

## מבחן משחק מלחמה

1. ב厰חן זהה נבנה משחק שמדמה את משחק הקלפים "מלחמה".
2. למי שלא מכיר, קר הולך המשחק:
  - a. משחקים עם חפיסת קלפים עם מספרים בין 10-2 ועם נסיך (J) מלכה (Q) מלך (K) ואו (A). זהה "דרגה" (**rank**) של הקלף.
  - b. בלבד הערך המספרי, לכל קלף יש גם "סוג" (**suit**) - לב (h), תלתן (club/c), יהלום (diamond/d), עליה (s/spade). קר שיש 4 קלפים מכל דרגה בחיפוי, ו52 קלפים בסך הכל.
  - c. לכל קלף יש ערך (**value**). ערכו של כל קלף מספרי שווה לדרגה שלו ( $2=3, 3=2$ ) וכן הלאה), נסיך = 11, מלכה = 12, מלך = 13 ואו = 14.
  - d. מערבבים את החפיסה ומחלקים את כל הקלפים בין שני מתמודדים. כל הקלפים הפוכים ומסתירים את הערך שלהם. זהה עירמת המשחק של השחקן - "היד" שלו (**hand**)
  - e. בכל סיבוב, כל שחקן שולף קלף אחד מראש העירימה האישית שלו ("היד"), והופר אותו - ומשווה לקלף שהיריב שולף.
  - f. השחקן עם הקלף הגבוה יותר - לוקח את שני הקלפים ששוחקו ושם אותם בעירימה נפרדת - "עירמת הרוח" (**won pile**)
  - g. אם הקלפים זרים, זהו מצב של "מלחמה", שב厰חן זהה נממש שם הקלפים זרים - לא קורה כלום, הקלפים יוצאים מהחפיסה של השחקן ולא נכנסים לשום עירמת רוח, ומשחקים את הקלף הבא בתור.
  - h. אפשר למש את הצורה האמיתית של המשחק כמשמעות בונוס - ראו פירוט בסוף המבחן.
3. אתם נדרשים לכתוב פונקציות בקבצים שונים, עם יבוא נכוון בינהן.
4. שימוש לב:
  - a. מבנה התיקיות שמתואר כאן **חייב** להיות זהה.
  - b. שמות הפונקציות והמשתנים **חייבים** להיות זרים.
  - c. אתם **חייבים** להציג את הקבצים **עם כל הפונקציות שנדרשות**. אם לא הספקתם למש אותן - כתבו רק את החתימה ואת ערך ההחזקה שנדרש כשהוא ריק. אין מצב של הגשה של קובץ ללא פונקציה שנדרשת.
  - d. לפני ההגשה **הסירו כל הרצה** של פונקציה שאתם מרכיבים בעצמכם (חו"ץ ממה שכותב במפורש נדרשים להריץ) והסירו כל הערה שענינה DEBUGGING מהקוד, או קוד שנמצא בתוך הערות. הסירו כל PRINT שלא נכתב במקום שנכתב במפורש שמותר להדפס.
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 בlynk של ריפו בגייטהוב
2. בהתאם הריפו לא יהיה שום קבצים שאינם קשורים ל מבחן 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 \_\_טיוח\_\_ ככל שתצטרכו.
4. אתם רשאים להוסיף קבצים ותיקיות **נוספיהם** למה שנדרש בסעיף 5, בתנאי שהם קשורים ל מבחן ואתם מודאים שהיבוא מהם לא מונע מהקוד לרוץ כמו שצירף. הקפידו על KISS ו על קוד נקי ו**SINGLE RESPONSIBILITY** בקבצים ובתיקיות שאתם יוצרם.

## **5. מבנה התיקיות:**

- a. game\_logic/
  - i. game.py

- b. utils/
  - i. deck.py

- c. main.py

קובץ שמאיל בשורה הראשונה שם פרטי שם משפחה כתה - readme.md ות"ז. עם רוחים ביניהם בעברית בלבד. **לא #** בתחילת השורה. **אי הכללת השורה הזאת לפסול את המבחן!**

## רשימת הפונקציות הנדרשת, פר קובץ (פירוט והסבר בהמשך, הסדר לא עקרוני בין הפונקציות באוטו 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 שתיהה הפעלה המשחק = הקריאה לפונקציות game וinit\_game
- b. הקריאה play\_round תיעשה **אך ורק** מתוך הקובץ זהה.

