בית הספר להנדסה ומדעי המחשב ע"ש רחל וסלים בנין

מבוא למדעי המחשב 67101

תרגיל 3 - לולאות

להגשה בתאריך 07/11/2018 בשעה 22:00

הקדמה

בתרגיל זה נתרגל שימוש בלולאות ומשתנים. דגשים לתרגיל:

- בתרגיל זה יש להגיש **שלושה** קבצים: `ex3.py`, `check_maximum.py` ו-`README` (שימו לב, ללא כל סיומת!), בתוך קובץ ארכיון יחיד בשם `ex3.zip`.
- חתימות הפונקציות (שם הפונקציה והפרמטרים שלה) בקבצים `ex3.py` ו-`check_maximum.py` צריכות להיות זהות במדויק לחתימות המתוארת במשימות. ניתן לשנות את חתימות הפונקציות על ידי הוספת ערכים דיפולטיביים לפרמטרים (במקומות בהם השימוש מתאים).
- לנוחיותכם, מסופקים לכם שני קבצים המכילים את חתימות כל הפונקציות שיש לממש. תוכלו להשלים את המימוש של כל אחת מהן על ידי החלפת הפקודה `pass` בפקודות הרלוונטיות.
- ניתן להוסיף פונקציות נוספות וניתן להשתמש בשאלות מאוחרות יותר בפונקציות שמומשו קודם.
- **סגנון:** הקפידו על תיעוד נאות ובחרו שמות משתנים משמעותיים. הקפידו להשתמש בקבועים (שמות משתנים באותיות גדולות), על פי ההסברים שנלמדו, ורק אם יש בכך צורך.
- **שימוש בכלים שלא נלמדו בקורס:** התרגיל ניתן לפתרון קצר ויעיל באמצעות החומר שנלמד. אולם, מותר להשתמש בכלים שטרם נלמדו בקורס כל עוד הם לא מייתרים את השאלה (והשימוש בהם הוא באחריותכם). למשל, אם צריך לסכום איברים ברשימה ע"י לולאות, ניתן לייתר את השאלה על ידי שימוש בפונקציה `sum` המובנית של פייתון. במקרה של ספק, יש לשאול בפורום התרגיל.
  אין להשתמש באף שאלה בכלי של list comprehension.
  מותר, בכל השאלות, להשתמש בפונקציה `append`.
- "רשימה ריקה" – רשימה שאינה מכילה איברים. היא עדיין נחשבת רשימה וגודלה הוא 0.
- "מחרוזת ריקה" – מחרוזת שאינה מכילה תווים, כלומר `""` (מירכאות פותחות וסוגרות) או `''` (גרש פתיחה וסגירה). זוהי עדיין מחרוזת, ואורכה 0. מחרוזת המכילה לפחות רווח אחד אינה ריקה.
- שימו לב מתי יש לקבל קלט מהמשתמש בעזרת הפונקציה `input` (השאלה הראשונה בלבד) ומתי הקלט מתקבל כארגומנט בעת קריאה לפונקציה.
- בכל שאלה מפורט מה ניתן להניח על הקלט - אין צורך לבצע בדיקות תקינות נוספות מעבר למפורט.
- בכל הסעיפים, כאשר פונקציה צריכה להחזיר ערך, הכוונה היא לשימוש במילה השמורה **return**. בפרט **הכוונה אינה להדפיס למסך בעזרת הפונקציה `print`**. שימו לב, **אין להדפיס למסך כל הודעה מלבד ההודעות הנדרשות במפורש**.
- בפונקציות המקבלות רשימות כקלט – אין לשנות את רשימות הקלט.
- **שימו לב כי בקובץ `ex3.py` אין לקרוא לפונקציות, אך יש לעשות זאת ב-`check_maximum.py`.**
- **שימו לב כי אורך השורות (בקוד וב-README) אינו חורג מהמגבלה שפורטה בנהלי הקורס.**

