

**תרגיל בית מס' 2 - להגשה עד 27/11/2025 בשעה 23:59**

קראו בעיון את הנחיות העבודה וההגשה המופיעות באתר הקורס, תחת התיקייה assignments.  
חריגת מההנחיות תגרור ירידת ציון / פסילת התרגיל.

**הגשה :**

- תשובהຕיכם יוגש בקובץ pdf ובקובץ עק בהתאם להנחיות בכל שאלה.
  - השתמשו בקובץ השלד skeleton2.py כבסיס לקובץ ה-עך אותו אתם מגישים.
  - לא לשכוח לשנות את שם הקובץ **למספר ת"ז** **שלכם** לפני ההגשה, עם סיום **.py**.
  - בסה"כ מגישים שני קבצים בלבד. עבור סטודנטית שמספר ת"ז שלה הוא 012345678 הקבצים שיש להגיש הם **.hw2\_012345678.pdf** ו- **.hw2\_012345678.py**.
  - הקפידו לענות על כל מה שנשאלתם.
  - לפני ההגשה ודאו כי הרצטם את הפונקציה **( ) test** שבקובץ השלד אך זכרו כי היא מבצעת בדיקות בסיסיות בלבד וכי בתהליך הבדיקה הקוד יבדק על פני מקרים מגוונים ומורכבים יותר.
  - שימו לב כי יש למחוק את **הקריאה לפונקציה** לפני ההגשה.
  - בכל שאלה, אלא אם מצוין אחרת באופן מפורש, ניתן להניח כי הקלט תקין.
  - תשיבות מילוליות והסבירים צריכים להיות תמציתיים, קולעים וברורים.
  - להנחה זו מטרה כפולה :
    1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
    2. כדי להרגיל אתכם להבעת טיעונים באופן מתומנת ויעיל, ללא פרטים חסרי הצד אחד ללא עודף.
- בלתי הכרחי מצד שני. זהה פרקטיקה חשובה במדעי המחשב.

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025-2026**

**שאלה 1**

בתרגילים הבאים נזכיר שלושה טיפוסי אוסףים בפייתון – tuple, dict, set –. מטרת התרגילים היא להכיר את הטיפוסים החדשניים הללו; ניתן ומומלץ להשתמש בחיפוש בגוגל, בטייעוד הרשמי של פייתון, וגם במודול שפה כגון ChatGPT. **שימו לב:** בשאלת זו יש לצרף את כל התשובות ב-pdf שתגישו (כולל קטע הקוד קצר בסעיף ג').

**סעיף א' – טיפוס tuple**

הטיפוס tuple מייצג רצף (אוסף סדור, ordered collection) של איברים בדומה לlist, אך בניגוד לשימושה, טיפוס זה הוא בלתי ניתן לשינוי (immutable). בהמשך התרגילים נבין בין היכן הבדל זה עשוי להיות שימושי. וודאו כי אתם מזוהים את השוני באיתחול הטיפוס למול איתחול רשותה.

1. הריצו את הקוד הבא :

```
a = (2, 3, 4)
b = [2, 3, 4]
print(type(a))

print(a == b)
print([a[i] == b[i] for i in range(3)])

b[0] = 99
print(b)

a[0] = 99
```

1.1 מה הודפס על המסך?

1.2 מדוע ההשמה 99 = [0]a גורמת לשגיאה?

2. עד כה ראיינו פונקציות בפייתון אשר מחזירות ערך ייחיד. הריצו את הפונקציה הבאה ובחנו כיצד ניתן "לעקוב" מגבלה זו בעזרת tuple :

הערה: שימושו לב'קייזור הדרכ' שפייתון מאפשרת לנו בהשמה האחורונה (נקראת "השמה בו זמנית") שבה מבצעים "בבת אחת" השמה ליוטר משתנה אחד).

```
def min_and_max(lst):
    """ Given a list of integers, returns its minimum and maximum """

    return min(lst), max(lst)

result = min_and_max([10, 3, 7, 1, 5])
print(result)
print(type(result))
x = result[0]
y = result[1]
print("min =", x, "max =", y)

x, y = min_and_max([10, 3, 7, 1, 5])
```

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025-2026**

```
print("min =", x, "max =", y)
```

2. מהם הטיפוס והערך המוחזרים מהפונקציה?

