עקרונות שפות תוכנה

תשע"ז, סמסטר א'

<u>עבודת הגשה מס' 3</u>

הנחיות:

- יש להגיש את העבודה עד **12/01/2017**
- על כל יום איחור בהגשה, ללא הצדקה, ירדו 3% מהציון. לא ניתן להגיש <u>כלל</u> באיחור של מעל שבוע
- מותר להכין את העבודה בזוגות או בודדים בלבד.
- חובה להשתמש בשמות הפונקציות המוגדרות.
- על הקובץ להכיל רק את הפונקציות ללא הרצה.
- מימוש המשימות חייב להיות מתואם לדוגמאות הרצה.
- יש לתעד (docstrings) בתוך כל פונקציה. המטרה היא שאם משתמש מפעיל help(func) אז הוא יקבל את כל המידע הדרוש להבנת הפונקציה.
- ההגשה היא באתר moodle – רק סטודנט/ית אחד מגיש - צריכים לעלות קובץ עם אותו השם שהוא צירוף ת"ז שלהם עם קו תחתון וקידומת מספר עבודה. לדוגמה:

  HW2_123456789_123456789.zip
- יש להגיש קובץ PY עבור החלק התכנותי וקובץ PDF עבור החלק התאורטי מכווצים יחד בקובץ ZIP. לכל 3 קבצים חייב להיות אותו שם (לדוגמה: HW2_123456789_123456789.zip/py/pdf)

<u>חלק 1: Data abstraction, immutable data</u>

1. יש ליצור מבנה של תאריך הכולל פירוט של שנה, חודש ויום בחודש. יש לממש פונקציות עזר כדי להדפיס את הרכיבים המרכיבים תאריך שלם, דוגמאות מופיעות למטה. אין להשתמש בטיפוסים מובנים של Python (חוץ ממספרים ופונקציה)! האם יצא לכם טיפוס (תאריך) שהוא mutable או immutable? הסבר.

<u>דוגמת הרצה</u>:

```
>>> d = make_date(2016, 12, 26)
>>> d
<function make_date.<locals>.dispatch at 0x02A880C0>
>>>year (d)
2016
>>>month (d)
December
>>>day (d)
26
>>>str_date(d)
'26th of December, 2016'
```

חלק 2: Conventional Interface, pipeline

עיבוד מוקדם של נתונים (data preprocessing) הוא מרכיב חשוב מאוד של תהליך כריית מידע (data mining). הוא אחראי על ניקוי, נירמול והשלמת נתונים מנותחים. במשימה הזאת אתם מתבקשים לבנות מנוע לעיבוד מוקדם של נתונים שהתקבלו ממערכת קבצים (Filesystem) מרוחקת. יתכן שעקב תקלות ברשת בזמן הורדת הנתונים מתקבלות תוצאות שגויות, או שעקב תקלה בשמירת הנתונים לא תירשם תוצאה כלל. המנוע מקבל נתונים בצורה של מחרוזת המורכבת מרצף שמות של קבצים המופרדים ע״י נקודה-פסיק. כל קובץ מוצג לפי הנתיב משורש הדיסק שהוא שמור בו לפי שיטת UNIX. ניתן להניח שכל קובץ מסתיים עם נקודה וסוג הקובץ. בשונה מתיקייה שאין לה נקודה וסוג קובץ בסוף.

לדוגמא:

״/Users/someuser/file.py;/tmp/download/file.zip;/;/usr/local/bin;/User/someuser/file..py״.

