המכללה האקדמית להנדסה סמי שמעון

עבודת הגשה מס' 4

מבוא:

מטרת התרגיל היא לבנות כלי להמרה בין 4 יחידות מדידה שונות, בעזרת העקרונות שנלמדו בקורס. את היחידות מדידה נייצג בעזרת 2 שיטות הOOP (בשפת פייטון ושפת שמייטון) שנלמדו בקורס, בהמשך נבנה כלים לחישוב של יחידות מדידה בעזרת Generic functions ולבסוף נאחסן נתונים של יחידות מדידה שונות במבנה נתונים רקורסיבי Binary search tree.

חלק 1 – lambda function & High order functions

נתונים:	**lambda name**
$1\ in = 0.0254\ m$	inches_to_meters
$1\ in = \frac{1}{12}\ m$	inches_to_feets
$1\ mi = 5280\ ft$	miles_to_feets

א. יש לישם 3 lambda לנותונים הנ"ל, ולשמור אותם במשתנים הרשומים בהתאמה:
2 lambda הראשונות מקבלות ערך inches ומחשבות meters או feets בהתאם למשוואה.
Lambda האחרונה מקבלת ערך miles ומחשבת feets בהתאם למשוואה.

ב. יש לכתוב 2 lambda שיעזרו בהמרה מיחידת מדידה אחת לאחרת:
1. lambda - composition המטפלת בהרכבה של 2 פונקציות, ומחזירה lambda חדשה
2. lambda - opposite המקבלת פונקציית המרה בין יחידת מדידה A ל-B, ומחזירה lambda חדשה של המרה מיחידת מדידה B ל-A

במידה ונרצה פונקציה הממירה מ-feets ל-inches, נשתמש ב-lambda: opposite, ונקבל פונקציה:
feets_to_inches = opposite(inches_to_feets)
במידה ונרצה פונקציה הממירה מ-miles ל-inches, נשתמש ב-lambda: composition, ונקבל פונקציה:
miles_to_inches = composition(feets_to_inches, miles_to_feets)
חשוב לזכור - שהרכבה של פונקציות: $f(g(x)) \neq g(f(x))$

ג. יש ליישם את כל סוגי ההמרות האפשריות:
המרות הנזכרות למעלה: $mi \to in, ft \to in, mi \to ft, in \to ft, in \to m$
המרות נוספות אשר מומלץ לישם תחילה, בעזרת שימוש lambda שלמעלה בלבד:
$mi \to m, in \to mi, ft \to mi, ft \to m, m \to mi, m \to ft, m \to in$

- שמות הlambda לפי פורמט הבא: <src unit>_to_<dest unit> בשמות שרשומים בנתונים (inches, meters, feets, miles)

דוגמאת הרצה:

```
>>> inches_to_meters(10)
0.254
>>> feets_to_inches(10)
120.00000000000001
>>> miles_to_inches(10)
633600.0
```

חלק 2 – תכנות מונחה עצמים ב-Python (OOP) ומימוש מערכת אובייקטים Shmython:

1. יש לבנות 2 מחלקות הבאות: Inches, Meters, כל אחת מהמחלקות מייצגת יחידת מדידה שונה.

דרישות לכל מחלקה, attributes & methods:

value – attribute של המחלקה מסוג float המייצג ערך של יחידת המדידה

__init__ - constructor המקבל ערך מספרי או מחרוזת לפי פורמט של יחידת מדידה (דוגמאות למטה), במידה ובבנאי מתקבל ערך מספרי כלשהו יש להמירו ל-float ולשמור בשדה value. במידה ומתקבל ערך str יש לבדוק האם המחרוזת בפורמט תקין (כפי שמודגם למטה). במידה וכן יש לבודד את ערך המספרי ולהמיר ל-float ולשמור גם כן ב-value, ובמקרה שהמחרוזת לא בפורמט תקין יש לזרוק Exception, עם הודעה בפורמט הבא:

'Format error: {}' - בתוך הסוגריים להציב את הערך שהוכנס בפרמטר של הבנאי

במקרה וערך המתקבל לא מחרוזת ולא מספר יש לזרוק Exception, עם הודעה בפורמט הבא:

Type error: <class 'list'> - דוגמה לקבלת ערך מסוג רשימה

דוגמאת הרצה:

```
for v in [25.8, "25.8 in", "25.8 ft", [], Inches]:
    try:
        print(str(Inches(v)))
    except Exception as e:
        print(e)
```

הפלט יהיה:

```
25.80 in
25.80 in
Format error: 25.8 ft
Type error: <class 'list'>
Type error: <class 'type'>
```

__str__ - method overriding אשר ידפיס את ערך ה-string של המחלקה, דוגמאת הרצה:

```
>>> str(Inches(25.8))
'25.80 in'
>>> str(Meters(25.8))
'25.80 m'
```