**סעיף ב' – טיפוס set**

הטיפוס **set** מייצג אוסף ללא סדר ולא כפליות הדומה לרעיון של קבוצה מתמטית. איברי **set** חייבים להיות בלתי ניתנים לשינוי (**immutable**), מסיבותן שיווגנו בהמשך הקורס. אך הקבוצה עצמה ניתנת לשינוי (**mutable**), כלומר, ניתן להוסיף לה איברים, או למחוק ממנה איברים. הטיפוס **set** מאפשר בוצע פעולות שונות שאופייניות לקבוצות במתמטיקה, ויש לו יתרון שימושו נספּ – חישוב איבר בו מתרחש ביעילות רבה (עוד על כך בחלק האחרון של תרגיל זה).

1. הריצו את הקוד הבא:

```
s = {1, 2, 3, "a", 0, 3.14}
print(type(s))
print(s)
s.add(3)
print(s)
s.add("3")
print(s)
print(s.intersection({1, 11, 111}))
print(s.union({1, 11, 111}))

for elem in s:
    print(elem)

print(3 in s)
print(33 in s)
```

1.1 האם מובטח שהסדר שהוגדר בעת איתחול קבוצה נשמר?

1.2 האם ב **set** יכולים להיות איברים כפולים?

2. מצאו לפחות שתי מתודות נוספות של **set** והריצו אותן; ציינו את המתודות הללו ותארו בשפה חופשייה מה הן מבצעות.

**סעיף ג' – טיפוס dict**

מילון (**dictionary**) מייצג אוסף של זוגות מפתח: ערך. ניתן לראות בו מיפוי של קבוצת איברים (ключи, keys) לערכים (values). המפתחות הם ייחודיים (לא חוזרים, כמו איברים של **set**) ובתלי נתונים לשינוי (**immutable**), כמו איברים של **set**, אך הערכים יכולים להיות מכל סוג.

לדוגמא המילון הבא (d) ממפה אותיות למספר סידורי המתאים להם; שימוש לב לאופן בו מתרחש האיתחול:

```
d = {'a': 1, 'b': 2, 'c': 3}
```

במילון, בשונה מרשימה,ograms, גם בה ניתן לאיברים באמצעות סוגרים מרובעים בהם מופיע אינדקס (מיקום), במילון מצינים בסוגרים אלו את המפתח (ואכן במילון אין סדר לאיברים ואין להם מיקום מובטח).

1. הריצו את הקוד הבא:

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025**

```
d = {"France": "Europe", "Germany": "Europe", "Japan": "Asia"}  
print(type(d))  
print(d)  
print("France" in d)  
print("Spain" in d)  
print("Asia" in d)  
  
print(d["Japan"])  
for country in d:  
    print(country, "is in", d[country])  
  
print(d["Spain"])  
print(d["Asia"])  
  
print("Israel" in d)  
d["Israel"] = "Asia"  
print("Israel" in d)  
print(d)
```

1. מה קורה כשמנסים לגשת למפתח שלא קיים? האם ניתן לגשת על פי הערך המתאים לו (value)? על מה עופרת לולאת for שפעילים על מיליון – על המפתחות או על הערכים?

2. השתמשו את הפונקציה הבאה, שמקבלת מחרוזת text ומחזירה מילון שבו המפתחות הם התווים שהופיעו בקלט, ולכלתו מספר המופעים שלו.

```
def char_count(text):  
    d = {}  
    for ch in text:  
        # TODO: COMPLETE THE FUNCTION HERE  
        # ..  
  
    return d
```

דוגמת ריצה:

```
> print(char_count("banana"))  
{ "a": 3, "b": 1, "n": 2 }
```

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025-2026**

**שאלה 2**

בשאלה זו נמשך מספר פונקציות אקראיות תוך שימוש בפונקציה הבסיסית `random.random`, וללא שימוש בפונקציות אחרות מהספירה `random`.

