אוניברסיטת תל אביב - בית הספר למדעי המחשב
מבוא מורחב למדעי המחשב, אביב 2015

תרגיל בית מספר 5 - להגשה עד 7 ביוני (יום ראשון) בשעה 23:55

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

הגשה:

- תשובותיכם יוגשו בקובץ pdf ובקובץ py בהתאם להנחיות בכל שאלה.
- השתמשו בקובץ השלד skeleton5.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 012345678.pdf ו- 012345678.py.

- 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.

שאלה 1

בשאלה זו עליכם להגדיר מחלקה חדשה בשם Polynomial, שתייצג פולינום במשתנה אחד. להזכירכם, פולינום במשתנה אחד מדרגה n ניתן לכתוב באופן הבא: $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ כאשר $n \geq 0$ מספר טבעי, ו- $a_n \neq 0$, למעט בפולינום האפס $p(x) = 0$.

המתודות __init__ ו-__repr__ כבר ממומשות עבורכם בקובץ השלד. המקדמים של הפולינום נשמרים בשדה **coeffs** שהינו רשימה (list), שבה האיבר במקום ה- i מייצג את המקדם של x^i. שימו לב כי המקדם האחרון חייב להיות שונה מ- 0, כדי למנוע מצב שבו פולינום מיוצג ע"י רשימה ארוכה מהמינימום הדרוש. בכל הסעיפים, לצורך ניתוח סיבוכיות הניחו כי חיבור וכפל של שני מספרים כלשהם רץ בזמן קבוע O(1).

א. השלימו בקובץ השלד את מימוש המתודות הבאות. בכל מתודה מצויינת מהי סיבוכיות זמן הריצה הדרושה - לקבלת ניקוד מלא יש לעמוד בדרישות סיבוכיות אלו. ניתן להניח כי הקלט תקין. היעזרו בדוגמאות ההרצה שמופיעות בהמשך. (שימו לב שבקובץ השלד מופיעה המתודה __lt__ אותה אין צורך לממש!)

- **degree** – מחזירה את הדרגה של הפולינום. סיבוכיות זמן דרושה O(1). הניחו כי הפונקציה len של מחלקת הרשימות (list) רצה בזמן קבוע O(1).
- **evaluate** – מקבלת כקלט מספר x (שלם או ממשי) ומחזירה את הערך של הפולינום עבור ה-x הנתון. סיבוכיות זמן דרושה O(n).

  <u>הערה מחייבת</u>: במתודה זו אין להשתמש כלל בפעולת העלאה בחזקה (pow או **). יש לחשב את התוצאה ע"י פעולות כפל וחיבור בלבד.
- **derivative** – מחזירה אובייקט <u>חדש</u> מטיפוס Polynomial שהוא הנגזרת של הפולינום המקורי. סיבוכיות זמן דרושה O(n). פעולת הנגזרת של פולינום מוגדרת כדלקמן:

$$derivative(a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0) = n \cdot a_n x^{n-1} + (n-1) \cdot a_{n-1}x^{n-2} + \cdots + a_1$$

- **__eq__** – מקבלת כפרמטר אובייקט נוסף other מטיפוס Polynomial, ומחזירה True אם הוא מייצג אותו פולינום כמו self, אחרת False. אם דרגת הפולינום הקיים (self) היא n ודרגת הפולינום הנוסף היא m אז סיבוכיות הזמן הדרושה היא O($max\{m,n\}$).
- **__add__** – מקבלת כפרמטר אובייקט נוסף other מטיפוס Polynomial, ומחזירה אובייקט <u>חדש</u> מטיפוס Polynomial, שמייצג את סכום שני הפולינומים. אם דרגת הפולינום הקיים (self) היא n ודרגת הפולינום הנוסף היא m אז סיבוכיות הזמן הדרושה היא O($max\{m,n\}$).
- **__mul__** – מקבלת כפרמטר אובייקט נוסף other מטיפוס Polynomial, ומחזירה אובייקט <u>חדש</u> מטיפוס Polynomial, שמייצג את מכפלת שני הפולינומים. אם דרגת הפולינום הקיים (self) היא n ודרגת הפולינום הנוסף היא m אז סיבוכיות הזמן הדרושה היא O(mn).
- **__sub__** – מקבלת כפרמטר אובייקט נוסף other מטיפוס Polynomial, ומחזירה אובייקט <u>חדש</u> מטיפוס Polynomial, שמייצג את תוצאת החיסור של הפולינום other מהפולינום הקיים (self), כלומר את

הפולינום self-other. אם דרגת הפולינום הקיים (self) היא n ודרגת הפולינום הנוסף היא m אז סיבוכיות הזמן הדרושה היא $O(max\{m,n\})$.

