תרגיל בית 5

מומלץ להשתמש בתרגילים בבדיקה הדרגתית של הפונקציות הקטנות על ידי הדפסות, בדיקה של הפונקציות בנפרד ודיבגר debugger כפי שהראיתי בסרטון. זה יקל מאד על הכתיבה והבדיקה שלכם.

1. (30%) **בשאלה זאת תממשו צופן לטקסט הכתוב באותיות לטיניות**

א. נגדיר פעולת חיבור (שנסמן אותה +) על האותיות 'a','b','c',...,'z' באופן הבא. לכל אות מותאם ערך מספרי: 'a' מקבלת ערך 0 , 'b' מקבלת את הערך 1, וכן הלאה עד 'z' שמקבלת ערך 25. כדי לחבר שתי אותיות מחברים את הערכים המספריים שלהן ואת התוצאה מתרגמים לאות המתאימה. אם התוצאה גדולה מ 25 לוקחים את השארית מ 26. למשל, כדי לחבר 'f' ו 'x' מחשבים 23+5=28, השארית מ 26 היא 2 ולכן התוצאה היא האות 'c'.
דוגמאות נוספות: 'a'+'a'='a', 'b'+'c'='d', 'y'+'g'='e'.
כתבו פונקציה add_letters המקבלת שתי מחרוזות. אם שתי המחרוזות באורך 1 ומכילות אותיות לטיניות בלבד, הפונקציה תחזיר את הסכום של האותיות לפי ההגדרה הנ"ל. אם לא – יוחזר None. התוצאה תמיד תהיה אות לטינית קטנה, גם אם אחד הפרמטרים או שניהם מייצגים אותיות לטיניות גדולות.
דוגמאות:
add_letters(‘a’, ‘b’) תחזיר ‘b’,
add_letters(‘x’, ‘F’) תחזיר ‘c’,
add_letters(‘Y’, ‘G’) תחזיר ‘e’,
add_letters(‘a’, ‘bcd’) תחזיר None,
add_letters(‘%’, ‘ ’) תחזיר None,

ב. כתבו פונקציה add_strings שמקבלת שתי מחרוזות המורכבות מאותיות לטיניות בלבד באורכים כלשהם ומחזירה מחרוזת שבה כל תו הוא סכום התווים המתאימים במחרוזות הקלט. **אורך הפלט הוא אורך המחרוזת הקצרה מבין השתיים**.
למשל add_strings('input, 'output') תחזיר 'whijn' כי 'i'+'o'='w', 'n'+'u'='h' וכן הלאה.
add_strings('hello','HI') תחזיר 'om'.
אם מחרוזות הקלט לא מורכבות אך ורק מאותיות לטיניות הפונקציה תחזיר None. יש להשתמש בפונקציה add_letters מסעיף א' לחישוב הסכום.

ג. צופן ויז'נר (Vigenere) הוא צופן עתיק מהמאה ה 16 להצפנת טקסט הכתוב באותיות לטיניות. כדי להצפין טקסט כלשהו s בוחרים מפתח סודי k (בדרך כלל מלה שקל לזכור), בונים מחרוזת t ע"י שיכפול של k מספר פעמים עד שהאורך של t הוא לפחות באורך של s ואז מחברים את s ו t באמצעות הפונקציה add_strings. התוצאה היא הטקסט המוצפן.
למשל, אם אנחנו רוצים להצפין את הטקסט s='attackatsixoclock' והמפתח k='input', אז המחרוזת t תהיה 'inputinputinputinput' ותוצאת ההצפנה תהיה 'igiuvsnimbfbrfhkx'
יש לכתוב פונקציה vigenere_encrypt שמקבלת מחרוזת s ומפתח k ומחזירה את תוצאת ההצפנה של s באמצעות המפתח k. אם k לא מורכב מאותיות לטיניות בלבד, אז יוחזר None. אם s לא מורכב מאותיות לטיניות בלבד, אז הפונקציה תתעלם מכל התווים שאינם אותיות לטיניות (הפונקציה תתעלם גם מרווחים). האורך של המחרוזת s יהיה אורכה ללא התווים שאינם אותיות.
למשל, vigenere_encrypt('Attack at Six (6) oclock', ‘input’) תצפין את המחרוזת 'AttackatSixoclock' ותחזיר את המחרוזת 'igiuvsnimbfbrfhkx’. (שימו לב שהפונקציה add_letters הופכת אותיות גדולות לקטנות) .
הפונקציה vigenere_encrypt תשתמש בפונקציה add_strings.
אבל עבור vigenere_encrypt('Attack at Six (6) oclock', ‘input1’) תחזיר None.

