

תרגיל 5: OOP

הערה: המסמך של העבודה ארוך כדי לתאר את המשימה באופן הברור ביותר. עם זאת,

התרגיל מכיל משימות רבות שרובן מאוד פשוטות. שימו לב כי ניתן לממש את רוב השיטות בעזרת שורת קוד יחידה.

תאריך פרסום: 29.11.21

תאריך הגשה: 16.12.21 בשעה 23:59

מתרגלת אחראית: שיר כהן

משקל תרגיל: 4 נקודות

הנחיות כלליות

קראו בעיון את ההנחיות והעבודה. לא יתקבלו ערעורים על טעויות שחרגו מההנחיות.

1. העבודה תבוצע ותוגש ביחידים.
2. מומלץ לקרוא את העבודה כולה לפני שאתם ניגשים לפתרון.
3. עליכם להוריד את קבצי הקוד שמסופק עם התרגיל ולהשלים את הקוד החסר. יש לממש את הפתרון לתרגיל אך ורק באזורים שהוגדרו לשם כך בקובץ! כתיבת קוד קריא:
- 4.1. השתמשו בשמות משתנים משמעותיים. שימוש בשמות לא משמעותיים עשוי לגרור לפגיעה בציון.
- 4.2. כתבו תיעוד (הערות) שמסביר את הקוד שלכם. יש לכתוב תיעוד docstring בכל פונקציה כפי שנלמד בכיתה.
- 4.3. אין לכתוב הערות בעברית! עבודה שתכיל טקסט בשפה שאינה אנגלית (או פייתון) תקבל ציון אפס ללא אפשרות ערעור.
5. אין להשתמש בחבילות או במודולים חיצוניים מלבד מה שהוגדר בתרגיל! אם יש ספק ניתן לשאול בפורום המתאים (ראו סעיף 10).
6. יש לכתוב קוד אך ורק באזורים שהוגדרו לשם כך!
7. הנחות על הקלט:
 - 7.1. בכל שאלה יוגדר מה הקלט שהקוד מקבל וניתן להניח כי הקלט שנבדוק מקיים את התנאים הללו. אין להניח הנחות נוספות על הקלט מעבר למה שהוגדר.
 - 7.2. בכל שאלה סיפקנו עבורכם דוגמאות לקלט והפלט הרצוי עבורו. עליכם לערוך בדיקות נוספות לקוד שמימשתם ולא להסתמך על דוגמאות אלו בלבד.
8. בדיקת העבודה:

8.1. העבודה תיבדק באופן אוטומטי ולכן על הפלטים להיות זהים לפלטים שמוגדר בתרגיל.

8.2. טרם ההגשה יש לעבור על המסמך [assignments checklist](#) שנמצא במודל.

8.3. מערכת הבדיקות קוראת לפונקציות שהוגדרו בתרגיל בצורה אוטומטית. אין לשנות את חתימות הפונקציות. חריגה מההנחיות תגרור ציון אפס.

9. העתקות:

9.1. אל תעתיקו!

9.2. העתקת קוד (משנים קודמות, מחברים או מהאינטרנט) אסורה בהחלט. בפרט אין להעביר קוד בין סטודנטים. צוות הקורס ישתמש בכלים אוטומטיים וידניים כדי לזהות העתקות. תלמיד שייתפס בהעתקה יועמד בפני ועדת משמעת (העונש המינימלי לפי תקנון האוניברסיטה הוא כישלון בקורס).

9.3. אנא קראו בעיון את המסמך שהכנו בנושא:

<https://moodle2.bgu.ac.il/moodle/mod/resource/view.php?id=1922556>

10. שאלות על העבודה:

10.1. שאלות בנוגע לעבודה ישאלו בפורום שאלות לתרגיל במודל או בשעות הקבלה של המתרגלת האחראית בלבד.

10.2. אין לפנות במייל לבודקת התרגילים או למתרגלים אחרים בנוגע לעבודות הגשה. מיילים בנושאים אלו לא יקבלו מענה.

10.3. לפני ששואלים שאלה בפורום יש לוודא שהשאלה לא נשאלה קודם!

10.4. אנו מעודדים סטודנטים לענות על שאלות של סטודנטים אחרים. המתרגלת האחראית תאשר שתשובה כזו נכונה.

11. הגשת העבודה:

11.1. עליכם להוריד את הקבצים מתיקיית "תרגיל בית 5" מהמודל.

התיקייה תכיל תיקייה נוספת ובה קבצי העבודה וקובץ הוראות.

עליכם למלא את הפתרון במקום המתאים ובהתאם להוראות התרגיל.

11.2. שימו לב: בנוסף לקבצי העבודה מצורף קובץ בשם `get_id.py`.

עליכם למלא במקום המתאים בקובץ את תעודת הזהות שלכם. הגשה

שלא תכיל את הקובץ הנ"ל עם תעודת הזהות הנכונה לא תיבדק

ותקבל ציון אפס!

11.3. את העבודה יש להגיש באמצעות תיבת ההגשה הייעודית במודל.

11.4. פורמט ההגשה הוא להלן:

11.4.1. את התיקייה בשם `hw5` שהורדתם מתיקיית העבודה

במודל, ובה נמצאים קבצים הקוד שלכם, יש לכווץ לפורמט

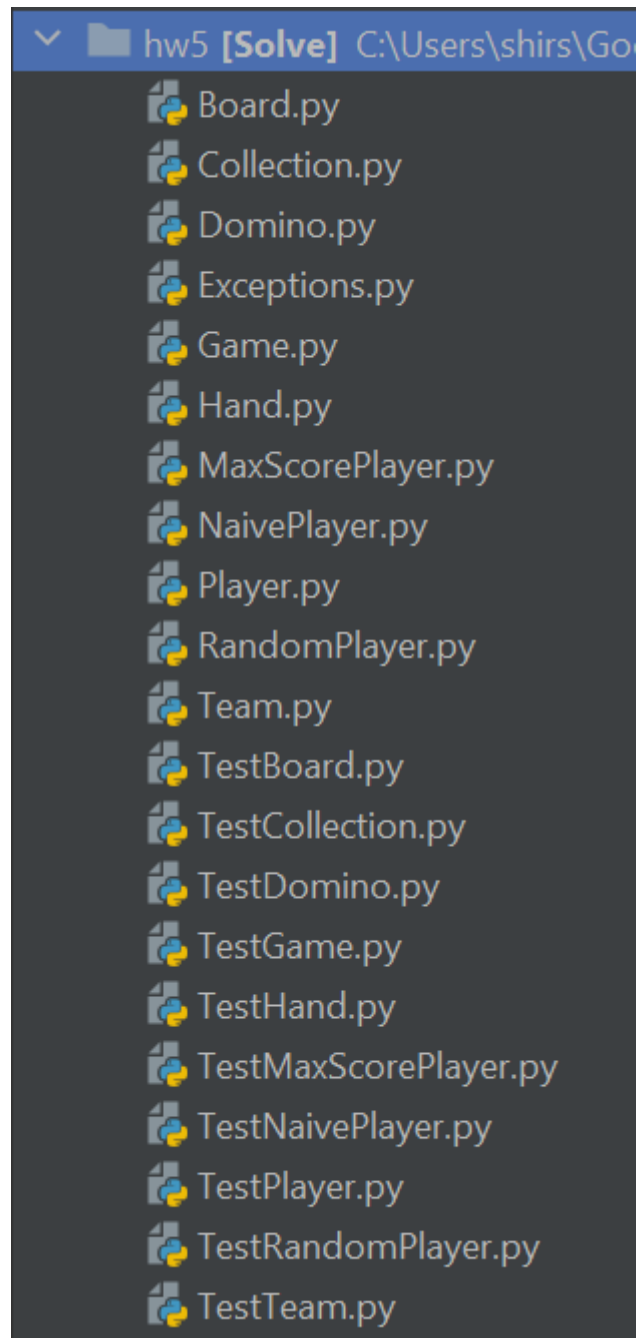
`zip` (קבצים אחרים, כגון קבצי `rar`, יקבלו ציון אפס).

11.4.2. השם של התיקייה המכוצת יהיה תעודת הזהות שלכם.

11.4.3. העלו את התיקייה המכוצת לתיבת ההגשה של העבודה.