- **__neg__** – מחזירה אובייקט חדש מטיפוס Polynomial שהוא הפולינום שמתקבל ע"י מכפלת כל אחד מהמקדמים של self ב (-1). ראו את דוגמת ההרצה בהמשך. סיבוכיות זמן דרושה O(n).

דוגמאות הרצה:

```
>>> q = Polynomial([0, 0, 0, 6])
>>> q
(6*x^3)
>>> Polynomial([0, 0, 0, -6])
(-6*x^3)
>>> q.degree()
3
>>> p = Polynomial([3, -4, 1])
>>> p
(3*x^0) + (-4*x^1) + (1*x^2)
>>> p.evaluate(10)
63
>>> dp = p.derivative()
>>> dp
(-4*x^0) + (2*x^1)
>>> ddp = p.derivative().derivative()
>>> ddp
(2*x^0)
>>> q==p
False
>>> r = p+q
>>> r
(3*x^0) + (-4*x^1) + (1*x^2) + (6*x^3)
>>> p + Polynomial([-1])
(2*x^0) + (-4*x^1) + (1*x^2)
>>> q == Polynomial([0, 0, 0, 5]) + Polynomial([0, 0, 0, 1])
True
>>> -p
(-3*x^0) + (4*x^1) + (-1*x^2)
>>> p-q
(3*x^0) + (-4*x^1) + (1*x^2) + (-6*x^3)
>>> p*q
(18*x^3) + (-24*x^4) + (6*x^5)
>>> Polynomial([0])*p
0
```

ב. השלימו את מימוש המתודה find_root ששייכת למחלקה Polynomial. על המתודה להשתמש בשיטת ניוטון רפסון (NR) למציאת קירוב לשורש כלשהו של הפולינום. שימו לב שאם לפולינום יותר משורש ממשי אחד, ייתכן כי בהרצות חוזרות יוחזר שורש שונה (בגלל הניחוש ההתחלתי האקראי של NR). במתודה find_root יש להשלים שורה אחת בודדת.

דוגמת הרצה:

```
>>> p.find_root()
0.9999999996240886
>>> p.find_root()
3.000000000000003
```

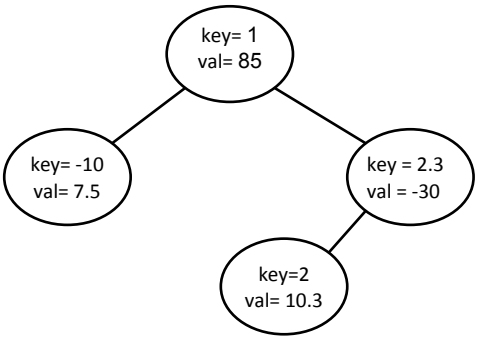
שאלה 2

בהרצאה ראינו מימוש למבנה הנתונים "עץ חיפוש בינארי". במימוש שראינו, צומת בעץ יוצג ע"י אובייקט מהמחלקה Tree_node, ואילו העץ עצמו **לא** יוצג בגישת OOP.

בשאלה זו נניח כי גם המפתחות (keys) וגם הערכים (vals) של הצמתים הם מספרים ממשיים כלשהם (חיוביים או שליליים).

א. נגדיר <u>מסלול כבד ביותר</u>: רצף צמתים שמתחיל בשורש העץ ומגיע עד עלה כלשהו, שסכום הערכים (val) לאורכו הוא מקסימלי. את הסכום המקסימלי הנ"ל נכנה <u>משקל העץ</u>.

לדוגמה, משקל העץ הבא הוא 92.5, שמתאים למסלול הכבד ביותר 1→-10 (הערך העליון בכל צומת הוא המפתח key, והתחתון הוא הערך val):



ממשו בקובץ השלד פונקציה רקורסיבית weight שמקבלת Tree_node שמייצג את שורש העץ ומחזירה את משקל העץ.

