

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

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

**הגשה:**

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton2.py כבסיס לקובץ ה-py אותו אתם מגישים. לא לשכוח לשנות את שם הקובץ למספר ת"ז שלכם לפני ההגשה, עם סיומת 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 הרצתם את הפונקציה test () שבקובץ השלד אך זכרו כי היא מבצעת בדיקות בסיסיות בלבד וכי בתהליך הבדיקה הקוד ייבדק 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.
- תשובות מילוליות והסברים צריכים להיות תמציתיים, קולעים וברורים. להנחיה זו מטרה כפולה:
  1. על מנת שנוכל לבדוק את התרגילים שלכם בזמן סביר.
  2. כדי להרגיל אתכם להבעת טיעונים באופן מתומצת ויעיל, ללא פרטים חסרים מצד אחד אך ללא עודף בלתי הכרחי מצד שני. זוהי פרקטיקה חשובה במדעי המחשב.

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

### שאלה 1

בתרגיל הבא נכיר שלושה טיפוסים אוספים בפייתון – tuple, dict, set. מטרת התרגיל היא להכיר את הטיפוסים החדשים הללו; ניתן ומומלץ להשתמש בחיפוש בגוגל, בתיעוד הרשמי של פייתון, וגם במודלי שפה כגון ChatGPT.

**שימו לב:** בשאלה זו יש לצרף את כלל התשובות ב-pdf שתגישו (כולל קטע הקוד הקצר בסעיף ג').

#### סעיף א' – טיפוס tuple

הטיפוס tuple מייצג רצף (אוסף סדור, ordered collection) של איברים בדומה ל list, אך בניגוד לרשימה, טיפוס זה הוא בלתי ניתן לשינוי (immutable). בהמשך התרגיל נבין היכן הבדל זה עשוי להיות שימושי. וודאו כי אתם מזהים את השוני באיתחול הטיפוס למול איתחול רשימה.

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 `a[0] = 99` גורמת לשגיאה?

2. עד כה ראינו פונקציות בפייתון אשר מחזירות ערך יחיד. הריצו את הפונקציה הבאה ובחנו כיצד ניתן "לעקוף" מגבלה זו בעזרת tuple:

הערה: שימו לב ל"קיצור הדרך" שפייתון מאפשרת לנו בהשמה האחרונה (נקראת "השמה בו זמנית" שבה מבצעים "בבת אחת" השמה ליותר ממשתנה 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 2025-6

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

2.1 מהם הטיפוס והערך המוחזרים מהפונקציה?

### סעיף ב' – טיפוס set

הטיפוס `set` מייצג אוסף ללא סדר וללא כפילויות הדומה לרעיון של קבוצה מתמטית. איברי `set` חייבים להיות בלתי ניתנים לשינוי (`immutable`), מסיבות שיובנו בהמשך הקורס. אך הקבוצה עצמה ניתנת לשינוי (`mutable`), כלומר, ניתן להוסיף לה איברים, או למחוק ממנה איברים. הטיפוס `set` מאפשרת לבצע פעולות שונות שאופייניות לקבוצות במתמטיקה, ויש לו יתרון משמעותי נוסף – חיפוש איבר בו מתבצע 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 חופשית מה הן מבצעות.

### סעיף ג' – טיפוס dict

מילון (`dictionary`) מייצג אוסף של זוגות *מפתח: ערך*. ניתן לראות בו מיפוי של קבוצת איברים (מפתחות, `keys`) לערכים (`values`). המפתחות הם ייחודיים (ללא חזרות, כמו איברים של `set`) ובלתי ניתנים לשינוי (`immutable`), כמו איברים של `set`, אך הערכים יכולים להיות מכל סוג.

לדוגמא המילון הבא (`d`) ממפה אותיות למספר סידורי המתאים להם; שימו לב לאופן בו מתבצע האיתחול:

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

במילון, בשונה מרשימה, שגם בה ניגשים לאיברים באמצעות סוגריים מרובעים בהם מופיע אינדקס (מיקום), במילון מציינים בסוגריים 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 2025-6**

```
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)
```

1.1 מה קורה כשמנסים לגשת למפתח שלא קיים? האם ניתן לגשת למפתח על פי הערך המתאים לו (value)?  
על מה עוברת לולאת for שמפעילים על מילון – על המפתחות או על הערכים?

