המכללה האקדמית להנדסה סמי שמעון

עבודת הגשה מס' 3

תאריך הגשה – 22/12/2022

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 הפתרון 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 הפונקציות המוגדרות.
- ✓ שימו לב, הפלט של דוגמאות ההרצה הוא בהתאם לסביבת הפיתוח Python IDLE (בהרצה מתוך הscript).
- ✓ חובה לכל פונקציה להוסיף doc strings .
- ✓ הגשה דרך מודל בלבד!

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 טיפוס שלא ניתן לשנות (immutable type) של כרטיס טיסה (make_ticket) הכולל מספר טיסה, תאריך (יום וחודש), זמן המראה (שעה ודקות).

המימוש חייב ליישם את עיקרון של הפשטת נתונים (**data abstraction**). יש לממש פעולות הבאות בשכבות הפשטה שונות:

א) **flight** – מקבלת מופע של כרטיס טיסה ומחזירה את מספר הטיסה (מחרוזת המתחילה באותיות A-B ומסתיימת בארבע ספרות).

ב) **date** - מקבלת מופע של כרטיס טיסה ומחזירה את תאריך הטיסה בפורמט DDMMM (*).

ג) **hour** – מקבלת מופע של כרטיס טיסה ומחזירה את רכיב השעה.

ד) **minute** – מקבלת מופע של כרטיס טיסה ומחזירה את רכיב הדקות.

ה) **print_ticket_info** – מקבלת מופע כרטיס טיסה ומחרוזת המייצגת את הפורמט הרצוי להדפסת המידע:

סוגי הפורמטים השונים הם:

- **'flight date hh:mm'**– מספר טיסה, תאריך, שעה – זוהי הדפסת ברירת המחדל של מופע מסוג כרטיס טיסה.
- **'flight hh:mm'** – מספר טיסה, שעה.
- **'date hh:mm'** – תאריך, שעה.
- **'hh:mm'** – שעה.

ו) **time_difference** – מחשבת את הפער בדקות בין הזמנים של שני מופעי כרטיס טיסה שמקבלת (יש להתייחס לתאריכים בחישוב).

ז) **time_correction** – מקבלת מופע כרטיס טיסה ופרמטר **דקות** ומחזירה מופע כרטיס טיסה **חדש** ע"י הוספת/חיסור הדקות (יש להתייחס לתאריכים בחישוב).

בסעיף ז' התאריך המקסימלי אשר יכול להתקבל עבור המופע החדש הוא 31DEC.

* DDMMM – פורמט תאריך אשר הימים מיוצגים ע"י שתי ספרות והחודשים הם שלושת האותיות הראשונות של החודש הלועזי.

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

דוגמת הרצה מחייבת:

```
>>> t1 = make_ticket('F0545', 8, 1, 7, 40)
>>> t1
<function make_time.<locals>.dispatch at 0x0342D588>
>>> date(t1)
'08JAN'
>>> flight(t1)
'F0545'
>>> hour(t1)
7
>>> minute(t1)
```

```
40
>>> print_ticket_info(t1)
'F0545 08JAN 07:40'
>>> t2 = make_ticket('TK0789', 9, 2, 10, 55)
>>> print_ticket_info(t2, 'date hh:mm')
'08JAN 10:55'
>>> print(time_difference(t1, t2))
'195'
>>> print_ticket_info(time_correction(t1, 220), 'flight hh:mm')
'F0545 11:20'
>>> time_correction(t2, -50)
>>> print_ticket_info(t2)
'TK0785 08JAN 10:05'
```

2) יש להגדיר טיפוס שלא ניתן לשנות (**immutable type**) של אלמנט בעץ בינארי (make_tree) הכולל ערך מספרי (ערך הצומת הנוכחי) ובנים-בן שמאלי ובן ימני.

המימוש חייב ליישם את עיקרון של הפשטת נתונים (**data abstraction**). יש למשש פעולות הבאות בשכבות הפשטה שונות:

א) **value** – ערך מספרי של הצומת הנוכחי בעץ.

ב) **left** – בן שמאלי (עץ, ואם לא קיים כזה אז ערכו הוא None).

ג) **right** – בן ימני (עץ, ואם לא קיים כזה אז ערכו הוא None).

ד) **print_tree** – מדפיסה את ערכי העץ לפי שיטת Inorder (תזכורת: ההדפסה תתחיל בערך הבן השמאלי, ערך האב, והערך הבן הימני עד אשר מגיעים לסוף העץ).