ב. ממשו בקובץ השלד את הפונקציה heavy_path שמקבלת Tree_node שמייצג את שורש העץ ומחזירה מסלול כבד ביותר בעץ. המסלול יוחזר כרשימה (list של פייתון) ובה רצף המפתחות (key) של הצמתים. אם יש יותר ממסלול כבד ביותר יחיד, יוחזר אחד מהם, לבחירתכם.

<u>רמז</u>: heavy_path תקרא תחילה ל- weight מהסעיף הקודם עם שינוי קל: במהלך פעילותה תעדכן הפונקציה weight בכל צומת בעץ שדה נוסף, ובו מידע שישמש לאחר מכן לחישוב המסלול הדרוש. יהיה עליכם להוסיף אם כן שדה למחלקה Tree_node.

ג. ממשו בקובץ השלד את הפונקציה find_closest_key שמקבלת Tree_node שמייצג את שורש העץ, ובנוסף מקבלת מספר ממשי (חיובי או שלילי) k ומחזירה את המפתח של צומת בעץ שערכו קרוב ביותר ל k. אם יש כמה כאלה, יוחזר אחד מהם, לבחירתכם. סיבוכיות הזמן הדרושה היא O(h) כאשר h הוא גובה העץ.

<u>הנחיות והבהרות לכלל השאלה</u>:

- בסעיף א' הגישו את weight לאחר השינוי הנדרש לצורך סעיף ב'. השינוי לא צריך להשפיע על הפלט של weight – ודאו שאכן זה כך (אחרת הטסטים האוטומטיים של סעיף ב' עלולים להיכשל).
- המחלקה Tree_node שראיתם בכיתה מופיעה בקובץ השלד. אין לשנות מחלקה זו, למעט תוספת השדה המוזכרת בסעיף ב'.

דוגמאות הרצה:

```
>>> t = None
>>> t = insert(t, 1, 85) #the first time we change t from None to a "real" Node
>>> insert(t, 2.3, -30)
>>> insert(t, -10, 7.5)
>>> insert(t, 2, 10.3)
>>> weight(t)
92.5
>>> heavy_path(t)
 [1, -10]
>>> find_closest_key(t, -5)
-10
>>> find_closest_key(t, 2.2)
2.3
```

שאלה 3

בשאלה זו נמצא את 10 המילים הנפוצות ביותר בספר "הרפתקאות אליס בארץ הפלאות" מאת לואיס קרול.
את הספר ניתן למצוא בגרסה אלקטרונית באתר של פרוייקט גוטנברג:

http://www.gutenberg.org/cache/epub/11/pg11.txt

בקובץ השלד תמצאו שתי פונקציות שמומשו עבורכם (ואין לשנות אותן):

- הפונקציה download(url) שמקבלת מחרוזת המייצגת כתובת url, ומחזירה את הטקסט שמכיל עמוד האינטרנט בכתובת זו. כדי לקבל את הסיפור הנ"ל כטקסט עליכם לכתוב:

```
>>> alice = download ("http://www.gutenberg.org/cache/epub/11/pg11.txt")
```

שימו לב שעליכם להריץ את הפקודה הבאה לפני הקריאה לפונקציה download:

```
>>> from urllib.request import urlopen
```

- הפונקציה clean(text) ש"מנקה" מחרוזת text שהיא מקבלת באופן הבא: מסירה כל תו שאיננו אות באנגלית, רווח או תווי ירידת שורה, וכן הופכת אותיות גדולות באנגלית לקטנות. למשל:

```
>>> clean("a\nBc"+chr(3)+"d")
'a\nbc d'
```

בשלב ראשון נרצה לחשב את תדירויות כל אחת מהמילים בטקסט.

א. להלן פונקציה שמקבלת רשימה (list) של מילים, ומחזירה את תדירויות המילים (כרשימות מהצורה [word, cnt] כאשר cnt הוא מספר הפעמים ש- word מופיעה בקלט):

```
def count_words_naive(words):
    count_list=[]
    words_set = set(words) #set of different words (no repetition)
    for word in words_set:
        count_list += [ [word, words.count(word)] ]
    return count_list
```

את הפונקציה הזו יש להפעיל אם כן כך:

```
>>> words = clean(alice).split()
>>> count_words_naive(words)
```

הריצו את הפונקציה. תנו הערכה לזמן שייקח לה לסיים. מהי סיבוכיות הפונקציה כתלות ב- n=len(words)? הפרידו למקרה הגרוע ולמקרה הטוב (ציינו מהו כל אחד).

