עבודת הגשה מס' 3

תאריך הגשה – 26/12/2019

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 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!
- ✓ כל שאלה בנוגע לתרגיל יש להפנות אך ורק לאחראי על התרגיל – מיכאל פינקלשטיין באימייל: misha@tcb.ac.il . פניות בכל בדרך אחרת – לא יענו! בפנייה, יש לציין את : שם הקורס, שם הקמפוס 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)

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 הסעיפים של שאלה יש להשתמש בפונקציות מובנות (**map**, **filter**, **reduce**), פונקציות ללא שם, טיפוסים (**dict**, **list**, **tuple**) ומתודות של הטיפוסים אלו. בכל הסעיפים צריך לכתוב פונקציה שמקבלת פרמטרים וכוללת רק פעולת **return** בלבד.

א) כתוב פונקציה **get_prices** שמקבלת כפרמטרים:

- שם חנות.
- רשימת מוצרים ומחיריהם (**tuples** באורך 2).
- רשימת חנויות ומבצעים(**tuples** באורך 2).

ומחזירה רשימת מוצרים עם מחירים סופיים (אחרי הנחה) עבור החנות המצוינת.

דוגמת הרצה מחייבת:

```
>>> products = (('p1',1000),('p2',2000),('p3',5000),('p4',100))
>>> sales = (('s1',0.2),('s2',0.3),('s3',0.1))
>>> get_prices('s1', products, sales)
(('p1', 800.0), ('p2', 1600.0), ('p3', 4000.0), ('p4', 80.0))
```

ב) כתוב פונקציה **get_prices_dict** שמקבלת כפרמטרים:

- שם חנות.
- מילון של מוצרים ומחיריהם.
- מילון של חנויות ומבצעים.

ומחזירה מילון מוצרים עם מחירים סופיים (אחרי הנחה) עבור החנות המצוינת.

דוגמת הרצה מחייבת:

```
>>> prod_dict = dict(products)
>>> sale_dict = dict(sales)
>>> get_prices_dict('s1', prod_dict, sale_dict)
{'p1': 800.0, 'p2': 1600.0, 'p3': 4000.0, 'p4': 80.0}
```

ג) כתוב פונקציה **get_prices_by_type** שמקבלת כפרמטרים:

- שם חנות.
- מילון של מוצרים ומחיריהם.
- מילון מקונן של חנויות ומבצעים לפי סוגים של מוצרים.
- מילון של רשימות מוצרים לפי סוגים.

ומחזירה מילון מוצרים עם מחירים סופיים(אחרי הנחה) עבור החנות המצוינת.

דוגמת הרצה מחייבת:

```
>>> sales = {'s1':{'t1':0.2, 't2':0.1}, 's2':{'t1':0.1, 't2':0.2},'s3':{'t1':0.3, 't2':0.5}}
>>> types = {'t1':('p2', 'p4'), 't2':('p1', 'p3')}
>>> get_prices_by_type('s1', prod_dict, sales, types)
{'p1': 900.0, 'p2': 1600.0, 'p3': 4500.0, 'p4': 80.0}
```

ד) כתוב פונקציה **accumulate_prices** שמקבלת כפרמטרים:

- שם חנות.
- מילון של מוצרים ומחיריהם.
- מילון של חנויות ומבצעים לפי סוגים של מוצרים.
- מילון של רשימות מוצרים לפי סוגים ופונקציית אגירה.

ומחזירה תוצאת אגירת מחירים סופיים(אחרי הנחה) עבור כל המוצרים בחנות המצוינת.

דוגמת הרצה מחייבת:

```
>>> accumulate_prices('s1', prod_dict, sales, types, add)
7080.0
```

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

4) יש לממש טיפוס (mutable) בשם **coding** שמאפשר להצפין ולפענח את טקסט המורכב ממילים עם רווח אחד בין המילים, בשיטת **message passing** עם **dispatch function**. ההצפנה מתבצעת באמצעות שינוי סדר המילים, שינוי סדר אותיות במילים והחלפת תווים.
לצורך ההצפנה, המערכת מייצרת את מפתח הצפנה(מילון) שמכיל את כל האותיות ושני דגלים(דגל ראשון מסמן האם יש צורך בהיפוך סדר מילים ודגל שני מסמן האם יש צורך בהיפוך סדר אותיות בכל המילים).
לביצוע פעילות, המערכת משתמשת בכלית ושיטות: **dict**, **dispatch function** ו-**message passing**, ומתודות של **str**.
יש לממש את הפעולות הבאות:

א) יצירת מפתח הצפנה (**'set_key'**) – באמצעות קבלת מספר שלם, שקובע את הזזה מעגלית שמאלה או ימינה לפי סימן של מספר, עם קבלת 0 מתבצעת הזזה למספר אקראי.
ב) אפוס מפתח הצפנה (**'empty_key'**).
ג) יצוא מפתח הצפנה (**'export_key'**).
ד) יבוא מפתח הצפנה (**'import_key'**).
ה) הצפנה (**'encoding'**).
ו) פיענוח (**'decoding'**).