2. יש לבנות מנוע לעיבוד מוקדם של נתונים תוך שימוש בממשק קונבנציונאלי. יש לממש את המנוע כפונקציה בשם data_preprocessing_*** ולבנות אותו כ- pipeline של שלבים הבאים (לפי סדר):
   a. **enumerate** – קריאה ומינוי ערכים. יש להחזיר רצף של ערכים כפלט של השלב הזה.
   b. **clean (noise removal)** – הסרת ערכים שגויים. ניתן לזהות ערכים כאלו כשיש שתי נקודות (..) או סלאש (//) ברצף.
   c. **complete missing values** – השלמת ערכים חסרים. יש להחליף קובץ שמסתיים בנקודה עם הערך ׳<שם הקובץ>.txt׳ (כלומר, כל קובץ ללא סיומת ייקלט כקובץ טקסט לפי ברירת מחדל)
   d. **accumulate** – יש לממש כמה סוגים של חישובים סטטיסטיים:
      i. **build tree** – לבנות עץ עבור הנתונים המנורמלים. מטרת העץ להראות היררכיה של הקבצים ושל התיקיות שהתקבלו. אתם צריכים לחשב ולהחזיר את הקבצים שהתקבלו כאוסף של מחרוזות ממוינים לפי שמות התיקיות והקבצים. כל תיקייה תופיע פעם אחת ברשימה כעבר ראשון בtuple ולאחריה תכולת כל הקבצים השמורים בו כtuple מקונן. יש לקרוא לפונקצית מנוע שבונה את העץ בשם data_preprocessing_tree.
      ii. **get file types** – לחשב כמות של סוגי הקבצים. ערך ההחזרה של הפונקציה הוא אוסף של (tuple) שבכל זוג יופיע סוג הקובץ ומספר קבצים בהיררכיה מאותו סוג, אין צורך להתייחס למספר התיקיות. יש לקרוא לפונקצית מנוע שמחשבת את הסכום הנ״ל data_preprocessing_file_types

<u>הערה</u>: אין להשתמש בלולאות ופונקציות עזר, אלא רק בפונקציות של ממשק קונבנציונאלי ו-lambda. מותר להשתמש בפעולות השמה ע"מ לשמור תוצאות ביניים.

<u>הערה</u>: יש להניח לגבי קלט שערכים חסרים לא יכולים להופיע בהתחלה ו/או בסוף של המחרוזת, ולא יתכן שיופיעו 2 או יותר ערכים חסרים ברצף.

<u>דוגמת הרצה</u>:

```
>>> data = "/Users/someuser/file.py;/tmp/download/file.zip;/tmp/download/file2.zip;/;/usr/local/bin;/User/someuser/file..py;/tmp/file."
>>> data_preprocessing_tree(data)
[("/", ()), ("/tmp/download", ("file.zip", "file2.zip", “unknown.txt”)), ("/usr/local/bin", ()), ("/User/someuser/", ("file.py", “uknown.txt”))]
>>> data_preprocessing_file_types(data)
[("py", 1), (“txt”, 2), ("zip", 2)]
```

חלק 3: Mutable data, message passing, dispatch function, dispatch dictionary

3. יש לממש טיפוס נתונים חדש בשם **currency** שמייצג שער יציג ע״י סכום וסימון של אותו מטבע תוך שימוש ב-dispatch function ו-message passing. יש למשש את הפעולות הבאות:
   a. גישה לערך הסכום וסימון של המטבע בהתאם להודעה: 'get_value'
   b. עידכון של ערך הסכום וסימון המטבע המתאים: 'set_value'
   c. ייצוג טקסטואלי. יש לייצג את הסכום ביחד עם הסימון דרך ההודעה המתאימה - 'str'
   d. החלפת מטבע תבוצע ע״י שליחה של סימון חדש ופונקציית lambda שעל ידו תמירו את ערך המטבע הקודמת.

<u>הערה</u>: אין להשתמש בטיפוסים מובנים של Python!!!

<u>דוגמת הרצה:</u>

```
>>> c = make_currency(10.50, '$')
>>> c('get_value')('amount')
10.50
>>> c('get_value')('symbol')
'$'
>>> c('set_value')('amount', 50)
>>> c('get_value')('amount')
50
>>> c('str')
'$50.00'
>>> c('convert')(lambda x: x*3.87, '₪')
>>> c('str')
'₪193.50'
```

4. בשאלה זו אתם מתבקשים לממש טיפוס נתונים חדש בשם **reverse_map_iterator** . ניתן ליצור אובייקט של **reverse_map_iterator** על רצף **s** ופונקציה **g** (של ארגומנט אחד) על מנת לעבור על ערכים של הרצף החדש שמתקבל על ידי הפעלת - **g** על ערכים של **s**.