כעת נייעל את המימוש ע"י שימוש במילון. לשם כך נשתמש במחלקה SimpleDict שמימוש חלקי שלו מצורף לקובץ השלד. מחלקה זו דומה למחלקה Hashtable שראיתם בהרצאה, אלא שהקוד מהכיתה מיצג מחלקה שדומה ל set של פייתון, ואילו SimpleDict מייצגת מחלקה שדומה ל- dict של פייתון. כלומר, איברים ב- SimpleDict

מורכבים ממפתח (key) ומערך נלווה (value), ובדומה למילון של פייתון חישוב ה- hash לצורך הכנסה וחיפוש במילון מתבצע על המפתח בלבד.

ב. השלימו בקובץ השלד את המימוש של המתודה find של המחלקה SimpleDict שחתימתה היא:

find (self, key)

המתודה מחזירה את הערך הנלווה value למפתח key אם המפתח key נמצא במילון, אחרת מחזירה None.

ג. השלימו בקובץ השלד את המימוש של המתודה insert של המחלקה SimpleDict שחתימתה היא:

insert(self, key, value)

המתודה מכניסה לטבלה איבר חדש שהמפתח שלו הוא key והערך הנלווה הוא value. אם המפתח key כבר קיים בטבלה אז המתודה תעדכן את הערך הנלווה השמור לו ל- value שהתקבל כפרמטר.

ד. השלימו בקובץ השלד את המימוש של המתודה values של המחלקה SimpleDict שחתימתה היא:

values(self)

המתודה מחזירה רשימה של כל הערכים (values) שקיימים בטבלה (אין חשיבות לסדר האיברים ברשימה המוחזרת).

דוגמאות הרצה לסעיפים ב' - ד':

```
>>> d = SimpleDict(5)
>>> d.insert("a", 10)
>>> d.insert("b", 20)
>>> d.find("a")
10
>>> d.find(10)     #should return None
>>> d.insert("a", 100)
>>> d.find("a")
100
>>>d.values()
[100, 20]           #[20, 100] is a valid output as well!
```

ה. השלימו בקובץ השלד את המימוש של count_words(words). הפונקציה מקבלת רשימת מילים words, ומחזירה מילון מטיפוס SimpleDict המכיל את המילים מתוך words כמפתחות (key), כאשר הערך (val) של כל מילה הוא כמות המופעים שלה. דוגמת הרצה (שימו לב שלסדר האיברים במילון אין חשיבות):

```
>>> d = count_words (["ab", "cd", "cd", "ef", "cd", "ab"])
>>> d.find("ab")
2
>>> d.find("cd")
3
>>> d.find("ef")
1
>>> d.find("xx") #should return None
>>>                #yey!
```

הערה: מכיוון שאיננו יודעים כמה איברים צפויים להיכנס למילון, קשה לקבוע את גודל הטבלה האופטימלי. לפיכך, נחליט לאתחל את הטבלה בגודל שרירותי m=200.

לצורך מציאת 10 המילים הנפוצות, נרצה כעת למיין את אוסף השכיחויות (מספרי המופעים) שקיבלנו. נשתמש בפלט של הפונקציה count_words מסעיף ד'.

ו. השלימו את הפונקציה sort_by_cnt, כך ש**תכיל שורה אחת בלבד ובה פקודת return**. הפונקציה מקבלת מילון שמוחזר מהפונקציה count_words, ומחזירה רשימה של tuples מהצורה (word,cnt), כאשר word הופיעה בטקסט cnt פעמים, והרשימה ממוינת לפי cnt מגדול לקטן. דוגמת הרצה:

```
>>> sort_by_cnt(count_words1(["ab", "cd", "cd", "ef", "cd", "ab"]))
[('cd', 3), ('ab', 2), ('ef', 1)]
```

<u>הערה</u>: אם יש מילים באותה תדירות, הסדר ביניהן לא משנה.

