אוניברסיטת תל אביב - בית הספר למדעי המחשב

מבוא מורחב למדעי המחשב, אביב 2022

תרגיל בית מספר 2 - להגשה עד 24/03/2022 בשעה 23:55

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

הנחיות כלליות:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- מומלץ מאוד **להקליד** את התשובות בקובץ ה-PDF. ניתן גם לכתוב את התשובות בכתב יד ברור ולסרוק אותן, אבל שימו לב שתשובות שיכתבו בכתב יד לא ברור **לא יבדקו**, וערעורים בנושא זה לא יתקבלו.
- 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.
- בכל שאלה, אלא אם מצוין אחרת באופן מפורש, ניתן להניח כי הקלט תקין.
- אין להשתמש בספריות חיצוניות פרט לספריות math, random, אלא אם נאמר במפורש אחרת.
- 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, אביב 2022

שאלה 1

עבור מספר שלם חיובי n נגדיר את $\boldsymbol{s(n)}$ להיות סכום כל המחלקים של n, לא כולל n עצמו.

לדוגמה, המחלקים של 4 הם $\{1,2\}$, ולכן $s(4) = 1 + 2 = 3$ (שימו לב ש-2 נספר פעם אחת).

מספר טבעי n נקרא **מספר משוכלל** (Perfect Number) אם $s(n) = n$. לדוגמה, 6 הוא מספר משוכלל כי:

$$s(6) = 1 + 2 + 3 = 6$$

סעיף א'

ממשו את הפונקציה $divisors(n)$ המחזירה רשימה של כל המחלקים של n, לא כולל n, בסדר עולה.
הנחיה מחייבת: יש לממש את הפונקציה באמצעות List Comprehension.

דוגמת הרצה:

```
>>> divisors(6)
[1, 2, 3]
>>> divisors(7)
[1]
```

סעיף ב'

ממשו את הפונקציה $perfect\_numbers(n)$ המחזירה רשימה של n המספרים המשוכללים הראשונים. **כתבו בקובץ ה-PDF** את זמני הריצה עבור $n = 2, 3$. מה קורה עבור $n = 5$?
דוגמת הרצה:

```
>>> perfect_numbers(1)
[6]
>>> perfect_numbers(2)
[6, 28]
```

סעיף ג'

מספר טבעי n נקרא **מספר שופע** (Abundant Number) אם $s(n) > n$. לדוגמה, 12 הוא מספר שופע כי:

$$s(12) = 1 + 2 + 3 + 4 + 6 = 16 > 12$$

ממשו את הפונקציה $abundant\_density(n)$ אשר מחשבת את צפיפות המספרים השופעים מ-1 עד n, כלומר את היחס:

$$\frac{|\{k \in \mathbb{N} \mid k \leq n \text{ } and \text{ } k \text{ } is \text{ } abundant\}|}{n}$$

הפלט צריך להיות מספר מטיפוס float בקטע [0,1].

דוגמת הרצה:

```
>>> abundant_numbers(20) # 12, 18, 20 are abundant numbers
0.15
```

סעיף ד'

ידוע כי צפיפות המספרים השופעים, כש-n שואף לאינסוף, היא בין 0.2474 ל-0.2480 (צפיפותם המדויקת היא שאלה פתוחה). על מנת להשתכנע בנכונות הטענה, הריצו את הפונקציה עבור ערכים $n = 50, 500, 5000$ **וכתבו את התוצאות בקובץ ה-PDF.** סכמו בקצרה את הממצאים.

<u>סעיף ה'</u>

מספר שלם חיובי n נקרא **מספר דמוי משוכלל** (Semi-perfect number) אם הוא שווה לסכום של **כל או חלק** מהמחלקים שלו (לא כולל n עצמו, ומבלי לסכום את אותו הגורם יותר מפעם אחת). למשל, המספר 18 הוא מספר דמוי משוכלל: המחלקים של 18 הם [1, 2, 3, 6, 9], ואכן

$$18 = 3 + 6 + 9$$

נאמר שמספר הוא **דמוי משוכלל מסדר k** אם הוא שווה לסכום של בדיוק k מן המחלקים שלו.

<u>לדוגמה</u>, 18 הוא מספר דמוי משוכלל מסדר 3 מכיוון שהמחלקים של 18 הם $[1,2,3,6,9]$ ואכן

$$18 = 3 + 6 + 9$$

ממשו את הפונקציה $semi\_perfect\_4(n)$ אשר מקבלת מספר n ומחזירה True אם המספר הוא דמוי משוכלל מסדר 4, ואחרת מחזירה False.