יש לממש פונקציה **get_reverse_map_iterator** היוצרת אוביקט של **reverse_map_iterator** לפי שיטת dispatch dictionary . שתי פעולות מוגדרות על הטיפוס:

a. **next** - פעולה מחזירה את האלמנט הבא של הרצף המתקבל

b. **has_more** - פעולה מחזירה – TRUE כל עוד יש עוד אלמנטים ברצף

<u>הערה</u>: במידה ופונקציה לא קיבלה את הארגומנט השני (g) אז היא תחזיר איטראטור על ערכים של הרצף הארגומנט (s) ללא שינוי.

<u>הערה</u>: אין לטפל בחריגות ש Python מעלה במקרים של חישובים כושלים כגון חלוקה ב-0 וכד'. (תטפלו בחריגות בעבודה הבאה).

<u>דוגמת הרצה:</u>

```
>>> it = get_reverse_map_iterator((1,3,6), lambda x: 1/x)
>>> while it['has_more']():
        it['next']()
0.16666666666666666
0.3333333333333333
1.0
>>> it = get_reverse_map_iterator((1,3,6))
>>> for i in range(1,6):
        it['next']()
6
3
1
no more items
no more items
```

5. שאלה זאת מתייחסת למימוש של רשימה שנעשה בשיעור (make_mutable_rlist) בשיטת message passing ו-dispatch dictionary. יש להשלים\לעדכן את המימוש ע"י שינויים הבאים:

a. פעולה לבנית ייצוג טקסטואלי (**str**) אמורה להחזיר מחרוזת המייצגת רשימה בדומה לייצוג של רשימה ב-Python (למשל, רשימה שמכילה 1, 2 ו-3 תוצג כ-"[1, 2, 3]").
b. יש להוסיף פעולה חדשה בשם **slice** שמחזירה תת-רשימה (בדומה לאופרטור slicing). הפעולה תקבל 2 ארגומנטים: אינדקס התחלה (כולל) ואינקס אחרון (לא כולל) ותחזיר אוביקט חדש של mutable_list
c. יש להוסיף פעולה חדשה בשם **extend** שמאפשרת לקבל אוביקט של mutable_list עם כמה אלמנטים ולהכניס אותם לסוף הרשימה לפי סדרם (בדומה ל-extend של list).
d. לאפשר הפעלת בנאי מעתיק (copy constructor), כלומר להעתיק איברים מרשימת-ארגומנט לפונקציה make_mutable_list לרשימה החדשה.
e. פעולה בשם **get_iterator** שמחזירה iterator על תוכן הרשימה.

<u>דוגמת הרצה:</u>

```
>>> my_list = make_mutable_rlist()
>>> for x in range(4)
            my_list['push_first'](x)
>>> my_list['str']()
[3, 2, 1, 0]
>>> ext = make_mutable_rlist(my_list)
>>> my_list['extend'](ext)
>>> my_list['str']()
[3, 2, 1, 0, 3, 2, 1, 0]
>>> my_list['slice'](0,2)['str']()
[3, 5]
>>> your_list = make_mutable_rlist(my_list)
>>> your_list['str']()
[3, 2, 1, 0, 3, 2, 1, 0]
>>> it = my_list['get_iterator']()
>>> while it['hasNext']():
          print(it['next']())
3
2
1
0
3
2
1
0
```

חלק 4: שאלות תאורטיות

6. סמנו אילו מהטענות נכונות והסבירו בקצרה לכל טענה:
   a. פונקציה מסדר גבוה (high-order function) ניתן להחזיר מפונקציה ולהעביר לפונקציה כארגומנט.
   b. ב Python 3 -משתמשים בהצהרה nonlocal על מנת לעדכן קשירה של משתנה במסגרת גלובאלית.
   c. בשפות עם Lexical Scoping ניתן לממש טיפוסי נתונים חדשים שהם mutable תוך שימוש בפונקציות והשמה לא לוקאלית (nonlocal).

בהצלחה!

כל שאלה ופניה בנוגע לתרגיל יש להפנות אך ורק לאחראי על התרגיל **דוב סוכצ'סקי**
באימייל: b@kloud.email
פניות בכל בדרך אחרת לא יענו!