דוגמת הרצה מחייבת:

```
>>> code1=coding()
>>> code1('set_key',(-3,'yes','yes'))
'done'
>>> key=code1('export_key')
>>> key
{'reverse_word': True, 'reverse_string': True, 'a': 'x', 'b': 'y', 'c': 'z', 'd': 'a', 'e': 'b', 'f': 'c', 'g': 'd', 'h': 'e',
'i': 'f', 'j': 'g', 'k': 'h', 'l': 'i', 'm': 'j', 'n': 'k', 'o': 'l', 'p': 'm', 'q': 'n', 'r': 'o', 's': 'p', 't': 'q', 'u': 'r', 'v': 's', 'w': 't', 'x':
'u', 'y': 'v', 'z': 'w'}
>>> cstr=code1('encoding','the london is the capital of great britain')
>>> cstr
'kfxqfoy qxbod cl ixqfmxz beq pf klakli beq'
>>> dstr=code1('decoding',cstr)
>>> dstr
'the london is the capital of great britain'
>>> code2=coding()
>>> dstr=code2('decoding',cstr)
>>> dstr
'key empty'
>>> code2('import_key',key)
'done'
>>> dstr=code2('decoding',cstr)
>>> dstr
'the london is the capital of great britain'
>>> code2('empty_key')
'done'
>>> code2('export_key')
'key empty'
```

5) בשאלה זו אתם מתבקשים לממש טיפוס בשם **parking** – לשימוש בניהול חניונים. קיימים שלושה סוגי חניונים: **Regular** - תשלום **רגיל**, **Priority** - תשלום **כפול**, **VIP** – תשלום פי-**3** מהתשלום הרגיל. עליך לרשום פונקציה **parking** שמקבלת פרמטרים: תשלום עבור שעת חניה וכמות מקומות בשלושת סוגי החניונים. על הפונקציה ליצור רצף שתאחסן נתונים עבור כל הרכבים, שחונים בחניונים אלו. הנתונים הנשמרים עבור כל רכב מכילים מספר רכב, סוג חניון ומס' שעות חניה*, שמאוחסנים בעזרת רשימה בגודל **3**. כל הפעולות מתבצעות לפי שיטת **dispatch_dictionary** בעזרת טיפוסים מובנים כמו **list**, **tuple** או **dict**.

הפעולות המוגדרות על הטיפוס:

א) הדפסת פרטים של כל הרכבים הנמצאים בחניונים(**'print_list'**).

ב) הדפסת פרטים של כל הרכבים הנמצאים בחניון מסוים(**'print_parking'**).

ג) קידום שעות חניה* לרכבים בחניונים(**'next_time'**).

ד) כניסת רכב לחניה(**'start_parking'**).

ה) סיום חניה(**'end_parking'**).

דוגמת הרצה מחייבת:

```
>>> park1=parking(10,3,3,3)
>>> park1
{'print_list': <function parking.<locals>.print_list at 0x03FC01E0>, 'print_parking': <function parking.<locals>.print_parking at 0x03FC0228>, 'next_time': <function parking.<locals>.next_time at 0x03FC0270>, 'start_parking': <function parking.<locals>.start_parking at 0x03FC02B8>, 'end_parking': <function parking.<locals>.end_parking at 0x03FC0300>}
>>> park1['start_parking'](222,'Regular')
>>> park1['start_parking'](223,'Regular')
>>> park1['next_time']()
>>> park1['start_parking'](224,'Regular')
>>> park1['start_parking'](225,'Regular')
Regular parking is full
>>> park1['start_parking'](225,'VIP')
>>> prn=park1['print_list']()
>>> prn
{'end': <function parking.<locals>.print_list.<locals>.end at 0x03FC0348>, 'next': <function parking.<locals>.print_list.<locals>.next at 0x03FC0390>}
>>> while not prn['end']():
        prn['next']()
car: 222, parking type: Regular, parking time: 2
car: 223, parking type: Regular, parking time: 2
car: 224, parking type: Regular, parking time: 1
car: 225, parking type: VIP, parking time: 1
>>> park1['print_parking']('VIP')
car: 225, parking time: 1
>>> park1['end_parking'](100)
car not found
>>> park1['end_parking'](223)
car: 223, parking type: Regular, parking time: 2
payment: 20
>>> park1['print_parking']('Regular')
car: 222, parking time: 2
car: 224, parking time: 1
```

חלק ד: שאלות תיאורטיות

6) עבור **פונקציה ללא שם** סמנו אילו מהטענות נכונות, הסבירו בקצרה לכל טענה ותנו דוגמא:

א) לא יכולה לקבל פונקציה כפרמטר.

ב) יכולה לקבל פרמטרים רק מטיפוסים פרימיטיביים (**int**, **float**, **bool**).

ג) יכולה להכיל יותר משורת קוד אחת.

ד) לא יכלה להחזיר אתד משלושת ערכים שונים, לפי התנאי המוגדר ב-**if**.

ה) פונקציה אחת יכולה להחזיר אובייקטים מטיפוסים שונים לפי תנאי (**לדוגמא**: **True** או **3**).

ו) לא ניתן לשכתב **dispatch function** (למימוש טיפוס שלא ניתן לשנות, **immutable**) עם פונקציה ללא שם.

ז) רק פונקציה ללא שם ניתן להעביר כארגומנט לפונקציות מובנות כמו: **map**, **filter**, **reduce**.

7) סמנו אילו מהטענות נכונות והסבירו בקצרה כל טענה:

א) בפייטון ניתן להגדיר שני משתנים שמצביעים לאותו אובייקט.

ב) בפייטון ניתן להגדיר שני משתנים שמצביעים לאותה פונקציה.

ג) לפי שיטת **dispatch function** עם **message passing** פונקציה **dispatch** יכולה לקבל פונקציה כפרמטר.

ד) לפי שיטת **dispatch function** עם **message passing** למימוש טיפוסי נתונים, ניתן ליצור רק טיפוסים שלא ניתן לשנות אותם (**immutable**).

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

ו) שניתן להכניס לתוכו.

ז) רשימה רקורסיבית (**rlist**) שמימשתם בכיתה (כפונקציה **dispatch function**) היא רצף (**sequence**).

בהצלחה !