מבנה ההגשה

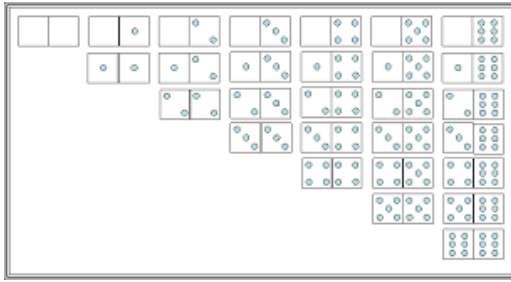
לפניכם רשימה של כל הקבצים שעליכם להגיש:



לתיקיה שקיבלתם hw5 (המכילה את הקבצים) בצעו zip והגישו את התיקיה שנוצרה לאחר ה-zip.

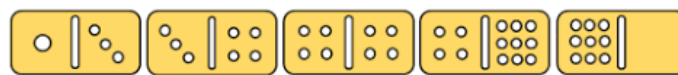
כללי המשחק

"דומינו הוא משחק המורכב מ-28 חלקים הנקראים אבנים, קלפים או דומינו. אבני המשחק מלבניות וקו מחלק אותן באמצע לשני ריבועים, שעל כל אחד מהם מסומנות מספר נקודות (ראה תמונה 1). נהוג להשתמש באבנים עם אפס עד שש נקודות, כאשר כל אבן מופיעה פעם אחת בלבד." (ויקיפדיה)



תמונה 1

בתחילת המשחק כל שחקן (Player) מקבל מספר זה של אבני דומינו (Domino) הנסחרות מעיני האחרים (כמות אבני הדומינו יכולה להשתנות ממשחק למשחק). בתחילה, לוח הדומינו (Board) ריק. השחקן הראשון מניח את האבן הראשונה. לאחריו, השחקן הבא בתור יכול להניח אבן בסמוך לאבן שהונחה על לוח הדומינו כך שאחד מריבועיה זהה לאחד הריבועים החיצוניים של האבנים בלוח (ראו תמונה 2).



סידור חוקי של אבני דומינו בשלב כלשהו במשחק

תמונה 2

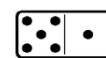
המשחק נגמר כאשר לאחד השחקנים לא נותרו אבנים או שאין אפשרות לאף שחקן להניח אבן בלוח. אם המשחק נגמר ולכל השחקנים נשארו אבני דומינו השחקן המנצח יוכרז כשחקן שסכום הערכים על אבני הדומינו שלו הוא הנמוך ביותר. למשל, בדוגמה שבתמונה 3 ניתן לראות דוגמה למצב בו לשחקנים נותרו אבנים אך המשחק נגמר מאחר ואף אבן לא מתאימה ללוח המשחק שמוצג בתמונה 4. במצב הזה הניקוד של שחקן 1 הוא 6 (אבן אחת שסכומה $5 + 1 = 6$) והניקוד של שחקן 2 הוא 9 (שתי אבנים שסכום כל הערכים הוא $0 + 2 + 2 + 5 = 9$). ולכן במצב זה שחקן 1 מנצח את שחקן 2.



תמונה 4 – לוח המשחק



שחקן 2



שחקן 1

תמונה 3

דגשים

1. נא לקרוא את **כל העבודה לפני** שאתם מתחילים לכתוב את הקוד.
2. הגדרות העבודה: **לא מצוין** מפורשות מתי יש לדרוס שיטה\להשתמש בשיטת האב\להגדיר מחלקה אבסטרקטית\שיטה אבסטרקטית\להגדיר שדה פרטי\מתי לבצע shallow copy \מתי לבצע deep copy וכדומה. לכן, עליכם להגדיר זאת **תוך עמידה בדרישות העבודה**. לדוגמה: **אין** להגדיר שדה פרטי אם זה לא נדרש כדי לעמוד בדרישות העבודה.
3. דוגמאות הרצה לחלק מהשיטות מצויות בקובץ נפרד בשם Tests.py.
4. עליכם לממש את העבודה בעזרת **מקסום** שימוש חוזר בקוד בעזרת שימוש בשיטות של מחלקת האב.
5. נא לקרוא את **כל העבודה לפני** שאתם מתחילים לכתוב את הקוד. הסעיף הזה נכתב שוב וזה לא בטעות.

חלק א'

אוסף (Collection)

המחלקה אוסף (Collection) מוגדרת על ידי השדות הבאים:

❖ **array** - רשימה (list) המכילה אובייקטים (object).

ממשו את המחלקה Collection:

```
def __init__(self, array):
```

בנאי המאתחל את שדה ה-array.

קלט:

○ **array [list]** - רשימה.

```
def get_collection(self):
```

השיטה תחזיר את השדה array.

פלט:

○ **[list]** - השדה array.

```
def add(self, item, option):
```

השיטה תוסיף איברים ל-array לפי מדיניות שתוגדר לאובייקטים אשר יממשו את הפונקציה. השיטה לא תמומש במחלקה הזו ועל כן יש לזרוק חריגה מסוג "[NotImplementedError](#)" עם הודעה אינפורמטיבית כרצונכם.

קלט:

○ **item [object]** - אובייקט אותו רוצה להוסיף לאוסף.

○ **option [object]** - אופציה להגדרת משתנה נוסף.

פלט:

○ **[Exception]**

דגש:

○ **ניתן** יהיה ליצור אובייקט מהמחלקה collection.

```
def __getitem__(self, i):
```

השיטה תחזיר את האיבר במקום ה-i בשדה array.

קלט:

○ **i [int]** - אינדקס.

פלט:

- **[object]** - ה-object במיקום ה-i. בכל מקרה בו לא ניתן להחזיר את ה-object במיקום ה-i יוחזר None.

```
def __eq__(self, other):
```

השיטה תחזיר ערך בוליאני (bool) המציין אם האובייקט הנוכחי (self) והאובייקט other שווים. שני אובייקטים שווים אם ה-array שלהם מכיל את אותם ה-objects במיקום זהה. שוויון בין ה-objects יוגדר על ידי השיטה __eq__ של האובייקטים עצמם. במקרה ולא ניתן להעריך אם האובייקטים שווים יוחזר False.

קלט:

- **[object] other** - אובייקט מולו משווים.

פלט:

- **[bool]** - True אם שני האובייקטים שווים. אחרת יוחזר False.

לדוגמה:

```
>>> [1, 2, 3, 4] == [1, 2, 4, 3]
```

```
False
```

```
>>> ["I", "Domino", None, 1] == ["I", "Domino", None, 1]
```

```
True
```

```
def __ne__(self, other):
```

השיטה תחזיר bool המציין אם שני אובייקטים לא שווים. שני אובייקטים יוגדרו כלא שווים אם אינם עומדים בתנאי השוויון שהוגדרו בשיטה __eq__.

קלט:

- **[object] other** - אובייקט מולו משווים.

פלט:

- **[bool]** - True אם שני האובייקטים שונים אחרת יוחזר False.

```
def __len__(self):
```

השיטה תחזיר את מספר האובייקטים בשדה array.

פלט:

- **[int]** - מספר האובייקטים בשדה array.

```
def __contains__(self, item):
```

השיטה תחזיר bool המציין אם item נמצא בשדה array. ההשוואה של ערכי ה-item תתבצע באמצעות השיטה `__eq__`.

קלט:

○ `[object] item` - אובייקט.

פלט:

○ `[bool]` - True אם האובייקט נמצא בשדה array. אחרת יוחזר False.

```
def __str__(self):
```

השיטה תחזיר מחרוזת המייצגת את ה-Collection הנוכחי. המחרוזת תכיל את הייצוג str של כל item בתוך שדה ה-array בלי רווחים ביניהם.

פלט:

○ `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט בדוגמה.

לדוגמה:

```
>>> [1, 2, 3, 4]
```

```
'1234'
```

```
>>> ["I", "Domino", None, 1]
```

```
'IDominoNone1'
```

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את ה-Collection הנוכחי כפי שהוגדר בשיטה `str`.

פלט:

○ `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה `__str__`.

חריגות (Exceptions)

הקובץ יכיל מספר מחלקות של חריגות שונות (Exceptions).

`EmptyBoardException`

החריגה תיזרק כאשר הלוח ריק.

`FullBoardException`

החריגה תיזרק כאשר הלוח מלא ולא ניתן להוסיף אבני דומינו.

`NoSuchDominoException`

החריגה תיזרק כאשר נרצה להסיר אבן דומינו שלא קיימת באוסף (Collection).

[InvalidNumberException](#)

החריגה תיזרק כאשר הוזן מספר בטווח לא תקין.

עליכם לדרוס את השיטה `__str__`:

```
def __str__(self):
```

השיטה תחזיר מחרוזת המייצגת תיאור של החריגה.

פלט:

○ `[str]` – מחרוזת המכילה את המילה `ERROR` (המילים הקבועות מסומנות באדום).