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

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

    return d
```

דוגמת הרצה:

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

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

## שאלה 2

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

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

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

### סעיף א'

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

### סעיף ב'

ממשו את הפונקציה `sample(v, p)`. הפונקציה מקבלת רשימה של ערכים  $v$  באורך  $d$ , ורשימה  $p$ , גם באורך  $d$ , שמהווה התפלגות-כלומר מתקיים כי כל איבר ברשימה תחום בין 0 ל-1 וגם כי סכום איברי הרשימה הוא 1; בכתוב מתמטי:

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

על הפונקציה לדגום ולהחזיר איבר  $v[i]$  בהסתברות של  $p[i]$ . 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 מגרילה כמצופה, ניתן לקחת השראה מדוגמא מההרצאה - בה הרצנו פונקציה אקראית מספר גדול של פעמים ובדקנו את התפלגות הפלטים שלה (אילו ערכים נצפה לשערך בשאלה?).

### סעיף ג'

בסעיף זה נממש את בעיית מונטי-הול המפורסמת. הבעיה מבוססת על שעשועון המורכב משחקן, מנחה, 3 דלתות, 2 סלעים, ומכונת אחת – היא הפרס השווה. מהלך המשחק:

- המנחה בוחר באקראי דלת אחת מתוך ה-3 ומציב מאחוריה את המכונת (מבלי שהשחקן יודע באיזה דלת בחר המנחה). מאחורי כל אחת משתי הדלתות האחרות הוא מציב סלע.
- לאחר מכן השחקן נכנס לחדר, ובוחר דלת אקראית אחת, מאחוריה הוא מהמר שיש מכונת.
- כעת המנחה פותח (וחושף) דלת אחת באקראי מבין הדלתות שהשחקן לא בחר ושמהאחוריה יש סלע (יתכן שיש רק דלת אחת כזו ואז המנחה יבחר אותה).
- 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.
- בסוף, המנחה חושף את הדלת שהשחקן בחר, רק אם מסתתרת מאחוריה מכונת השחקן זוכה בפרס השווה.

נרצה להעריך מה האסטרטגיה האידאלית לשחקן: האם בשלב 4 כדאי לו להחליף את הדלת או לא? לשם כך ממשו את הפונקציה `monty_hall(switch, times)` אשר מקבלת משתנה בוליאני `switch` ומספר שלם חיובי `times`. על

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

הפונקציה לבצע סימולציה של המשחק המתואר למעלה 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, ואחרת הוא לא ישנה את הדלת שבחר). בסוף, הפונקציה מחזירה את סיכויי ההצלחה של האסטרטגיה  $switch$  (כלומר את כמות הסימולציות בהן השחקן זכה במכונית, חלקי  $times$ ). לדוגמא: 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  $monty\_hall(True, 10000)$ ,  $monty\_hall(False, 10000)$ . כתבו בקובץ  $pdf$  את אחוזי ההצלחה שיצאו לכם עבור כל אחת מהאסטרטגיות. האם יש אסטרטגיה עדיפה לשחקן? (רשות: נסו להסביר למה זו התוצאה המתקבלת. בקורס בהסתברות בוודאי תחזרו לדוגמה הזו ותוכיחו את התוצאה הידועה).

### סעיף ד'

השתמשו בסעיף ב' על מנת לממש את הפונקציה  $sample\_anagram(st)$ , אשר מקבלת מחרוזת  $st$  בעלת תווים ייחודיים (= כל תו שמופיע בא, יופיע בה פעם אחת) ומחזירה אנגרמה אקראית שלה (במילים אחרות, [פרמוטציה אקראית](#) של רצף התווים הנתון), כאשר כל אנגרמה נדרשת להיות מוחזרת בסיכוי שווה.

לדוגמא, עבור המחרוזת  $st = 'abc'$  הפונקציה תחזיר את אחת משש המחרוזות הבאות בסיכוי שווה:

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

דוגמת הרצה:

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

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

>>> sample_anagram("SILENT_PAR")
"PANTS_ILER"
```

הערה: ניתן ומומלץ, אם כי לא חובה, להשתמש בפונקציה  $sample(v, p)$  שמימשתם בסעיף ב'.

תזכורת ממטלה 1: [אנגרמה](#) היא מחרוזת שנוצרה מסידור מחדש של תווי מחרוזת אחרת (ללא הוספה/מחיקת אף תו). לדוגמא 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 שמימשתם בסעיף ד': הריצו את הפונקציה 10,000 פעמים על הקלט  $st = 'abc'$ , ופרטו בטבלה בקובץ ה-PDF כמה פעמים כל אחת מ-6 המחרוזות האפשריות התקבלה כפלט של הפונקציה. לאור התוצאות שצורפו, האם המימוש שלכם תקין? ציינו זאת ואת הסיבה לכך.

