# עבודה מסכמת – מבוא לפייתון

בעבודה הבאה תתבקשו ליצור חמש מחלקות שונות העוסקות במשחק קלפים (מחלקה לקלף, מחלקות לחפיסה, ומחלקות למשחק "מלחמה"). יש לכתוב קוד כמה שיותר ברור וקריא, ורצוי להוסיף הערות המסבירות שורות או מהלכים מורכבים במיוחד. כל המחלקות צריכות להופיע באותו קובץ הפייתון, אותו תגישו לי, במייל.

ישנה שאלה מילולית בתור בונוס בסוף התרגיל, שאותה ניתן להגיש בקובץ טקסט (וורד, pdf, txt, מה שנוח לכם) בנפרד.

שם קובץ הקוד צריך להיות:

war_cards_<full name>.py

כאשר <full name> מייצג את השם המלא שלכם, מופרד בקו תחתון. למשל, לו אני הייתי מגיש את הקובץ הוא היה נקרא:

war_cards _itamar_shturm.py

את קובץ הטקסט יש להגיש בשם:

word_portion_<full name>

באופן דומה.

לגבי הקוד עצמו:

- לא צריך לבדוק את תקינות הקלט אלא אם נאמר אחרת. אם כתוב שפונקציה מסוימת תקבל מחרוזת, לא צריך להתכונן למצב בו ייכנס בוליאן במקום (או כל דבר אחר). כן צריך להתכונן לכל המצבים הנכללים בהגדרת השאלה, למשל אם יכניסו מחרוזת ריקה (כיוון שזו עדיין מחרוזת).

- ניתן לכתוב כמה מתודות עזר נוספות שאתם רוצים בתוך כל מחלקה, ולהוסיף אטריביוטים אם בא לכם. אסור לשנות את חתימות הפונקציות (לשנות את שם הפונקציה, את הארגומנטים שהיא מקבלת, או את הערכים שהיא מחזירה).

- מדובר בתרגיל במסגרת Object Oriented Programming, אז שימו לב מתי אתם נדרשים לממש מתודות קסם, או מתי ואיך כדאי להשתמש בירושה.

- אתם יכולים לעשות אילו בדיקות שאתם רוצים בקובץ שלכם, אך בסוף עליכם להגיש אותו עם המחלקות בלבד, ובלי קוד נוסף מחוץ להן.

לגבי אופן הכנת והגשת העבודה:

- מותר להתייעץ אחד עם השני בפרטי או בקבוצת הוואטסאפ. אל תעתיקו קוד אחד מהשני, זה אסור, אבל מותר לדבר על מה הפונקציות שלכם עושות ואיך.

- מותר ואף רצוי להתייעץ בגוגל.

- אם שכחתם דבר מה אתם יכולים להיעזר בחומר שבדרייב או בי. ניתן להפנות אליי שאלות כלליות בכל שלב, רצוי בוואטסאפ או במייל.

קראו היטב את נוסחי השאלות וכתבו את המחלקות והמתודות בהתאם. היעזרו בדוגאמות ההרצה שמופיעות מתחת לכל מחלקה כדי לבדוק שהקוד שלכם תקין (בתיקיית התרגיל ישנו קובץ קוד עם כל הבדיקות המופיעות בתמונות).

## מחלקת Card

המחלקה הראשונה שתכתבתו מייצגת קלף משחק.

<u>בנאי המחלקה (פונקציית init)</u>

הבנאי מקבל שני ארגומנטים:

val – ערך מסוג int המייצג את הערך המספרי של הקלף (מ-1 עבור אס עד 13 עבור מלך, ו-14 לג'וקר)

suit – ערך מסוג string המייצג את הצורה של הקלף (עבור ג'וקר הערך הזה יהיה None, לא מחרוזת)

הבנאי מאתחל שלושה אטריביוטים (משתנים עם self ונקודה לפניהם) של המחלקה:

val – הערך המספרי שהועבר בארגומנט val. יש לוודא שהמספר הוא בן אחד לארבע עשרה, ואם לא להעלות שגיאה מסוג Value Error עם מסר הגיוני לבחירתכם.

suit – המחרוזת שהועברה בארגומנט suit. יש לשמור את המחרוזת באותיות קטנות, לא משנה איך נכנסה, ולוודא שלאחר המעבר לאותיות קטנות, היא אחת מהמחרוזות:

diamond, club, heart, spade

או הערך None.