ERROR message

לדוגמה, בהרצת קטע הקוד הבא:

```
try:
```


```
    raise InvalidNumberException("Invalid")
```

```
except Exception as e:
```

```
    print(str(e))
```

יודפס:

'ERROR Invalid'

בדיקה סופקה בקובץ `Tests.py`. 

דומינו (Domino)

המחלקה דומינו (Domino) מייצגת אבן דומינו. המחלקה מוגדרת על ידי השדות הבאים:

❖ **left_side** - מספר שלם בתחום [0-6] המגדיר את הערך השמאלי של אבן הדומינו.

❖ **right_side** - מספר שלם בתחום [0-6] המגדיר את הערך הימני של אבן הדומינו.

דגש – שני שדות המחלקה לא ניתנים לשינוי מחוץ למחלקה.

ממשו את המחלקה Domino:

```
def __init__(self, left, right):
```

בנאי המאתחל את שדות המחלקה.

קלט:

○ `[int] left` - מספר שלם. אם המספר לא בתחום [0-6] יש לזרוק חריגה מסוג

[InvalidNumberException](#) עם הודעה אינפורמטיבית כרצונכם.

- `right [int]` - מספר שלם. אם המספר לא בתחום [0-6] יש לזרוק חריגה מסוג `InvalidNumberException` עם הודעה אינפורמטיבית כרצונכם.

```
def get_left(self):
```

שיטה המחזירה את ה-`left_side` של אבן הדומינו.

פלט:

- `[int]` - הערך שבצד השמאלי של האבן.

```
def get_right(self):
```

שיטה המחזירה את ה-`right_side` של אבן הדומינו.

פלט:

- `[int]` - הערך שבצד הימני של האבן.

```
def __str__(self):
```

השיטה תחזיר מחרוזת המייצגת את אבן הדומינו.

פלט:

- `[str]` - מחרוזת עם ערכי אבן הדומינו לפי הפורמט המצורף (התווים הקבועים מסומנים באדום והטקסט השחור הוא ערכי השדות). שימו לב! אין רווחים. המחרוזת מוגדרת על ידי: סוגר ריבוע פותח, הערך השמאלי, קו (|), הערך הימני וסוגר סוגר ימני.

```
[left_side|right_side]
```

לדוגמה:

```
>>> tile1 = Domino(1,2)
```

```
>>> tile1
```

```
[1|2]
```

בדיקה סופקה בקובץ `Tests.py`. 🚀

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את אבן הדומינו כפי שהוגדר בשיטה `str`.

פלט:

- `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה `__str__`.

```
def __eq__(self, other):
```

השיטה תחזיר ערך בוליאני (bool) המציין אם האובייקט הנוכחי (self) והאובייקט other שווים. שני אובייקטים שווים אם שניהם מכילים אותם ערכים ללא חשיבות למיקום (שמאל וימין). במקרה ולא ניתן להעריך אם האובייקטים שווים יוחזר False.

קלט:

○ **other [object]** - אובייקט מולו משווים.

פלט:

○ **[bool]** - True אם שני האובייקטים שווים אחרת יוחזר False.

לדוגמה:

```
>>> Domino(1,2) == Domino(2,1)
```

```
True
```

```
def __ne__(self, other):
```

השיטה תחזיר bool המציין אם שני אובייקטים לא שווים. שני אובייקטים יוגדרו כלא שווים אם אינם עומדים בתנאי השוויון שהוגדרו בשיטה `__eq__`.

קלט:

○ **other [object]** - אובייקט מולו משווים.

פלט:

○ **[bool]** - True אם שני האובייקטים שונים אחרת יוחזר False.

```
def __gt__(self, other):
```

השיטה תחזיר bool המציין אם אבן הדומינו הנוכחית (self) גדולה יותר מהאובייקט other. אבן דומינו תוגדר כגדולה מאבן דומינו אחרת אם סכום הערכים על האבן (צד ימני ושמאלי) **גדול ממש** מסכום הערכים של אבן הדומינו other. במקרה ולא ניתן להעריך אם האובייקטים יוחזר False.

קלט:

○ **other [object]** - אובייקט.

פלט:

○ **[bool]** - True אם self גדול ממש מ-other אחרת יוחזר False.

```
def __contains__(self, key):
```

השיטה תחזיר bool המציין אם באחד הצדדים (שמאל או ימין) של אבן הדומינו יש את הערך key.

קלט:

○ **key [int]** - מספר שלם.

פלט:

○ **[bool]** - True אם באבן הדומינו יש את הערך key אחרת יחזור False.

לדוגמה:

```
>>> tile = Domino(1,2)
```

```
>>> 1 in tile
```

```
True
```

```
>>> 2 in tile
```

```
True
```

```
>>> 3 in tile
```

```
False
```

```
def flip(self):
```

השיטה תבצע היפוך לאבן הדומינו. השיטה תחזיר אובייקט חדש מסוג Domino שהערך שבצד השמאלי שלו הינו הערך הצד הימני של self, והערך שבצד הימני שלו הינו הערך שבצד השמאלי של self.

פלט:

○ **[Domino]** - אובייקט מסוג אבן דומינו בו ערכי האבן הפוכים.

לדוגמה:

```
>>> tile = Domino(1,2)
```

```
>>> flip_tile = tile.flip()
```

```
>>> tile
```

```
[1|2]
```

```
>>> flip_tile
```

```
[2|1]
```

לוח (Board)

המחלקה לוח (Board) יורשת מהמחלקה Collection.

המחלקה מוגדרת על ידי השדות הבאים:

❖ **max_capacity** - מספר שלם בתחום [1-28] המציין את המספר המקסימלי של אבני דומינו

שיכולות להיות בלוח.

❖ **array** - רשימה (list) המכילה בתוכה אבני דומינו (Domino).

ממשו את המחלקה Board:

```
def __init__(self, max_capacity):
```

בנאי המאתחל את שדות האובייקט. השדה array יוגדר באתחול כרשימה ריקה.

קלט:

○ `max_capacity [int]` - מספר שלם. אם המספר אינו בתחום של [1-28] יש לזרוק חריגה

מסוג [InvalidNumberException](#) עם הודעה אינפורמטיבית כרצונכם.

🚦 בדיקה סופקה בקובץ Tests.py.

`def in_left(self):`

שיטה שמחזירה את הערך השמאלי ביותר בלוח. במידה והלוח ריק תיזרק חריגה מסוג

[EmptyBoardException](#) עם הודעה אינפורמטיבית כרצונכם.

פלט:

○ `[int/Exception]` - הערך השמאלי של האבן השמאלית ביותר בלוח או חריגה מתאימה.

לדוגמה, עבור הלוח הבא:

[3|1][1|2]

יוחזר הערך 3.

`def in_right(self):`

שיטה שמחזירה את הערך הימני ביותר בלוח. במידה והלוח ריק תיזרק חריגה מסוג

[EmptyBoardException](#) עם הודעה אינפורמטיבית כרצונכם.

פלט:

○ `[int/Exception]` - הערך הימני של האבן הימנית ביותר בלוח או חריגה מתאימה.

לדוגמה, עבור הלוח הבא:

[3|1][1|2]

יוחזר הערך 2.

`def add(self, domino, add_to_right=True):`

השיטה תוסיף אבן דומינו ללוח במידה והמהלך חוקי. אם המשתנה `add_to_right` מוגדר כ-True

השיטה תנסה להוסיף את אבן הדומינו לצד הימני של הלוח. אחרת, השיטה תנסה להוסיף את אבן הדומינו

לצד השמאלי של הלוח. בהתחלה ננסה להוסיף את האבן ללוח לפי צורתה המקורי ואם הדבר לא

מתאפשר ננסה להוסיפה על ידי היפוך של אבן הדומינו (באמצעות flip). השיטה תחזיר bool המציין אם

הפעולה בוצעה בהצלחה או תזרוק חריגה (exception) במידה והלוח מלא. אם הפעולה הצליחה יש

להוסיף את אבן הדומינו ללוח.

קלט:

- **domino [Domino]** - אבן דומינו אותה רוצים להוסיף לוח.
 - **add_to_right [bool]** – מאותחל בערך ברירת מחדל עם True. אם True, השיטה מוסיפה את האבן החדשה לקצה הימני של הלוח. אחרת היא מוסיפה אותה לקצה השמאלי.
- פלט:
- **[bool/Exception]** - True אם אבן הדומינו נוספה ללוח בהצלחה ו-False אחרת. במידה והלוח מלא תיזרק חריגה מסוג [FullBoardException](#) עם הודעה אינפורמטיבית כרצונכם.

