אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2021

<u>תרגיל בית מספר 2 - להגשה עד 08/04/2021 בשעה 23:55</u>

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

<u>הגשה</u>:
- 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.
- מכיוון שניתן להגיש את התרגיל בזוגות, עליכם בנוסף למלא את המשתנה SUBMISSION_IDS שבתחילת קובץ השלד. רק אחת מהסטודנטיות בזוג צריכה להגיש את התרגיל במודל.

- הקפידו לענות על כל מה שנשאלתם.

- 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.

תזכורת: מילון הוא מבנה נתונים המחזיק זוגות של ערכים מהצורה *Key : Value*.
נחשוב על המילון כעל אוסף של מפתחות, ללא חשיבות לסדר וללא חזרות (בדומה ל-*set* בה תתקלו בשאלה 5), ולכל מפתח משויך ערך כלשהו.
בשאלה 1 תצטרכו להשתמש בפעולות בסיסיות על מילון, כגון: הכנסה למילון, שליפה ממילון, מחיקת זוג מהמילון, ומעבר על מילון באמצעות לולאת *for*. לפניכם מספר הרצות לדוגמה של פקודות אלה.

```
>>> d = {"Banana":"Yellow", "Strawberry":"Red"} #new dictionary
>>> d
{'Banana': 'Yellow', 'Strawberry': 'Red'}


>>> d["Banana"]
'Yellow'


>>> d["Apple"] = "Green" #adds "Apple":"Green" to the dictionary
>>> d
{'Banana': 'Yellow', 'Strawberry': 'Red', 'Apple': 'Green'}


>>> d["Apple"] = "Red" #changes Apple's value to "Red"
>>> d
{'Banana': 'Yellow', 'Strawberry': 'Red', 'Apple': 'Red'}

>>> d.pop("Banana") #removes Banana from d (and returns its value)
'Yellow'
>>> d
{'Strawberry': 'Red', 'Apple': 'Red'}


>>> for key in d:    #iterates over d's keys
     print(key, d[key])

Strawberry Red
Apple Red
```

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2021

שאלה 1

לאור ההנחיות החדשות של משרד הבריאות, החליטה האוניברסיטה לנקוט באסטרטגיה של חזרה הדרגתית וזהירה לשיעורים פרונטליים בקמפוס. נרצה לבנות מדד פשוט למדידת הסיכון בכיתות הלימוד.

הערה:
לצורך התרגיל נניח כי כל מי שמתחסנ/ת בפעם הראשונה מתחסנ/ת גם בפעם השנייה לאחר 21 ימים. לדוגמה, אם סטודנט/ית התחסנ/ה בפעם הראשונה לפני 25 ימים, **בהכרח** נובע שהתחסנ/ה בפעם השנייה לפני 4 ימים.

נגדיר את **פקטור הסיכון של סטודנט/ית**, $r(n)$, באופן הבא:

1) אם הסטודנט/ית לא מחוסנ/ת, פקטור הסיכון הוא $r(n) = 1$.
2) אם הסטודנט/ית התחסנ/ה פעם ראשונה, ועברו n ימים **מאז מועד החיסון הראשון** ($0 \le n < 21$), פקטור הסיכון נתון על ידי $r(n) = 1 - \left(\frac{n}{21} \cdot 0.5\right)$
3) אם הסטודנט/ית התחסנ/ה בפעם השנייה, ועברו n ימים **מאז מועד החיסון השני** ($n \ge 0$), פקטור הסיכון נתון על ידי $r(n) = 0.5 - \left(\frac{\min(n,14)}{14} \cdot 0.4\right)$

לדוגמה:

- פקטור הסיכון של סטודנט/ית שהתחסנ/ה בפעם הראשונה לפני כ-10 ימים הוא $\frac{16}{21}$ (בקירוב 0.761).
- פקטור הסיכון של סטודנט/ית שהתחסנ/ה בפעם השנייה לפני כ-4 ימים הוא $\frac{27}{70}$ (בקירוב 0.385).

א. ממשו את הפונקציה $risk\_factor(n)$ המחזירה את פקטור הסיכון של הסטודנט/ית, כאשר:
- n מספר הימים **מאז החיסון הראשון** (שימו לב להערה למעלה).
- הניחו כי $n \ge 0$ שלם ו-$n = 0$ אם הסטודנט/ית לא מחוסנ/ת.