אם נכנס ערך אחר יש להעלות שגיאה מסוג Value Error עם מסר הגיוני לבחירתכם. שימו לב שאם הערך של val הוא 14, אז ה-suit חייב להיות None, ולהיפך (הייצוג של ג'וקר). אם מצב זה לא מתקיים, גם כאן יש להעלות שגיאה מסוג Value Error עם מסר הגיוני לבחירתכם.

name – מחרוזת המייצגת את השם של הערך המספרי של הקלף, כלומר קלף שערכו 3 יקבל את השם three. קלף שערכו 12 יקבל את השם queen, וכן הלאה. עליכם להשתמש ב-val כדי לשמור את הערך המילולי הנכון כמחרוזת.

*רמז: אתם יכולים לכתוב פונקציית עזר שתבצע את ההמרה ממספר לשם. אתם יכולים להשתמש במילון.*

<u>הדפסה</u>

היינו רוצים שבעת הדפסת קלף בעזרת print, נקבל מידע יעיל על מי הוא הקלף. ממשו פונקציית repr אשר תגרום להדפסת קלף להופיע בפורמט הבא:

<name> of <shape>s

כלומר, עבור הקלף עם הערכים 2 ולב, יודפס:

Two of Hearts

עבור הקלף עם הערכים 11 ועלה, יודפס:

Jack of Spades

וכן הלאה. שימו לב לאותיות הגדולות והקטנות, ולריבוי (האות s) בסוף הצורה.

אם הקלף הוא ג'וקר, צריכה פשוט להיות מודפסת המחרוזת 'Joker!'.

## השוואה

היינו רוצים להיות יכולים לדעת איזה קלף גדול או קטן או שווה לקלפים אחרים, כדי שנוכל למיין רשימות קלפים, או כדי שנוכל לשאול איזה קלף מנצח בכל סיבוב של "מלחמה".

לשם כך, ממשו את הפונקצייה __lt__ אשר קובעת, עבור מחלקה מסוימת, את התנהגות הסימן "קטן מ" (הסימן <).

הפונקצייה מחזירה True כאשר האובייקט עצמו (self) קטן מאובייקט אחר (other) ו-False אחרת.

ניתן לראות דוגמא למימוש של הפונקצייה כאן:

https://blog.finxter.com/python-__lt__-magic-method/

הפונקציה הזו מספיקה עבור פייתון כדי להבין את כל הסימנים האחרים (גדול, שווה, לא שווה וכו').

```
c1 = Card(5, 'heart')
c2 = Card(14, None)
c3 = Card(11, 'club')

print(c1)
print(c2)
print(c3 > c2)
print(c1 < c2)
```

```
assign
C:\Users\itama\venv\Scripts\python
Five of Hearts
Joker!
False
True

Process finished with exit code 0
```

```
c4 = Card(5, 'bla')
```

```
assign
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
Traceback (most recent call last):
  File "C:/Users/itama/Desktop/assign.py", line 16, in <module>
    c4 = Card(5, 'bla')
  File "C:\Users\itama\Desktop\war_cards.py", line 15, in __init__
    raise ValueError('invalid suit')
ValueError: invalid suit
```

```
c5 = Card(16, 'heart')
```

```
assign
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
Traceback (most recent call last):
  File "C:/Users/itama/Desktop/assign.py", line 17, in <module>
    c5 = Card(16, 'heart')
  File "C:\Users\itama\Desktop\war_cards.py", line 7, in __init__
    raise ValueError('invalid val')
ValueError: invalid val
```

```
c6 = Card(14, 'heart')
```

```
assign
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
Traceback (most recent call last):
  File "C:/Users/itama/Desktop/assign.py", line 18, in <module>
    c6 = Card(14, 'heart')
  File "C:\Users\itama\Desktop\war_cards.py", line 11, in __init__
    raise ValueError('val 14 cannot have a not-None suit')
ValueError: val 14 cannot have a not-None suit
```

```
c7 = Card(7, None)
```

```
assign
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
Traceback (most recent call last):
  File "C:/Users/itama/Desktop/assign.py", line 19, in <module>
    c7 = Card(7, None)
  File "C:\Users\itama\Desktop\war_cards.py", line 18, in __init__
    raise ValueError('if suit is None val must be 14')
ValueError: if suit is None val must be 14
```

## מחלקת Deck

המחלקה השנייה שתכתבו מייצגת חפיסת קלפים.