ענן מחשבה –

1. חשבו מה המטרה (או מטרות) של השיטה add?
2. אם לשיטה יש מספר מטרות מומלץ לפצל אותה למספר שיטות.

בדיקה סופקה בקובץ Tests.py. 🚩

def add_left(self, domino):

השיטה תוסיף אבן דומינו לצד השמאלי של הלוח במידה והמהלך חוקי. בהתחלה ננסה להוסיף את האבן ללוח לפי צורתה המקורי ואם הדבר לא מתאפשר ננסה להוסיפה על ידי היפוך של אבן הדומינו (באמצעות flip). השיטה תחזיר bool המציין אם הפעולה בוצעה בהצלחה. אם הפעולה הצליחה יש להוסיף את אבן הדומינו ללוח.

קלט:

- **domino [Domino]** - אבן דומינו אותה רוצים להוסיף לוח.

פלט:

- **[bool]** - True אם אבן הדומינו נוספה ללוח בהצלחה ו-False אחרת.

def add_right(self, domino):

השיטה תוסיף אבן דומינו לצד הימני של הלוח במידה והמהלך חוקי. בהתחלה ננסה להוסיף את האבן ללוח לפי צורתה המקורי ואם הדבר לא מתאפשר ננסה להוסיפה על ידי היפוך של אבן הדומינו (באמצעות flip). השיטה תחזיר bool המציין אם הפעולה בוצעה בהצלחה. אם הפעולה הצליחה יש להוסיף את אבן הדומינו ללוח.

קלט:

- **domino [Domino]** - אבן דומינו אותה רוצים להוסיף לוח.

פלט:

○ **[bool]** - True אם אבן הדומינו נוספה ללוח בהצלחה ו-False אחרת.

```
def __getitem__(self, i):
```

השיטה תחזיר את ה-domino במקום ה-i.

קלט:

○ **[int] i** - אינדקס.

פלט:

○ **[Domino/None]** - ה-domino במיקום ה-i. בכל מקרה בו לא ניתן להחזיר את ה-domino

במיקום ה-i יוחזר None.

```
def __contains__(self, key):
```

השיטה תחזיר bool המציין אם ה-key נמצא ב-array.

קלט:

○ **key [Domino]** - אובייקט.

פלט:

○ **[bool]** - True אם הדומינו נמצא ב-array אחרת יוחזר False. ההשוואה של ערכי הדומינו

תתבצע לפי השיטה `__eq__` של אבן דומינו.

```
def __eq__(self, other):
```

השיטה תחזיר ערך בוליאני (bool) המציין אם האובייקט הנוכחי (self) והאובייקט other שווים. שני

אובייקטים שווים אם ה-max_capacity שלהם שווה וה-array שלהם מכיל את אותם אבני דומינו

במיקום זהה ובסדר זהה (כלומר, יש חשיבות לערך השמאלי והימני של כל אחת מאבן הדומינו). במקרה

ולא ניתן להעריך אם האובייקטים שווים יוחזר False.

קלט:

○ **other [object]** - האובייקט מולו משווים.

פלט:

○ **[bool]** - True אם שני האובייקטים שווים. אחרת יוחזר False.

לדוגמה:

עבור Board עם max_capacity של 8 ומכיל את האבנים:

```
[1|2]
```

עבור Board עם max_capacity של 8 ומכיל את האבנים:

```
[2|1]
```

נגדיר כי שני הלוחות אינם שווים.

```
def __ne__(self, other):
```

השיטה תחזיר bool המציין אם שני אובייקטים לא שווים. שני אובייקטים יוגדרו כלא שווים אם אינם עומדים בתנאי השוויון שהוגדרו בשיטה `__eq__`.

קלט:

○ `[object] other` - אובייקט מולו משווים.

פלט:

○ `[bool]` True אם שני האובייקטים שונים אחרת יוחזר False.

```
def __len__(self):
```

השיטה תחזיר את כמות האבנים בלוח.

פלט:

○ `[int]` - אורך ה-array.

```
def __str__(self):
```

השיטה תחזיר מחרוזת המייצגת את הלוח. המחרוזת תכיל את הייצוג str של כל אבן דומינו בלי רווחים ביניהם.

פלט:

○ `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט בדוגמה.

לדוגמה: עבור לוח עם שלוש אבנים תוחזר המחרוזת

'[2|1][1|3][3|2]'

בדיקה סופקה בקובץ Tests.py. 🚀

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את הלוח כפי שהוגדר בשיטה str.

פלט:

○ `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט

המפורט עבור השיטה `__str__`.

בדיקה סופקה בקובץ Tests.py. 🚀

ענן מחשבה – חשבו אילו שיטות נרצה לדרוס ובאילו שיטות נרצה להשתמש מהאב.

יד (Hand)

המחלקה יד (Hand) יורשת מהמחלקה Collection.

המחלקה מוגדרת על ידי השדות הבאים:

❖ array - רשימה (list) המכילה בתוכה אבני דומינו (Domino).

ממשו את המחלקה Hand:

```
def __init__(self, dominoes):
```

בנאי המאתחל את שדות האובייקט. שדות האב יאותחלו כמתואר במחלקת Collection.
קלט:

○ **dominoes** [list] - רשימה (list) המכילה בתוכה אבני דומינו (Domino).

```
def add(self, domino, index=None):
```

השיטה תוסיף אבן דומינו ל-array במיקום האינדקס.
קלט:

○ **domino** [Domino] - אבן דומינו אותה רוצים להוסיף ליד.

○ **index** [None/int] - אם ה-index הוא None אבן הדומינו תתווסף לסוף ה-array. אחרת,

אבן הדומינו תתווסף למיקום ה-index ואבני הדומינו לאחר מיקום ה-index יועברו מקום אחד ימינה. ניתן להניח כי ה-index הוא גדול שווה ל-0 וקטן מאורך הרשימה.

לדוגמה:

```
>>> tile1 = Domino(1, 2)
```

```
>>> hand = Hand([tile1])
```

```
>>> hand
```

```
'[1|2]'
```

```
>>> tile2 = Domino(1, 3)
```

```
>>> hand.add(tile2, 0)
```

```
>>> hand
```

```
'[1|3][1|2]'
```

```
def remove_domino(self, domino):
```

השיטה תסיר אבן דומינו מה-array ותחזיר את האינדקס ב-array שבו הייתה אבן הדומינו. אבני דומינו ישווה על ידי השיטה `__eq__` של האובייקט Domino. במידה ואבן הדומינו לא קיימת בלוח, תיזרק חריגה מסוג [NoSuchDominoException](#) עם הודעה אינפורמטיבית כרצונכם. לאחר ביצוע השיטה האבן תוסר מה-array.

קלט:

○ **[Domino] domino** - אבן דומינו אותה רוצים להסיר מהיד.

פלט:

○ **[int/Exception]** - מיקום (אינדקס) האבן ב-array או חריגה במידה ולא ניתן להסיר את האבן.

לדוגמה:

```
>>> tile1 = Domino(2,1)
>>> tile2 = Domino(4,1)
>>> tile3 = Domino(1,4)
>>> hand = Hand([tile1, tile2])
>>> hand.remove_domino(tile3)
1
```

נשים לב כי אבן הדומינו [1|4] שווה על פי השיטה `__eq__` לאבן הדומינו [4|1] ולכן יוחזר אינדקס 1.

```
def __getitem__(self, i):
```

השיטה תחזיר את ה-domino במקום ה-i.

קלט:

○ **[int] i** - אינדקס.

פלט:

○ **[Domino/None]** - ה-domino במיקום ה-i. במקרה ולא ניתן להחזיר את ה-domin במיקום

ה-i יוחזר None.

```
def __contains__(self, key):
```

השיטה תחזיר bool המציין אם ה-key נמצא ביד.

קלט:

○ **[Domino] key** - אובייקט.

פלט:

○ **[bool]** - True אם הדומינו נמצא ב-array אחרת יוחזר False. ההשוואה של ערכי הדומינותתבצע לפי השיטה `__eq__` של דומינו.

```
def __eq__(self, other):
```

השיטה תחזיר ערך בוליאני (bool) המציין אם האובייקט הנוכחי (self) והאובייקט other שווים. שני

אובייקטים שווים אם ה-array מכיל שני אבני דומינו זהות (לפי השיטה `__eq__` של אבן דומינו)**במיקום זהה.** במקרה ולא ניתן להעריך אם האובייקטים שווים יוחזר False.

