

עבודת הגשה מס' 3

תאריך הגשה – 12/08/2024

הוראות הגשה: (אי קיום הוראות אלו עלול לגרום להורדת ציון!)

1. יש להגיש עד התאריך **12/08/2024** בשעה 23:55 למטלה הקשורה ב-Moodle בלבד.
 2. אין להגיש בשום פנים ואופן למייל של מרצה או מתרגל - אך ורק ב-Moodle.
 3. דחיית העבודה ניתנת רק במקרה של מילואים או אישור מחלה. יש להגיש בקשת סטודנט בצירוף המסמכים. **אין לפנות במייל למתרגל או למרצה בבקשת דחיית העבודה! מייל עם בקשת דחיית העבודה לא יקבל מענה כלל.**
 4. **אין להגיש באחזור!**
 5. ניתן להגיש את העבודה או ביחיד או בזוגות. **יש לרשום את כל השותפים לעבודה בתוך הקובץ.**
 6. את העבודה בזוגות יש להגיש על ידי סטודנט אחד עם שם הקובץ שיהיה מורכב מהמילה "HW3" ושני מספרי ת"ז מופרדים בקו תחתון ביניהם.
לדוגמא: HW3_123456789_123456789.zip
 7. במקרה של העתקה מלאה או חלקית של העבודה (מסטודנטים אחרים, מ-Internet או מכל מקום אחר), יינתן ציון 0 על העבודה של **כלל הסטודנטים המעורבים** והם יעלו לוועדת משמעת.
 8. כל שאלה בנוגע לתרגיל יש להפנות אך ורק לאחראי על התרגיל – מלכ באימייל "malekgh@ac.sce.ac.il"
- פניות בכל בדרך אחרת – לא יענו! בפנייה, יש לציין את : שם הקורס

חלק א: Data abstraction, Immutable data

(1) יש להגדיר טיפס שלא ניתן לשנות (**immutable type**) של מספר בחזקה b^p (**make_power**). המימוש חייב ליישם את עיקרון של הפשטת נתונים (**data abstraction**). יש לממש פעולות הבאות (**API** או ממשק)

בשכבות הפשטה שונות:

(א) **base** – מחזירה בסיס.

(ב) **power** – מחזירה חזקה.

(ג) **print_power** – מדפיסה מספר בחזקה בפרמטר b^p .

(ד) **calc_power** – מחשבת ומחזירה את התוצאה.

(ה) **mul_power** – מכפילה בין שני מספרים.

(ו) **div_power** – מחלקת בין שני מספרים.

(ז) **improve_power** – בודקת האם ניתן להקטין בסיס ע"י שינוי (הגדלת) חזקה ומחזירה את המספר החדש ($b^p = a^{n^p}$).

הערה: אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים ופונקציות)!

דוגמת הרצה מחייבת:

```
>>> x=make_power(4,5)
>>> x
<function make_power.<locals>.dispatch at 0x0421DB70>
>>> base(x)
4
>>> power(x)
5
>>> print_power(x)
4^5
>>> print_power(improve_power(x))
2^10
>>> print_power(mul_power(improve_power(x),make_power(2,5)))
2^15
>>> y=make_power(9,2)
>>> print_power(improve_power(y))
3^4
>>> print_power(mul_power(x,y))
82944
>>> print_power(mul_power(improve_power(y),make_power(3,5)))
3^9
>>> print_power(div_power(improve_power(y),make_power(3,5)))
3^-1
>>> print_power(div_power(mul_power(make_power(2,3),make_power(2,8)),
make_power(2,4)))
2^7
>>> print_power(div_power(mul_power(improve_power(make_power(8,1)),
improve_power(make_power(256,1))),improve_power(make_power(16,1))))
2^7
>>> print_power(make_power(12,1))
12
>>> print_power(make_power(12,0))
1
```