כזכור, הספירה `random` מכילה פונקציה בשם `random` שמחזירה מספר מתייחס בקטע  $(0, 1]$ , כאשר לכל מספר יש סיכוי שווה להיבחר (ליתר דיוק, לכל מספר שפיגטן יודע ליתג' בקטע  $(0, 1]$  יש סיכוי שווה להיבחר). לצורך הפשטות נניח לאורך השאלה שעבור כל מספר  $1 \leq p \leq 0$ , הסיכוי לקבל מספר קטן מ- $p$  בקטע  $(0, 1]$  על ידי הפונקציה `random` הוא  $p$ .

```
>>> import random  
>>> random.random()  
0.13937543523525686  
>>> random.random()  
0.6376812941041776
```

**סעיף א'**

שימוש את הפונקציה `coin()` שמחזירה `True` בסיכוי חצי ו-`False` בסיכוי חצי.

**סעיף ב'**

שימוש את הפונקציה `sample()`. הפונקציה מקבלת רשימה של ערכים  $u$  באורך  $d$ , ורשימה  $u$ , גם באורך  $d$ , שהויה התפלגות-כלומר מתקיים כי כל איבר ברשימה בתחום בין 0 ל-1 וגם כי סכום איברי הרשימה הוא 1 ; בכתיב מתמטי :

$$\sum_{i=0}^{d-1} p[i] = 1 \wedge \forall i: p[i] \in [0,1]$$

על הפונקציה לדגום ולהחזיר איבר  $[i]$  בהסתברות של  $[i]$ . ניתן להניח כי הקלט תקין וכי שתי הרשימות אינן ריקות.

```
>>> sample([5, 3, "s"], [0.1, 0.2, 0.7])  
3 # with probability 0.2  
>>> sample([5, 3, "s"], [0.1, 0.2, 0.7])  
"s" # with probability 0.7
```

**הצעה:** על מנת לוודא שהפונקציה שלכם מגילה כמצופה, ניתן לחתת השראה מדוגמא מההרצאה - בה הרצינו פונקציה אקראית מספר גדול של פעמים ובדקנו את התפלגות הפלטים שלה (אילו ערכים נצפה לשערך בשאלת?).

**סעיף ג'**

בסעיף זה נמשך את בעיית מונטי-הול המפורסמת. הבעיה מבוססת על שעשווען המורכב משחקן, מנהה, 3 דלתות, 2 סלעים, ומכונית אחת – היא הפרט השווה. מהלך המשחק :

1. המנהה בוחר באקראי דלת אחת מתוך ה-3 ומצביע מאחוריה את **המכונית** (ambil שהשחקן יודע באיזה דלת בחר המנהה). מאחוריו כל אחת משלתי הדלתות האחרות הוא מצביע סלא.
  2. לאחר מכן השחקן נכנס לחדר, ובוחר דלת **אקרדיאת אחת**, מאחוריה הוא מהמר שיש מכונית.
  3. כעת המנהה פותח (וחושף) דלת אחת באקראי מבין הדלתות שהשחקן לא בחר ושמאחוריה יש סלע (יתכן שיש רק דלת אחת צזו ואז המנהה יבחר אותה).
  4. בשלב זה נשארו רק שתי דלתות סגורות, ולשחקן ניתנת האפשרות לבחור האם ברצונו **להחליף** את הדלת שבחר בשלב 2 (לדלת הסגורה השנייה שנותרה) או לא.
  5. בסוף, המנהה חושף את הדלת שהשחקן בחר, רק אם מסתתרת מאחוריה מכונית השחקן זוכה בפרס השווה.
- נרצה להעריך מה האסטרטגייה האידיאלית לשחקן: האם בשלב 4 כדאי לו להחליף את הדלת או לא? לשם כך נמשו את הפונקציה `monty_hall(switch, times)`. על

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025-2026**

הfonקציה לבצע סימולציה של המשחק המתואר לעליה במשך *times* פעמים, כאשר בשלב 4 השחקן בוחר אם לשנות את הדלת או לא לפי הערך של *switch* (כלומר אם *switch* שווה *True* השחקן ישנה את הדלת הנבחרת בשלב 4, ואחרית הוא לא ישנה את הדלת שבחר). בסוף, הפונקציה מחזירה את סיכויי ההצלחה של האסטרטגיה *switch* (כלומר את כמות הסימולציות בהן השחקן זכה במכונית, חלקי *times*). לדוגמה: אם  $times = 10$  והשחקן זכה במכונית ב-5 סימולציות, הפונקציה תחזיר 0.5. דוגמת הרצה:

```
>>> monty_hall(True, 10)
```