דוגמת הרצה:

```
>>> semi_perfect_4(20) # 20 = 1 + 4 + 5 + 10
True
>>> semi_perfect_4(28)
False
```

שימו לב ש-20 הוא מספר דמוי משוכלל מסדר 4 שאינו משוכלל, ו-28 הוא מספר משוכלל שאינו דמוי משוכלל מסדר 4.

<u>סעיף ו' (בונוס)</u>

הוכיחו שלכל מספר n שלם חיובי, n הוא דמוי משוכלל מסדר 3 <u>אם ורק אם</u> n מתחלק ב-6.

שאלה 2

בשאלה זו נממש מספר פונקציות אקראיות תוך שימוש בפונקציה הבסיסית $random.random$,
ללא 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.*

```
>>> 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 $roll\_dice(d)$, שמדמה הטלת קובייה עם d פאות ומחזירה מספר שלם אקראי בין 1 ל-d, כאשר לכל מספר סיכוי שווה להיבחר. הניחו כי $d \geq 2$ ושלם.

סעיף ג'

השתמשו בסעיף ב' על מנת לממש את הפונקציה $roulette(bet\_size\ , parity)$ אשר מדמה **משחק רולטה** עם הימור בגודל bet_size על ערך $parity \in \{"even", "odd"\}$, ומחזירה את **ערך הזכייה**. חוקי המשחק:

- "מסובבים את הרולטה" – מגרילים מספר אקראי בין 0 ל-36, כולל (שימו לב שניתן להגריל 0 בשונה מסעיף ב'), כאשר לכל מספר סיכוי שווה להיבחר.
- אם הרולטה נופלת על 0 – הפסדנו, ללא תלות במשתנה $parity$, והפונקציה מחזירה 0.
- אחרת:
  i. אם הזוגיות של המספר שהוגרל תואמת למשתנה $parity$, ניצחנו – והפונקציה מחזירה $2 * bet\_size$. לדוגמה, אם $parity = "odd"$ והוגרל המספר 3 אז ניצחנו. זאת אומרת שאם bet_size היה 7, אז הפונקציה תחזיר 14.
  ii. אחרת, הפסדנו והפונקציה מחזירה 0.

הנחיה מחייבת: בסעיף זה יש לקרוא לפונקציה $roll\_dice(d)$ שמימשתם בסעיף ב', **ואסור לקרוא ישירות** לפונקציה $random.random$ (או לכל פונקציה אחרת מספרייה חיצונית). $roll\_dice(d)$ יכולה לקרוא ל-$random.random$.

סעיף ד'

ממשו את הפונקציה $roulette\_repeat(bet\_size, n)$ המחשבת את **הרווח המצטבר** מ-n משחקים חוזרים ברולטה, על ידי קריאות חוזרות לפונקציה $roulette(bet\_size, parity)$. בכל קריאה לפונקציה $roulette$, השתמשו ב-$coin()$ על מנת להגריל את ערך המשתנה $parity$ שאיתו תקראו לפונקציה. שימו לב שהרווח במשחק יחיד מורכב מסכום הזכייה פחות סכום ההימור. בפרט, הרווח במשחק יחיד הוא **שלילי** כאשר אנחנו מפסידים, **וחיובי** כאשר אנחנו מנצחים. ענו בקובץ ה-PDF: האם בממוצע המשחק משתלם לשחקן?

סעיף ה'

ממשו את הפונקציה $shuffle\_list(lst)$ המקבלת רשימה שלהי ו"מערבבת אותם" באופן אקראי, בדומה למה שעושה הפונקציה המובנית shuffle של מחלקת הרשימות בפייתון. מותר להשתמש רק בפונקציות הבאות (אין צורך להשתמש בכולן): הפונקציות המובנות של רשימות append, extend, pop, remove (לא את כולן ראיתם בהרצאה / תרגול – אתם מוזמנים לקרוא על פונקציות אלה) והפונקציות שמימשתם בסעיפים הקודמים.