קלט:○ **other [object]** - אובייקט מולו משווים.פלט:○ **[bool]** - True אם שני האובייקטים שווים. אחרת יוחזר False.דוגמאות:

1. דוגמה 1:

עבור Hand שמיוצגת כך:

[1|2][3|1]

ועבור Hand שמיוצגת כך:

[2|1] [3|1]

נגדיר כי שני ה-Hands שווים והשיטה תחזיר True.

2. דוגמה 2:

עבור Hand שמיוצגת כך:

[3|1] [1|2]

ועבור Hand שמיוצגת כך:

[2|1] [3|1]

נגדיר כי שני ה-Hands לא שווים והשיטה תחזיר False.

def __ne__(self, other):

השיטה תחזיר bool המציין אם שני אובייקטים לא שווים. שני אובייקטים יוגדרו כלא שווים אם אינם עומדים בתנאי השוויון שהוגדרו בשיטה `__eq__`.

קלט:○ **other [object]** - אובייקט מולו משווים.פלט:○ **[bool]** - True אם שני האובייקטים שונים אחרת יוחזר False.**def __len__(self):**

השיטה תחזיר את כמות האבנים ב-hand.

פלט:○ **[int]** - אורך ה-array.**def __str__(self):**

השיטה תחזיר מחרוזת המייצגת את היד. המחרוזת תכיל את הייצוג str של כל item בלי רווחים ביניהם.

פלט:

○ **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט בדוגמה.

לדוגמה: עבור יד עם שלושה אבנים תוחזר המחרוזת:

'[2|1][1|3][3|2]'

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את היד כפי שהוגדר בשיטה `.str`.

פלט:

○ **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה `.__str__`.

ענן מחשבה – חשבו אילו שיטות

נרצה לדרוס ובאילו שיטות נרצה

להשתמש מהאב.

חלק ב'

החל מחלק זה לא יוגדרו ענני מחשבה. אין זה אומר שעליכם להפסיק לחשוב כיצד לממש בצורה הנכונה ביותר העונה על דרישות המשימה.

שחקן (Player)

המחלקה שחקן (Player) מגדירה שחקן במשחק הדומינו.

המחלקה מוגדרת על ידי השדות הבאים:

❖ **name** - שם השחקן.

❖ **age** - גיל השחקן.

❖ **hand** - היד של השחקן.

השלימו את מימוש המחלקה Player:

```
def __init__(self, name, age, hand):
```

בנאי המאתחל את שדות השחקן.

קלט:

○ **name [str]** - מחרוזת לא ריקה המכילה את שם השחקן.

○ **age [int]** - גיל השחקן. הניחו כי הגיל הוא בין 0 ל-120.

○ **hand [Hand]** - אובייקט מטיפוס Hand. הניחו כי האובייקט מכיל רק אובייקטים מסוג

Domino עם ערכים בין 0 ל-6.

ניתן להניח שכל הערכים או האובייקטים אינם None או מכילים שדות/ערכים שערך None.

```
def score(self):
```

שיטה המגדירה את מספר הנקודות של השחקן – הסכום המצטבר של ערכי אבני הדומינו שברשות

השחקן (hand) כפי שהוגדר בתחילת העבודה.

פלט:

○ **[int]** - מספר הנקודות של השחקן.

```
def has_dominoes(self):
```

שיטה שמחזירה bool אם לשחקן יש עוד אבני דומינו ביד (hand).

פלט:

○ **[bool]** - True אם קיימות אבני דומינו אצל השחקן ו-False אחרת.

```
def play(self, board):
```

המטרה של השיטה `play` היא להפעיל אסטרטגית משחק בהינתן לוח (`board`).
לשחקן מטיפוס `Player` לא מוגדרת אסטרטגיה ולכן יש להגדיר את שיטה זו אך לא לממש אותה.

```
def __str__(self):
```

השיטה תחזיר מחרוזת המייצגת את השחקן.

פלט:

○ `[str]` - מחרוזת המייצגת את השחקן לפי הפורמט הבא: המילים הקבועות מסומנות באדום והערכים בשחור (קיים רווח לאחר כל פסיק ולאחר הנקודתיים).

Name: name, **Age:** age, **Hand:** hand, **Score:** self.score()

לדוגמה:

'Name: shir, Age: 26, Hand: [1|2][1|3][1|4], Score: 12'

בדיקה סופקה בקובץ `Tests.py`. 🚀

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את השחקן כפי שהוגדר בשיטה `str`.

פלט:

○ `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה `__str__`.

שחקן נאיבי (NaivePlayer)

המחלקה `שחקן נאיבי` (NaivePlayer) יורשת מהמחלקה `Player` ולכן מוגדרת על ידי אותם שלושת השדות (`name`, `age` ו-`hand`). שחקן נאיבי הינו שחקן שמניח את האבן הראשונה המתאימה לאבנים הנמצאות בלוח.

ממשו את המחלקה `NaivePlayer`:

```
def __init__(self, name, age, hand):
```

בנאי המאתחל את שדות השחקן כפי שמוגדר במחלקה `Player`.

קלט:

- **name** `[str]` - מחרוזת לא ריקה המכילה את שם השחקן.
- **age** `[int]` - גיל השחקן. הניחו כי הגיל הוא בין 0 ל-120.
- **hand** `[Hand]` - אובייקט מטיפוס `Hand`. הניחו כי האובייקט מכיל רק אובייקטים מסוג Domino עם ערכים בין 0 ל-6.

ניתן להניח שכל הערכים או האובייקטים אינם None או מכילים שדות/ערכים שערך None.

def score(self):

שיטה המגדירה את מספר הנקודות של השחקן – הסכום המצטבר של ערכי אבני הדומינו שברשות השחקן (hand) כפי שהוגדר בתחילת העבודה.
פלט:

○ **[int]** - מספר הנקודות של השחקן.

def has_dominoes(self):

שיטה שמחזירה bool אם לשחקן יש עוד אבני דומינו ביד (hand).
פלט:

○ **[bool]** - True אם קיימות אבני דומינו אצל השחקן ו-False אחרת.

def play(self, board):

שיטה שמבצעת מהלך יחיד עבור השחקן NaivePlayer. שחקן נאיבי עובר על היד (hand) החל מאבן הדומינו במיקום ה-0. השחקן מניח בלוח את האבן הראשונה שהוא יכול לשים בלוח. תחילה השחקן מנסה להוסיף את האבן לצד הימני של הלוח ואז לצד השמאלי. אם השחקן הצליח להוסיף אבן דומינו ללוח, ה-board מתעדכן בהתאם והאבן מוסרת מה-hand של השחקן.
קלט:

○ **[Board] board** - לוח משחק דומינו.

פלט:

○ **[bool]** - True אם השחקן הצליח לשים אבן דומינו על הלוח ו-False אחרת. שימו לב! על

השיטה להחזיר רק ערך bool.

לדוגמה:

עבור שחקן בעל ה-hand:

'[2|1][1|3][3|2]'

השחקן ינסה קודם להוסיף את אבן הדומינו [2|1]. אם אבן זאת אינה מתאימה ללוח השחקן ינסה להוסיף את דומינו [1|3] וכך הלאה.

def __str__(self):

השיטה תחזיר מחרוזת המייצגת את השחקן.

פלט:

○ **[str]** - מחרוזת המייצגת את השחקן לפי הפורמט הבא: המילים הקבועות מסומנות באדום

והערכים בשחור (קיים רווח לאחר כל פסיק ולאחר הנקודתיים).

Name: name, **Age:** age, **Hand:** hand, **Score:** self.score()

לדוגמה:

'Name: shir, Age: 26, Hand: [1|2][1|3][1|4], Score: 12'

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את השחקן כפי שהוגדר בשיטה str.

פלט:

○ [str] - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה __str__.

שחקן רנדומלי (RandomPlayer)

המחלקה שחקן רנדומלי (RandomPlayer) יורשת מהמחלקה Player ולכן מוגדרת על ידי אותם שלושת השדות (name, age ו-hand). שחקן רנדומלי לפני כל מהלך מסדר בצורה רנדומלית את ה-Hand שלו ומניח את האבן הראשונה המתאימה לאבנים הנמצאות בלוח.

השלימו את מימוש המחלקה RandomPlayer:

```
def __init__(self, name, age, hand):
```

בנאי המאתחל את שדות השחקן כפי שמוגדר במחלקה Player.

קלט:

○ **name [str]** - מחרוזת לא ריקה המכילה את שם השחקן.

○ **age [int]** - גיל השחקן. הניחו כי הגיל הוא בין 0 ל-120.

○ **hand [Hand]** - אובייקט מטיפוס Hand. הניחו כי האובייקט מכיל רק אובייקטים מסוג

Domino עם ערכים בין 0 ל-6.