<u>עזרה והנחיה</u>: השתמשו בפונקציה sorted, ובמתודה items של המחלקה SimpleDict הממומשת כבר עבורכם.

ז. ציינו בקובץ התשובות (pdf) מהן 10 המילים הנפוצות. אפשר פשוט לצרף את הפלט של הפקודות הבאות:

```
>>> from urllib.request import urlopen
>>> alice = download ("http://www.gutenberg.org/cache/epub/11/pg11.txt")
>>> words = clean(alice).split()
>>> word_cnt = count_words(words)
>>> word_cnt_sorted = sort_by_cnt(word_cnt)
>>> word_cnt_sorted[:10]
```

עזרה נוספת בבדיקת תקינות:

```
>>> len([cnt for cnt in word_cnt.values() if cnt==1])
1330 #number of words that appear only once
```

ח. האם לדעתכם הבחירה בגודל טבלה m=200 הינה בחירה אופטימלית עבור הטקסט הנ"ל? הסבירו בקצרה.

שאלה 4

שאלה זו עוסקת בפונקציות גנרטור.

משולש פסקל הוא משולש אינסופי המורכב משורות של מספרים, כך שבשורה ה- n, האיבר ה- r מוגדר כ:
$a_{nr} = \binom{n}{r} = \frac{n!}{r!(n-r!)}$.
7 השורות הראשונות במשולש הן כדלהלן:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

נשים לב שניתן לקבל את השורה ה-i במשולש פסקל מהשורה ה-i-1 באופן הבא:

1. האיבר הראשון והאיבר האחרון בשורה הם תמיד 1.
2. פרט לכך, האיבר ה-j בשורה ה-i הוא סכום האיבר ה-j-1 והאיבר ה-j בשורה ה-i-1.

נייצג כל שורה במשולש על ידי רשימה כך שהשורה ה-0 היא הרשימה [1], השורה ה-1 היא הרשימה[1,1] וכו'.

סעיף א

השלימו בקובץ השלד את הפונקציה `next_row(row)`, המקבלת את השורה ה-i-1 במשולש פסקל ומחזירה את השורה ה-i.

סעיף ב

נייצג את משולש פסקל כזרם אינסופי של רשימות בצורה הבאה:

```
[[1], [1,1], [1,2,1], [1,3,3,1],…
```

השלימו בקובץ השלד את פונקצית הגנרטור `generate_pascal()` אשר מחזירה גנרטור המייצר את משולש פסקל. השתמשו בפונקציה מסעיף א'.

סעיף ג

משולש ברנולי הוא המשולש שבו כל שורה היא רשימת הסכומים החלקיים של המקדמים הבינומיים, כלומר $a_{nk} = \sum_{i=0}^{k} \binom{n}{i}$.

במילים אחרות, כל שורה במשולש ברנולי היא רשימת הסכומים החלקיים של השורה המתאימה במשולש פסקל. בפרט, המספר ה- i בשורה ה- j של משולש ברנולי הוא סכום i המספרים הראשונים של השורה ה-j במשולש פסקל.

7 השורות הראשונות במשולש ברנולי הן כדלהלן:

1

1 2

1 3 4

1 4 7 8

1 5 11 15 16

1 6 16 26 31 32

1 7 22 42 57 63 64

נייצג את משולש ברנולי בצורה הדומה למשולש פסקל, כזרם אינסופי של רשימות:

```
[[1], [1,2], [1,3,4], [1,4,7,8] …
```

השלימו בקובץ השלד את פונקצית הגנרטור `generate_bernoulli()` אשר מחזירה גנרטור המייצר את משולש ברנולי.

<u>הנחיות הגשה:</u>
ממשו את הפונקציות מסעיפים א', ב' ו ג' בקובץ השלד.

<u>שאלה 5</u>

שאלה זו עוסקת בעיבוד תמונה.

שימו לב שעל מנת להריץ את הפונקציות בשני הסעיפים הבאים עליכם לדאוג שבתיקיה (folder) בה נמצא קובץ הקוד שלכם נמצאים גם הקובץ matrix.py . בנוסף יהיה עליכם להוסיף את הפקודה הבאה לקובץ השלד:

```
from matrix import *
```

א. השלימו בקובץ השלד את <u>שלוש השורות החסרות</u> בפונקציה upside_down, שמקבלת מטריצה שמייצגת תמונה ומחזירה מטריצה חדשה שמייצגת את התמונה שמתקבלת ע"י שיקוף התמונה על הציר האופקי.