0.5

נסו להריץ את הפונקציה עם הפרמטרים (True, 10000), *monty\_hall(False, 10000)*. כתבו בקובץ pdf את אחוזי ההצלחה שייצאו לכם עבור כל אחת מהאסטרטגיות. האם יש אסטרטגיה עדיפה לשחקן? (שווות: נסו להסביר למה זו התוצאה המתבקשת. בקורס בהסתברות בוודאי תחזרו לדוגמה זו ותוכחו את התוצאה הידועה).

#### סעיף ד'

השתמשו בסעיף ב' על מנת למש את הפונקציה (*st*) *sample\_anagram*, אשר מקבלת מחרוזת *s* בעלת תוים ייחודיים (= כל TWO שמויע בא, יופיע בה פעם אחד) ומחזירה **אנגרמה אקראית** שלה (במילים אחרות, **פרומטיצה אקראית** של רצף התווים הנתון), כאשר כל אנגרמה נדרשת להיות מוחזרת בסיכוי שווה.

לדוגמה, עבור המחרוזת 'abc' הפונקציה תחזיר את אחת מש המחרוזות הבאות בסיכוי שווה:

```
'abc', 'acb', 'bac', 'bca', 'cba', 'cab'
```

דוגמאות הרצה:

```
>>> sample_anagram("abcde")
"abcde" # or any other anagram with the same probability...
```

```
>>> sample_anagram("abcde")
"cadeb"
```

```
>>> sample_anagram("SILENT_PAR")
"PANTSILER"
```

עזרה: ניתן ומומלץ, אם כי לא חובה, להשתמש בפונקציה *sample(n, p)* שמימושם בסעיף ב'.

**תזכורת מטלה 1: אנגרמה** היא מחרוזת שנוצרה מסידור מחדש של תווי מחרוזת אחרת (ללא הוספה/מחיקת אף תו). לדוגמה המחרוזת *silent* היא אנגרמה של המחרוזת *listen* (וההפך). בפרט, כל מחרוזת היא אנגרמה (טריוויאלית) של עצמה, וכל היפוך מחרוזת הוא אנגרמה של המחרוזת המקורית.

#### סעיף ה'

וודאו את נכונות הפונקציה שמימושם בסעיף ד': הריצו את הפונקציה 10,000 פעמים על הקלט 'abc', ופרטו בטבלה בקובץ PDF כמה פעמים כל אחת מ-6 המחרוזות האפשרות התקבלה כפלט של הפונקציה. לאור התוצאות שצורפו, האם המימוש שלכם תקין? ציינו זאת ואת הסיבה לכך.

עזרה: כתבו קוד בפייתון שבוצע את הבדיקה. וודאו כי מחקתם את הקוד הרלוונטי לסעיף זה לפני ההגשה.

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025-2026**

**שאלה 3**

בשאלה זו עוסוק בIMPLEMENTATION פועלות ארכיטקטניות על מספרים שלמים ואי שליליים בייצוג בינארי.  
להלן מספר הערות והנחיות התקפות לכל הטעיפים בשאלה:

- לאורך השאלה נייצג מספרים בינאריים באמצעות המכילה את התווים "0" ו-"1" בלבד.
- לאורך השאלה אין לבצע המרה של אף מספר בינארי לבסיס עשרוני או לכל בסיס אחר. בפרט, אין להשתמש כלל פונקציות `str` ו-`int` של פיתון או בפונקציה `convert_base` שראינו בתרגול 3.
- לאורך השאלה, ניתן להניח כי מהירות הניתנתה כקלט היא "תקינה", כלומר, מכילה אך ורק את התווים "0" ו-"1", וכי התו השמאלי ביותר במקורוות הוא "1" (מלבד המחרוזת "0" אשר מייצגת את המספר 0). בפרט, מהירות למספר שאינו אפס לא תכיל אפסים מובילים ומהירות המייצגת את אפס תכיל "0" ייחיד.
- לכל פונקציה בשאלה אשר מחזירה כפלט מהירות בינארית יש לוודא כי המחרוזות תקינה על פי ההגדרה הקודמת. (למשל, הפלטים "0100" ו-"000" אינם תקינים ואילו הפלטים "100" ו-"0" תקינים).
- לאורך השאלה נusband עם מספרים אי-שליליים בלבד. בפרט, ניתן להניח כי המחרוזות הבינאריות הניתנות כקלט לפונקציות השונות מייצגות מספרים אי-שליליים בלבד.
- הרצה של הפונקציות בשאלה על מהירות באורך של עד 10 ספרות צריכה להסתiens בזמן קצר (לכל היוטר שנייה).