בנאי המחלקה (פונקציית init)

בנאי המחלקה לא מקבל אף ארגומנט, ומייצר את האטריביוט הבא:

card_list – רשימה של כל הקלפים 52 הקלפים בחפיסה (לא כולל ג'וקרים).

יש לייצר רשימה של אובייקטים מסוג Card, שתכיל את כל 52 הקלפים הדרושים בחפיסה. חשבו איך לייצר את כל הקלפים הנחוצים ולהכניסם לרשימה. לאחר יצירת הרשימה המלאה, יש "לערבב" אותה (לבלגן את סדר הרשימה בצורה ראנדומלית). *רמז: חפשו בגוגל איך לערבב רשימה בצורה ראנדומלית בפייתון, יש לשם כך פונקצייה.*

שליפת קלף

כתבו פונקציה בשם draw_card שמסירה מהרשימה את הקלף הראשון, ומחזירה אותו (בעזרת return). אם החפיסה ריקה, יש להחזיר None.

שליפת מספר קלפים

כתבו פונקציה בשם draw_multiple אשר מקבלת מספר (num), זורקת את num הקלפים הראשונים בחפיסה לפח (מסירה אותם מרשימת הקלפים) ומחזירה אותם בליסט. אם לא נותרו קלפים ברשימה הפונקציה מחזירה None. כלומר – אם num הוא 5, למשל, יש להסיר את 5 הקלפים הראשונים ברשימה ולהסירם ממנה.

ערבוב

כתבו פונקציה בשם shuffle ה"מערבבת את החבילה" (משנה את סדר רשימת הקלפים כפי שעשיתם בבנאי).

אתחול מחדש

כתבו פונקציה בשם reset אשר מחזירה לחבילה את כל הקלפים ומערבבת אותה, לקבלת חפיסה כמו-חדשה. *רמז: חשבו איזו פונקציה שכבר כתבתם מבצעת פעולה זו.*

הדפסה

ממשו פונקציית repr כך שבעת הדפסת חבילה, תודפס מחרוזת בפורמט הבא:

A Deck Containing <num of current cards> Cards

כאשר <num of current cards> מתייחס למספר הקלפים שנותרו בחבילה בעת ההדפסה.

השוואה

גם כאן נרצה לממש את הפונקציה __lt__ כך שחבילות מושוות על ידי מספר הקלפים שבהן. אם מספר הקלפים קטן יותר, החבילה קטנה יותר, אם גדול יותר גדולה יותר, וכן הלאה.

```python
d1 = Deck()
print(d1)

c1 = d1.draw_card()
print(c1)

cards = d1.draw_multiple(5)
print(cards)

print(d1)

d1.reset()
print(d1)
```

assign

```
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
A Deck Containing 52 cards
Queen of Diamonds
[King of Clubs, Six of Clubs, Two of Diamonds, Three of Hearts, Seven of Clubs, Three of Diamonds]
A Deck Containing 46 cards
A Deck Containing 52 cards

Process finished with exit code 0
```

```python
deck1 = Deck()
deck2 = Deck()
c = deck2.draw_card()
print(deck1 > deck2)
```

assign

```
C:\Users\itama\venv\Script
True
```

## מחלקת JokerDeck

המחלקה השלישית שתכתבו היא חפיסת קלפים המכילה שני ג'וקרים, בנוסף לשאר הקלפים. כל שאר ההתנהגות שלה זהה לחפיסה רגילה.