דוגמת הרצה:

```
>>> risk_factor(10)
0.7619047619047619
>>> risk_factor(25)
0.38571428571428573
```

הערה: בסעיפים ב'-ה' הניחו כי אין שני סטודנטים בכיתה בעלי אותו שם.

ב. העזרו בפונקציה מסעיף א' על מנת לממש את הפונקציה $students\_risk\_factors(students)$, כאשר:
- $students$ – מילון, כאשר כל מפתח הוא שם של סטודנט/ית בכיתה, וערכו של כל מפתח הוא מספר הימים שעברו **מאז החיסון הראשון** של הסטודנט/ית.

על הפונקציה להחזיר מילון חדש עם אותם המפתחות של $students$ (שמות הסטודנטים), כאשר ערכו החדש של כל מפתח הוא פקטור הסיכון של הסטודנט/ית.

דוגמת הרצה:

```
>>> students = {"Alon": 10, "Gal": 25}
>>> students_risk_factors(students)
{'Alon': 0.7619047619047619, 'Gal': 0.38571428571428573}
```

ג. נגדיר את **פקטור הסיכון של הכיתה** להיות **ממוצע** פקטורי הסיכון של הסטודנטים בכיתה.
ממשו את הפונקציה $class\_risk\_factor(student\_factors)$ שמקבלת מילון $student\_factors$ של סטודנטים ופקטורי הסיכון שלהם (בדומה **לפלט** של הפונקציה מסעיף ב') ומחזירה את פקטור הסיכון של הכיתה.

דוגמת הרצה:

```
>>> student_factors = {"Tom": 0.65, "Alon": 0.3, "Gal": 0.55}
>>> class_risk_factor(student_factors)
0.5
```

ד. ממשו את הפונקציה $convert\_to\_list(student\_factors)$ אשר מקבלת מילון של סטודנטים ופקטורי הסיכון שלהם, ומחזירה רשימה ממוינת של זוגות $(student, risk\_factor)$ (כל זוג כזה הוא אובייקט מטיפוס tuple – כלומר הפונקציה מחזירה רשימה של tuples) של כל הסטודנטים שמופיעים במילון, בסדר עולה לפי ערכי ה-$risk\_factor$.

ניתן להיעזר בפונקציות sort ו-sorted שראיתם בתרגול 2 ובמשתנה האופציונלי key שראיתם בתרגול 3.

דוגמת הרצה:

```
>>> student_factors = {"Tom": 0.65, "Alon": 0.3, "Gal": 0.55}
>>> convert_to_list(student_factors)
[('Alon', 0.3), ('Gal', 0.55), ('Tom', 0.65)]
```

ה. במטרה להטמיע שיטה של "למידה היברידית" (לימודים פרונטליים המשודרים במקביל בזום), תוך שמירה על ביטחון הסטודנטים, החליטה האוניברסיטה לחלק כל כיתה לשתי קבוצות – קבוצת הסטודנטים שרשאים להגיע לקמפוס, וקבוצת סטודנטים שלא רשאים להגיע לקמפוס.
ממשו את הפונקציה $partition\_class(student\_factors, threshold)$ אשר מקבלת מילון של סטודנטים ופקטורי הסיכון שלהם, ומספר $0 < threshold < 1$, ומחזירה את ה-tuple:
$(campus, home)$, כאשר $campus$ ו-$home$ הם מילונים בהם המפתחות הם שמות הסטודנטים שרשאים / לא רשאים להגיע לקמפוס בהתאמה, וערך כל מפתח הוא פקטור הסיכון של הסטודנט/ית.

החלוקה תתבצע על פי הכללים הבאים:

1) פקטור הסיכון של **הכיתה** $campus$ חייב להיות קטן ממש מ-$threshold$ (שימו לב שיתכנו סטודנטים בכיתה עם פקטור סיכון אישי גדול מ-$threshold$. התנאי מתייחס לפקטור הסיכון הכיתתי).
2) פקטור הסיכון האישי של כל סטודנט/ית ב-$campus$ קטן או שווה לזה של כל סטודנט/ית ב-$home$.
3) יש למזער את מספר הסטודנטים בקבוצה $home$ (ובתנאי ש-1 ו-2 מתקיימים).