חלק א' - קבלת קלט מהמשתמש

1. קבלת רשימה כקלט

הקדמה

הפונקציה **input** מקבלת שורת קלט מהמשתמש וממירה אותה למחרוזת אותה ניתן לשמור לתוך משתנה. לדוגמה, בעת הרצת קטע הקוד הבא התכנה תמתין לקלט מהמשתמש והקלט אותו יכניס המשתמש (למשל, המחרוזת "Hello") יישמר לתוך המשתנה string_from_user. במשתנה זה ניתן להשתמש, לדוגמה על מנת להדפיס את ערכו (למסך תודפס המחרוזת "Hello").

```
string_from_user = input()

print(string_from_user)
```

מימוש

- כתבו פונקציה אשר מקבלת מחרוזות רבות מהמשתמש.
- הפונקציה תקבל מחרוזות מהמשתמש עד אשר יכניס מחרוזת ריקה.
  - שימו לב - רווח יחיד או מספר רווחים הם שונים ממחרוזת ריקה.
- הפונקציה תחזיר רשימה המכילה את כל המחרוזות שהמשתמש הכניס כקלט.
  - כל תא ברשימה יכיל קלט אחד בדיוק של המשתמש.
  - הקלט האחרון (המחרוזת הריקה) לא צריך להיות איבר ברשימה.
  - סידור הערכים ברשימה יהיה על פי סדר קבלתם כך שקלט המשתמש הראשון יימצא בתא ה – 0 ברשימה והקלט ה – n יימצא בתא ה –n-1 ברשימה.
- במקרה והקלט הראשון הוא מחרוזת ריקה, תחזיר הפונקציה רשימה ריקה.
- חתימת הפונקציה הינה : **def input_list()**
- במימוש הפונקציה (וכן פונקציות אחרות) מותר לעשות שימוש בפונקציה **append** של פייתון.
- יש לקרוא לפונקציה **input** ללא מחרוזת (כלומר אין להדפיס למסך הודעה לבקשת קלט).

דוגמה לשימוש בפונקציה (הטקסט בירוק הוא הסבר לתרגיל ולא מופיע בקלט המשתמש)

עבור הקלט:

```
Hello world
What
 # 1 Space
a nice day
3
# Enter with no input
```

הפונקציה תחזיר את הרשימה: ["Hello world", "What", " ", "a nice day","3"]

2. **שרשור אברי רשימה למחרוזת אחת**

מימוש

- כתבו פונקציה המקבלת כקלט רשימה של מחרוזות ומחזירה את שרשור (concatenation) איברי הרשימה כמחרוזת אחת, כאשר בין כל שתי מחרוזות ישנו רווח (כלומר, יש להכניס רווח לאחר כל מחרוזת משורשרת **מלבד האחרונה**).
- הנחות על הקלט:
  - ניתן להניח כי קלט הפונקציה הוא רשימה.
  - לא ניתן להניח כי הרשימה אינה ריקה.
  - ניתן להניח כי כל איבר ברשימה הוא אכן מחרוזת.
- חתימת הפונקציה הינה: `def concat_list(str_list):`
- ניתן לשרשר שתי מחרוזות ע"י האופרטור +.
- לדוגמה, שימוש באופרטור + על שתי המחרוזות "or" ו-"ange" באופן "ange"+"or" יניב את המחרוזת "orange".
- עבור רשימה ריקה, תחזיר הפונקציה מחרוזת ריקה ("").
- **במימוש הפונקציה אסור להשתמש בפונקציה `join()` של שפת Python.**
- שימו לב כי בשאלה זו וכל השאלות הבאות בתרגיל, הקלט לפונקציה נשלח כפרמטר: **אין להשתמש בפונקציה `input()` כדי לקבל קלט מהמשתמש**.

דוגמה לשימוש בפונקציה

עבור הקריאה (רשימת מחרוזות הנשלחת לפונקציה):

```
concat_list (['Hello world', 'What', ' ', 'a nice day','3'])
```

הפונקציה תחזיר את המחרוזת:

```
"Hello world What   a nice day 3"
```

(שימו לב לרווחים "הכפולים" בין What ו-a, זאת מכיוון שאחת המחרוזות ברשימת הקלט הייתה מחרוזת שהכילה את תו הרווח (" ") – הוספה של רווח (על פי דרישות הפונקציה) לפני ואחרי המחרוזת הזו, מניבה תוצאה של שלושה רווחים במחרוזת הפלט. כמו כן, שימו לב כי אין רווח בסוף).