**סעיף א'**

ممשו את הפונקציה `inc(binary)` (קיוצר של `increment`) אשר מקבלת מהירות המציינת מספר שלם אי שלילי בכתיב בינארי (כולומר מהירות המורכבות מאפסים ואחדות בלבד). הפונקציה תחזיר מהירות המייצגת את המספר הבינארי לאחר תוספת של 1.  
להלן הממחשה של אלגוריתם החיבור של מספרים בינאריים (בדומה לחיבור מספרים עשרוניים עם נשא (carry)):

```
    1 (carried digits)
+ 1 0 1 (binary)
-----+
= 1 1 0
```

**הנחייה מחייבת**: יש למש את האלגוריתם בהתאם להמחשה: ישירות באמצעות לולאות.  
דוגמאות הרצה:

```
>>> inc("0")
'1'
>>> inc("1")
'10'
>>> inc("101")
'110'
>>> inc("111")
'1000'
>>> inc(inc("111"))
'1001'
```

**סעיף ב'**

ممשו את הפונקציה `add(bin1, bin2)` אשר מקבלת שתי מהירות המציינות מספרים שלמים בכתיב בינארי (כולומר מהירות המורכבות מאפסים ואחדות בלבד). הפונקציה תחזיר מהירות המייצגת את המספר הבינארי המתקבל מחיבור `bin1`-`bin2`.

**הנחייה מחייבת**: יש למש את האלגוריתם בהתאם להמחשה בסעיף א': ישירות באמצעות לולאה ואין להשתמש בפונקציה `inc`.  
דוגמאות הרצה:

```
>>> add("1", "0")
'1'
```

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6**

```
>>> add("1", "1")
'10'
>>> add("11", "110")
'1001'
```

**סעיף ג'**

משו את הפונקציה (*mod\_two*) אשר מקבלת מחרוזות המייצגת מספר בינארי אי שלילי, *binary*, ומספר אי שלילי (מтиיפוס *int*), *power*. הפונקציה תחזיר במחuzeות את הייצוג הבינארי של *binary* מודולו 2 בחזקת *power*, זאת אומרת שהפונקציה תחזיר את הייצוג הבינארי של  $(2^{power} \% binary)$ .

```
>>> mod_two("11110", 3)
```

```
'110'
```

```
>>> mod_two("1001", 1)
```

```
'1'
```

```
>>> mod_two("100", 2)
```

```
'0'
```

```
>>> mod_two("100", 4)
```

```
'100'
```

**סעיף ד'**

משו את הפונקציה (*max\_bin*) אשר מקבלת רשימה *lst* המכילה  $k \geq 2$  (לט) מחרוזות המייצגות מספרים שלמים אי-שליליים בכתב בינארי (כלומר מחרוזות המורכבות מאפסים ואחדות בלבד). הפונקציה תחזיר את המחרוזת הבינרית **המייצגת** את המספר הגדול ביותר בראשימה (ניתן להניח שכל המחרוזות שונות ותקינות).

```
>>> max_bin(["1010", "1011", "1011"])
```

```
"1011"
```

```
>>> max_bin(["10", "0", "1"])
```

```
"10"
```

**סעיף ה'**

השאלות הבאות עוסקות בשיטת ייצוג ה-*floating point* (ב-64 ביטים ; IEEE 754) כפי שנראית בכתיבה. עבור כל שאלה ציינו והסבירו את תשובהכם **בקובץ ה-PDF**.

**סעיף ה' 1**

כמה מספרים בטוחה ( $1 + 2^{10}$ ,  $2^{10}$ ] ניתן לייצג בשיטה הניל?

