המכללה האקדמית להנדסה סמי שמעון

עבודת הגשה מס' 3

תאריך הגשה – 20/12/2020

בעבודה זו חל איסור להשתמש בפתרונות המבוססים על נושאים שטרם נלמדו.

- ✓ ניתן להכין את המטלה בזוגות
- ✓ רק **חבר אחד** בצמד יגיש בפועל את העבודה (במידה ומוגש כעבודה זוגית, יש לרשום בשם הקובץ את **2 מספרי הזהות המגישים**).
- ✓ יש להגיש את קבצי הפתרון **תחת שם המכיל את מספרי ת"ז של המגישים**.
- ✓ את החלק התיאורטי יש להגיש בפורמט PDF ואת החלק המעשי יש להגיש בקובץ PY עם שם קובץ מס' ת.ז. (לדוגמא אם מס' ת.ז. 123456789, קובץ להגשה 123456789[1].py).
- ✓ חובה להשתמש **בשמות הפונקציות המוגדרות**.
- ✓ שימו לב, הפלט של דוגמאות ההרצה הוא בהתאם לסביבת הפיתוח Python IDLE.
- ✓ חובה לכל פונקציה להוסיף doc strings .
- ✓ הגשה דרך מודל בלבד!
- ✓ כל שאלה בנוגע לתרגיל יש להפנות אך ורק לאחראי על התרגיל – ילנה באימייל: elena.chk@gmail.com . פניות בכל בדרך אחרת – לא יענו! בפנייה, יש לציין את : שם הקורס ופרטים מזהים.
- ✓ אישורי ההארכה יינתנו ע"י מרצה בלבד !

* שימו לב: קיים הבדל עקרוני בין הדפסה לבין החזרה של ערך מפונקציה! ברירת המחדל בהיעדר הוראת הדפסה מפורשת היא החזרה בלבד.

[1] 2 מס' ת"ז יש לכתוב בשם הקובץ דרך קו (דוגמה: 123456789-987654321.zip)

המכללה האקדמית להנדסה סמי שמעון

חלק א: Data abstraction, Immutable data

1) יש להגדיר טיפס שלא ניתן לשנות (**immutable type**) של מלבן (**make_rectangle(x,y,leghth,width**).
המלבן מיוצג ע"י המיקום של הפינה השמאלית העליונה של המלבן (**x,y**), אורכו ורוחבו.
המימוש חייב ליישם את עיקרון של הפשטת נתונים (**data abstraction**). יש לממש פעולות הבאות בשכבות הפשטה שונות:

א) **x** - מקבלת מלבן ומחזירה קואורדינטה x של המיקום המלבן.

ב) **y** – בקבלת מלבן ומחזירה קואורדינטה y של המיקום המלבן.

ג) **length** – מקבלת מלבן ומחזירה אורך של המלבן.

ד) **width** – מקבלת מלבן ומחזירה רוחב של המלבן.

ה) **diagonal** – מקבלת מלבן ומחזירה את האורך של האלכסון של המלבן.

ו) **print_rectangle** – מקבלת מלבן ומדפיסה את המלבן.

ז) **center** – מקבלת מלבן ומחזירה את קואורדינטת המרכז של המלבן.

ח) **distance** – מקבלת שני מלבנים ומחזירה את המרחק בין המרכזים של שני המלבנים.

ט) **move** – מקבלת מלבן, deltaX ו-deltaY ומזיזה את המלבן במערכת צירים. מחזירה מלבן חדש אחרי השינויים.

י) **resize** – מקבלת מלבן ו-resize factor. משנה את הרוחב ואת האורך של המלבן בהתאם לפקטור. מחזירה מלבן חדש אחרי השינויים.

יא) **average_rectangle** – מקבלת שני מלבנים ומחזירה את המלבן הממוצע בין שני המלבנים (ממוצע של רוחב ואורך) וממקמת אותו באמצע בין שני המלבנים שקיבלה בקלט.

הערה: אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים, tuple ופונקציות).