הערות:

- הנחה מקלה: בסעיף זה הניחו כי פקטורי הסיכון ייחודיים. כלומר, אין שני סטודנטים עם אותו פקטור סיכון.
- ניתן ומומלץ להיעזר בפונקציות מסעיפים ג' וד'.

דוגמת הרצה (דוגמאות נוספות ב-tester):

```
>>> student_factors = {"Tom": 0.65, "Alon": 0.3, "Gal": 0.55}
>>> partition_class(student_factors, 0.45)
({'Alon': 0.3, 'Gal': 0.55}, {'Tom': 0.65})
```

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
```

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

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.

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, ניצחנו.
  ii. אחרת, הפסדנו והפונקציה מחזירה 0.

הנחיה מחייבת: בסעיף זה יש לקרוא לפונקציה $roll\_dice(d)$, **ואסור** לקרוא לפונקציה $random.random()$ (או לכל פונקציה אחרת מספרייה חיצונית).

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.

ה. הריצו את $roulette\_repeat$ 100 הרצות נפרדות, עם הפרמטרים $bet\_size = 100$ ועם $n = 200$, וספרו כמה פעמים הפונקציה סיימה את ההרצה עם רווח מצטבר **חיובי**.
כעת נסו זאת עם $n = 1,000$ ועם $n = 10,000$. **פרטו את התוצאות בקובץ ה-PDF** בטבלה קצרה המסכמת כמה פעמים סיימה ההרצה עם רווח מצטבר חיובי, עבור כל n (אין צורך לפרט כל אחת מ-100 התוצאות השונות). סכמו בקצרה את התוצאות במילים.

הערה כללית לשאלה 2: ה-tester בקובץ השלד לא בודק שהפונקציות מחזירות תשובות בהסתברות הנכונה. וודאו זאת בעצמכם באמצעות הרצה חוזרת של כל אחת מהפונקציות.

העשרה:
אתם מוזמנים לקרוא עוד על רולטה בויקיפדיה, ובפרט על ה-"House Edge" הנוגע לשאלה זו:
https://en.wikipedia.org/wiki/Roulette#House_edge
וכן על חוק המספרים הגדולים:
https://en.wikipedia.org/wiki/Law_of_large_numbers

אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2021

<u>שאלה 3</u>

בשאלה זו נעסוק במימוש פעולות אריתמטיות על מספרים 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 שנראה בתרגול 4.
- 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).

א. 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'
```

ב. ממשו את הפונקציה dec(binary) (קיצור של decrement) אשר מקבלת מחרוזת המייצגת מספר טבעי בכתיב בינארי (כלומר מחרוזת המורכבת מאפסים ואחדות בלבד). הפונקציה תחזיר מחרוזת המייצגת את המספר הבינארי לאחר חיסור של 1.

הערות:

- ניתן להניח שהמחרוזת "0" לא תינתן כקלט לפונקציה.

דוגמאות הרצה:

```
>>> dec("1")
'0'
>>> dec("101")
'100'
>>> dec("100")
'11'
>>> dec(dec("100"))
'10'
```

ג. 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.

דוגמאות הרצה:

```
>>> add("1","0")
'1'
>>> add("1","1")
'10'
>>> add("11" ,"110")
'1001'
```

ד. 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.

דוגמאות הרצה:

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

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

ה. ממשו את הפונקציה is_divisor(bin1, bin2) אשר מקבלת שתי מחרוזות המייצגות מספרים טבעיים (גדולים או שווים ל 1) בכתיב בינארי (כלומר מחרוזות המורכבות מאפסים ואחדות בלבד). הפונקציה תחזיר True אם bin2 הוא מחלק של bin1, ו-False אחרת.

<u>הנחיה מחייבת</u>: יש לממש את הפונקציה תוך שימוש בפונקציות add ו-leq.

דוגמאות הרצה:

```
>>> is_divisor("1000","100")
True
>>> is_divisor("1001","101")
False
>>> is_divisor("111","100")
False
```

<u>שאלה 4</u>

בהינתן מחרוזת לא ריקה s באורך n ומספר טבעי k שמקיים $1 \leq k \leq n$, נרצה לדעת האם s מכילה תת מחרוזת רצופה באורך k שחוזרת על עצמה.

למשל המחרוזת $s = "ababa"$ מכילה תת מחרוזת באורך 3 שחוזרת עצמה ("aba") אבל לא מכילה תת מחרוזת כזו באורך 4.