3. מציאת איבר מקסימלי ברשימת מספרים

מימוש

- כתבו פונקציה המקבלת רשימה של מספרים (מסוג float או int) ומחזירה את האיבר המקסימלי.
- חתימת הפונקציה הינה `def maximum(num_list):`
- הנחות על הקלט:
  - ניתן להניח כי מקבלים רשימה (ולא None או משהו שאינו רשימה).
  - ניתן להניח כי אם קיימים איברים ברשימה, הם כולם מספרים (מסוג float או int).
  - לא ניתן להניח כי ברשימה קיימים איברים: עבור רשימה ריקה, הפונקציה תחזיר את הערך המוגדר None (שימו לב – **לא 0**).
  - לא ניתן להניח כי מספר אינו חוזר על עצמו (למשל, הרשימה [1, 1, 1] תקינה).
  - **ניתן להניח כי כל המספרים הם אי-שליליים (גדולים או שווים ל-0).**
- **במימוש הפונקציה אסור להשתמש בפונקציה `max()` של שפת פייתון.**

דוגמאות לשימוש בפונקציה

maximum([]) → None

maximum ([0]) → 0

maximum ([1, 1, 1]) → 1

maximum ([1, 5, 2, 3]) → 5

חלק ב' – לולאות ורשימות

4. הסטה מעגלית

הקדמה

תהא רשימה באורך k, ו-m מספר שלם אי שלילי כלשהו. **הסטה מעגלית** של הרשימה ב-m מקומות, היא הזזת כל האיברים m מקומות ימינה כך שאיברים שחורגים מסוף הרשימה מושמים חזרה בתחילתה. כלומר, איבר עובר מאינדקס i לאינדקס (i + m) % k, כאשר % הוא סימן עבור פעולת מודולו.

לדוגמה הסטה מעגלית של m=1 עבור הרשימה [1, 2, 3, 4] תיתן:

[1, 2, 3, 4] → [4, 1, 2, 3]

כיוון שהאינדקסים ברשימה המקורית ממופים לאינדקסים ברשימה החדשה באופן הבא:

[0 → 1, 1 → 2, 2 → 3, 3 → 0]

פרמוטציה ציקלית של רשימה היא הסטה מעגלית (ציקלית) של רשימה ב-m מקומות, עבור m כלשהו.

לדוגמה לרשימה בגודל 3 (למשל [0, 1, 2]) קיימות 3 פרמוטציות ציקליות:

1) [0, 1, 2] 2) [2, 0, 1] 3) [1, 2, 0]

מימוש

- כתבו פונקציה המקבלת שתי רשימות מספרים ומחזירה True אם רשימה אחת היא פרמוטציה ציקלית של השנייה. אחרת, היא מחזירה False. במימוש הפתרון אין לשנות את רשימות הקלט.
- חתימת הפונקציה הינה: `def cyclic(lst1, lst2):`
- הנחות על הקלט:
  - lst1 ו lst2 הן רשימות, אך לא ניתן להניח כי הרשימות אינן ריקות.
  - אם הרשימות אינן ריקות, הן מכילות מספרים שלמים (int) בלבד.
- אם שתי הרשימות אינן באורך זהה הפונקציה מחזירה False.
- שתי רשימות ריקות נחשבות כהסטה ציקלית אחת של השנייה.
- **מותר להשתמש בלולאות מקוננות (nested)**, אך קיים פתרון שאינו משתמש בשיטה זו.

דוגמאות לשימוש בפונקציה (הפונקציה מופעלת עם קלט מסוים, וה-"→" מראה מה היא תחזיר):

cyclic([1], [1, 2]) → False	cyclic([1, 2], [1,2]) → True
cyclic([1, 2, 3], [1, 3, 2]) → False	cyclic([1, 2], [2, 1]) → True
cyclic([1, 2, 3], [4, 2, 3]) → False	cyclic([1, 2, 3], [2, 3, 1]) → True
cyclic([1], [5]) → False	cyclic([], []) → True
cyclic([89], [98]) → False	cyclic([17, 1, 2], [2, 17, 1]) → True
cyclic([0, 0, 1, 1], [0, 1, 0, 1]) → False	cyclic([0, 0, 1], [1, 0, 0]) → True