ניתן להניח שכל הערכים או האובייקטים אינם None או מכילים שדות/ערכים שערך None.

```
def score(self):
```

שיטה המגדירה את מספר הנקודות של השחקן – הסכום המצטבר של ערכי אבני הדומינו שברשות השחקן (hand) כפי שהוגדר בתחילת העבודה.

פלט:

○ [int] - מספר הנקודות של השחקן.

```
def has_dominoes(self):
```


שיטה שמחזירה bool אם לשחקן יש עוד אבני דומינו ביד (hand).

פלט:

○ **[bool]** True אם קיימות אבני דומינו אצל השחקן ו-False אחרת.

def play(self, board):

שיטה שמבצעת מהלך יחיד עבור השחקן RandomPlayer. בכל קריאה לשיטה, השחקן רנדומלי מסדר את ה-hand בצורה אקראית ולאחר מכן עובר על ה-hand החל מאבן הדומינו במיקום ה-0. השחקן מניח בלוח את האבן הראשונה שהוא יכול לשים בלוח. תחילה השחקן מנסה להוסיף את האבן לצד הימני של הלוח ואז לצד השמאלי. אם השחקן הצליח להוסיף אבן דומינו ללוח ה-board מתעדכן בהתאם והאבן מוסרת מה-hand של השחקן.

על מנת לסדר את ה-hand בצורה אקראית השתמשו בשיטה [random.shuffle](#). לאחר הפעלת השיטה, ה-hand תישאר באותו הסדר שהוגדר **טרם** הפעלת השיטה. כלומר, מיקומי אבני הדומינו ב-hand לא ישתנה כתוצאה מפעולת ה-random.

קלט:

○ **board [Board]** - לוח משחק דומינו.

פלט:

○ **[bool]** True אם השחקן הצליח לשים אבן דומינו על הלוח ו-False אחרת. שימו לב! על

השיטה להחזיר רק ערך bool.

דגשים:

1. בקובץ של המחלקה הוגדרה השורה הבאה:

`random.seed(12)`

אין לשנות את המלל או את מיקום השורה!!! הנכם נדרשים לכתוב את הקוד לאחר שורה זאת.

אם ברצונכם להרחיב את הידע שלכם על `random.seed` אתם יכולים לקרוא [כאן](#).

2. אסור להוסיף באף קטע קוד (מלבד בקבצי הטסטים) שורה נוספת מהסוג: `random.seed`

def __str__(self):

השיטה תחזיר מחרוזת המייצגת את השחקן.

פלט:

○ **[str]** - מחרוזת המייצגת את השחקן לפי הפורמט הבא: המילים הקבועות מסומנות באדום

והערכים בשחור (קיים רווח לאחר כל פסיק ולאחר הנקודתיים).

Name: name, **Age:** age, **Hand:** hand, **Score:** self.score()

לדוגמה:

'Name: shir, Age: 26, Hand: [1|2][1|3][1|4], Score: 12'

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את השחקן כפי שהוגדר בשיטה `str`.

פלט:

○ `[str]` - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה `__str__`.

שחקן ממקסם נקודות (`MaxScorePlayer`)

המחלקה שחקן ממקסם נקודות (`MaxScorePlayer`) יורשת מהמחלקה `Player` ולכן מוגדרת על ידי אותם שלושת השדות (`name`, `age` ו-`hand`). שחקן ממקסם נקודות מנסה להיפטר תחילה מאבני דומינו בעלי ניקוד גבוה.

ממשו את המחלקה `MaxScorePlayer`:

```
def __init__(self, name, age, hand):
```

בנאי המאתחל את שדות השחקן כפי שמוגדר במחלקה `Player`.

קלט:

○ `name [str]` - מחרוזת לא ריקה המכילה את שם השחקן.

○ `age [int]` - גיל השחקן. הניחו כי הגיל הוא בין 0 ל-120.

○ `hand [Hand]` - אובייקט מטיפוס `Hand`. הניחו כי האובייקט מכיל רק אובייקטים מסוג

`Domino` עם ערכים בין 0 ל-6.

ניתן להניח שכל הערכים או האובייקטים אינם `None` או מכילים שדות/ערכים שערךם `None`.

```
def score(self):
```

שיטה המגדירה את מספר הנקודות של השחקן – הסכום המצטבר של ערכי אבני הדומינו שברשות השחקן (`hand`) כפי שהוגדר בתחילת העבודה.

פלט:

○ `[int]` - מספר הנקודות של השחקן.

```
def has_dominoes(self):
```

שיטה שמחזירה `bool` אם לשחקן יש עוד אבני דומינו ביד (`hand`).

פלט:

○ `[bool]` - `True` אם קיימות אבני דומינו אצל השחקן ו-`False` אחרת.

```
def play(self, board):
```

שיטה שמבצעת מהלך יחיד עבור השחקן MaxScorePlayer. השחקן מעוניין להקטין כמה שניתן את ה-score במהלך הנוכחי ועל כן יניח את אבן הדומינו בעלת הערך המקסימלי. אם יש שני אבני דומינו בעלות אותו ערך ניתן להחליט על הסדר ביניהם באופן אקראי. למשל, עבור ה-hand:

[3|2][1|4][3|6]

השחקן ינסה להניח קודם את אבן הדומינו [3|6] שכן הערך באבן דומינו זאת הינו 9 וערך זה גדול מהערך 5 שעל יתר אבני הדומינו. אם לא ניתן להניח את האבן [3|6], השחקן ינסה להניח את אחת מאבני הדומינו האחרות ללא חשיבות לסדר (מאחר ושתיהן בעלות ערך זהה). תחילה השחקן מנסה להוסיף את האבן לצד הימני של הלוח ואז לצד השמאלי. אם השחקן הצליח להוסיף אבן דומינו ללוח, ה-board מתעדכן בהתאם והאבן מוסרת מה-hand של השחקן. לאחר ביצוע השיטה, ה-hand תישאר באותו הסדר שהוגדר **טרם** ביצוע השיטה, כלומר, מיקומי אבני הדומינו ב-hand לא ישתנה.

קלט:

○ **board [Board]** - לוח משחק דומינו.

פלט:

○ **[bool]** True אם השחקן הצליח לשים אבן דומינו על הלוח ו-False אחרת. שימו לב! על השיטה להחזיר רק ערך bool.

```
def __str__(self):
```

השיטה תחזיר מחרוזת המייצגת את השחקן.

פלט:

○ **[str]** - מחרוזת המייצגת את השחקן לפי הפורמט הבא: המילים הקבועות מסומנות באדום והערכים בשחור (קיים רווח לאחר כל פסיק ולאחר הנקודתיים).

Name: name, Age: age, Hand: hand, Score: self.score(), I can win the game!

לדוגמה:

'Name: shir, Age: 26, Hand: [], Score: 0, I can win the game!'

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את השחקן כפי שהוגדר בשיטה str.

פלט:

○ **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה __str__.

קבוצה (Team)

המחלקה קבוצה (Team) מגדירה קבוצת שחקנים במשחק הדומינו. המחלקה מוגדרת על ידי השדות הבאים:

❖ **name** - שם הקבוצה.

❖ **players** - רשימה של שחקנים מטיפוס Player. לא ניתן יהיה לגשת לשדה זה מחוץ למחלקה.

ממשו את המחלקה Team:

```
def __init__(self, name, players):
```

בנאי המאתחל את שדות הקבוצה.

קלט:

○ **name [str]** - מחרוזת לא ריקה המכילה את שם הקבוצה.

○ **players [list]** - רשימה (list) המכילה אובייקטים של Players. הניחו כי הרשימה מכילה רק

אובייקטים מסוג Players עם הנחות הקלט של ה-`__init__` של ה-Player.

ניתן להניח שכל הערכים או האובייקטים אינם None או מכילים שדות/ערכים שערבם None.

```
def get_team(self):
```

השיטה תחזיר את רשימת השחקנים (players). אין לאפשר שינוי של רשימת השחקנים או שדות

השחקנים מחוץ למחלקה.

פלט:

○ **[list]** - רשימה של שחקנים.

```
def score_team(self):
```

שיטה המחשבת את מספר הנקודות של הקבוצה. מספר הנקודות מהווה סכימה של ערכי ה-score של כל

השחקנים.

פלט:

○ **[int]** - מספר הנקודות של הקבוצה.

```
def has_dominoes_team(self):
```

שיטה שמחזירה bool אם לשחקני הקבוצה יש עוד אבני דומינו.