חישבו מה הדרך הכי פשוטה לממש מחלקה זו, ללא העתקה והדבקה של הרבה שורות קוד.

```
d2 = JokerDeck()
print(d2)

c2 = d2.draw_card()
print(c2)

cards = d2.draw_multiple(5)
print(cards)

print(d2)

d2.reset()
print(d2)
```

```
assign
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
A Deck Containing 54 cards
One of Spades
[Six of Spades, Three of Hearts, Two of Clubs, Six of Clubs, Two of Hearts, Jack of Spades]
A Deck Containing 48 cards
A Deck Containing 54 cards

Process finished with exit code 0
```

## מחלקת WarGame

המחלקה הרביעית שתכתבו מייצגת את המשחק "מלחמה" והחוקים שלו, ויכולה להריץ משחק בין שני "שחקנים" (חבילות קלפים).

בגרסא זו החוקים מעט פשוטים יותר. ישנם שני שחקנים שמתחילים עם חבילות מלאות.

כל סיבוב, שני השחקנים שולפים וחושפים שני קלפים (שהולכים לערימה).

הקלף המנצח – שני הקלפים בערימה הולכים לסוף החפיסה של השחקן ששלף אותו (בלי לערבב).

אם הקלפים שווים בכוחם – שני השחקנים מוסיפים לערימה שלושה קלפים כל שחקן, ואז מתחילים סיבוב חדש (כלומר, המנצח בסיבוב הבא לוקח את כל הערמה עם לפחות שמונה קלפים בתוכה, שלושה מכל אחד במלחמה + אחד מכל אחד בסיבוב החדש).

השחקן הראשון שהחבילה שלו מתרוקנת לגמרי מפסיד.

### בנאי המחלקה (פונקציית init)

הבנאי המחלקה מקבל ארגומנט בוליאני אחד בשם has_jokers, שקובע האם במשחק יהיה עם חבילות עם ג'וקרים או בלי ג'וקרים.

יש לבדוק שהארגומנט הוא אכן בוליאן ולהעלות TypeError לבחירתכם במקרה ולא.

הבנאי מאתחל שלושה אטריביוטים:

d1 – חפיסת הקלפים (מסוג Deck, או JokerDeck אם יש ג'וקרים) של השחקן הראשון.

d2 – חפיסת הקלפים (מסוג Deck, או JokerDeck אם יש ג'וקרים) של השחקן השני.

card_plie – רשימה ריקה אליה יצטרפו קלפים בערימה, וילקחו על ידי השחקן המנצח כל סיבוב.

### ריקון הערמה

כתבו פונקציה בשם give_pile המקבלת ארגומנט מסוג int בשם player, שיכול להיות 1 או 2, ו"נותנת את הערמה" לשחק שצוין.

כלומר, אם נכנס 1, הפונקצייה צריכה לשים את ערימת הקלפים הנוכחית בסוף החפיסה d1, ולהפוך את הערמה לרשימה ריקה שוב. אם 2, אותו הדבר, רק לסוף החפיסה d2. ניתן להניח שלא יכנס מספר אחר.

### סיבוב משחק יחיד

כתבו פונקציה בשם round המקבלת int בשם i המייצג את מספר הסיבוב (סיבוב משחק ראשון, שני, מאה).

הפונקציה מריצה סיבוב אחד של משחק המלחמה.

הפונקציה תבצע את הפעולות הבאות:

1. תשלוף קלף מכל חבילה.

2. תדפיס מחרוז בפורמט הבא:

Round <i>: <player 1 card> vs <player 2 card>

לדוגמא:

Round 3: Seven of Diamonds vs Six of Spades

3. תוסיף את שני הקלפים שנשלפו לערמה.

4. תבדוק מי מבין השחקנים ניצח בסיבוב:

   a. אם השחקן הראשון – יודפס:

'Player 1 Won\n'

   והערמה תינתן (give_pile) לשחקן הראשון.

   b. אם השחקן השני – יודפס:

'Player 2 Won\n'

   והערמה תינתן לשחקן השני.

   c. אם יש תיקו, יודפס:

'War!\n.\n.\n.\n'

   ("מלחמה!" ושלוש נקודות אחריה, כל אחת בשורה משל עצמה)

   ויתווספו לערמה שלושת הקלפים הראשונים בחפיסה של כל שחקן (לא משנה הסדר).

<u>הרצת המשחק</u>

כתבו פונקציה בשם run_game אשר מריצה משחק שלם של מלחמה בעזרת החפישות שאותחלו בבנאי המחלקה.