5. "שבע בום"

הקדמה

במשחק "שבע בום" יש לעבור על כל המספרים בין 1 ל-N, כאשר N הוא מספר טבעי כלשהו. יש לומר כל מספר, מלבד מספרים שמתחלקים ב-7 (לדוגמא 7, 14, 49) או מספרים שמכילים את הספרה 7 (למשל 17), בהם אומרים "boom". למשל, אם משחקים עד 24, נאמר:

1, 2, 3, 4, 5, 6, boom, 8, 9, 10, 11, 12, 13, boom, 15, 16, boom, 18, 19, 20, boom, 22, 23, 24

במקרה שבו המספר גם מתחלק ב-7 וגם מכיל את הספרה 7 (למשל, 70), גם כן אומרים "boom" (כלומר אין טיפול מיוחד במקרה זה).

מימוש

- כתבו פונקציה המקבלת כקלט מספר שלם (`int`) ומחזירה רשימה של תוצאות הקריאה במשחק "שבע בום", עד המספר n **כולל**, בסדר עולה.
- הרשימה תורכב ממספרים **בתור מחרוזות** (כלומר, "4" ולא 4), ומהמילה "boom" במקומות המתאימים, בלבד.
- הנחות על הקלט:
  - ניתן להניח כי קלט הפונקציה הוא מספר טבעי (שלם הגדול מ-0).
  - שימו לב שהספירה היא עד המספר שניתן לפונקציה (קלט הפונקציה), **כולל**.
- חתימת הפונקציה הינה: `def seven_boom(n):`

דוגמאות לשימוש בפונקציה

`seven_boom`(1) → ["1"]

`seven_boom`(6) → ["1", "2", "3", "4", "5", "6"]

`seven_boom`(10) → ["1", "2", "3", "4", "5", "6", "boom", "8", "9", "10"]

`seven_boom`(18) → ["1", "2", "3", "4", "5", "6", "boom", "8", "9", "10", "11", "12", "13", "boom", "15", "16", "boom", "18"]

6. היסטוגרמה

הקדמה

יהא מספר n ותהא רשימת מספרים l המכילה רק מספרים שלמים בטווח R=[0, …, n-1].

היסטוגרמה של הרשימה היא מספור של הפעמים שכל איבר בטווח המספרים הופיע ברשימה. כלומר לכל אחד מהערכים ב-R, יופיע כמה פעמים הוא הופיע ב-l. לפיכך, עבור n, היסטוגרמה היא רשימה באורך n (ללא תלות בתוכן של l) אשר בתא ה-i שלה מופיע מספר המציין את מספר הפעמים בו הופיע המספר i ב-l.

מימוש

- כתבו פונקציה המקבלת רשימה של מספרים שלמים ומספר שלם n, ומחזירה את ההיסטוגרמה המתאימה. ערך ההיסטוגרמה עבור מספרים שלא מופיעים ברשימה הוא אפס.
- חתימת הפונקציה הינה `def histogram(n, num_list):`
- הנחות על הקלט:
  - n הוא מספר שלם (int) חיובי.
  - אין ב – num_list איברים שאינם מסוג int.
  - האיברים ב-num_list הם שלמים אי שליליים הקטנים מ-n (כלומר, בטווח R).
  - לא ניתן להניח כי num_list מכילה איברים (עשויה להיות ריקה).
- **במימוש <u>אין להשתמש</u> במודול numpy, בפונקציה count וב-collections.Counter.**

דוגמאות לשימוש בפונקציה

histogram (3, []) → [0, 0, 0]

histogram (3, [0]) → [1, 0, 0]

histogram (3, [1]) → [0, 1, 0]

histogram (3, [1, 2]) → [0, 1, 1]

histogram (3, [1, 2, 2]) → [0, 1, 2]

histogram (4, [1, 2, 2, 3, 3, 3]) → [0, 1, 2, 3]

histogram (4, [3, 2, 3, 1, 3, 2]) → [0, 1, 2, 3]

histogram (4, [3]) → [0, 0, 0, 1]

histogram (5, [3]) → [0, 0, 0, 1, 0]

חלק ג' – לולאות מקוננות

השאלות בפרק זה ניתנות לפתרון על ידי שימוש בלולאות מקוננות. עם זאת, אין חובה לקודד לולאות מקוננות "באופן מפורש" ובפרט, אפשר מתוך לולאה לקרוא לפונקציה אחרת אשר מריצה בעצמה לולאה.