הקריאה לפונקציות תעשה מתוך התנאי:

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

<הקריאה לפונקציות, וכל קוד אחר שלכם שאתם רוצים>

## **dagshim nosofim:**

1. ודאו כי יש לכם את **כל** הקבצים הנדרשים ואת **כל** הפונקציות הנדרשות (גם אם הן ריקות ומחזירות רק ערך ריק שתואם לחתימת הפונקציה - קלומר, פוקנציה שדורשת להחזיר STR ולא הספקתם למש, כתבו אותה ותחזרו STR ריק, וכן לערך החזרה של DICT או כל ערך אחר!).
2. הרצת המשחק תיעשה מתוך התיקיה הראשית, בעזרת הפקודה: `python main.py` עם הפקודה הזו!
3. קראו את פירוט הפונקציות **לאט ובהקפדה**. ודאו שכל פונקציה שאתם כותבים **עומדת לחלוtin** בכל הדרישות ומימוש ב**בדיקה** את מה שנדרש ממנה. כל דרישת שהتعلמתם ממנה, לא שמתם לב אליה או שכחתם ממנה תגרום להורדת ניקוד, עד כדי פסילת הפונקציה לחלוtin.
4. הדוגמאות של הקלט והפלט לכל פונקציה שמוצגות בפירוט הפונקציות (במהרשך) הן דוגמה עבורכם בלבד. שימו לב שאתם לא משאים אותם בהגשה הסופית, ושימו לב שאתם לא מתעלמים בכלל מחתימת הפונקציה. חתימה הפונקציה היא המקור היחיד לשאלת מה אמרו להתקבל כפרמטר ומה הפונקציה אמורה להחזיר. הדוגמאות תמיד נכונות ותואמות לחתימה, אבל הן להמחשה בלבד.
5. וודאו שהקובד שלכם נקי ומסודר, עם שמות הגינויים **לפונקציות ול משתנים**.
6. הוסיפו תייעוד בהערות אם אתם מרגישים זהה נוח. שימו לב שאתם לא משאים הערות מיותרות או קוד שנמצא בתוך הערה.
7. התחלו את המבחן ביצירת כל הקבצים וכל הפונקציות שאתם נדרש ליצור. ודאו שכל פונקציה מחזירה את הערך הנכון לפי החתימה שלה. לדוגמה - פונקציה שדורשת להחזיר מילון - תחזיר בתור התחלה מילון ריק. אתם תמשו את הקוד בפועל ככל שתתקדמו במבחן.
8. הקוד **שייבדק** הוא הקוד שנמצא בענף **MAIN** בלבד. ודאו שהקובד שלכם נמצא **שם לפניו סוף המבחן**.

**להן פירוט הפונקציות.**  
**שיהיה בהצלחה!**

## **פירוט דרישות פונקציות:**

פונקציות בתחום `: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**:
  - o 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. וודאו כי בתוך התנאי אתם קוראים לפונקציות `play_round` ו `init_game` אם `name == "main"`:
2. וודאו כי בתוך התנאי אתם מפעילים את `game_logic` שאותם מיבאים מ�וך `game_logic`
3. וודאו כי אתם מרים את `play_round` כל עוד יש לשחקנים קלפים במשחק שלהם
4. וודאו כי כשהחפיסה של אחד או שני השחקנים מסתיים מתם מחשבים נכון את המנצח לפי גודל חפיסת הרווח.
5. הדריסו 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 ומהזירה את הערך שנדרש בחיתימה ללא כלום.

להלן משלימות בonus, שיש לעשות אם סייםتم את **משימות החובה במלואן**

## **משימות בונוס:**

1. ממשו את מנגנון "המלחמה" במקרה שני הקלפים שלפוי השחקנים זרים.
  - a. כישיש מלחמה שני השחקנים מוסיפים שלושה קלפים הפוכים (הפרט) והופכים קלח נוסף ומשווים.
    - i. אם הקלח הנוסף של שחן אחד גדול מהקלח הנוסף של שחן השני - השחקן עם הקלח הגבוה יותר ניצח, והוא לוקח את הקלפים לעירימת הרוח שלו, כולל כלפי הפרט שלו ושל יריבו.
    - ii. אם הקלח הנוסף שוב שווה לקלח הנוסף - עושים מלחמה נוספת.
    - iii. ממשיכים כך עד שלשחן אין יותר קלפים בעירימה שלו. במקרה זה, 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 ניצח.
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!**