ה) **min_value** – מדפיסה את הערך המינימלי בעץ.

ו) **mirror_tree** – מקבלת מופע של עץ בינארי ומחזירה מופע **חדש** בו סדר הבנים הפוך משל העץ שקיבלה (העץ המתקבל יהיה **מראה** של העץ המקורי).

רמז: סעיפים ד-ה צריכות להיות ממומשות בצורה רקורסיבית.

הערות: אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים שלמים ופונקציות).

דוגמת הרצה מחייבת:

```
>>> tree = make_tree(12,make_tree(6,make_tree(8,None,None),None),make_tree(7,mak
e_tree(2,None,None),make_tree(15,None,None)))
>>> tree
<function make_tree.<locals>.dispatch at 0x0342D8A0>
>>> value(tree)
12
>>> value(left(tree))
6
>>> value(right(tree))
<function make_tree.<locals>.dispatch at 0x0342D7C8>
>>> value(left(right(tree)))
2
>>> print_tree(tree)
8 6 12 2 7 15
```

```
>>> min_value(tree)
2
>>> tree1 = mirror_tree(tree)
15 7 2 12 6 8
```

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.

הערה: אסור להשתמש בלולאות בשאלה הנ"ל, כולל for בכל צורה שהיא.

1. לכתוב פונקציה avg_salary כך שבהינתן רשימה זוגות: שם מחלקה ורשימת משכורות של כמה מעובדי המחלקה, הפונקציה מחזירה את רשימת המחלקות עם המשכורת הממוצעת עבור אותה מחלקה.

דוגמת הרצה:

```
>>> salaries = (('a', [12675, 7876, 8765])), ('b', [9500, 6987]), ('c', [7500, 4576]),
('d', [11654])
>>> print(avg_salary(salaries))
(('a', 9772.0), ('b', 8243.5), ('c', 6038.0), ('d', 11654))
```

2. לכתוב פונקציה add_bonus כך שבהינתן שתי רשימות:

א) רשימה זוגות: (כמו הקלט לעיל).

ב) רשימת זוגות: שמות מחלקות וגובה בונוס בכל מחלקה.

הפונקציה תוסיף לכל המשכורות בכל הרשימות את הבונוס המתאים לפי המחלקה. עבור כל אחד מהרשימות של המשכורות שהתקבלו לאחר הוספת הבונוס, הפונקציה תחזיר משכורת מינימלית, משכורת מקסימלית ואת הממוצע.

דוגמת הרצה:

```
>>> salaries = (('a', [12675, 7876, 8765])), ('b', [9500, 6987]), ('c', [7500, 4576]),
('d', [11654])
>>> bonus = (('a', 1000), ('b', 2000), ('c', 900), ('d', 3500))
>>> print(add_bonus(salaries, bonus))
((8876.0, 13675.0, 10772.0), (7987.0, 10500.0, 9243.5), (5576.0, 8500, 7038.0),
(12654.0, 12654.0, 12654.0))
```

הערה: אפשר להשתמש בפונקציות min, max, sum ו-len מובנות.

המכללה האקדמית להנדסה סמי שמעון

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

4) יש לממש טיפוס נתונים חדש בשם **temperature** מעלות המיוצגות ע"י מספר וסימון המעלות תוך שימוש ב dispatch function ו-message passing . יש לממש את הפעולות הבאות:

א) גישה לערך המעלות וסימון המעלות בהתאם להודעה 'get_value'

ב) עדכון של ערך המעלות וסימון של הייצוג המתאים 'set_value'

ג) ייצוג טקסטואלי .יש לייצג את המעלות ביחד עם הסימון דרך ההודעה המתאימה - 'str'

ד) החלפת ייצוג תבוצע ע"י שליחה של סימון חדש ופונקציית lambda שבאמצעותם תמירו את ערך המעלות הקודם.

בפתרון שלכם תצטרכו לספק המרה יחידות מידה: Celsius (C), Fahrenheit (F), Kelvin (K) טבלת המרות מפורטת ניתן למצוא כאן – Temperature Scale Formulae And Equations

הערה :אין להשתמש בטיפוסים מובנים של Python !!!