7. **פירוק לגורמים**

הקדמה

על פי המשפט היסודי של האריתמטיקה לכל מספר טבעי קיים פירוק יחיד למספרים ראשוניים. כלומר כל מספר טבעי יכול להיכתב כמכפלה ייחודית של מספרים ראשוניים (ייחודית, עד כדי שינוי סדר הגורמים). לדוגמה הגורמים הראשוניים של המספר 12 הינם [3, 2, 2].

דרך אחת למציאת הגורמים הראשוניים של מספר n היא על ידי מעבר על כל המחלקים הפוטנציאליים (מ-2 ועד n) ועבור כל מחלק פוטנציאלי לבדוק כמה פעמים ניתן לחלק את n במספר זה ללא שארית.

שאלה למחשבה (ללא ניקוד): האם ניתן להקטין את כמות המספרים הנבדקים כך שתהיה קטנה מ-*n-2*? איך ניתן לייעל את מציאת הגורמים?

מימוש

- כתבו פונקציה המקבלת מספר שלם חיובי - n ומחזירה את רשימת כל הגורמים הראשוניים שלו (מהקטן לגדול).
- חתימת הפונקציה הינה: `def prime_factors(n):`
- הנחות על הקלט – ניתן להניח כי n הוא מספר שלם (int) גדול או שווה ל-1.
- 1 מחלק את כל המספרים ואין לכלול אותו ברשימות הפלט; רשימת המחלקים של 1 היא ריקה.
- עבור מספרים ראשוניים – יש להחזיר את המספר עצמו, כאיבר יחיד ברשימה (ראו דוגמאות).
- במקרה של ריבוי שורשים, כלומר אם מספר מסוים מחלק את n יותר מפעם אחת, יש לכלול אותו ברשימת הפלט כמספר הפעמים אותה הוא מחלק את המספר.
- מותר להשתמש (בשאלה זו ובשאלות ככלל) בפונקציה **append**.

דוגמאות לשימוש בפונקציה

```
prime_factors(1)→ []

prime_factors (2)→ [2]

prime_factors (17)→ [17]

prime_factors (6)→ [2, 3]

prime_factors (8)→ [2, 2, 2]

prime_factors (15)→ [3, 5]
```

8. מכפלה קרטזית

הקדמה

מכפלה קרטזית של שתי קבוצות היא אוסף כל הזוגות בהם איבר אחד הוא מהקבוצה הראשונה והאיבר השני מקבוצה השנייה. המכפלה הקרטזית של כל רשימה עם רשימה ריקה, היא רשימה ריקה.

מימוש

- כתבו פונקציה אשר מקבלת שתי רשימות ומחזירה את המכפלה הקרטזית שלהן.
- חתימת הפונקציה הינה: `def cartesian(lst1, lst2):`
- הנחות על הקלט:
  - איברים ברשימה יכולים להיות מכל טיפוס נתונים – הטיפוס עשוי להיות שונה בתוך הרשימה ובין שתי הרשימות.
  - הרשימות לא חייבות להיות באותו האורך.
  - הרשימות עשויות להיות ריקות (אם אחת מהן ריקה, התוצאה היא רשימה ריקה).
  - הרשימות עשויות להכיל איברים זהים בתוך הרשימה ובין הרשימות. בפרט, אם איבר מסוים מופיע פעמיים ברשימה, זהו מצב תקין, ושני המופעים של אותו איבר ייצרו כל אחד זוגות עם כל האיברים מהרשימה השנייה (ראו בדוגמאות להלן).
- פלט הפונקציה הוא רשימה של כל הזוגות האפשריים בהם האיבר הראשון הוא איבר מ-lst1 והאיבר השני הוא איבר מ-lst2:
  - הזוגות יהיו גם הם רשימות (באורך 2) – כלומר, הפלט הוא רשימה של רשימות.
  - סדר האיברים ברשימה החיצונית (כלומר סדר הזוגות) אינו חשוב. אך הסדר בתוך זוג כן חשוב, האיבר הראשון צריך להיות מתוך lst1 והשני מתוך lst2.