פלט:

○ **[bool]** True - אם קיים לפחות שחקן אחד שיש לו אבני דומינו (אחת או יותר) ו-False אחרת.

def play(self, board):

שיטה שמבצעת מהלך יחיד עבור הקבוצה (משמע, רק שחקן אחד משחק!) הקבוצה תעבור על רשימת השחקנים החל מהשחקן במיקום ה-0 והלאה בסדר עולה (תחילה השחקן במיקום ה-0 ינסה לשחק ולאחריו השחקן במיקום ה-1 וכך הלאה). אם שחקן הצליח להוסיף אבן דומינו ללוח ה-board מתעדכן בהתאם, האבן מוסרת מה-hand של השחקן ונגמר הסיבוב של הקבוצה (כלומר, לא בודקים אם עוד שחקנים יכולים להניח אבני דומינו בלוח). לאחר ביצוע השיטה רשימת השחקנים תישאר באותו הסדר שהוגדר **טרם** ביצוע השיטה, כלומר, מיקומי השחקנים לא ישתנו.

קלט:

○ **[Board] board** - לוח משחק דומינו.

פלט:

○ **[bool]** True - אם הקבוצה הצליחה לבצע מהלך ו-False אחרת. שימו לב! על השיטה להחזיר רק ערך bool.

def __str__(self):

השיטה תחזיר מחרוזת המייצגת את הקבוצה.

פלט:

○ **[str]** - מחרוזת המייצגת את הקבוצה לפי הפורמט הבא: המילים הקבועות מסומנות באדום והערכים בשחור (קיים רווח לאחר כל פסיק ולאחר הנקודתיים).

Name: name, **Score team:** self.score_team(), **Players:** players

לדוגמה:

'Name Blue, Score team: 87, Players: Name: shir, Age: 26, Hand: [2|6][4|5][3|5][0|5][0|6][0|1][1|4], Score: 42, I can win the game! Name: chen, Age: 26, Hand: [0|4][1|5][5|5][2|5][3|6][1|2][2|4], Score: 45'

בדיקה סופקה בקובץ Tests.py. 🚀

def __repr__(self):

השיטה תחזיר מחרוזת המייצגת את הקבוצה כפי שהוגדר בשיטה str.

פלט:

○ **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט עבור השיטה __str__.

משחק (Game)

המחלקה משחק (Game) מגדירה את משחק הדומינו. המחלקה מוגדרת על ידי השדות הבאים:

❖ **board** - לוח משחק דומינו.

❖ **team1** - קבוצה ראשונה.

❖ **team2** - קבוצה שנייה.

ממשו את המחלקה Game:

```
def __init__(self, board, team1, team2):
```

בנאי המאתחל את שדות המשחק.

קלט:

○ **board [Board]** - לוח משחק דומינו. הניחו כי הלוח מוגדר על ידי max_capacity בין 1 ל-

28 וכי הלוח מכיל רק אובייקטים מסוג Domino עם ערכים בין 0 ל-6.

○ **team1 [Team]** - קבוצה ראשונה. הניחו כי הקבוצה מאותחלת לפי הנחות הקלט של הבנאי של Team.

○ **team2 [Team]** - קבוצה שנייה. הניחו כי הקבוצה מאותחלת לפי הנחות הקלט של הבנאי של Team.

ניתן להניח שכל הערכים או האובייקטים אינם None או מכילים שדות/ערכים שערך None.

```
def play(self):
```

שיטה שמנהלת את המשחק. הקבוצה הראשונה שתשחק תהיה team1 ולאחריה תשחק team2. כל קבוצה תנסה בתורה להוסיף אבן דומינו ללוח (board). המשחק מסתיים כאשר לאחת מהקבוצות נגמרו האבנים או שאף קבוצה לא יכולה להניח אבן דומינו בלוח. הלוח (board) מתעדכן במהלך המשחק וכך גם ה-hand של השחקנים.

פלט:

○ **[str]** - מחרוזת המגדירה איזו קבוצה ניצחה. הקבוצה המנצחת מוגדרת כקבוצה שה-score שלה הוא הנמוך ביותר. באדום מסומנות המילים הקבועות.

אם team1 ניצחה יוחזר:

"Team team1.name wins Team team2.name"

אם team2 ניצחה יוחזר:

"Team team2.name wins Team team1.name"

אם לשתי הקבוצות score זהה יוחזר:

"Draw!"

לדוגמה:

'Team Red wins Team Blue'

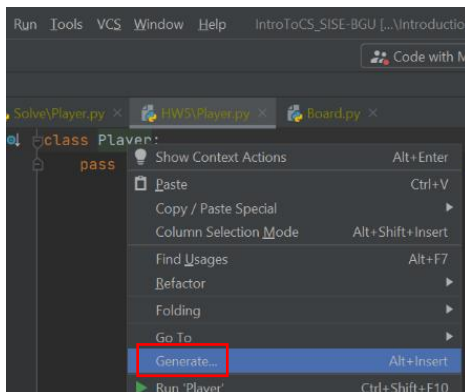
בדיקה סופקה בקובץ Tests.py. 🚦

חלק ג' - Unit test

זאת הפעם הראשונה שהינכם מתמודדים עם מספר רב של קבצים, שיטות ומחלקות. על מנת שתוכלו לבדוק את התרגיל שלכם בצורה קלה ונוחה תשתמשו בכלי בדיקות תכנה שנקרא **unit test**, שימש אתכם בהמשך דרככם המקצועית. אנחנו מעודדים אתכם להרחיב את הידע שלכם בנושא מעבר למה שתלמדו בעבודה ובתרגול.

שלבם לפתיחת קובץ Test

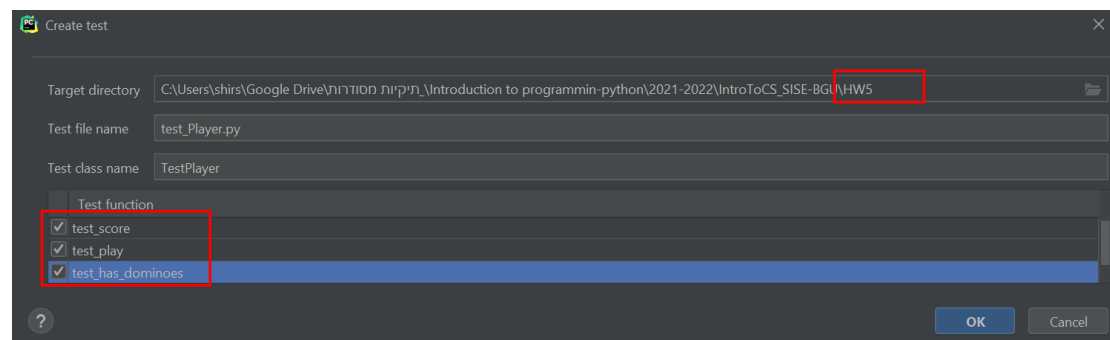
על מנת לפתוח קובץ test בצורה נוחה בצעו את השלבים הבאים:
לחצן ימיני בעכבר על שם המחלקה ובחירה ב- Generate. לאחר מכן, בחרו test.



יפתח החלון הבא:

בתיקית היעד (target directory) הגדירו את התיקיה hw5 שסופקה לכם.
שם הקובץ ושם המחלקה נתון לבחירתכם.

סמנו ב- V את כל השיטות המופיעות תחת קטגוריות test function על מנת ליצור טסטים מובנים.



לאחר לחיצה על OK יוצר הקובץ הבא:

```
from unittest import TestCase

class TestPlayer(TestCase):
    def test_score(self):
        self.fail()

    def test_play(self):
        self.fail()

    def test_has_dominoes(self):
        self.fail()
```

שימו לב כי לא נוצרו טסטים אוטומטיים לשיטות שדרסתם (override) כמו `__str__` ועליכם ליצר להם טסטים בצורה ידנית.

שלבם להגדרת test חדש

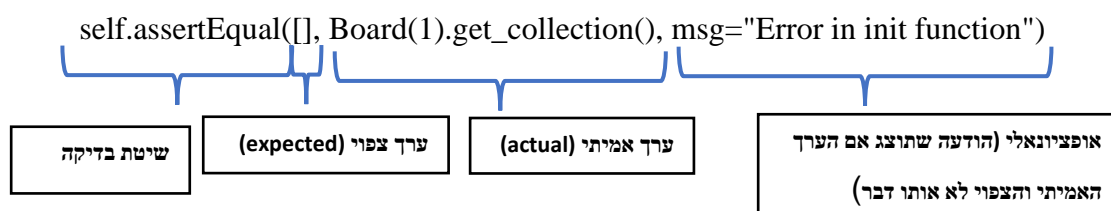
כדי ליצור טסט חדש נגדיר פונקציה כך ששם הפונקציה מתחיל עם המילה `test`. למשל:

```
def test_str(self):
```

בתוך הפונקציה נבצע תרחיש מסוים שבודק שיטה בקוד. למשל, בדוגמה אנו בודקים כי הבנאי של המחלקה `Board` מבוצע כהלכה.