- (2) יש להגדיר טיפס שלא ניתן לשנות (immutable type) של עץ בינארי (`make_tree`) כולל ערך מספרי ובנים שמאלי וימני (אם אין משתמשים ב-`None`). המימוש חייב ליישם את עיקרון של הפשטת נתונים (`data abstraction`). יש לממש פעולות הבאות (`API` או ממשק) בשכבות הפשטה שונות:
- (א) `value` – מחזירה ערך מספרי.
 - (ב) `left` – מחזירה בן שמאלי.
 - (ג) `right` – מחזירה בן ימני.
 - (ד) `print_tree` – מדפיסה עץ לפי שיטת Inorder הכוללת הדפסת בן שמאלי, ערך ובן ימני.
 - (ה) `count_value` – מקבלת ערך כפרמטר ומחזירה כמה פעמים הוא מופיעה בעץ.
 - (ו) `tree_BST` – מחזירה `True` אם עץ הוא עץ חיפוש.
 - (ז) `tree_depth` – מחזירה גובה עץ.
 - (ח) `tree_balanced` – מחזירה `True` אם עץ הוא עץ מאוזן: עץ שבו הפרש גובהם של שני תתי-העצים של הבנים של כל צומת הוא לכל היותר 1.
- הערה: אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים שלמים ופונקציות)!
- רמז: חלק מפונקציות הן פונקציות רקורסיביות.

דוגמת הרצה מחייבת:

```
>>> tree1=make_tree(12,make_tree(6,make_tree(8,None,None),None),make_tree(7,make_tree(8,None,None),make_tree(15,None,None)))
>>> tree2=make_tree(12,make_tree(6,make_tree(3,make_tree(1,None,None),None),make_tree(8,make_tree(7,None,None),None)),make_tree(15,None,make_tree(20,make_tree(17,None,None),None)))
>>> tree1
<function make_tree.<locals>.dispatch at 0x03E6DA08>
>>> tree2
<function make_tree.<locals>.dispatch at 0x03E6DC90>
>>> value(tree1)
12
>>> value(left(tree1))
6
>>> value(right(left(tree2)))
8
>>> print_tree(tree1)
8 6 12 8 7 15
>>> print_tree(tree2)
1 3 6 7 8 12 15 17 20
>>> count_value(tree1,8)
2
>>> tree_BST(tree1)
False
>>> tree_BST(tree2)
True
>>> tree_depth(tree1)
2
>>> tree_depth(tree2)
3
>>> tree_balanced(tree1)
True
>>> tree_balanced(tree2)
False
```

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) כתוב פונקציה `grades_avg` שבהינתן:

- רשימת זוגות – שם של הקורס ורשימת הציונים שקיבל סטודנט בקורס הנ"ל.

הפונקציה מחזירה רשימת הקורסים עם ציון ממוצע עבור כל קורס.

דוגמת הרצה מחייבת:

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

(ב) כתוב פונקציה `add_factors` שבהינתן

• רשימת זוגות - קורסים עם ציון כמו בפלט של סעיף 1.

• רשימת זוגות – קורסים ופקטור עבור קורסים מסוימים שעבורם צריך לחשב פקטור.

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

- רשימת זוגות - קורסים עם ציון כמו בפלט של סעיף 1.

- רשימת זוגות – קורסים ופקטור עבור קורסים מסוימים שעבורם צריך לחשב פקטור.

הפונקציה מחזירה רשימת הקורסים עם הציונים מעודכנים אחרי הפקטור.