דוגמאות לשימוש בפונקציה

```
cartesian([], []) →[]
cartesian (['a'], []) →[]
cartesian (['a'], ['x'])→[['a', 'x']]
cartesian (['a', 'a'], ['x']) →[['a', 'x'], ['a', 'x']]
cartesian (['a', 'b'], ['x', 'y']) →[['a', 'x'], ['b', 'x'], ['a', 'y'], ['b', 'y']]
cartesian (['a', 'b', 'c'], ['x', 'y']) → [['a', 'x'], ['b', 'x'], ['a', 'y'], ['b', 'y'], ['c', 'x'], ['c', 'y']]
cartesian (['Hello', 'World', 'dog', 'cat'], []) → []
cartesian (['Hello', 1], ['Hello']) → [[1, 'Hello'], ['Hello', 'Hello']]
```

9. זוגות שווי סכום

מימוש

- כתבו פונקציה המקבלת רשימה של מספרים שלמים בשם num_list, ומספר שלם נוסף n, ומחזירה את רשימת כל הזוגות (רשימות באורך 2) שסכומם שווה בדיוק ל-n.
- חתימת הפונקציה הינה : `def pairs(num_list, n)`
- הנחות על הקלט :
  - n הוא מספר שלם (int).
  - num_list היא רשימה של מספרים שלמים (int-ים) השונים זה מזה (כל מספר מופיע לכל היותר פעם אחת ברשימה).
  - num_list עשויה להיות ריקה.
- במידה ואין אף זוג שסכומו n תחזיר הפונקציה רשימה ריקה.
- אי אפשר לחבר מספר לעצמו.
- אין חשיבות לסדר הזוגות ברשימת הפלט ואין חשיבות לסדר האיברים בתוך כל זוג. בפרט :

אין להחזיר את אותו הזוג, רק בסדר הפוך: למשל, עבור num_list = [2, 3, 4] ו-n=7, לא נחזיר גם [3, 4] וגם [4, 3], אלא רק אחד מהם (לא חשוב איזה אחד).

דוגמאות לשימוש בפונקציה

pairs([], 5) → []

pairs ([4], 5) → []

pairs ([5], 5) → []

pairs ([2], 4) → []

pairs ([4, 1], 5) → [[1, 4]]

pairs ([4, 2], 5) → []

pairs ([4, 2, 7, 10, 0], 5) → []

pairs ([4, 1, 2], 5) → [[1, 4]]

pairs ([4, 1, 5], 5) → [[1, 4]]

pairs ([4, 1, 0, 5], 5) → [[1, 4], [5, 0]]

pairs ([-1, 4, 1, 7, 2, 5], 6) → [[2, 4], [5, 1], [7, -1]]

חלק ד' – בדיקת הפונקציה maximum.

את הפונקציה בחלק זה, יש לכתוב בקובץ בשם **check_maximum.py** (ולא ב-**ex3.py**).

כתבו פונקציה (בקובץ **check_maximum.py**) שאינה מקבלת פרמטרים, בשם לבחירתכם.

בפונקציה זו, עליכם לבצע מספר בדיקות על קלטים שונים (לבחירתכם) עבור הפונקציה maximum.

עבור כל בדיקה, הפונקציה תודיע (בהדפסה אינפורמטיבית למסך עם טקסט לבחירתכם), האם הפונקציה maximum החזירה את התוצאה הנכונה (המספר המקסימלי), או לא. בנוסף, הפונקציה תחזיר True אם כל הבדיקות עברו בהצלחה, ו-False אחרת. **ב-README, כתבו מדוע החלטתם לבחור במקרים אלו.**

לדוגמא, אם הגדרנו 4 רשימות קלט ו-4 תוצאות מצופות עבור הפונקציה test, הדפסה לדוגמא יכולה להיות:

```
Test 0 OK
Test 1 OK
Test 2 OK
Test 3 FAIL
```

ובמקרה זה תחזיר הפונקציה False.