על מנת להריץ את ה-`test`, עלינו להשתמש באובייקט `self` ובשיטת בדיקה.

למשל, בדוגמה:

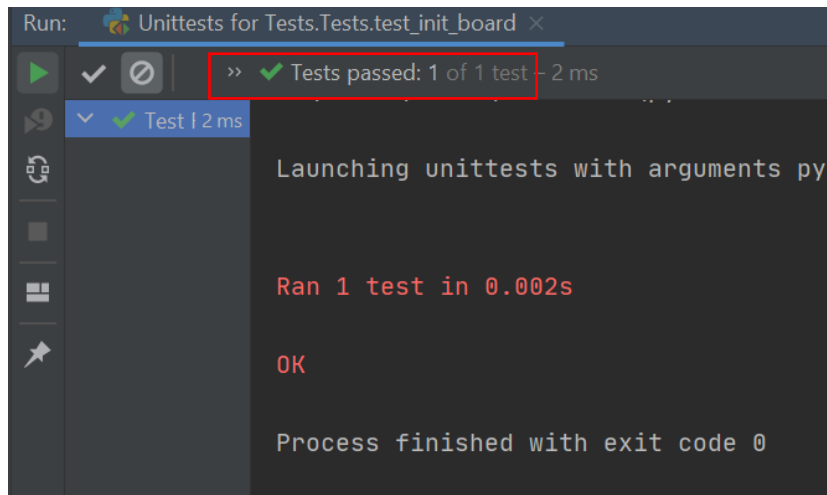


מוזמנים לקרוא על כל המתודות להשוואה [בקישור](#).

ניתן להריץ טסט בודד על ידי לחיצה על החץ הירוק בצד שמאל ליד שם הפונקציה:

```
class Tests(TestCase):
    def test_init_board(self):
        self.assertRaises(InvalidNumberException, Board, 0)
        self.assertEqual([], Board(1).get_collection())
```

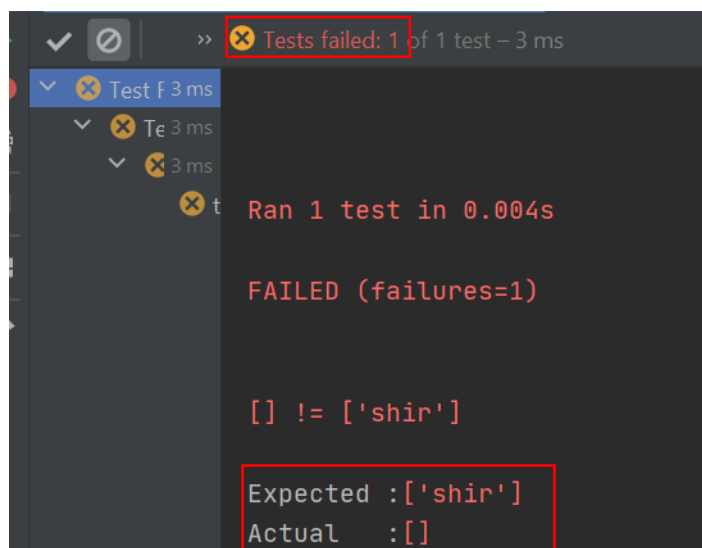
לאחר ההרצה נקבל פלט כי הטסט עבר:



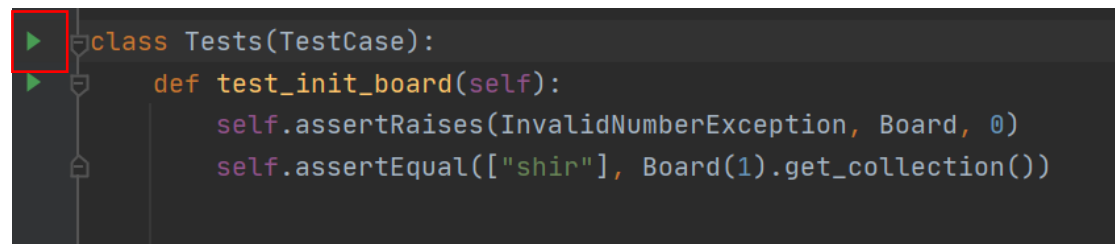
אם למשל נשנה את הטסט לטסט שגוי:

```
class Tests(TestCase):
    def test_init_board(self):
        self.assertRaises(InvalidNumberException, Board, 0)
        self.assertEqual(["shir"], Board(1).get_collection())
```

כאשר נריץ נקבל הודעה כי הטסט נכשל:



על מנת להריץ את כל הטסטים לחצו על החץ הירוק ליד שם המחלקה:



```
class Tests(TestCase):
    def test_init_board(self):
        self.assertRaises(InvalidNumberException, Board, 0)
        self.assertEqual(["shir"], Board(1).get_collection())
```

מחלקת Test לדוגמה

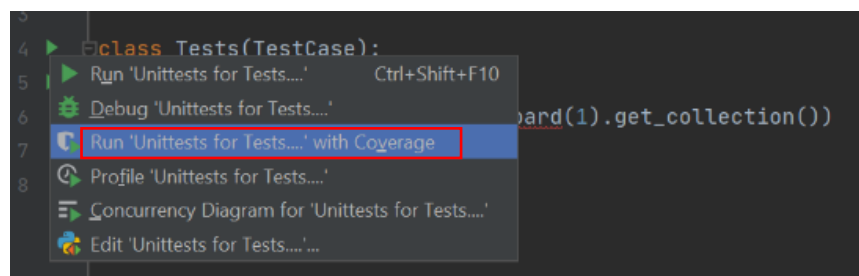
סיפקנו לכם מחלקה בשם Tests.py שמכילה דוגמאות לטסטים שבהם תוכלו לבדוק מספר שיטות שונות. שימו לב, המחלקה תוכל לרוץ רק לאחר שסיימתם לממש את כל הדרישות של העבודה.

דרישות התרגיל

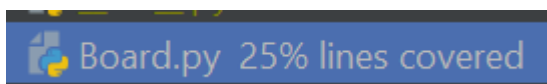
כחלק מדרישות העבודה עליכם לכתוב קובץ test עבור כל מחלקה שהוגדרה בעבודה (מלבד מחלקת ה-Exception).

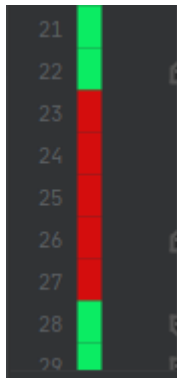
המלצה חמה: לכל מחלקה נסו לכתוב טסטים לפני שאתם כותבים את הקוד על פי השיטה TDD (test driven development) שהוצגה בתרגול 7. באופן הזה אתם כותבים את הקוד כדי שיעבור את הטסטים ולא להיפך. זוהי שיטה מקובלת לפיתוח תכנה.

כדי שתוכל לדעת האם הטסטים שלכם מכסים את מרבית הקוד שכתבתם תוכלו להשתמש באופציית ה-coverage. לחצן ימני על החץ הירוק ובחירה "run with coverage".



לאחר ההרצה תוכל לראות את השורות שהטסטים כיסו ליד שם המחלקה. למשל:
















בנוסף, כאשר תיכנסו לקובץ של המחלקה תוכלו לראות את הצבעים הבאים:

- ירוק – הטסט עבר בשורה זאת ולכן השורה נבדקה.
 - אדום – הטסט לא עבר בשורה זאת ולכן השורה לא נבדקה.
- לקריאה נוספת על ה-coverage מוזמנים לקרוא [כאן](#).

עליכם לכתוב עבור כל מחלקה קבצי טסט שיכסו לפחות 90% משורות כל מחלקה שכתבתם. לדוגמה:

	Board.py	94% lines covered
	Collection.py	96% lines covered
	Domino.py	96% lines covered
	Exceptions.py	90% lines covered
	Game.py	94% lines covered
	Hand.py	94% lines covered
	MaxScorePlayer.py	100% lines covered
	NaivePlayer.py	100% lines covered
	Player.py	96% lines covered
	RandomPlayer.py	100% lines covered
	Team.py	96% lines covered

שימו לב כי אסור לשתף טסטים בין סטודנטים.

בהצלחה ועבודה מהנה! 😊