דוגמת הרצה:

```
>>> t = make_temperature(25, 'C')
>>> t('get_value')('degrees')
25
>>> t('get_value')('unit')
'C'
>>> t('set_value')('degrees', 31)
>>> t('get_value')('degrees')
31
>>> t('str')
'31 C'
>>> t('convert')( lambda x: x*1.8 + 32, 'F')
>>> t('str')
'87.8 F'
```

5) בשאלה זו אתם מתבקשים לממש טיפוס נתונים חדש בשם **traveler_trip** המייצגת יעדים בטיול אשר מטייל מתכנן להגיע אליהם תוך כדי שימוש ב-**dict**, ב- **dispatch dictionary** ו- **message passing**.

למטייל קיים תיק וירטואלי אשר בו קיים תיעוד ליעדים (ערים, טרקים, מסעדות, פארקים וכו') אליהם רוצה להגיע, מיקום (עיר או מדינה), ועלויות בשקל, דולר או יורו (המידע נשמר כמחרוזת ומאוחסן במילון).

המטייל מוסיף את היעדים לתיק בהנחה לפי סדר הגעתו ליעדים הללו.

עליכם לממש את הפעולות הבאות:

א) **add_destination –** מקבלת יעד, מיקום (עיר או מדינה), עלות, סמל מטבע.

ב) **print_trip** – מדפיסה את כל המידע שהמטייל הוסיף לתיק הווירטואלי.

ג) **view** – מדפיסה את המידע המצוי בתיק לפי פרמטר שקיבלה: יעד – תדפיס את כל היעדים | מיקום – תדפיס את כל המיקומים (ערים/מדינות) ללא חזרות.

ד) **calculate_expenses** – מקבלת סוג מטבע (ILS, USD, EUR) ומחזירה את סך העלויות לטיול עם סוג המטבע המתאים.

ה) **delete_destination** – מקבלת את שם היעד ומוחקת אותו מהתיק.

ו) **search_destination** – מקבלת שם יעד ומדפיסה את המידע עבורו.

דוגמת הרצה:

```
>>> mt=make_traveler_trip('Shahar', 1)
>>> mt
{add_destination: <function make_traveler_trip.<locals>.add_destination at 0x0425E808>,
'print_trip': …, '': …, 'view: …, 'calculate_expenses': …, 'delete_destination': …}, 'search':
…}

>>> mt['add_destination']('Vondelpark', 'Amsterdam', '15', 'EUR')
>>> mt['search_ destination']('Vondelpark')
'Vondelpark, Amsterdam, 15 EUR'
>>> mt['add_destination']('ADAM Lookout, 'Amsterdam', '153', 'ILS')
>>> mt['add_destination']('Lauterbrunnen', 'Switzerland', '1000', 'ILS')
>>> mt['add_destination']('Greendelwald, 'Switzerland', '167', 'EUR')
>>> mt['add_destination']('Lake Como, 'Italy, '30, 'EUR')
>>> mt['view']('destinations')
'Vondelpark, ADAM Lookout, Lauterbrunnen, Greendelwald, Lake Como'
>>> mt['view']('locations')
'Amsterdam, Switzerland, Italy'
>>> mt['calculate_expenses']('ILS')
'1911 ILS'
>>> mt['delete_destination']('ADAM Lookout')

>>> pt = mt['print_trip']()
'Shahar 1'
>>> pt
{'hasMore': <function make_medical_Record.<locals>.printRecord.<locals>.hasMore at
0x0425E970>, 'next': …}
>>> pt['next']
'Vondelpark, Amsterdam, 15 EUR'
>>> while pt['hasMore']():
        pt['next']()
'Lauterbrunnen Switzerland 1000 ILS'
'Greendelwald Switzerland 167 EUR'
'Lake Como Italy 30 EUR'
```

חלק ד: שאלות טאורטיות

6) סמנו אילו מהטענות נכונות והסבירו בקצרה לכל טענה:

א) ניתן להחזיר מפונקציה ולהעביר כארגומנט לפונקציה פונקציה אחרת שהיא מסדר גבוה (**high-order function**).

ב) פונקציה ללא שם (למבדה) יכולה להחזיר אובייקטים מטיפוסים שונים לפי תנאי.

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

ד) לפונקציות מובנות כמו map אפשר להעביר כארגומנט רק פונקציות ללא שם.

ה) מילון (dictionary) הוא מבנה גמיש לחלוטין , שאין שום מגבלה על הטיפוסים של אובייקטים.

ו) לפי שיטת dispatch function עם message passing פונקציה dispatch יכולה לקבל רק פרמטר אחד בלבד.

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

בהצלחה !