__repr__ - method overriding לישם כפי שנלמד, דוגמאת הרצה:

```
>>> repr(Inches(25.8))
'Inches(25.8)'
>>> repr(Meters(25.8))
'Meters(25.8)'
>>> str(eval(repr(Inches(25.8))))
'25.80 in'
>>> str(eval(repr(Meters(25.8))))
'25.80 m'
>>>m = eval(repr(Meters(25.8)))
>>>m.value
25.8
```

2. יש לבנות 2 מחלקות ב-Shmython הבאות:
Feets – require implement: **make_feets_class()**
Miles – require implement: **make_miles_class()**

<u>דרישות למחלקות הנ"ל:</u>

א. יש לצרף קוד של מערכת האובייקטים (Shmython) שמצורף לעבודה זו
ב. יש לממש את כלל הדרישות כפי שהוגדרו בסעיף 1
ג. יש להגדיר משתנה אחד לכל make_*_class(), בשמות הנתונים למעלה
ד. יש להוסיף משתנה סטאטי שישמש את פונקציות גנריות בחלק 3, בשם **__type__** משתנה זה יהיה "משתנה סטאטי" שישמור ערך של שם של מחלקה בShmython, דוגמאת הרצה:

```
>>> Feets['get']('__type__')
'Feets'
>>> Miles['get']('__type__')
'Miles'
```

3. יש לממש 3 פונקציות עזר:
מטרת הפונקציות הבאות לתמוך גם ב-OOP הרגיל וגם ב-Shmython (פונקציות אילו יתמכו בכל 4 המחלקות למעלה)

א. to_str – פונקציה מקבלת instance (מופע) של אחד המחלקות שנבנו בחלק זה, ומחזירה את ערך __str__ של המופע (תמיכה גם בShmython)
ב. to_repr - פונקציה מקבלת instance (מופע) של אחד המחלקות שנבנו בחלק זה, ומחזירה את ערך __repr__ של המופע (תמיכה גם בShmython)
ג. type_of - פונקציה מקבלת instance (מופע) של אחד המחלקות שנבנו בחלק זה, ומחזירה את ערך __type__ אם המופע של Shmython, או את הערך עצמו במידה התקבל פרמטר של שם של מחלקה או type(instance) בכל מקרה אחר

- דוגמאת הרצה כללית:

```
>>>Feets = make_feets_class()
>>>f = Feets['new']('25.8 ft')
>>>f['get']('__str__')()
'25.80 ft'
>>> to_str(f)
'25.80 ft'
>>>f['get']('__repr__')()
'Feets['new'](25.8)'
>>> to_repr(f)
'Feets['new'](25.8)'
>>>Miles = make_miles_class()
>>>m = Miles['new'](25.8)
>>>m['get']('__str__')()
'25.80 mi'
>>>m = eval(to_repr(Miles['new']( (25.8)))
>>>m.value
25.8
>>> to_repr(Inches(25.8))
Inches(25.8)
```

חלק 3 – פונקציות גנריות:

לאחר יישום של כלל פונקציות המרה בחלק 1, ו- 4 המחלקות **בחלק 2**, יש לממש פונקציות גנריות שמטפלות בחישוב והמרה של יחידת מדידה אחת ליחידת מדידה אחרת (שימו לב שחובה לטפל גם במקרה הקצה של בקשה להמרה של יחידת מדידה לאותה יחידת מדידה), כפי שנלמד בכיתה, ובהתאם לדרישות הבאות:

1. יש לממש פונקציה גנרית apply בהינתן שם של פעולה, על 2 המחלקות הרגילות (Meters & Inches) ושמות טיפוסים של ארגומנטים. פונקציה תבצע את 2 הפעולות הבאות: חיבור (add) חיסור (sub)

```
>>>apply('add', Inches(1), Meters(1.5))
Inches(60.05511811023623)
>>>apply('add', Meters(1.5), Inches(1))
Meters(1.5254)
>>>apply('sub', Inches(1), Meters(1.5))
Inches(-58.05511811023623)
>>>apply('sub', Meters(1.5), Inches(1))
Meters(1.4746)
```

2. יש לממש פונקציה גנרית apply בהינתן שם של פעולה, על 2 המחלקות Shmython (Feets & Miles) ושמות טיפוסים של ארגומנטים. פונקציה תבצע את 2 הפעולות הבאות: חיבור (add) חיסור (sub)

```
>>>to_repr(apply('add', Feets['new'](1.5), Miles['new'](1)))
Feets['new'](5281.5)
>>>to_repr(apply('add',Miles['new'](1), Feets['new'](1.5)))
Miles['new'](1.0002840909090909)
>>>to_repr(apply('sub', Feets['new'](1.5), Miles['new'](1)))
Feets['new'](-5278.5)
>>>to_repr(apply('sub',Miles['new'](1), Feets['new'](1.5)))
Miles['new'](0.9997159090909091)
```

