המכללה האקדמית להנדסה סמי שמעון

משימות למעבדה מס' 12

(Recursive data structures, memoization and exceptions)

1. כתבו פונקציה רקורסיבית בשם **Merge** המקבלת שתי רשימות ממוינות בסדר עולה ומחזירה מיזוג של שתי הרשימות לרשימה אחת חדשה, שגם היא תמוין בסדר עולה.

דוגמאות להרצה:

lst1= [1,3,5,7,9]

lst2= [2,3,4,6,8,10]

print (**Merge** (lst1, lst2)) # => [1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10]

2. יש ליישם את האלגוריתם **Mergesort**, שהוא מבין אלגוריתמי המיון המהירים והשכיחים:

א) אם קיים איבר אחד (או אפס איברים) ברשימה אזי היא כבר ממוינת.

ב) במידה וקיימים יותר מ-1 איברים ברשימה, אז נחלק את הרשימה שבידנו לשניים, נמיין כל חלק בצורה רקורסיבית ואז נמזג את התוצאות (באמצעות הפונקציה משאלה 1). התוצאה תהינה רשימה ממויינת.

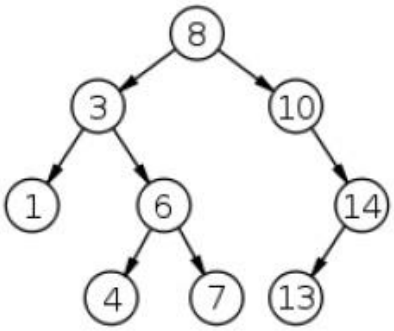
יש לכתוב פונקציה רקורסיבית על-ידי שימוש באלגוריתם שתואר לעיל

דוגמאות להרצה:

lst=[1,19,3,10,5,3,9]

print (**Mergesort** (lst)) => [1, 3, 3, 5, 9, 10, 19]

3. בנה עץ בצורה הבאה(אפשר להשתמש בפונקציות משלך):



א) בנה פונקציה **print_tree** אשר מדפיסה את כל העץ בצורה רקורסיבית בצורת **Inorder**.

ב) בנה פונקציה אשר מקבלת **node** הראשי של העץ ומבצעת **Mirror reflection**.

דוגמאות להרצה:

```
root = Node(8)
root.insert(3)
root.insert(10)
root.insert(1)
root.insert(6)
root.insert(4)
root.insert(7)
root.insert(14)
root.insert(13)
root.print_tree()
output:
1
3
4
6
7
8
10
13
14
root.mirror()
root.print_tree()
output:
14
13
10
8
7
6
4
3
1
```

4. גנב קמצן בא לשוד עם תרמיל שיכול להכיל **n** קילוגרם. הוא לא יכול לקחת יותר ממה שהתרמיל שלו יכול להכיל, אך גם לא מסכים לקחת פחות מהמקסימום. השלם את הפונקציה **isAnyOptionToSteal** כפונקציה רקורסיבית שבינתן **n** ורשימת משקלים של דברים (יש להניח שיש אינסוף דברים מכל משקל) מחשבת האם יש איזושהי קומבינציה של דברים לגניבה "המושלמת", כך שהמשקל המשותף שלהם שווה ל-**n** ומחזיר ערך הבוליאני בהתאם:

```
def isAnyOptionToSteal(n, weights):
```

דוגמאות להרצה:

```
print(isAnyOptionToSteal(10, (4,3))) => True
print(isAnyOptionToSteal(10, (4,))) => False
print(isAnyOptionToSteal(10, (4,3,2))) => True
```

5. **memoization** - טכניקה **לשיפור זמן ריצה** של פונקציה רקורסיבית תוך **שמירת ערכים** עבור קלט שהתקבל קודם.

להלן מימוש של ה-**memo**.

```
def memo(function):
    cache={}
    def memoized(*args):
        if args not in cache:
            cache[args]=function(*args)
        return cache[args]
    return memoized
```

בהינתן פונקציה הרקורסיבית הבאה:

```
def f(n):
    if n<3:
        return n
    else:
        return f(n-1)+f(n-2)+f(n-3)
```

יש להריץ אותה בלי **memoization**:

```
print(f(35))
```

ולאחר מכן עם **memoization**:

יש לשקול כיצד להריץ

מה תוכלו להגיד לגבי זמני הריצה בשני המקרים?

חריגות (Exceptions)

פייתון תומכת בטיפול חריגות. במקרה שתוכנית נתקלת בסיטואציה חריגה, על פי רוב שגיאה או נתונים עבורם תוצאת החישוב הרצויה איננה מוגדרת, ניתן "להרים" (או "לזרוק") חריגה באמצעות המילה השמורה **raise** (מקביל ל-**throw** בשפות אחרות). החריגה הנזרקת היא אובייקט, שיילכד במעלה הקריאה לפונקציה שזרקה אותו, בבלוק ייעודי מהצורה **try ... except**, שם יטופל באופן ספציפי. אם חריגה לא נלכדת על ידי המתכנת, היא נלכדת על ידי המפרש .

בפייתון כל שגיאה היא חריגה, וכל שגיאה ניתן ללכוד - חלוקה באפס, שגיאות קלט-פלט, שמות שטרם הוגדרו בקוד, וכו' - למעט שגיאות תחביר.

6. כתוב פונקציה אשר מקבלת מספר רציונלי – מספר ראשון מונה ומספר שני כמכנה ובעזרת TUPLE ומחזירה את שבר העשרוני שלו (הדוגמא מהרצאה 4). אליך לתפוס חריגות הבאות ולהחזיר הודעות מתאימות למשתמש:

א) שהמשתמש לא הכניס מספרים.

ב) חילוק באפס.

```
from operator import getitem

def make_rat(n, d):
        return (n, d)
def numer(x):
        return getitem(x, 0)
def denom(x):
        return getitem(x, 1)
def make(x):
        .........
        return result
def str_rat(x):
        """Return a string n/d for numerator n and denominator d."""
        return '{0}/{1}={2}'.format(numer(x), denom(x), make(x))
def main():
        n=input("Enter numer number:")
        d=input("Enter denom number:")
        ns=0.0
        .........
        print( str_rat(ns))

main()
```

7. שאלה:

א) כתוב פונקציה **avg_grade** המחשבת ציון ממוצע עבור קורס (הציונים נמצאים בתוך קובץ טקסט). במידה והציון לא תקין (שלילי או לא מספרי) יש לעלות חריגה וב-**FINALLY** לסגור את הקובץ.

ב) כתוב פונקציה המחשבת ציון ממוצע עבור כל קורס בסמסטר וממוצע כללי עבור כל הקורסים (בלולאה יש להריץ **avg_grade** ב-**TRY**).

בהצלחה !