דוגמת הרצה מחייבת:

```
1. >>> courses = (('a', [81, 78, 57]), ('b', [95, 98]), ('c', [75, 45]), ('d', [58]))
2. >>> factors = (('c', 15), ('a', 20))
3. >>> print(add_factors(grades_avg(courses), factors))
4. (('a', 92.0), ('b', 96.5), ('c', 75.0), ('d', 58.0))
5.
```

(ג) כתוב פונקציה `total_average` שבהינתן:

- רשימת זוגות - קורסים עם ציון כמו בפלט של סעיף 1

- 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 מחייבת:

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



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

(4) יש יש לממש טיפוס נתונים חדש בשם `magic_box` שמאפשר לאחסן ולהפעיל פריטים קסומים בשיטת `message passing` עם `dispatch function`. כל פריט יכול להיות כלי קסם עם יכולות מיוחדות. יש לממש את הפעולות הבאות:

- a. הוספת פריט קסום ל-`box`: `add_item`.
- b. הסרת פריט קסום מה-`box` לפי שם: `remove_item`.
- c. הפעלת פריט קסום לפי שם עם פרמטרים: `use_item`. (רמז: יש להשתמש ב-`args*` עבור פרמטרים)
- d. הצגת רשימת כל הפריטים הקסומים ב-`box`: `list_items`.
- e. קבלת פריט קסום לפי שם: `get_item`.

הערה: אין להשתמש בטיפוסים מובנים של Python !!!

דוגמת הרצה מחייבת:

```
>>> box = make_magic_box()
>>> box('add_item')('Healing Potion', lambda hp: f"Healed {hp} HP")
>>> box('add_item')('Fireball', lambda damage: f"Caused {damage} damage")
>>> box('add_item')('Teleport', lambda x, y: f"Teleported to coordinates ({x}, {y})")
>>> box('list_items')()
['Healing Potion', 'Fireball', 'Teleport']
>>> box('use_item')('Healing Potion', 50)
'Healed 50 HP'
>>> box('use_item')('Fireball', 100)
'Caused 100 damage'
>>> box('use_item')('Teleport', 10, 20)
'Teleported to coordinates (10, 20)'
>>> box('remove_item')('Healing Potion')
>>> box('list_items')()
['Fireball', 'Teleport']
>>> box('get_item')('Fireball')
<function <lambda> at 0x...>
```

- (5)** בשאלה זו אתם מתבקשים לממש טיפוס נתונים חדש בשם `transform_sequence_iterator`. ניתן ליצור אובייקט של `transform_sequence_iterator` על רצף `seq` ופונקציה `f` (של ארגומנט אחד) על מנת לעבור על ערכים של הרצף החדש שמתקבל על ידי הפעלת `f` על ערכים של `seq`. יש לממש פונקציה `get_transform_sequence_iterator` היוצרת אובייקט של `transform_sequence_iterator` לפי שיטת `dispatch dictionary` פעולות מוגדרות על הטיפוס:
1. `next` פעולה מחזירה את האלמנט הבא של הרצף המתקבל.
 2. `has_next` פעולה מחזירה `True` כל עוד יש עוד אלמנטים ברצף.
 3. `reset` פעולה שמאפסת את האיטרטור להתחלה של הרצף.

הערה: במידה ופונקציה לא קיבלה את הארגומנט השני (`f`) אז היא תחזיר איטרטור על ערכים של הרצף הארגומנט (`seq`) ללא שינוי.
 הערה: אין לטפל בחריגות ש Python-מעלה במקרים של חישובים כושלים כגון חלוקה ב-0 וכד'. (תטפלו בחריגות בעבודה הבאה).

דוגמת הרצה:

```

1. >>> it = get_transform_sequence_iterator([2, 4, 8], lambda x: x**2)
2. >>> while it['has_next']():
3. ...     print(it['next']())
4. ...
5. 4
6. 16
7. 64
8. >>> it['reset']()
9. >>> print(it['next']()) # returns 4 again
10. 4
11. >>> print(it['skip']()) # skips 16 and returns it
12. 16
13. >>> print(it['next']()) # returns 64
14. 64
15. >>> it = get_transform_sequence_iterator([2, 4, 8])
16. >>> for _ in range(5):
17. ...     print(it['next']())
18. ...
19. 2
20. 4
21. 8
22. no more items
23. no more items
24.

```

בהצלחה !