הפונקציה תפעל באופן הבא:

1. תדפיס את המחרוזת:

   STARTING WAR...

2. **עד שאחת החפיסות מתרוקנת**, תמשיך להפעיל שוב ושוב את הפונקציה round עם מספר הסיבוב המתאים (בהתחלה סיבוב אפס, אחר כך סיבוב אחד, וכן הלאה וכן הלאה)

3. לאחר שלפחות חפיסה אחת מתרוקנת...

   אם התרוקנה רק השנייה, יש להדפיס:

   PLAYER 1 IS THE VICTOR!

אם התרוקנה רק הראשונה, יש להדפיס:

PLAYER 2 IS THE VICTOR!

אם שתיהן התרוקנו בו זמנית, יש להדפיס:

IT'S A TIE!

**שימו לב**: אין בדיקות ספציפיות למחלקה זו, הבדיקות שלה שקולות לאלו של המחלקה הבאה.

## מחלקת LimitedWarGame

משחק מלחמה יכול להימשך, תיאורטית, לנצח. לא נרצה להריץ ולבדוק כזה משחק בפייתון. כדי להוכיח שהלוגיקה שלנו עובדת, נרצה ליצור מחלקה זהה עם מספר סיבובים מוגבל – אם נכניס לה את המספר 10,000 לדוגמא, המשחק יסתיים אחרי עשרת אלפים סיבובים ויהי מה.

המחלקה  האחרונה שתכתבו היא גרסא של משחק המלחמה, מוגבלת במספר הסיבובים. כמעט כל המתודות של המחלקה הזו זהות לשל WarGame המקורית.

הבנאי שלה מקבל, בנוסף ל-has_joker מהמקור, גם משתנה מסוג int בשם rounds אשר קובע כמה סיבובים המשחק יקח.

ערכו את הבנאי של המחלקה בהתאם. בנוסף, יש לערוך עוד פונקציה כך שהמשחק אכן יסתיים לאחר rounds סיבובים בדיוק.

לאחר מספר זה של סיבובים, המנצח מוכרע על פי למי יש יותר קלפים בחבילה באותו הרגע (אם המספר שווה, יש תיקו).

חשבו, איזו פונקציה מ-WarGame המקורית יש לדרוס ב-LimitedWarGame כדי להגביל את מספר הסיבובים? ומה צריך לשנות בה?

דאגו להשאיר את הודעות הניצחון זהות לאיך שהיו ב-WarGame המקורית.

```
lg = LimitedWarGame(True, 500)
lg.run_game()
```

```
assign
C:\Users\itama\venv\Scripts\python.exe C:/Users/itama/Desktop/assign.py
STARTING WAR...
Round 0: One of Spades vs Five of Spades
Player 2 Won

Round 1: Four of Hearts vs One of Hearts
Player 1 Won

Round 2: Five of Diamonds vs Seven of Hearts
Player 2 Won

Round 3: Three of Diamonds vs Nine of Spades
Player 2 Won
```

```
lg = LimitedWarGame(True, 500)
lg.run_game()
```

```
assign
Round 497: Two of Hearts vs Six of Spades
Player 2 Won

Round 498: Jack of Diamonds vs Six of Hearts
Player 1 Won

Round 499: Three of Spades vs Six of Spades
Player 2 Won

PLAYER 1 IS THE VICTOR!

Process finished with exit code 0
```

## שאלה מילולית פתוחה (בונוס)

אפשר להסתכל על המשחק "מלחמה", באופן כללי, כמשחק רקורסיבי.

לו היינו כותבים את מהלך המשחק כרקורסיה, כיצד פונקציה זו הייתה נראית?

תארו במילים את:

- תנאי העצירה
- מה הארגומנטים הנדרשים (מה כל סיבוב ברקורסיה צריך לדעת)
- צעד הרקורסיה (מתי ואיך קוראים לפונקציה בתוך עצמה, ואיך הארגומנטים משתנים)

כיוון שאנו משתמשים במילים לא צריך להגדיר היטב את כל הפעולות, וניתן להשתמש בביטויים כמו "לשלוף קלף מחפיסה אחת" ללא הסבר נוסף.

**בהצלחה!**