לדוגמא עבור התמונה:

לאחר הפעלת הפונקציה תתקבל התמונה:

ב. בני ואמיר רוצים לקיים פגישה סודית, שבה יכתבו יחד שאלות למבחן. אמיר רוצה להציע מקום מפגש כלשהו בקמפוס האוניברסיטה. השמועות מספרות, שביום רביעי, ה- 20 במאי בשעות הערב, הסתובב אמיר עם המצלמה הדיגיטלית המשוכללת שלו, וצילם את מקום המפגש המיועד, כדי לשלוח את התמונה לבני במייל. אבל כדי שמקום המפגש לא ידלוף חלילה, הוא שלח את התמונה באופן מוצפן: התמונה מפורקת לפאזל בגודל m x m (כלומר הפאזל מורכב מ-m שורות ו-m עמודות של חלקים), כאשר m=20. כל חלקי הפאזל הם מלבניים, ובאותו הגודל. כדי שבני יצליח להרכיב בחזרה את התמונה, אמיר דואג שבין כל שני חלקים סמוכים של הפאזל תהיה חפיפה של שורת או עמודת פיקסלים (למשל, עמודת הפיקסלים הימנית ביותר של חלק מסויים זהה לעמודת הפיקסלים השמאלית ביותר של שכנו מימין, או שורת הפיקסלים התחתונה של חלק מסוים זהה לשורת הפיקסלים העליונה של שכנו מלמטה). אך אבוי - בני, שקיבל את חלקי הפאזל, העלה אותם לאתר הקורס בטעות!

בקובץ ה-zip באתר תמצאו את חלקי הפאזל של מקום המפגש הסודי. מטרתכם היא להרכיב מחדש את התמונה, ולגלות את מקום המפגש.

ממשו את הפונקציה `reconstruct_image`. הפונקציה תקבל כפרמטר את m שמוזכר לעיל. הפונקציה תחזיר מטריצה (אובייקט מטיפוס Matrix) המייצגת את התמונה המפוענחת. כלומר הפקודה:

`reconstruct_image(20).display()` תציג את התמונה המפוענחת.

הפונקציה תפעל תחת ההנחות הבאות:

1. בתיקיה (folder) בה נמצא קובץ הקוד שלכם נמצאים גם הקובץ matrix.py.
2. בתוך אותה תיקיה נמצאת גם תיקיה בשם puzzle, אשר מכילה את כל חלקי הפאזל. כלומר פקודה מהסוג:

```
im = Matrix.load("./puzzle/im1.bitmap")
```

תטען את התמונה מהקובץ im1.bitmap לאובייקט im מסוג מטריצה.

3. לשם פשטות נניח כי: אין שני חלקים בעלי אותה שורת פיקסלים עליונה, אין שני חלקים בעלי אותה שורת פיקסלים תחתונה, אין שני חלקים בעלי אותה עמודת פיקסלים שמאלית ואין שני חלקים בעלי אותה עמודת פיקסלים ימנית.

<u>הנחיות הגשה:</u>

ממשו את הפונקציה `reconstruct_image` בקובץ השלד. בנוסף, צרפו לקובץ ה pdf את התמונה המשוחזרת (אפשר ברזולוציה נמוכה יחסית, כדי לא שהקובץ לא יהיה גדול מדיי) וכן הסבר מילולי קצר לאלגוריתם שבעזרתו הרכבתם את הפאזל. אין צורך להסביר את הפרטים הקטנים; רשמו רק כמה משפטים שסוקרים את אופן הפעולה של התוכנית שלכם. אם זיהיתם את אתר המפגש, ציינו מהו.

<u>הערה חשובה:</u> מצורף לתרגיל קובץ בשם solved_example.zip שמכיל תיקייה בשם solved_example עם חלקי פאזל בגודל 3x3 והפתרון שלו (הקובץ solved_exmaple.bmp). תוכלו לוודא שהקוד שכתבתם תקין באמצעות דוגמא זו.

<u>סוף</u>