3. יש לממש בapply גם את הפעולות הבאות: גדול מ (gt) שווה(eq) , בין 4 סוגי היחידות שמומשו בחלק 2 (תמיכה בכל סוג של OOP גם רגיל וגם Shmython), הערך המוחזר יהיה Boolean

דוגמאת הרצה:

```
>>>to_repr(apply('gt', Miles['new'](1), Inches(1.5)))
True
>>>to_repr(apply('gt', Inches(1.5), Miles['new'](1)))
False
>>>to_repr(apply('>', Feets['new'](1), Inches(1.5)))
True
>>>to_repr(apply('>', Inches(1.5), Feets['new'](1)))
False
>>>to_repr(apply('eq', Feets['new'](1), Inches(feets_to_inches(1))))
True
>>>to_repr(apply('eq', Inches(1.5), Feets['new'](inches_to_feets(1.5))))
True
>>>to_repr(apply('==', Feets['new'](1), Inches(1.5)))
False
>>>to_repr(apply('==', Inches(1.5), Feets['new'](1)))
False
```

4. יש לממש פונקציה גנרית apply_coerce, שבהינתן שם של פעולה ושמות טיפוסים של ארגומנטים, תבצע את הפעולה לאחר המרה של כל יחידת מדידה ל-Meters
הערה: יש ליישם שיטה coercion
הערה: התוצאה תמיד תהיה מטיפוס Meters
הערה: יש לתמוך כאן אך ורק במימוש של מחלקות הרגילות שהוגדרו ב-Python (Meters & Inches)

דוגמאת הרצה:

```
>>>coerce_apply('add', Meters(1.5), Inches(1))
Meters(1.5254)
>>>coerce_apply('add', Inches(1), Meters(1.5))
Meters(1.5254)
>>>coerce_apply('sub', Meters(1), Inches(1.5))
Meters(0.9619)
>>>coerce_apply('sub', Inches(1.5), Meters(1))
Meters(-0.9619)
```

חלק 4 – Recursive Data Structures & Exception:

1. יש ליישם 3 מחלקות ב-Python אשר יורשו ממחלקת Exception:
ValueExistsException – מקבלת בבנאי ערך ושולחת למחלקת האב את ההודעה בפורמט הבא:
"Same Value Exists: {}"
ValueNotExistsException – בדומה ל-ValueExistsException שולחת לאב את ההודעה הבאה:
"Value Not Exists: {}"
EmptyTreeException – לא מקבלת ערך בבנאי אבל שולחת לבנאי של האב את ההודעה הבאה:
"The Tree Is Empty"

2. יש ליישם מבנה נתונים Binary Search Tree, מבנה נתונים זה מורכב מ 2 מחלקות (OOP הרגיל) והעץ יתמוך ב-2 סוגי המופעים (גם OOP הרגיל וגם Shmython):
TreeNode – מחלקה אשר מגדירה צומת של עץ. ידוע שבצומת של עץ נשמר הערך ומצביעים לבן השמאלי ולבן הימני (במידה וקיימים) שהם גם כן צמתים מסוג TreeNode.
BSTree – מחלקה אשר שומרת את המופע של שורש העץ: root שיהיה מסוג מחלקת TreeNode.

דרישות:
"Attribute "value" in "TreeNode - מטרת מבנה נתונים זה היא לתמוך בטיפוסים של 4 יחידות מדידה שיושמו בחלק 2, אזי בערך value יהיה שמור מופע של יחידת מדידה כלשהיא, כלומר המימוש.

את מתודות הבאות יש ליישם ב2 המחלקות:

insert – תמיכה בהכנסה של ערכים לתוך עץ בדגש שכל הערכים לא יחזרו שנית.
1. במחלקה BSTree:
   א. יש לבנות מתודה שמפעילה את Insert של root.
   ב. יש לתמוך בהכנסה של רשימה של ערכים במתודה זו.
   ג. יש לטפל בחריגות שיכלו להיזרק מהפעלה של insert על root.
   ד. יש להחזיר את root העדכני
2. במחלקה TreeNode:
   א. יש לבנות מתודה **רקורסיבית** אשר מקבלת ערך וממקמת אותו עפ"י אלגוריתם של "Binary Search Tree", בשונה מ"BSTree" אין לתמוך ברשימה.
   ב. במידה והערך כבר קיים באותו עץ – אין לשנות את העץ ויש לזרוק חריגה מסוג "ValueExistsException".
   ג. יש להחזיר את הצומת שלתוכה נכנס הערך החדש (באחד הבנים left or right).

search – מתודה מקבלת ערך ומחזירה את הצומת בעלת הערך המבוקש, במידה וערך לא קיים בעץ יש להחזיר None.

height – מתודה המחזירה את גובה העץ המקסימלי או צומת מבוקשת.

in_order – מתודה עוברת על עץ בשיטת in order ובונה מערך כללי (ממוין) של כל ערכי העץ (או צומת המבוקשת). (אסור להשתמש בפונקציות מיון כמו: sort & sorted)

delete – תמיכה במחיקה של ערכים ועדכון העץ לעץ BST עדכני ותקין.