דוגמת הרצה מחייבת:

```
>>> r1 = make_rectangle(3, 4, 10, 26)
>>> r1
<function make_rectangle.<locals>.dispatch at 0x031C8BB8>
>>> x(r1)
3
>>> y(r1)
4
>>> length(r1)
10
>>> width(r1)
26
>>> diagonal(r1)
27.85677655436824
>>> print_rectangle(r1)
Rectangle: point = (3,4); size = 10x26
>>> center(r1)
(8.0, 17.0)
>>> distance(r1, make_rectangle(6, 9, 5, 8))
4.031128874149275
```

```
>>> print_rectangle(move(r1, 2, -3))
Rectangle: point = (5,1); size = 10x26
>>> print_rectangle(resize(r1, 0.5))
Rectangle: point = (3,4); size = 5.0x13.0
>>> print_rectangle(move(resize(r1, 1.5), -8, 2))
Rectangle: point = (-5,6); size = 15.0x39.0
>>> r2 = make_rectangle(6, 9, 5, 8)
>>> print_rectangle(average_rectangle(r1, r2))
Rectangle: point = (4.5,6.5); size = 7.5x17.0
>>> print_rectangle(average_rectangle(move(r1, -1, -2), resize(average_rectangle(r2,
make_rectangle(6, 9, 5, 8)), 2)))
Rectangle: point = (4.0,5.5); size = 10.0x21.0
>>> average_rectangle(move(r1, -1, -2), resize(average_rectangle(r2, make_rectangle(6, 9, 5, 8)),
2))
<function make_rectangle.<locals>.dispatch at 0x031C8A98>
```

2) יש להגדיר טיפס שלא ניתן לשנות (**immutable type**) של וקטור (**make_vector(size,list)**).
הווקטור יהיה מורכב מגודל של הווקטור ורשימת ערכים. המימוש חייב ליישם את עיקרון של הפשטת נתונים (**data abstraction**). יש לממש פעולות הבאות בשכבות הפשטה שונות:

א) **size** – מקבלת ווקטור ומחזירה את האורך של הווקטור.

ב) **values** – מקבלת ווקטור ומחזירה את רשימה של הערכים.

ג) **print_vector** – מקבלת ווקטור ומדפיסה את ערכי הווקטור וגודלו.

ד) **value_at** – מקבלת ווקטור ואינדקס ומחזירה את הערך באינדקס נתון.

ה) **reverse** – מקבלת ווקטור ומחזירה ווקטור חדש כך שסדר האיברים בווקטור הפוך.

ו) **norm1** – מקבלת ווקטור ומחזירה את הנורמה 1 של הווקטור (סכום של ערכים מוחלטים של האיברים בווקטור).

ז) **norm2** – מקבלת ווקטור ומחזירה את הנורמה 2 של הווקטור (שורש של הסכום של הריבועים של הערכים בווקטור).

ח) **insert** – מקבלת ווקטור וערך חדש ומוסיפה ערך לסוף של הווקטור. מחזירה ווקטור חדש.

ט) **delete** – מקבלת ווקטור ואינדקס ומוחקת את הערך באינדקס נתון. מחזירה ווקטור חדש.

י) **sort_vector** – מקבלת ווקטור ומחזירה ווקטור חדש ממוין מהקטן לגדול.

יא) **add_vector** – מקבלת שני ווקטורים ומחזירה ווקטור חדש שהוא חיבור בין ווקטורים.

יב) **mult_scalar** – מקבלת ווקטור וסקלר ומחזירה וקטור חש של מכפילת הווקטור בסקלר.

הערות: אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים, רשימות, tuple ופונקציות).

דוגמת הרצה מחייבת:

```
>>> v1 = make_vector(5, (1,2,3,4,5))
>>> v1
<function make_vector.<locals>.dispatch at 0x03308B70>
```

```
>>> size(v1)
5
>>> values(v1)
(1, 2, 3, 4, 5)
>>> print_vector(v1)
size = 5; values = (1, 2, 3, 4, 5)
>>> print_vector(reverse(v1))
size = 5; values = (5, 4, 3, 2, 1)
>>> print_vector(v1)
size = 5; values = (1, 2, 3, 4, 5)
>>> value_at(v1, 3)
4
>>> print(value_at(v1, 8))
None
>>> norm1(v1)
15
>>> norm2(v2)
11.832159566199232
>>> print_vector(insert(v1,6))
size = 6; values = (1, 2, 3, 4, 5, 6)
>>> v2 = reverse(insert(insert(v1,6),7))
>>> print_vector(v2)
size = 7; values = (7, 6, 5, 4, 3, 2, 1)
>>> print_vector(delete(v2,3))
size = 6; values = (7, 6, 5, 3, 2, 1)
>>> v3 = delete(delete(delete(v1,0),3),1)
>>> print_vector(v3)
size = 2; values = (2, 4)
>>> print_vector(delete(v3,4))
size = 2; values = (2, 4)
>>> v3 = delete(delete(v3,0),0)
>>> print_vector(v3)
size = 0; values = ()
>>> print_vector(delete(v3, 2))
size = 0; values = ()
>>> print_vector(sort_vector(v2))
size = 7; values = (1, 2, 3, 4, 5, 6, 7)
>>> print_vector(mult_scalar(v2,3))
size = 7; values = (21, 18, 15, 12, 9, 6, 3)
>>> print_vector(add_vector(v1,reverse(v2)))
size = 7; values = (2, 4, 6, 8, 10, 6, 7)
>>> print_vector(add_vector(reverse(add_vector(v1,reverse(v2))),v2))
size = 7; values = (14, 12, 15, 12, 9, 6, 3)
```