ד. יש לכתוב פונקציה vigenere_decrypt שמקבלת מחרוזת w ומפתח k ומחזירה את תוצאת הפיענוח של w באמצעות המפתח k. אם k לא מורכב מאותיות לטיניות בלבד, אז יוחזר None.
למשל, vigenere_decrypt('igiuvsnimbfbrfhkx', 'input') תחזיר את המחרוזת 'attackatsixoclock' (שימו לב שהתוצאה תמיד באותיות קטנות ובלי רווחים).

ה. יש לכתוב תוכנית enc_dec.py שעושה את הפעולות הבאות, לאחר שהמשתמש התבקש לבחור בין e ל d :

i. אם המשתמש הקיש e (קיצור של encrypt), אז התוכנית תבקש ממנו מפתח הצפנה ושם קובץ. התוכנית תצפין באמצעות הפונקציה vigenere_encrypt את תוכן הקובץ. התוצאה תודפס לקובץ בשם דומה לקובץ המקורי, אבל עם תוספת vig_. למשל, אם שם הקובץ המקורי הוא pooh.txt אז תוצאת ההצפנה תישמר לקובץ pooh_vig.txt. התוכנית תצפין רק אותיות אנגליות ותתעלם מכל תו שאינו אות אנגלית (כולל רווחים).

ii. אם המשתמש הקיש d (קיצור של decrypt) אז התוכנית תבקש ממנו מפתח הצפנה ושם קובץ ותפענח את הטקסט שנמצא בתוך הקובץ באמצעות הפונקציה vigenere_decrypt. תוצאת הפיענוח תודפס לקובץ המורכב משם הקובץ המקורי עם תוספת dec_. למשל, כאשר נפענח את הקובץ Alibaba_vig.txt נדפיס את התוצאה לקובץ Alibaba_dec.txt.

iii. אם המשתמש הקיש כל דבר אחר, אז התוכנית לא תעשה כלום ותפסיק את פעולתה

השתמשו ב __main__ לצורך הרצת התכנית כפי שלמדנו עבור הגדרת מודולים.
לצורך בדיקה עצמית מצורפים כמה קבצים מקוריים, הקבצים המוצפנים המתאימים והמפתחות ששימשו להצפנה.

מפתח	קובץ מקורי	קובץ מוצפן
robbers	alibaba.txt	alibaba_vig.txt
childhood	catcher.txt	catcher_vig.txt
tragedy	hamlet.txt	hamlet_vig.txt

שימו לב שהתוכנית תקרא את הקבצים מאותה תיקייה של הקוד כדי שנוכל לבדוק בקלות.
נסו לפענח את הקבצים באמצעות התוכנית שאתם כותבים. ודאו שאתם מקבלים טקסט באנגלית (באותיות קטנות וללא רווחים). נסו להצפין את הקבצים המקוריים ולפענח אותם.
שיומו לב שלא תקבלו את הטקסט המקורי מכוון שנמחקו בפיענוח התווים שאינם אותיות אבל עדיין תוכלו לבדוק שהטקסט המקורי שוחזר. נתונים לכם שלושת הקבצים הראשונים לבדיקה.

2. (40%) בתרגיל זה תתרגלו מימוש אלגוריתם הנתון ב pseudocode. שימו לב, אף שהתרגיל נראה ארוך וקשה, זהו למעשה תרגיל קל! כל שעליכם לעשות הוא לפעול לפי ההוראות.

ראינו בהרצאה ששיטות ההצפנה המודרניות משתמשות במספרים ראשוניים גדולים מאוד. בתרגיל זה נלמד איך למצוא מספרים ראשוניים תוך שימוש באלגוריתם Miller-Rabin לבדיקת ראשוניות. אלגוריתם זה לא נותן תשובה ודאית במקרה שהמספר ראשוני, אבל, אל דאגה! ההסתברות שהאלגוריתם טועה נמוכה מאוד. את הפונקציות שבסעיפים א-ה יש לכתוב בתוך מודול בשם prime_miller_rabin.py. את התוכנית שבסעיף ו' עליכם לכתוב בקובץ primes.py.

א. עבור מספר שלם חיובי נתון n נגדיר את **הדרגה הזוגית** (even degree) של n בתור המעריך של החזקה הגבוהה ביותר של 2 שמחלקת אותו. נסמן את זה ב even(n).
למשל, even(12)=2 (כי 2^2 מחלק את 12, אבל 2^3 לא מחלק את 12). even(64)=6, even(2)=1, even(k)=0 לכל מספר אי-זוגי k.