1. במחלקה BSTree:
   א. יש לבנות מתודה שמפעילה את delete של root ומעדכנת את root שמתקבל לאחר מחיקה של צומת.
   ב. יש לתמוך במחיקה של רשימה של ערכים במתודה זו.
   ג. יש לטפל בחריגות שיכלו להיזרק מהפעלה של delete על root.
   ד. יש להחזיר את root העדכני.
2. במחלקה TreeNode:
   א. יש לבנות מתודה אשר מחפשת את הערך לאחר מכן מוחקת את הצומת ומעדכנת את העץ כך שצורתו תהיה לפי הגדרה של BST
   ב. יש לזרוק חריגה מסוג "ValueNotExistsException" במידה והערך לא קיים.
   ג. ערך המוחזר יהיה מסוג "TreeNode" בלבד

__repr__ - לממש כפי שנלמד בכיתה
__str__ - יש לתמוך בהדפסה של עץ (להחזיר אותה מחרוזת כמו ב__repr__)

דגשים לפונקציות: delete, search, height, in_order בלבד!

- במחלקה BSTree יש לבדוק אם קיים root במידה ולא יש לזרוק חריגה "EmptyTreeException".
- במחלקה TreeNode מתודות אילו חייבות להיות רקורסיביות.

- בכל מקום שיש צורך להשתמש בהשוואה (==,<,>) יש להשתמש בפונקציות גנריות בלבד מחלק 3.
- ניתן ליצור פונקצית עזר רקורסיבית אחת בלבד בתוך מחלקה TreeNode, בשם successor, כפונקציית עזר למתודה delete.
- בפונקציות שתומכות ברשימה – יש לקבל בפרמטר משתנה אחד ולבדוק האם הוא מסוג רשימה כלשהי (list, tuple) לא לתמוך בvarargs

דוגמאת הרצה כללית לחלק 4:

```
tree = BSTree()
print(tree)
tree.insert(Meters(10))
tree.insert(Inches(10))
tree.insert(Feets['new'](10))
tree.insert(Miles['new'](10))
print(tree)
print(tree.in_order())
print('-- in order print after insert ------------------------------')
for v in tree.in_order():
    if (isinstance(v, dict)):
        print(v['get']('__str__')(), end=" = ")
    else:
        print(v, end=" = ")
    print(apply('add', Meters(0), v))

tree.delete(Meters(0.254))
print('-- in order print after delete ------------------------------')

for v in tree.in_order():
    if (isinstance(v, dict)):
        print(v['get']('__str__')(), end=" = ")
    else:
        print(v, end=" = ")
    print(apply('add', Meters(0), v))
```

פלט:

```
BSTree(None)
BSTree(TreeNode(Meters(10.0), left=TreeNode(Inches(10.0),
right=TreeNode(Feets['new'](10.0))), right=TreeNode(Miles['new'](10.0))))
[Inches(10.0), {'get': <function make_class.<locals>.new.<locals>.get at
0x0000020893B70378>, 'set': <function make_class.<locals>.new.<locals>.set at
0x0000020893B70400>}, Meters(10.0), {'get': <function
make_class.<locals>.new.<locals>.get at 0x0000020893B70488>, 'set': <function
make_class.<locals>.new.<locals>.set at 0x0000020893B70510>}]
-- in order print after insert ------------------------------
10.00 in = 0.25 m
10.00 ft = 3.05 m
10.00 m = 10.00 m
10.00 mi = 16093.44 m
-- in order print after delete ------------------------------
10.00 ft = 3.05 m
10.00 m = 10.00 m
10.00 mi = 16093.44 m
```

חלק 5 – תאורטי

ענה נכון/לא נכון, והסבר את תשובתך בכמה משפטים.

א. "במערכת אוביקטים שמימשנו (שמייטון) אין תמיכה בפונקציות גנריות מסוג פולימורפיזם בהורשה".

ב. בעת הגדרת מחלקה ויצירת אובייקט ממנה אנו יוצרים 2 אובייקטים.

ג. שימוש במילון dispatch תמיד עדיף על שימוש בפונקציית dispatch במימוש הטיפוסים.

ד. העמסת אופרטור ממומשת בעזרת פונקציות גנריות המוגדרות על הטיפוסים ללא קשר הורשה אך בעלי ממשק משותף (הפונקציות האלו הן חלק מהממשק המשותף הזה).

בהצלחה!