המכללה האקדמית להנדסה סמי שמעון

חלק ב: Conventional Interface, Pipeline

3) בכל משימות הנתונות בסעיף זה יש להשתמש בפונקציות מובנות שנלמדו בכיתה: map, filter, reduce, וכד'. כל הפונקציות שתכתבו בתרגיל זה צריכות לתמוך בכל רצף ש-Python תומך בו, כלומר, כל רצף שהפונקציות לעיל תומכות בו או שלולאת for יודעת לעבור עליו. אם על הפונקציה להחזיר רצף, אז סוג הרצף לא חשוב (למשל, אפשר להחזיר tuple או רשימה, או להחזיר את מה ש-map או filter החזירו).

הערה: אסור להשתמש בלולאות בשאלה הנ"ל.

1. לכתוב פונקציה avg_grades שבהינתן
a. רשימת זוגות – שם של הקורס ורשימת הציונים שקיבל סטודנט בקורס הנ"ל.

הפונקציה מחזירה רשימת הקורסים עם ציון ממוצע עבור כל קורס. דוגמת הרצה:

```
>>> courses = (('a', [81, 78, 57])), ('b', [95, 98]), ('c', [75, 45]), ('d', [58])
>>> print(avg_grades(courses))
(('a', 72.0), ('b', 96.5), ('c', 60.0), ('d', 58.0))
```

2. לכתוב פונקציה add_factors שבהינתן
a. רשימת זוגות - קורסים עם ציון (כמו בפלט של סעיף 1).
b. רשימת זוגות – קורסים ופקטור עבור קורסים מסוימים שעבורם צריך לחשב פקטור.

הפונקציה מחזירה רשימת הקורסים עם הציונים מעודכנים אחרי הפקטור. דוגמת הרצה:

```
>>> courses = (('a', [81, 78, 57])), ('b', [95, 98]), ('c', [75, 45]), ('d', [58])
>>> factors = (('c', 15), ('a', 20))
>>> print(add_factors(avg_grades(courses), factors))
(('a', 92.0), ('b', 96.5), ('c', 75.0), ('d', 58.0))
```

3. לכתוב פונקציה total_average שבהינתן
a. רשימת זוגות - קורסים עם ציון (כמו בפלט של סעיף 1).
b. רשימת זוגות - קורסים ונקודות הזכות שלהם. לכל קורס מ-a צריך להיות זוג עם נקודות זכות, אך סדר של הקורסים יכול להיות שונה מרשימה ב-a.

הפונקציה מחזירה את הממוצע הכללי של כל הקורסים. דוגמת הרצה:

```
>>> courses = (('a', [81, 78, 57])), ('b', [95, 98]), ('c', [75, 45]), ('d', [58])
>>> credits = (('b', 2.5), ('d', 4), ('c', 3.5), ('a', 5))
>>> print(total_average(avg_grades(courses), credits))
69.55
```

המכללה האקדמית להנדסה סמי שמעון

חלק ג: Mutable data, message passing, dispatch function, dispatch dictionary

4) בסעיף זה ניתן להשתמש ולהיעזר בפונקציות הממומשות בסעיף הקודם.
עליכם להשתמש ב- Dispatch Dictionary ו-Message Passing

יש לכתוב פונקציה שבהינתן:

a. מילון של קורסים והציון ממצע בהם.
b. מילון של קורסים ונקודות הזכות שלהם.
c. מילון של רשימות קורסים לפי סוגים.

יוצרת מחסן נתונים (**courses_warehouse**) המאפשר להפעיל פעולות שונות על נתוניו:

a. ציון בקורס עם מספר נקודות זכות מינימאלי\מקסימלי.
עונה להודעות: min_credits/max_credits.

b. ציון ממוצע\מינימאלי\מקסימאלי עבור קורסים מטיפוס מסויים.
עונה להודעות: avg/min/max_course, פונקציה תקבל שם של טיפוס.