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

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

### שאלה 3

בשאלה זו נעסוק במימוש פעולות אריתמטיות על מספרים שלמים ואי שליליים בייצוג בינארי. להלן מספר הערות והנחיות התקפות לכלל הסעיפים בשאלה:

- לאורך השאלה נייצג מספרים בינאריים באמצעות מחרוזות 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 בפונקציות `int` ו-`bin` של פייתון או בפונקציה `convert_base` שראינו בתרגול 3.
- לאורך השאלה, ניתן להניח כי מחרוזות הניתנות כקלט היא "תקינה", כלומר, מכילה אך ורק את התווים "0" ו-"1", וכי התו השמאלי ביותר במחרוזת הוא "1" (מלבד המחרוזת "0" אשר מייצגת את המספר 0). בפרט, מחרוזת למספר שאינו אפס לא תכיל אפסים מובילים והמחרוזת המייצגת את אפס תכיל "0" יחיד.
- לכל פונקציה בשאלה אשר מחזירה כפלט מחרוזת בינארית יש לוודא כי המחרוזת תקינה על פי ההגדרה הקודמת. (למשל, הפלטים "0100" ו-"000" אינם תקינים ואילו הפלטים "100" ו-"0" תקינים).
- לאורך השאלה נעבוד 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 מחרוזות באורך של עד 10 ספרות צריכה להסתיים בזמן קצר (לכל היותר שניה).

### סעיף א'

ממשו את הפונקציה `inc(binary)` (קיצור של increment) אשר מקבלת מחרוזת המייצגת מספר שלם אי שלילי בכתובי בינארי (כלומר מחרוזת המורכבת מאפסים ואחדות בלבד). הפונקציה תחזיר מחרוזת המייצגת את המספר הבינארי לאחר תוספת של 1. להלן המחשה של אלגוריתם החיבור של מספרים בינאריים (בדומה לחיבור מספרים עשרוניים עם נשא (carry)):

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

הנחיה מחייבת: יש לממש 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'

ממשו את הפונקציה `add(bin1, bin2)` אשר מקבלת שתי מחרוזות המייצגות מספרים אי שליליים שלמים בכתובי בינארי (כלומר מחרוזות המורכבות מאפסים ואחדות בלבד). הפונקציה תחזיר מחרוזת המייצגת את המספר הבינארי המתקבל מחיבור `bin1` ו-`bin2`.

הנחיה מחייבת: יש לממש 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 2025-6**

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

### סעיף ג'

ממשו את הפונקציה  $mod\_two(binary, power)$  אשר מקבלת מחרוזת המייצגת מספר בינארי אי שלילי,  $binary$ , ומספר אי שלילי (מטיפוס `int`),  $power$ . הפונקציה תחזיר במחרוזת את הייצוג הבינארי של  $binary$  מודולו 2 בחזקת  $power$ , זאת אומרת שהפונקציה תחזיר את הייצוג הבינארי של  $binary \% (2^{power})$ .

```
>>> mod_two("11110", 3)
'110'
>>> mod_two("1001", 1)
'1'
>>> mod_two("100", 2)
'0'
>>> mod_two("100", 4)
'100'
```

### סעיף ד'

ממשו את הפונקציה  $max\_bin(lst)$  אשר מקבלת רשימה  $lst$  המכילה  $k$  ( $k \geq 2$  שלם) מחרוזות המייצגות מספרים שלמים אי-שליליים בכתוב בינארי (כלומר מחרוזות המורכבת מאפסים ואחדות בלבד). הפונקציה תחזיר את המחרוזת הבינארית המייצגת את המספר הגדול ביותר ברשימה (ניתן להניח שכל המחרוזות שונות ותקינות).

```
>>> max_bin(["1010", "1011"])
"1011"
>>> max_bin(["10", "0", "1"])
"10"
```

### סעיף ה'

השאלות הבאות עוסקות בשיטת ייצוג ה-`floating point` (ב-64 ביטים; IEEE 754) כפי שנראתה בכיתה. עבור כל שאלה ציינו והסבירו את תשובתכם בקובץ ה-PDF.

### סעיף ה'1

כמה מספרים בטווח  $[2^{10}, 2^{10} + 1]$  ניתן לייצג בשיטה הנ"ל?

### סעיף ה'2

הסבירו את התוצאות הבאות, וכיצד הן מסתדרות עם שיטת הייצוג הנ"ל. התייחסו בתשובתכם לגודל המרווח הממוצע 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 2025-6**

#### שאלה 4

בשאלה זו נשאף למצוא שעת קבלה שתספק את כולם, אך כמו תמיד – לא בטוח שנצליח.  
לצורך הפשטות נתרכז ביום ספציפי בשבוע בו אנו מעוניינים לקבוע את השעה.