א. השלימו בקובץ השלד את מימוש הפונקציה *has_repeat1(s, k)* שמקבלת מחרוזת *s* לא ריקה ומספר טבעי k ומחזירה *True* אם *s* מכילה תת מחרוזת רצופה באורך *k* שחוזרת על עצמה, אחרת מחזירה *False*. אין צורך לבדוק את תקינות הקלטים שהפונקציה מקבלת.

במימושכם עליכם להשתמש בזיכרון עזר מסוג אוסף (למשל רשימה, *set* וכו׳) שיכיל לכל היותר *n* איברים, כאשר *n* הינו אורך המחרוזת *s*. בנוסף, אין להשתמש בלולאה מקוננת (לולאה בתוך לולאה).

ב. השלימו בקובץ השלד את מימוש הפונקציה *has_repeat2(s, k)* שפועלת בדומה לפונקציה מסעיף א׳. בשונה מהמימוש הקודם הפעם אין להשתמש בזיכרון עזר באורך משתנה (כמו רשימה או מבנה נתונים דומה. מותר מספר קבוע של משתני עזר), אך מותר להשתמש בלולאה מקוננת.

<u>שאלה 5</u>

<u>הקדמה</u>: שאלה זו משתמשת במבנה נתונים מסוג set. זהו מבנה נתונים שמייצג את האובייקט המתמטי קבוצה – אוסף של ערכים (למשל מספרים) ללא חשיבות לסדר וללא חזרות. set הוא טיפוס מובנה בפייתון, שמממש פעולות יעילות על קבוצה, כגון הוספה לקבוצה, חיתוך בין קבוצות וכו׳. ניתן ליצור קבוצות בפייתון בעזרת סוגריים מסולסלים, או ע״י המרת רשימה לקבוצה בעזרת פקודת set:

```
>>> {2, 3, 5, 7, 7, 5}
{2, 3, 5, 7}
>>> set([2, 3, 5, 7, 7, 5])
{2, 3, 5, 7}
```

בנוסף, ניתן לבדוק האם איבר x נמצא בקבוצה s ע״י הפקודה x in s:

```
>>> 7 in {2, 3, 5, 7}
True
>>> 11 in {2, 3, 5, 7}
False
```

בשאלה זו נעסוק בבדיקת השערת גולדבאך. השערת גודלבאך אומרת כי כל מספר זוגי גדול מ-2 הוא סכום של שני מספרים ראשוניים כלשהם.
לקובץ התרגיל צורף הקובץ primes.txt אשר מכיל את 10000 המספרים הראשוניים הראשונים. עליכם לדאוג שקובץ זה ישב באותה תיקייה בה יושב קובץ השלד. השתמשו במתודה parse_primes שנתונה לכם בקובץ השלד, המחזירה set של כל המספרים הראשוניים בקובץ primes.txt.

א. השלימו בקובץ השלד את הפונקציה check_goldbach_for_num(n, primes_set) אשר מקבלת מספר זוגי גדול מ-2 בשם n ורשימת מספרים ראשוניים, primes_set, ומחזירה True אם המספר מקיים את ההשערה עבור הקבוצה (כלומר, ישנם שני ראשוניים בקבוצה אשר סכומם הוא n) ו- False אחרת. ניתן להניח כי הקלט תקין.
דוגמאות הרצה:

```
>>> check_goldbach_for_num(10, {2, 3})
False
>>> check_goldbach_for_num(10, {2, 3, 5, 7})
True
```

ב. השלימו את הפונקציה check_goldbach_for_range(limit, primes_set) אשר מקבלת מספר גדול מ-2 בשם limit וקבוצת מספרים ראשוניים primes_set ובודקת את ההשערה עבור כל המספרים הזוגיים הגדולים מ 2 עד ל- limit (לא כולל). כלומר, הפונקציה תחזיר True אם כל מספר זוגי הגדול מ-2 וקטן מ-limit ניתן להצגה כסכום של שני ראשוניים בקבוצה primes_set ו-False אחרת.
דוגמאות הרצה:

```
>>> check_goldbach_for_range(20, {2, 3, 5, 7, 11})
True
>>> check_goldbach_for_range(21, {2, 3, 5, 7, 11})
False
```