**סעיף ה' 2**

הסבירו את התוצאות הבאות, וכייזד הן מסתדרות עם שיטת הייצוג הניל. התיחסו בתשובתכם לגודל המרווה המומוצע בין שני מספרים הניתנים לייצוג אשר נמצאים בין שתי חזקות עוקבות של 2.

```
>> 2.0**53
```

```
9007199254740992.0
```

```
>> 2.0**53+1
```

```
9007199254740992.0
```

```
>> 2.0**53+2
```

```
9007199254740994.0
```

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6 2025-2026**

**שאלה 4**

בשאלה זו נדרש למצוא שעת קבלה שתספק את כלם, אך כמו תמיד – לא בטוח שנצלה.  
לצורך הפשטות נתרכז **ביום ספציפי** בשבוע בו אנו מעוניינים לקבוע את השעה.

ניצג **סלוט זמן** (למשל שיעור) ע"י tuple באורך 2 (התחלה של השיעור וסוף השיעור, בסדר זה, ובאורך זמן חיווי). לצורך פשטות נניח כי כל שיעור מתייחס ומסתיים בשעה עגולה שתווצג ע"י מספר שלם ב-tuple. למשל סלוט הזמן שבין 15:00 עד 18:00 יוצג ע"י –

(15, 18)

ניצג את **מערכת שעות** של סטודנטית ביום המذובר כרשימה student\_schedule וכל איבר בה מייצג סלוט של זמן בו הסטודנטית בשיעור, לדוגמא :

```
student_schedule = [(8, 10), (10, 12), (15, 18)]
```

במקרה זה הסטודנטית לוקחת שיעור מ-00:08 עד 00:10, לאחר מכן שיעור נוסף מ-00:10 עד 00:12 ומסיימת את היום עם שיעור מ-00:15 עד 00:18. ניתן להניח כי, כמו בדוגמה, אין חפיפות בין השיעורים והרשימה ממוקנת לפי זמני תחילת השיעורים.

הרשימה יכולה להיות גם ריקה, במקרה זה אין לסטודנטית שיעורים באותו יום.

כשמתකבת החלטה על שעת הקבלה, כמובן, יש לשקלן מערכות שעوت של סטודנטים שונים. ניצג את **אוסף מערכות שעות** שלהם ע"י מילון, student\_schedules\_dict, כך שם הסטודנט ממופף למערכת השעות שלו. ניתן להניח כי המילון אינו ריק (כלומר מכיל לפחות סטודנט אחד). לדוגמא :

```
student_schedules_dict = {  
    'noam': [(8, 10), (10, 12), (15, 18)],  
    'larry': [(10, 11), (14, 16)],  
}
```

את **שעת הקבלה** ביום המذובר, office\_hour, ניצג כסלוט-זמן בן שעה אחת.

בשאלה ניתן להניח את תקינות הקלטים על פי הפורמט המתואר.

**סעיף א'**

משו את הפונקציה is\_student\_available(office\_hour, student\_schedule) המקבלת סלוט זמן פוטנציאלי לשעת הקבלה ומערכת שעות של סטודנט מסוים (רשימה כנ"ל, ומחזירה True אם הסטודנט זמין לשעת הקבלה, אחרת False).

**סעיף א''**

משו את הפונקציה assess\_office\_hour(office\_hour, student\_schedules\_dict) שמקבלת את זמן שעת הקבלה שברצוננו להערך, ואת מילון מערכות השעות של הסטודנטים לאותו יום (בפורמט המתויר לעיל). הפונקציה תחזיר tuple אשר : ערכו הראשון הוא רשימת השמות שזמינים להגיע לשעת הקבלה, וערךו השני הוא יחס של סטודנטים שזמינים להגיע לשעת הקבלה מבין כל הסטודנטים.

הערה : ניתן ורצוי להשתמש בסעיף הקודם.