בסוף הקובץ, עליכם להוסיף את השורה (עליה יורחב בהמשך הקורס): `if __name__ == "__main__":`

ולאחריה, בהזחה (indentation), קריאה לפונקציה שכתבתם. למשל, אם פונקציית הבדיקה נקראת test:

```
if __name__ == "__main__":
    test()
```

יקרא לפונקציית הבדיקה שלכם.

שימו לב: לא מומלץ להעתיק את השורה מקובץ זה, אלא לכתוב אותה בעצמכם. תוכלו גם להשתמש בקובץ המצורף, check_maximum.py, המכיל את שלד הקוד, כולל שורה זו.

<u>הבהרות</u>:

- כדי לקרוא לפונקציה maximum מתוך check_maximum.py, רשמו בשורה הראשונה בקובץ check_maximum.py את הפקודה **from ex3 import maximum**
- אין צורך לבדוק מקרים של קלט לא תקין (רשימות של מספרים לא חיוביים, או רשימות שמכילות טיפוסי נתונים שאינם מספרים – כלומר אינם int או float).
- יש לבצע בדיקות שונות, עם קלטים מגוונים. שימו לב לבדוק מקרים שונים, כולל מקרי קצה. **הסבירו ב-README מדוע החלטתם לבדוק את המקרים שבדקתם.**
- בשאלה זו, אין צורך להשתמש בקבועים עבור רשימות הבדיקה והתוצאות המצופות עבורן.
- שימו לב כי בקובץ check_maximum.py יש להריץ את פונקציית הבדיקה, אולם בקובץ ex3.py אין להריץ דבר – הקובץ הסופי צריך להכיל את הפונקציות בלבד, ללא קריאה להן.

בית הספר להנדסה ומדעי המחשב ע״ש רחל וסלים בנין

חלק ה׳ – שאלות תאורטיות

על הסעיפים הבאים יש לענות באופן מילולי (באנגלית, לא בקוד) בקובץ ה – README.

הסבירו מה יחזיר הקוד שלכם עבור הקריאות הבאות ומדוע.

שימו לב:

- **מצוינים להלן גם מקרים בהם לא נדרשתם לטפל**, לכן אם התכנה שלכם קורסת או מחזירה תשובה לא נכונה **זהו מצב תקין**.
- אין צורך להשיב מה עשוי להחזיר כל מימוש שהוא לבעיה אלא מה יחזיר הקוד שאתם כתבתם.

1. `cyclic(123 ,321)`
2. `maximum([-3, -2, -1])`
3. `maximum([1, 10, 100, 'intro'])`
4. `histogram(3, [1,2,3,4])`
5. `prime_factors(0)`
6. `pairs([0,0,1,1,2,2], 2)`

הוראות הגשה

עליכם להגיש את הקובץ **`ex3.zip`** (בלבד) בקישור ההגשה של תרגיל 3 דרך אתר הקורס (moodle) על ידי לחיצה על ״Upload file״. ההגשה היא עד יום רביעי, ה-07/11/2018 בשעה 22:00.

`ex3.zip` צריך לכלול את הקבצים הבאים בלבד:

1. `ex3.py`
2. `check_maximum.py`
3. README (כמפורט בנהלי הקורס)

שימו לב שב-`ex3.py` אין הרצה של פונקציות וב-`check_maximum.py` יש הרצה של פונקציית הבדיקה.

הנחיות כלליות בנוגע להגשה

- הנכם רשאים להגיש תרגילים דרך מערכת ההגשות באתר הקורס מספר רב של פעמים. ההגשה האחרונה בלבד היא זו שקובעת ושתיבדק.
- לאחר הגשת התרגיל, ניתן ומומלץ להוריד את התרגיל המוגש ולוודא כי הקבצים המוגשים הם אלו שהתכוונתם להגיש וכי הקוד עובד על פי ציפיותיכם.
  - באחריותכם לוודא כי – PDF הבדיקות נראה כמו שצריך.
- קראו היטב את קובץ נהלי הקורס לגבי הנחיות נוספות להגשת התרגילים.
- שימו לב - יש להגיש את התרגילים בזמן!

בהצלחה!