחשה בלבד.
5. וודאו שהקוד שלכם נקי ומסודר, ועם שמות הגיוניים לפונקציות ולמשתנים.
6. הוסיפו תיעוד בהערות אם אתם מרגישים שזה נחוץ. שימו לב שאתם לא משאירים הערות מיותרות או קוד שנמצא בתוך הערה.
7. התחילו את המבחן ביצירת כל הקבצים וכל הפונקציות שאתם נדרשים ליצור. ודאו שכל פונקציה מחזירה את הערך הנכון לפי החתימה שלה. לדוגמה - פונקציה שדורשת להחזיר מילון - תחזיר בתור התחלה מילון ריק. אתם תממשו את הקוד בפועל ככל שתתקדמו במבחן.
8. הקוד שייבדק הוא הקוד שנמצא בענף MAIN בלבד. ודאו שהקוד שלכם נמצא שם לפני סוף המבחן.

להלן פירוט הפונקציות.

שיהיה בהצלחה!

פירוט דרישות פונקציות:

פונקציות בתוך `utils/deck.py`

`create_card(rank:str, suite:str) -> dict`

Creates a card dictionary with fields: rank, suite, value.

rank = card number/letter, suite = type of card (hearts, clubs etc) - single letter (uppercase), value is a number based on rank:

2 = 2, ..., 10 = 10, J = 11, Q = 12, K = 13, A = 14

Suits are one of: H, C, D, S.

Input → Output examples

- `create_card("A", "S") → {"rank": "A", "suite": "S", "value": 14}`
- `create_card("10", "H") → {"rank": "10", "suite": "H", "value": 10}`

`compare_cards(p1_card: dict, p2_card: dict) -> str`

Compare **two card dicts** by their **value**.

- `p1_card1` is the **human** players (p1) card.
- `p2_card2` is **AI** (p2) players card.
Return: a **string** 'p1' or 'p2' indicating who wins. If its a draw, return "WAR"

Input → Output examples (card dict is showing only value, for short. In the test the entire card dict needs to be passed as an argument)

- `compare_cards({"value": 14, suite...}, {"value": 10, suite...}) → 'p1'`
- `compare_cards({"value": 9, ...}, {"value": 13, ...}) → 'p2'`
- `compare_cards({"value": 5, ...}, {"value": 5, ...}) → 'WAR'`

create_deck() -> list[dict]

Returns a list of **52 card dicts** (full deck).

Input → Output examples

- `print(len(create_deck())) → 52`
- `create_deck()[0:4] →`
`[{"rank": "2", "suite": "S", "value": 2},`
`{"rank": "2", "suite": "H", "value": 2}, ...]`
(Exact order is up to you, but the deck must include all 52 unique cards.)

shuffle(deck: list[dict]) -> list[dict]

Shuffles the deck using this **exact algorithm**:

1. Repeat **1000 times**:
 - Pick two **random** numbers that correspond to indexes in the deck list → `index1, index2`.
 - If `index1 == index2`, pick again.
 - **Swap** the cards at those two indexes. For example, if `index 1 = 12` and `index 2 = 34`, place the card from index 12 at index 34, and the card at index 34 at index 12.
2. Return the deck (the same one you got as input - after you shuffled it).

Hints for random index - read about these functions if you want to use them:

- `random.randrange`
- `random.randint`

Input → Output examples

- `d = create_deck(); d = shuffle(d); len(d) → 52`

פונקציות בתוך :game_logic/game.py

create_player(name: str) -> dict

Creates a player dictionary that has the following keys:

- name: the name of the player, taken from the argument. If no name is passed - the name will be "AI"
- hand: list of cards (start with an empty list - []). These are the player's cards to be used in the game.
- won_pile: list of cards the player won (start with an empty list - [])

Input → Output examples

- create_player("p1") →
{"name": "p1", "hand": [], "won_pile": []}
- create_player() → {"name": "AI", "hand": [], "won_pile": []}

init_game() -> dict

- **Creates the 2 players** using create_player. one with any name that you want, one with the name of AI.
- **Creates a new deck** and **shuffles** it. (*using the functions from utils*)
- **Deals** the cards:
 - p1 (human) gets the **top 26** of the shuffled deck.
 - p2 (AI) gets the **remaining 26**.
 - Each player stores his cards in his "hand" - in the player dict.
- returns the **game dict**: {
 "deck": <the deck dict you created, shuffled>,
 "player_1": <human player dict you created>,
 "player_2": <AI player dict you created>
}

play_round(player_1: dict, player_2: dict) -> None

Plays **one round** of the game, using the 2 player dicts we created before, that are passed as parameters to this function.

1. Each player plays the **top card** from their hand. **Take out** the first card from the "hand" of each player (use the player dict properties!).
2. **Compare** the two cards by **value** (using the compare function we created)
3. If one is **higher**:
 - The winner **takes both** cards into their won_pile.
4. if the cards are the same - do nothing (see bonus tasks for different implementation)
5. you can add any prints you want.

פונקציות [main.py](#):

1. if __name__ == "__main__":
ודאו כי יש את התנאי: if __name__ == "__main__":
2. ודאו כי בתוך התנאי אתם קוראים לפונקציות init_game, play_round ו game_logic
ודאו כי אתם מריצים את play_round כל עוד יש לשחקנים קלפים בחפיסה שלהם
3. ודאו כי אתם מריצים את play_round כל עוד יש לשחקנים קלפים בחפיסה שלהם
4. ודאו כי כשהחפיסה של אחד או שני השחקנים מסתיימת אתם מחשבים נכון את המנצח לפי גודל חפיסת הרווח.
5. הדפיסו את המנצח או תיקו, אם הגודל של חפיסות הרווח שווה.

אלו כל דרישות **החובה**. כאמור - אי הגשה של פונקציה עלולה לגרום לפסילת המבחן. הקפידו להגיש הכל - גם אם הפונקציה ריקה ומחזירה את הערך שנדרש בחתימה ללא כלום.

להלן משימות בonus, שיש לעשות אם סיימתם את משימות **החובה במלואן**.

משימות בונוס:

1. ממשו את מנגנון "המלחמה" במקרה ששני הקלפים ששלפו השחקנים זהים.
 - a. כשיש מלחמה שני השחקנים מוסיפים שלושה קלפים הפוכים (הפרס) והופכים קלף נוסף ומשווים.
 - i. אם הקלף הנוסף של שחקן אחד גדול מהקלף הנוסף של השחקן השני - השחקן עם הקלף הגבוה יותר ניצח, והוא לוקח את הקלפים לערימת הרווח שלו, כולל כלפי הפרס שלו ושל יריבו.
 - ii. אם הקלף הנוסף שוב שווה לקלף הנוסף - עושים מלחמה נוספת.
 - iii. ממשיכים כך עד שלשחקן אין יותר קלפים בערימה שלו. במקרה כזה, עצרו את המשחק מיד וחשבו את המנצח
2. ממשו מנגנון הכרעה במקרה שהמשחק הסתיים ולשני השחקנים יש את אותה כמות קלפים. ממשו איזה מנגנון שאתם רוצים - לדוגמה החליטו שסוג קלף מסוים (למשל "לבבות") הוא שווה יותר מהקלפים האחרים, והשחקן שיש לו יותר לבבות ינצח.
3. שימו לב שאסור שאף משימת בונוס לא תפריע או תשנה את ערכי ההחזרה של הפונקציות הקיימות או של צורת ההרצה של המשחק, כפי שתואר בחלק של משימות החובה.

בהצלחה!