בדקו את ההשערה עם limit=10,000 ורשימת הראשוניים שבקובץ המצורף. קראו את הקובץ בעזרת הפונקציה הנתונה parse_primes. <u>**כתבו בקובץ ה-PDF – מהו זמן הריצה של הפונקציה.**</u>

ג. כעת נרצה לאסוף סטטיסטיקות על המספרים שמקיימים את ההשערה ובפרט, כמה זוגות ראשוניים יכולים להרכיב מספר מסוים. למשל, 23+7=30 אבל גם 19+11=30 .כלומר יתכנו מספר זוגות ראשוניים שמרכיבים את אותו מספר זוגי. כמו בסעיפים הקודמים, הניחו כי הקלט primes_list הינו רשימה של מספרים ראשוניים.

- השלימו בקובץ השלד את הפונקציה check_goldbach_for_num_stats(n, primes_list) אשר תחזיר את מספר זוגות הראשוניים ב-primes_list שיכולים להרכיב מספר n המתקבל כקלט.
- השלימו בקובץ השלד את הפונקציה check_goldbach_stats(limit, primes_set) אשר תחזיר מילון בו המפתחות יהיו מספר זוגות הראשוניים בprimes_set המרכיבים את כל המספרים עד limit (לא כולל). הערך של כל מפתח יהיה כמות המספרים הזוגיים שיכולים להיות מורכבים ממספר זוגות זה. למשל, אם יש חמישה מספרים זוגיים שיכולים להיות מורכבים ע"י שני זוגות ראשוניים, המילון יכיל את הכניסה d[2] = 5.
<u>הערה</u>: שימו לב לא לספור זוגות פעמיים!
- הריצו את check_goldbach_stats(limit, primes_set) כאשר limit=1000 וprimes_set היא קבוצת הראשוניים שנקראה מהקובץ ע"י parse_primes **<u>וצרפו לקובץ ה-pdf את המילון המוחזר</u>**.

דוגמאות הרצה:

```
>>> check_goldbach_for_num_stats(20, primes_set)
2 # (שני זוגות ראשוניים מרכיבים את 20)
>>> check_goldbach_for_num_stats(10, primes_set)
2 # (שני זוגות ראשוניים מרכיבים את 10)
>>> check_goldbach_stats(11, primes_set)
{1: 3, 2: 1} # 4, 6, 8 מורכבים מזוג אחד ו10 מורכב משני זוגות
```

<u>שאלה 6</u>

עבור מספר טבעי n נגדיר את $\boldsymbol{s(n)}$ להיות סכום כל המחלקים של n, <u>לא כולל</u> 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:

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

א. ממשו את הפונקציה $divisors(n)$ המחזירה רשימה של כל המחלקים של n, לא כולל n, בסדר עולה.
<u>הנחיה מחייבת</u>: יש לממש את הפונקציה באמצעות List Comprehension.
דוגמת הרצה:

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

ב. ממשו את הפונקציה $perfect\_numbers(n)$ המחזירה את רשימת n המספרים המשוכללים הראשונים, בסדר עולה. **<u>כתבו בקובץ ה-PDF</u>** את זמני הריצה עבור $n = 3, 4$. מה קורה עבור $n = 5, 6$?
דוגמת הרצה:

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

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

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

ערך ההחזרה צריך להיות מספר מטיפוס float בקטע [0,1].
דוגמת הרצה:

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

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 $n = 50, 500, 5000$ **<u>וכתבו את התוצאות בקובץ ה-PDF</u>**. סכמו בקצרה את הממצאים.

ה. מספר טבעי 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

$$18 = 3 + 6 + 9$$

ממשו את הפונקציה $semi\_perfect\_3(n)$ אשר מקבלת מספר n ובודקת האם ניתן לכתוב אותו **כסכום של 3 מהמחלקים שלו**. אם כן, הפונקציה תחזיר את רשימת המחלקים שסכומם שווה ל-n, בסדר עולה. אחרת, הפונקציה תחזיר את הערך None.

הערה:

- במידה שיש יותר משלשה אחת מתאימה, החזירו שלשה כלשהי.

דוגמת הרצה (שימו לב ש-20 מספר דמוי משוכלל, אבל אינו סכום של אף שלשה מהמחלקים שלו):

```
>>> semi_perfect_3(18)
[3, 6, 9]
>>> semi_perfect_3(20) # 20 = 1 + 4 + 5 + 10
None
```

<u>סוף.</u>