נציג **סלוט זמן** (למשל שיעור) ע"י tuple באורך 2 (תחילת השיעור וסוף השיעור, בסדר זה, ובאורך זמן חיובי). לצורך פשטות נניח כי כל שיעור מתחיל ומסתיים בשעה עגולה שתיוצג ע"י מספר שלם ב-tuple. למשל סלוט הזמן שבין 15:00 עד ל-18:00 ייוצג ע"י –

(15, 18)

נציג את **מערכת שעות** של סטודנטית ביום המדובר 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-8:00 עד 10:00, לאחר מכן שיעור נוסף מ-10:00 עד 12:00 ומסיימת את היום עם שיעור מ-15:00 עד 18:00. ניתן להניח כי, כמו בדוגמא, אין חפיפות בין השיעורים והרשימה ממוינת לפי זמני תחילת השיעורים.  
הרשימה יכולה להיות גם ריקה, במקרה זה אין לסטודנטית שיעורים באותו יום.

כשמתקבלת החלטה על שעת הקבלה, כמובן, יש לשקול מערכות שעות של סטודנטים שונים. נציג את **אוסף מערכות השעות** שלהם ע"י מילון, student\_schedules\_dict, כך ששם הסטודנט ממופה 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 המדובר, office\_hour, נציג כסלוט-זמן בן שעה אחת.

בשאלה **ניתן להניח** את תקינות הקלטים על פי הפורמט המתואר.

#### סעיף א'1

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

#### סעיף א'2

ממשו את הפונקציה assess\_office\_hour(office\_hour, student\_schedules\_dict) שמקבלת את זמן שעת הקבלה שברצוננו להעריך, ואת מילון מערכת השעות של הסטודנטים לאותו יום (בפורמט המתואר לעיל). הפונקציה תחזיר tuple אשר: ערכו הראשון הוא רשימת השמות שזמינים להגיע לשעת הקבלה, וערכו השני הוא היחס של הסטודנטים 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 2025-6**

```
'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'

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

### סעיף ב'1

ממשו את פונקציית העזר `merge_intervals(intervals)` שמקבלת רשימה של אינטרוולים (`interval`) שלמים, כלומר כל איבר ברשימה הוא `tuple` באורך 2 של מספרים שלמים בסדר עולה (דוגמא לאינטרוול:  $(-2, 43)$ ). על הפונקציה להחזיר רשימה של אינטרוולים **זרים** אשר **ממוינת** לפי ערך ההתחלה של כל אינטרוול.

הפונקציה תעשה זאת ע"י מיזוג כל האינטרוולים החותכים זה את זה. למשל, האינטרוול  $(-2, 43)$  **חותך** את האינטרוול  $(20, 52)$ , וניתן **למזגם** לאינטרוול  $(-2, 52)$ .

**רמז:** נסו לחשוב על סדר בו יהיה נוח לעבור על רשימת האינטרוולים. בהינתן סדר כזה, נסו למנות את המקרים השונים (המעטים) שייתכנו בעת מעבר על הרשימה.

### דוגמת 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

ממשו את הפונקציה `find_perfect_slots(student_schedules_dict)` שמקבלת את מערכות השעות של כל הסטודנטים, ומחזירה את כל הסלוטים בני-שעה שבהם **כל** הסטודנטים זמינים לשעת הקבלה, או רשימה ריקה אם אין כאלה. הניחו כי שעת הקבלה יכולה להתחיל לכל המוקדם ב-7:00 וכן להתחיל לכל המאוחר ב-19:00.

הנחיה: יש להשתמש בפונקציית העזר שמימשתם 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 2025-6**

```
}
```

```
>> 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
```

ציירו והסבירו בקובץ ה-PDF את תמונת הזיכרון מבחינת מרחב הכתובות ומרחב השמות לאחר שהמפרש מבצע כל אחת מהשורות בהן מופיעה ההערה `breakpoint`. על התרשים להיות דומה לאופן בו הצגנו את תמונת הזיכרון בכיתה. בשאלה זו ניתן לצייר בכתב יד ולסרוק באופן ברור את הציור. הקפידו שהציור יהיה ברור לחלוטין.

יש לזכור, כפי שנאמר בהרצאה, שהאתר "python tutor" לא תמיד משקף באופן מדויק את תמונת הזיכרון ולכן מומלץ לבחון את הדברים באמצעות בדיקת כתובות בזיכרון. בנוסף, שימו לב שב-`breakpoint 2` יש לצייר את כל תמונת הזיכרון ולא רק את הסקופ (scope) הפנימי של הקריאה לפונקציה.