**משימות למעבדה מס' 7**

**(Conventional Interfaces)**

1. נרצה לממש פעולה דומה ל-**reduce** אבל כללית יותר (לא מוגבלת לסכימת הערכים) – **accumulate**. זו פונקציה אשר מקבלת רצף, פונקציה (שמחליפה את הסכימה), וערך התחלתי.

**לדוגמא:**

>>> print**(accumulate** (0, add, (1, 2, 3, 4, 5)))

15

>>> print(**accumulate**(1, lambda x,y: x\*y, (1, 2, 3, 4, 5)))  
120  
>>> print(**accumulate**(1, lambda x,y:4\*x-y, (1, 2, 3, 4, 5)))  
571

>>> from operator import add

>>> print**(accumulate** (1, add, **()**))

1

1. ממש\י פונקציה **mymap** אשר מבצעת פעולה זהה ל-**map** של פייתון (עבודה עם tuple).

**לדוגמא:**

>>> print(tuple (**map** (lambda x: x\*x, range (1, 5))))

(1, 4, 9, 16)

>>>print( tuple (**mymap** (lambda x: x\*x, range (1, 5))))

(1, 4, 9, 16)

1. ממש\י פונקציה **orfilter** אשר מקבלת שני פילטרים ומחזירה פילטר (פונקציה) חדש שמורכב מפעולת **or** בניהם.

**לדוגמא:**

>>> import math

>>> def **iseven**(x):

return x % 2 == 0

>>> def **issquare**(x):

return int(math.sqrt(x)) \*\*2 == x

>>>print( list (filter (**orfilter** (iseven, issquare), range (20))))

# [0, 1, 2, 4, 6, 8, 9, 10, 12, 14, 16, 18]

>>>print(list (filter (**accumulate** (lambda x: False, **orfilter**, (**iseven**, **issquare**)), range (20))))

#[0, 1, 2, 4, 6, 8, 9, 10, 12, 14, 16, 18]

>>> print(list (filter (**mymap** (lambda f: **orfilter** (f, **issquare**), (**iseven**,)) [0], range (20))))

#[0, 1, 2, 4, 6, 8, 9, 10, 12, 14, 16, 18]

list**(**filter**(mymap(**lambda f: orfilter**(**f, **issquare)**, **(iseven**,**))**[0], range**(**20**)))**

1. מרצה מחשב ציון ממוצע של סטודנטים שעברו את המבחן אחרי פקטור, בהינתן רשימת ציוני מבחנים. הרשימה מכילה ציונים בטווח **0-100**, וכן ציון **199** עבור סטודנטים שלא ניגשו למבחן. ציון המעבר (אחרי פקטור) הוא **56**. הפקטור (המכונה פקטור-שורש) מחושב ככפל של **10** בשורש ריבועי של ציון:
2. יש להשלים את הפונקציה:

def **average\_passed\_grade** (grades):

s = tuple (filter ( **<1>**, map ( **<2>**, filter ( **<3>** ))))

return reduce ( **<4>** ) / **<5>**

שימו לב: אין להגדיר פונקציות עזר, ומותר להשתמש בפונקציה **sqrt** ופונקציות מתמטיות מובנות נוספות. אין לבצע חישובים חוזרים מיותרים.

1. יש להכליל את הפונקציה מהסעיף הקודם, כך שתקבל כפרמטר נוסף פונקציה המחשבת פקטור. יש להתחשב בכך שציון אחרי פקטור לא יכול להיות גדול מ-**100** – **ציון כזה יש להגביל ל-100.**

**דוגמת הרצה:**

>>> print(**average\_passed\_grade** ([23, 64, 199, 20, 77, 98, 100, 199]))

91.686148310009472

>>> **print(average\_passed\_grade** ([23, 64, 199, 20, 77, 98, 100, 199], lambda x: x+15))

92.75

**5**.עליכם להשלים את הפונקציה**text\_preprocessing** - פונקציה היוצרת **pipline** לעיבוד טקסט וסופרת תדירות של מילים בטקסט. ניתן להניח כי הטקסט מגיע ללא סימני פיסוק ומכיל אותיות קטנות וגדולות, מספרים, וכן רווחים בין המילים. תהליך העיבוד מורכב מהשלבים הבאים:

* 1. הפרדה בין המילים.
  2. סינון מילות עצירה (**stopwords**) ומספרים.
  3. חישוב תדירות מילים.

הפונקציה שומרת את התוצאות של **שלב 2** (מילים אחרי סינון, **<1>**,**<2>**) במשתנה **filtered** ולאחר מכן מחשבת תדירות **(<3>)** לכל אחת מהמילים.

def **text\_preprocessing**(text, stopwords):

bag\_words = {}

**filtered** = tuple(filter(**<1>**, map(**<2>**,text.split())))

**<3**>

return bag\_words

**דוגמת הרצה:**

>>> **stop\_list** = (*'is', 'it', 'a', 'the', 'my', 'and'*)

>>> **print(text\_preprocessing** (*'My cat is 10 and it is a very fat cat'*, **stop\_list**))

{*'fat'*: 1, *'very'*: 1, *'cat'*: 2}

להלן חתימות הפונקציות שניתן להיעזר בהן(שתיהן שייכות לממשק של str):

**isnumeric()** - מחזירה True כאשר מופעלת על מחרוזת המורכבת מתווים מספריים בלבד.

**lower()** - מחזירה עותק של מחרוזת עם כל התווים מומרים לאותיות קטנות.

**בהצלחה !**