דוגמת הרצה :

```
>> office_hour = (11, 12)  
>> student_schedules_dict = {
```

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף-6**

```
'noam': [(8, 10), (10, 12), (15, 18)],  
'larry': [(10, 11), (14, 16)],  
}  
  
>> assess_office_hour(office_hour, student_schedules_dict)  
(['larry'], 0.5)
```

**סעיף ב'**

נסהה בכל זאת לספק את כולם. נרצה לדעת באילו שעות במהלך היום נוכל לקיים שעת קבלה עבורה **כל** הסטודנטים זמינים.

**סעיף ב'2**

משוו את פונקציית העזר `merge_intervals(intervals)` שמקבלת רשימה של אינטראולים (interval) שלמים, כלומר כל איבר ברשימה הוא tuple באורך 2 של מספרים שלמים בסדר עולה (דוגמא לאינטראול: (-2, 43)). על הפונקציה להחזיר רשימה של אינטראולים **זרים** אשר **מominת** לפי ערך התחלה של כל אינטראול.

הפונקציה תעשה זאת ע"י מייזוג כל האינטראולים החותכים זה את זה. למשל, האינטראול (-2, 43) **חותך** את האינטראול (20, 52), וניתן **mezgem** לאינטראול (52, -2).

**רמז:** נסו לחשב על סדר בו יהיה נכון לעבור על רשימות האינטראולים. בהינתן סדר כזה, נסו למנוע את המקרים השונים (המעטים) שייתכנו בעת מעבר על הרשימה.

**דוגמאות הרצה:**

```
>> merge_intervals([(−2, 43), (−700, −9), (20, 52), (52, 60)])  
[(−700, −9), (−2, 60)]  
  
>> merge_intervals([(−2, 43), (−700, −9)])  
[(−700, −9), (−2, 43)]  
  
>> merge_intervals([(8, 10), (10, 12), (10, 11), (15, 18), (14, 16)])  
[(8, 12), (14, 18)]
```

**סעיף ב'2**

משוו את הפונקציה `find_perfect_slots(student_schedules_dict)` שמקבלת את מערכות השעות של כל הסטודנטים, ומוציאיה את כל הسلطים בני-שעה שבהם **כל** הסטודנטים זמינים לשעת הקבלה, או רשימה ריקה אם אין כאלה. הניחו כי שעת הקבלה יכולה **להתחליל** לכל הזמן בין 00:00 ו-19:00.

**הנחייה:** יש להשתמש בפונקציית העזר `shimshutim` בסעיף ב'1.

**דוגמאות הרצה:**

```
>> student_schedules_dict = {  
    'noam': [(8, 10), (10, 12), (15, 18)],  
    'larry': [(10, 11), (14, 16)],
```

**אוניברסיטת תל אביב - בית הספר למדעי המחשב**  
**מבוא מורחב למדעי המחשב, חורף 6-2025**

}

```
>> find_perfect_slots(student_schedules_dict)  
[(7, 8), (12, 13), (13, 14), (18, 19), (19, 20)]
```

## שאלה 5

לפניכם קטע קוד :

```
x = 8
y = 'cs'
z = x
y = x
y += 12
lst1 = [x, z]
lst2 = [x, y, lst1]          # breakpoint 1
def what(x, lst):
    x -= 10
    lst[0] = "i"
    lst2 = [x, y, what]      # breakpoint 2
    return lst2
lst1 = lst1 + lst2 * 2       # breakpoint 3
lst3 = what(x, lst1)        # breakpoint 4
```

צירו והסבירו **בקובץ ה-PDF** את תמונה הזיכרון מבחן מרחיב הכתובות ומרחיב השמות לאחר שהמפרש מבצע כל אחת מהפעולות בהן מופיעה הערה [breakpoint](#). על התשושים להיות דומה לאופן בו הצגנו את תמונה הזיכרון בciteth.

בשאלה זו ניתן לצייר בכתב יד ולסroke באופן ברור את הציגו. הקפידו שהציגו יהיה ברור לחולティ. יש לזכור, כפי שנאמר בהרצאה, שהאתר “python tutor” לא תמיד משקף באופן מדויק את תמונה הזיכרון ולכן מומלץ לבחון את הדברים באמצעות בדיקת כתובות בזיכרון. בנוסף, שימוש לב ש-[breakpoint 2](#) יש לצייר את כל תמונה הזיכרון ולא רק את הסקופ (scope) הפנימי של הקריאה לפונקציה.