ולעדכן מצב לוקאלי של מחסן נתונים:

c. הוספת קורס חדש יחד עם הציון בו וטיפוסו.
עונה להודעות: add_course, פונקציה תקבל : שם של קורס, ציון וטיפוס

הנחות:

a. כל קורס שייך לטיפוס(קטגוריה)אחד בלבד.
b. לא ניתן להוסיף קורס ששייך לטיפוס הלא קיים במחסן נתונים.
c. לא ניתן להוסיף טיפוס חדש.

דוגמת הרצה:

```
>>> courses = (('a', 80), ('b', 95), ('c', 75), ('d', 58))
>>> credits = (('a', 2.5), ('b', 4), ('c', 3.5), ('d', 5))
>>> courses_dict = dict(courses)
>>> credits_dict = dict(credits)
>>> types = {'t1':('a', 'b'), 't2':('c',), 't3':('d',)}
>>> w = make_warehouse(courses_dict, credits_dict, types)
>>> print(w['min_credits']())
80
>>> print(w['max_credits']())
58
>>> print(w['min_course']('t1'))
80
>>> print(w['max_course']('t1'))
95
>>> print(w['avg_course']('t1'))
87.5
>>> w['add_course']( 'e', 90, 't2')
>>> print(w['max_course']('t2'))
90
>>> print(w['min_course']('t2'))
75
>>> print(w['avg_course']('t2'))
82.5
```

המכללה האקדמית להנדסה סמי שמעון

5) בשאלה זו אתם מתבקשים לממש טיפוס נתונים חדש בשם **sequence** שעובד על כל סוגי הרצף האפשריים. יש לרשום פונקציה **make-sequence** אשר תיצור אובייקט של **sequence** לפי שיטת **dispatch_dictionary** וישמור את האלמנטים שלו ברשימה (ניתן להשתמש בטיפוס מובנה כמו list או tuple).

הפעולות המוגדרות על טיפוס:

א) **filter** שתקבל כפרמטר פונקציה של ארגומנט אחד ותחזיר **tuple** של ערכים מסוננים.

ב) **filter_iterator** שתקבל כפרמטר פונקציה של ארגומנט אחד ותחזיר iterator מעגלי שיחזיר ערכים מסוננים על ידי קידום ('**next**') או ע"י קידום בכיוון הפוך ('**reverse**').

ג) **reverse** שתחזיר **tuple** עם כל ערכי הרשימה מסודרים בסדר הפוך.

ד) **extend** שתקבל כפרמטר רצף ותשלים את הרשימה הקיימת ע"י ערכים שהתקבלו.

במקרה שפעולות **filter** ו-**filter_iterator** לא יקבלו ארגומנטים יש להחזיר את כל האלמנטים ללא סינון.

דוגמת הרצה:

```
>>> s1=make_sequence((1,2,3,4,5))
>>> s1['filter'](lambda x: x%2==0)
(2, 4)
>>> p1=s1['filter_iterator'](lambda x: x<4)
>>> p1['next']()
1
>>> p1['next']()
2
>>> p1['next']()
3
>>> p1['next']()
1
>>> p1['next']()
2
>>> p1['reverse']()
2
>>> p1['reverse']()
1
>>> p1['reverse']()
3
>>> s1['extend'](s1['filter'](lambda x: x%2!=0))
>>> s1['filter'](lambda x: x>2)
(3, 4, 5, 3, 5)
>>> s1['filter']()
(1, 2, 3, 4, 5, 1, 3, 5)
>>> (make_sequence(s1['reverse']()))['filter'](lambda x: x<4)
(3, 1, 3, 2, 1)
```

המכללה האקדמית להנדסה סמי שמעון

חלק ד: שאלות טאורטיות

6) סמנו אילו מהטענות נכונות והסבירו בקצרה לכל טענה:

א) פונקציה lambda לא יכולה לקבל פונקציה רגילה כפרמטר.

ב) מותר לעשות פקודות השמה פשוטות (כמו x=5) בפונקציה ללא שם.

ג) מותר להשתמש בלולאת for בתוך פונקציה מסדר גבוהה (high-order function).

ד) פונקציה ללא שם ניתן להחזיר מפונקציה ללא שם אחרת ולהעביר לפונקציה ללא שם אחרת בתור ארגומנט.

ה) ב Python 3 -משתמשים בהצהרה nonlocal על מנת לעדכן קשירה של משתנה במסגרת גלובאלית.

ו) לפי מודל הסביבות, הפעלת פונקציה יוצרת קשירה חדשה לשם של הפונקציה במסגרת שמרחיבה את הסביבה הנוכחית.

ז) בשפות עם Lexical Scoping ניתן לממש טיפוסי נתונים חדשים שהם mutable תוך שימוש בפונקציות והשמה לא לוקאלית (nonlocal).

בהצלחה !