עבור מספר שלם חיובי נתון n נגדיר את **החלק האי-זוגי** (odd part) של n בתור המספר האי-זוגי הגבוה ביותר שמחלק אותו. נסמן את זה ב odd(n). למשל, odd(12)=3 , odd(64)=1, odd(k)=k לכל מספר אי-זוגי k, ו- odd(k)=1 לכל מספר k שהוא חזקה של 2.

עליכם לכתוב פונקציה get_even_odd_parts שמקבלת מספר שלם חיובי n ומחזירה זוג מספרים: even(n) ו odd(n).

רמז: שימו לב שאם s=even(n) ו t=odd(n) אז $n=t\cdot 2^s$.

ב. בסעיף זה נממש פונקציה is_probably_prime() לקביעה האם מספר נתון הוא ראשוני: (הסיבה שהפונקציה נקראת is_probably_prime ולא is_prime היא שהתשובה של הפונקציה לא נכונה ב 100%, אבל הסתברות השגיאה מאוד נמוכה)

<u>ארגומנט</u>:

n = מספר שלם חיובי אי-זוגי (שאותו רוצים לבדוק)

<u>ערך מוחזר</u>:

True – אם יש סיכוי גבוה ש n ראשוני

False – אם בטוח ש n לא ראשוני

<u>אופן פעולת הפונקציה</u>:

i. הפונקציה תחשב את s=even(n-1) ואת t=odd(n-1) באמצעות הפונקציה מסעיף א'.

ii. הפונקציה תריץ 10 פעמים את הקריאה is_suspected_prime(n,t,s) (ר' סעיף ג').

iii. אם **<u>בכל</u>** הפעמים הוחזר True, אז is_probably_prime תחזיר True. אם באחת הפעמים הוחזר False – הפונקציה תחזיר False.

ג. בסעיף זה עליכם לממש את האלגוריתם הבא בפונקציה is_suspected_prime:

<u>ארגומנטים</u>:

n = מספר שלם חיובי אי-זוגי

t = מספר שלם חיובי

s = מספר שלם חיובי

<u>ערך מוחזר</u>:

True – אם יתכן ש n ראשוני

False – אם בטוח ש n לא ראשוני.

<u>אופן פעולת הפונקציה</u>: (עליכם "לתרגם" את האלגוריתם הבא לשפת פייתון):

```
choose a random positive integer a between 2 and n-1
let d= a^t mod n
if d==1 or d==n-1 then return True
for i=1 to s – 1
    d= d^2 mod n
    if d == n – 1
        then return True
return False
```

את החזקה a^t mod n יש לחשב בעזרת הפונקציה modular_power שלמדנו בהרצאה (יש לכלול גם פונקציה זאת במודול).

ד. כיתבו פונקציה make_random_odd_number שמקבלת כפרמטר מספר שלם חיובי m ומחזירה מספר אקראי <u>אי-זוגי</u> באורך m ספרות עשרוניות.

(רמז: יש לבחור לחוד באופן אקראי את כל אחת מהספרות של המספר. שימו לב לראשון ואחרון.)

ה. כיתבו פונקציה make_prime שמקבלת פרמטר m (מספר שלם חיובי) ומחזירה מספר ראשוני באורך m ספרות עשרוניות. (רמז: יש להשתמש בפונקציה make_random_odd_number כדי לבחור מספרים שוב ושוב, עד שמקבלים מספר שהפונקציה is_probably_prime מאשרת שהוא ראשוני).

ו. כיתבו תוכנית נוספת (כלומר קובץ נפרד) primes.py שמקבלת מהמשתמש שני מספרים שלמים חיוביים k ו m. התוכנית תשתמש בפונקציה make_prime כדי למצוא k מספרים ראשוניים באורך m ספרות כל אחד. את המספרים יש להדפיס לקובץ פלט primes.txt, כל מספר בשורה נפרדת (ראו דוגמה מצורפת).

דוגמת הרצה:

```
Enter number of primes: 3
Enter size of primes: 50
```

מצורף קובץ הפלט primes.txt עם מספרים גדולים, חלקם ראשוניים וחלקם לא. שימו לב שאתם לא תקבלו את אותם מספרים.

לצורך בדיקת הפונקציה is_probably_prime מצורף קובץ primes_examples.txt ובו המספרים מ primes.txt , ליד כל מספר כתוב אם הוא ראשוני או לא. מצורף גם קובץ examples_2.txt עם מספרים קטנים ו primes_examples_2.txt לבדיקה. שימו לב שאלה לא פלטים לדוגמא אלה נועדים בשבילכם לבדוק את הפונקציות שלכם.

