המכללה האקדמית להנדסה סמי שמעון

עבודת הגשה מס' 3

תאריך הגשה – 24/03/2024

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 בהערה את שמות המגישים ואת מספרי הזהות שלהם). יש להגיש את קבצי הפיתרון תחת שם המכיל את מספרי ת"ז של המגישים.
- ✓ את החלק התיאורטי יש להגיש בפורמט PDF ואת החלק המעשי יש להגיש בקובץ ZIP עם שם קובץ מס' ת.ז. (לדוגמא אם מס' ת.ז. 123456789, קובץ להגשה 123456789[1].zip) הכולל קובץ PY עם שם "solution.py" .
- ✓ חובה להשתמש בשמות וחתימות הפונקציות כפי שמוגדרות בעבודה – אי-עמידה בדרישה הזאת עלולה לגרום לפסילה בבדיקה אוטומטית!
- ✓ שימו לב, הפלט של דוגמאות ההרצה הוא בהתאם לסביבת הפיתוח Python IDLE (בהרצה מתוך הscript). לבדיקה ניתן להשתמש IDLE גרסאות 3.11 או 3.12
- ✓ חובה לכל פונקציה להוסיף doc strings .
- ✓ הגשה דרך מודל בלבד!
- ✓ כל שאלה בנוגע לתרגיל יש להפנות אך ורק לפורם של עבודה 3.
- ✓ אישורי ההארכה יינתנו ע"י מרצה בלבד !
- ✓ יום איחור בהגשת עבודת הגשה יעלה 3% מהציון. שבוע איחור יעלה 20% מהציון. אחרי שבוע של איחור, העבודה לא תתקבל. אין ערעורים במקרה של איחור!

* 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)

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 התחתונה 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 ליצור מבנה של מטריצה (**matrix**) כולל ממדים של מטריצה ורשימת מספרים (לא רשימות). יש לממש פונקציות עזר כדי להדפיס את הרכיבים המרכיבים מטריצה (פונקציות ממשק): ממדים של מטריצה (**n**,**m**), ערכים של מטריצה (**matrix**) ופונקציות; הדפסת מטריצה (**PrintMatrix**), חיבור מטריצות (**AddMatrix**), הפיכת מטריצה (**TransposeMatrix**) ומכפלת מטריצות (**MulMatrix**), דוגמאות מופיעות למטה. אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים, רשימה ופונקציה)!

דוגמת הרצה מחייבת:

```
>>> m1=make_matrix(2,3,[1,2,3,4,5,6])
>>> m1
<function make_matrix.<locals>.dispatch at 0x000001637E0A5598>
>>> n(m1)
2
>>> m(m1)
3
>>> matrix(m1)
[1, 2, 3, 4, 5, 6]
>>> m2=make_matrix(2,3,[3,4,5,6,7,8])
>>> PrintMatrix(m1)
  1  2  3
  4  5  6
>>> PrintMatrix(m2)
  3  4  5
  6  7  8
>>> PrintMatrix(AddMatrix(m1,m2))
  4  6  8
 10 12 14
>>> PrintMatrix(MulMatrix(m1, TransposeMatrix(AddMatrix(m1,m2))))
 40 76
 94 184
>>>
```

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.

א) 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))
```

ב) 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))
>>>
(('a', 92.0), ('b', 96.5), ('c', 75.0), ('d', 58.0))
```

ג) 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
```

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

4) יש לממש טיפוס נתונים חדש בשם **sets** שמייצג קבוצה תוך שימוש ב- **tuple**, **dispatch function** ו-**message passing**. הקבוצה אוניברסלית מורכבת ממספרים שלמים מ **0-** עד **20** כולל. יש לממש את הפעולות הבאות:

א) החלפת איברים בקבוצה.

ב) הצגת קבוצה - שמחזירה איברים של קבוצה בצורת מחרוזת.

ג) בדיקה אם איבר שייך לקבוצה.

ד) בדיקה אם איבר לא שייך לקבוצה.

ה) פעולה (**not**) שמחזירה קבוצה משלימה.

ו) פעולה (**+**) שמחזירה קבוצה שמורכת מאיברים של הקבוצה ואיברים של קבוצה נוספת.

ז) פעולה (*) שמחזירה קבוצה שמורכת מאיברים משותפים של הקבוצה ואיברים של קבוצה נוספת.

ח) פעולה (\\) שמחזירה קבוצה חדשה המורכבת מאיברים הקבוצה ללא איברים של קבוצה נוספת.

ט) פעולה (**xor**) שמחזירה קבוצה חדשה המורכבת מאיברים הקבוצה ואיברים של קבוצה נוספת ללא איברים משותפים.

דוגמת הרצה מחייבת:

```
>>> s1=sets((1,2,3,4,5,100))
>>> s1
<function sets.<locals>.dispatch at 0x03D3DD68>
>>> s1('view')
'{1, 2, 3, 4, 5}'
>>> s1('in',3)
True
>>> s1('not in',31)
True
>>> s1('not in',3)
False
>>> s2=s1('not')
>>> s2
<function sets.<locals>.dispatch at 0x03D4F108>
>>> s2('view')
'{6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}'
>>> s1('set',(1,2,3,4,5,7,9,12,17))
>>> s2('set',(2,4,5,10,14,16,20))
>>> s1('+',s2)('not')('view')
'{6, 8, 11, 13, 15, 18, 19}'
>>> s1('*',s2)('xor',s1('\\',sets((2,3,5,12))))('view')
'{1, 2, 5, 7, 9, 17}'
```

5) 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:

a. 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
```

6) 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 function** וישמור את האלמנטים שלו ברצף (ניתן להשתמש בטיפוס מובנה כמו **list** או **tuple**).

<u>הפעולות המוגדרות על טיפוס:</u>

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

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

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

ד) **extend** שתקבל כפרמטר רצף ותשלים את הרצף 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
<function make_sequence.<locals>.dispatch at 0x000002568E992B00>
>>> p1=s1('filter_iterator')(lambda x: x<4)
>>> p1
<function make_sequence.<locals>.get_filter_iterator.<locals>.dispatch at 0x000002568E992D40>
>>> for _ in range(5):
        p1('next')()
1
2
3
1
2
>>> for _ in range(4):
        p1('reverse')()
3
2
1
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

7) 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) רשימה רקורסיבית (**rlist**) שמימשתם בכיתה (כפונקציה **dispatch**) היא רצף.

בהצלחה !