```
>>> shuffle_list([1, 2, 3, 4])
[2, 4, 1, 3]
>>> shuffle_list(["a", "b", "c", "d"])
["c", "b", "d", "a"]
```

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 מחרוזת 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 מחרוזת הניתנת 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 10 ספרות צריכה להסתיים בזמן קצר (לכל היותר שניה).

<u>סעיף א'</u>

ממשו את הפונקציה $inc(binary)$ (קיצור של increment) אשר מקבלת מחרוזת המייצגת מספר שלם אי שלילי בכתיב 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
```

<u>הנחיה מחייבת</u>: 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'
```

<u>סעיף ב'</u>

ממשו את הפונקציה $add(bin1, bin2)$ אשר מקבלת שתי מחרוזות המייצגות מספרים אי שליליים שלמים בכתיב בינארי (כלומר מחרוזות המורכבות מאפסים ואחדות בלבד). הפונקציה תחזיר מחרוזת המייצגת את המספר הבינארי המתקבל מחיבור bin1 ו-bin2.
<u>הנחיה מחייבת</u>: יש לממש את האלגוריתם בהתאם להמחשה בסעיף א': ישירות באמצעות לולאה ואין להשתמש בפונקציה inc.
דוגמאות הרצה:

```
>>> add("1","0")
'1'
>>> add("1","1")
'10'
```

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

<u>סעיף ג'</u>

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

```
>>> pow_two("1010", 2)
'101000'
>>> pow_two("1", 3)
'1000'
```

<u>סעיף ד'</u>

ממשו את הפונקציה $div\_two(binary, power)$ אשר מקבלת מחרוזת המייצגת מספר בינארי אי שלילי ומספר אי שלילי (מטיפוס int). הפונקציה תחזיר את הייצוג הבינארי של $binary$ חלקי 2 בחזקת power מעוגל כלפי מטה, זאת אומרת שהפונקציה תחזיר את הייצוג הבינארי של $\left\lfloor \frac{binary}{2^{power}} \right\rfloor$.

```
>>> div_two("1010", 2)
'10'
>>> div_two("1", 3)
'0'
```

<u>סעיף ה'</u>

ממשו את הפונקציה $leq(bin1, bin2)$ אשר מקבלת שתי מחרוזות המייצגות מספרים שלמים אי שליליים בכתיב בינארי (כלומר מחרוזות המורכבת מאפסים ואחדות בלבד). הפונקציה תחזיר True אם $bin1 \leq bin2$ ו-False אחרת.

```
>>> leq("1010","1010")
True
>>> leq("1010","0")
False
>>> leq("1010","1011")
True
```

<u>סעיף ו'</u>

ממשו את הפונקציה $to\_decimal(binary)$ אשר מקבלת מחרוזת המייצגת מספר בינארי אי שלילי וממירה אותו למספר דצימאלי, זאת אומרת מחזירה int שהוא הערך של המחרוזת. אסור להשתמש בפונקציה זו באף אחד מהסעיפים האחרים.

```
>>> to_decimal("1000")
8
>>> to_decimal("1001")
9
>>> to_decimal("1")
1
```

<u>סעיף ז'</u>

בהרצאה ראינו כי מספר בבסיס b בעל d ספרות דורש בבסיס c מספר ספרות הוא לכל היותר $\lceil d * \log_c b \rceil$.
<u>הוכיחו טענה זו בקובץ ה-PDF.</u>
<u>הערה</u>: אין צורך להסתבך בהוכחה ארוכה, ניתן להוכיח את הטענה בשורות בודדות כאשר מסתמכים על החסמים העליון והתחתון שראינו גם כן בהרצאה לגודלו של מספר בעל n ספרות בבסיס b.

אוניברסיטת תל אביב - בית הספר למדעי המחשב

מבוא מורחב למדעי המחשב, אביב 2022

שאלה 4

בהינתן מספר שלם חיובי n כלשהו, נתאר את התהליך האיטרטיבי הבא:

1. נחשב את n^R – המספר המתקבל מהפיכת סדר הספרות במספר n (בייצוג העשרוני שלו).
2. נעדכן את הערך של n: $n = n + n^R$
3. אם n פלינדרום (כלומר $n == n^R$) – נעצור. אחרת, נחזור ל-1.

לדוגמה עבור המספר ההתחלתי $n = 28$:

<u>סיבוב ראשון:</u>

$n^R = 82$ ולכן נעדכן את n להיות $n = 28 + 82 = 110$. מכיוון ש-110 הוא לא פלינדרום נחזור על התהליך.

<u>סיבוב שני:</u>

$n^R = 011$ ולכן $n = 110 + 11 = 121$ (שימו לב שאפסים מובילים מושמטים). מכיוון ש-121 הוא פלינדרום, נעצור.

<u>הגדרה:</u> מספר שעבורו התהליך הנ"ל **לא עוצר (כלומר ערכו של n בשורה 3 אף פעם לא פלינדרום)** נקרא **מספר ליישרל**. למשל, 28 **<u>אינו</u>** מספר ליישרל מכיוון שהתהליך נעצר לאחר שני סיבובים.

<u>הערה:</u> השאלה "האם קיים מספר ליישרל" היא שאלה פתוחה, אבל קיימים מספרים "חשודים" כמספרי ליישרל, כלומר מספרים שעבורם לא הצליחו להראות שהתהליך המתואר לעיל נפסק לאחר מספר סופי של סיבובים. המספר הקטן ביותר שחשוד להיות מספר ליישרל הוא 196 (נסו להריץ עליו כמה סיבובים של התהליך!).

<u>סעיף א'</u>

ממשו את הפונקציה $lychrel\_loops(n)$ המקבלת מספר שלם חיובי n ומחזירה את מספר הסיבובים של התהליך המתואר עד שמגיעים לפלינדרום. בסעיף זה ניתן להניח שהקלט של הפונקציה הוא מספר שאינו מספר ליישרל.