3. (30%) בשאלה זו תממשו אלגוריתם רקורסיבי.
יהי n מספר שלם חיובי. נגדיר חלוקה ממויינת של n כסידרה ממויינת בסדר **יורד** של מספרים שלמים חיוביים, לא בהכרח שונים, שהסכום שלהם הוא n.
דוגמאות:
4,3,3,2,2 היא חלוקה ממויינת של 14
6,5,5,3,2,2,1 היא חלוקה ממויינת של 24.
סידרה באורך 1 נחשבת לחלוקה ממויינת חוקית, לכן הסידרה הכוללת את המספר 24 בלבד היא חלוקה ממויינת של המספר 24.

א. כיתבו פונקציה get_partitions שמקבלת כפרמטר מספר שלם חיובי n ומחזירה רשימה מקוננת של כל החלוקות הממויינות של n. הפונקציה תשתמש בפונקציה רקורסיבית get_partitions_helper שמקבלת פרמטרים נוספים מלבד n, לפי הבנתכם.

דוגמה: הפקודה get_partitions(4) תחזיר את הרשימה המקוננת

[[1, 1, 1, 1], [2,1, 1], [3,1], [2, 2], [4]]

רמז: תוכלו להגדיר

get_partitions_helper(n, current_partition, all_partitions, max_size, current_index)

- **n** - n המספר שאותו מנסים לחלק בסכום (מספר שמחפשים עבורו סכום בקריאה הנוכחית).
- **current_partition** - רשימה זמנית ששומרת את החלוקה הנוכחית שמרכיבים באיטרציה הנוכחית. הרשימה מאותחלת לאפסים בגודל המספר המקורי.

- **all_partitions** - רשימה מקוננת שבה שומרים את כל החלוקות שמצאנו עד כה. כל חלוקה מיוצגת כרשימה בפני עצמה.
- **max_size** - הערך המקסימלי שמותר לבחור בשלב זה כדי להבטיח חלוקות בסדר יורד.
- **current_index** - האינדקס ברשימה current_partition שבו אנחנו מוסיפים את הערך הנוכחי שמרכיב את החלוקה.

זאת רק הצעה. כמובן שיש דרכים אחרות לפתרון. אך חייב להיות פתרון רקורסיבי.

ב. תוכנית generate_partitions.py שקוראת מקובץ partition_sizes.txt מספרים שלמים חיוביים המופרדים ברווחים ומדפיסה לקובץ partitions.txt את כל החלוקות הממוינות של כל אחד מהמספרים שבקובץ הקלט, כמו בדוגמה המצורפת partitions.txt . אם אחד המספרים בקלט אינו חוקי (אינו מספר חוקי או שאינו מספר שלם חיובי) התוכנית תתעלם ממנו באמצעות מנגנון exceptions. יש להקפיד על פלט בדיוק על פי הפורמט שבקובץ לדוגמה.

0111401 מבוא למדעי המחשב
החוג למדעי המחשב, המכללה האקדמית תל-חי
סמסטר סתיו תשפ"ה

הנחיות הגשה:

1- יש להגיש תוכניות שרצות ללא שגיאות. תוכנית שתוגש עם שגיאות ריצה תקבל לכל היותר חצי מהנקודות.
2- בכל התרגילים יש להשתמש במנגנון exceptions כדי למנוע קריסת התוכנית במקרה של קובץ קלט לא קיים ובמקרה של קלט לא תקין.
3- יש לכתוב הערות לתוכנית: docstring בתחילת כל פונקציה, הסבר קצר בתחילת התוכנית, הסבר בתחילת לולאות.
4- אין להשתמש במודולים מלבד מודולים סטנדרטיים כמו math, random, sys,
5- יש לפתור כל שאלה בקובץ נפרד עם סיומת py. (אלא אם כן נאמר אחרת).
6- יש להגיש את כל הקבצים בקובץ אחד מכווץ עם סיומת zip. שם הקובץ המכווץ צריך להיות id_ex5.zip, כאשר id הוא מספר הת"ז שלכם. למשל 111112113_ex5.zip.
יש לכלול רק קבצים עם סיומת py. אין לכלול קובצי קלט ופלט.
7- כל קובץ יתחיל בהערה ובה המידע הבא:
   א. שם הסטודנט
   ב. מס' תעודת זהות
   ג. מספר תרגיל התרגילים
   ד. שם התוכנית

למשל, עבור שאלה 2 בתרגיל 5:

```
"""
Student: Lionel Messi
ID: 111111111
Assignment no. 5
Program: primes.py
"""
```

שימו לב: יש להקפיד על הנחיות ההגשה האלה. הגשה שלא בפורמט הזה לא תקבל את מלוא הנקודות ואף עלולה להיפסל.