```
>>> lychrel_loops(28)
2
>>> lychrel_loops(110)
1
```

<u>סעיף ב'</u>

נאמר שמספר n הוא "t-חשוד ליישרל" אם לא הגענו לפלינדרום באף אחד מ-t הסיבובים הראשונים של התהליך.
ממשו את הפונקציה $is\_lychrel\_suspect(n, t)$ המקבלת מספרים שלמים וחיוביים n, t ומחזירה True אם n הוא t-חשוד ליישרל, ו-False אחרת.

```
>>> is_lychrel_suspect(28, 1)
True
>>> is_lychrel_suspect(28, 2)
False
```

<u>סעיף ג'</u>

ממשו את הפונקציה $lychrel\_sort(numbers, t)$ המקבלת רשימה numbers של מספרים שלמים, חיוביים ושונים זה מזה, ומספר שלם חיובי t, ומחזירה רשימה של אותם המספרים ב-numbers ממוינים באופן הבא:

1. תחילה יופיעו כל המספרים שאינם t-חשודים, ממוינים בסדר עולה לפי מספר הסיבובים הנדרשים על מנת להגיע לפלינדרום. במקרה של שוויון במספר הסיבובים, הסדר ברשימת הפלט יקבע לפי הסדר ברשימה המקורית (כלומר אם x, y שני מספרים ב-numbers עם אותו מספר סיבובים ו-x מופיע לפני y ב-numbers, אז x יופיע לפני y גם ברשימת הפלט).
2. אחריהם יופיעו כל המספרים ה-t-חשודים, ממוינים לפי הסדר ביניהם ברשימת הקלט (כפי שהוסבר ב-1)

<u>הערה</u>: מומלץ להיעזר בקלט האופציונלי key של הפונקציה $sorted$ שראיתם (או תראו) בתרגול 4, וכן מומלץ להיעזר בפונקציה שמימשתם בסעיפים א' וב'. ניתן לראות דוגמת הרצה בטסטר בקובץ השלד.

<u>שאלה 5</u>

בשאלה זו נעסוק במספר דרכים שונות לחישוב ציון סופי בקורס "מבוא מוארך למדעי המחשב". הקורס מכיל 3 תרגילי בית ומבחן סופי. נייצג את ציוני הסטודנטים בקורס בעזרת הרשימה $grades$: כל איבר ברשימה מייצג את הציונים של סטודנט אחד בקורס, והינו tuple באורך 2, כאשר האיבר הראשון ב-tuple הוא ציון המבחן (מספר שלם בין 0 ל-100), והאיבר השני הוא tuple באורך 3 של שלושת ציוני תרגילי הבית (מספרים שלמים בין 0 ל-100). לדוגמה, סטודנטית שקיבלה 90 במבחן, וציוני תרגילי הבית שלה הם 90,92,100 מיוצגת על ידי האיבר $(90,(90,92,100))$ ברשימה $grades$.

<u>סעיף א'</u>

בסעיף זה הציון הסופי יקבע על פי הממוצע המשוקלל של ציון המבחן וממוצע תרגילי הבית, כאשר ממוצע תרגילי הבית מהווה "מגן" של 10%. כלומר, אם ממוצע תרגילי הבית גבוה מציון המבחן, אז ציון המבחן נלקח במשקל 0.9 וציון התרגילים נלקח במשקל 0.1. אחרת, הציון הסופי שווה לציון המבחן.

<u>לדוגמה</u>, עבור הציונים $(90,(90,92,100))$ הציון הסופי הוא $0.9 \cdot 90 + 0.1 \cdot \frac{90+92+100}{3} = 90.4$ ועבור הציונים $(95,(85,90,95))$ הציון הסופי הוא 95.

ממשו את הפונקציה $calculate\_grades\_v1(grades)$ המקבלת רשימה $grades$ של ציונים, ומחזירה רשימה חדשה שבה האיבר במקום ה-i הוא ציונו הסופי של הסטודנט ה-i ברשימת הקלט $grades$.

דוגמת הרצה:

```
>>> calculate_grades_v1([(95, (85, 90, 95)), (90, (90, 92, 100))])
[95, 90.4]
```

<u>סעיף ב'</u>

בסעיף זה הציון הסופי יקבע על פי הממוצע המשוקלל של ציון המבחן <u>לאחר פקטור</u>, וציון תרגילי הבית (ללא "מגן").
כמו כן, הפעם נרצה לחשב את הממוצע ביחס למשקל w *כלשהו* (בסעיף א' קבענו משקל של 0.9 למבחן. הפעם נרצה שהפונקציה שלנו תקבל כקלט את המשקל הרצוי).

<u>לדוגמה</u>, עבור הציונים $(95,(85,90,95))$, פונקציית פקטור $f(x) = \min\{x+3,100\}$ ומשקל $w = 0.7$, ציונו הסופי של הסטודנט הוא $0.7 \cdot 98 + 0.3 \cdot \left(\frac{85+90+95}{3}\right) = 95.6$ (שימו לב שהתרגילים אינם מהווים "מגן").

ממשו את הפונקציה $calculate\_grades\_v2(grades, w, f)$ אשר מקבלת רשימה $grades$ של ציונים, משקל w בין 0 ל-1 המייצג את משקל המבחן, ופונקציית פקטור f אשר לכל מספר בין 0 ל-100 מחזירה מספר בין 0 ל-100, ומחזירה רשימה חדשה שבה האיבר ה-i הוא הציון הסופי של הסטודנט במקום ה-i ברשימת הקלט $grades$.

דוגמת הרצה:

```
>>> grades = [(95, (85, 90, 95)), (90, (90, 92, 100))]
>>> w = 0.7
>>> f = lambda x: min(100, x+5)
>>> calculate_grades_v2(grades, w, f)
[95.6, 93.3]
```

סעיף ג׳

בסעיף זה הציון הסופי של סטודנט בקורס יקבע על פי הממוצע המשוקלל של ציון המבחן והממוצע של שני תרגילי הבית הטובים ביותר מתוך שלושת תרגילי הבית (גם בסעיף זה אין ״מגן״).

לדוגמה, עבור הציונים $((85,90,95), 95)$ והמשקל $w = 0.7$ הציון הסופי הוא $0.7 \cdot 95 + 0.3 \cdot \frac{90+95}{2} = 94.25$.

i. ממשו את הפונקציה $calculate\_grades\_v3(grades, w)$ המקבלת רשימה $grades$ ומספר w המייצג את משקל המבחן (כמו בסעיף ב׳), ומחזירה רשימה חדשה שבה האיבר ה-i הוא הציון הסופי של הסטודנט במקום ה-i ברשימת הקלט $grades$.

ii. בהינתן ממוצע יעד שהציב בית הספר, ובהינתן רשימת הציונים $grades$, על צוות הקורס למצוא את המשקל w המתאים כך שהממוצע הכיתתי של הקורס יהיה שווה לממוצע היעד.
ממשו את הפונקציה $calculate\_w(grades, target\_average)$ שמקבלת רשימה $grades$ ומספר $target\_average$ בין 0 ל-100 המייצג את ממוצע היעד, ומחזירה את המשקל המתאים w (מספר בין 0 ל-1) כך שממוצע הציונים הכיתתי לפי שיטת ציון זו יהיה שווה ל-$target\_average$. אם לא קיים w מתאים, הפונקציה תחזיר $None$.
כלומר לכל קלט $grades, target\_average$ צריך להתקיים:

```
>>> w = calculate_w(grades, target_average)
>>> final_grades = calculate_grades_v3(grades, w)
>>> real_average = sum(final_grades) / len(final_grades)
>>> (real_average == target_average) or (w == None)
True
```

כאשר $w == None$ אמ״ם לא קיים w שאיתו ניתן לקבל את ממוצע היעד.

הערה: ניתן להתעלם משגיאות אי-דיוק הקשורות לייצוג float במחשב. מספיק לבדוק שהתוצאה מחזירה w קרוב מאוד ל-w האמיתי (הסתכלו על הבדיקה בטסטר שבקובץ השלד, וודאו שהמימוש שלכם עובר בדיקות דומות עבור קלטים שונים).