מטלת בית מס׳ 2

הנחיות:

- הגשה:
- יש להגיש את המטלה עד לתאריך 10.5.18.
- ההגשה תתבצע ע״י קובץ zip המכיל את הקוד שנכתב בהתאם לדרישות. פורמט קובץ ה-zip יהיה ID01_ID02.zip. שם קובץ ההרצה יהיה Ex2.py.
- ההגשה הינה **בזוגות**.
- חבר קבוצה אחד בלבד יעלה את הפתרון לאתר.
- בעיות אישיות בנוגע למועד ההגשה יש להפנות לבודק התרגילים הקורס טרם מועד ההגשה.
- **כל חריגה מנהלים אלו, ללא אישור בכתב מצוות הקורס, מהווה עילה לפסילת המטלה או להפחתת נקודות.**
- **אין להעתיק פתרונות ואין לשתף קוד בין סטודנטים. אין להעתיק קוד מוכן באינטרנט!**
- **לפתרון המטלה יש להשתמש בגרסאת פייטון 2.7 בלבד.**
- להבהרות, הכוונות או כל עזרה אחרת, ניתן לשאול שאלות בפורום המתאים למטלה זו באתר הקורס.
- בדיקת המטלה תתייחס בין השאר לפרמטרים הבאים: נכונות הקוד, יעילות הקוד וזמני ריצה. יש לבדוק מקרי קצה.

שאלה 1 – מימוש באמצעות OOP (36 נק')

עליכם לממש מחלקה בשם ComplexNum אשר מתארת מספר מרוכב. מספר מרוכב מיוצג ע"י שני מספרים, a ו-b, אשר מייצגים רכיב ממשי re ורכיב מדומה im במספר מרוכב: a + bi. שני רכיבים אלו ישמרו כמשתנים פנימיים של המחלקה ComplexNum. על המחלקה להיות immutable, כלומר, לאחר יצירת מופע חדש של ComplexNum, המתודות של המחלקה לעולם לא ישנו את המשתנים הפנימיים שלו, אלא יחזירו מופע חדש של המחלקה (בדומה ל-Rational מההרצאה).

במתודות שמקבלות מרוכב נוסף כמשתנה (למשל, מתודת __eq__) יש צורך לבדוק את הטיפוס של המשתנה הנוסף ולא ניתן להניח כי הוא גם מסוג מרוכב.

את כל אחת מהמתודות יש לכתוב בצורה היעילה ביותר.

1. ממשו את הבנאי __init__. הבנאי יקבל את שני הרכיבים וישמור אותם במשתנים פנימיים.
דוגמא:
```
>>> z = ComplexNum(1, 2)
```

2. ממשו את המתודות re ו- im. המתודות לא מקבלות משתני קלט מעבר ל-self ומחזירות את הרכיב הממשי (re) והמדומה (im), בהתאמה.
3. ממשו את המתודה to_tuple. המתודה אינה מקבלת משתני קלט מעבר ל-self ומחזירה tuple באורך 2 ובו הרכיב הממשי והרכיב המרוכב, לפי הסדר הזה.
4. ממשו את המתודה __repr__ אשר מייצרת מחרוזת לפי הסימון המתמטי של מספר מרוכב "a + bi" משני הרכיבים. שימו לב במיוחד למקרה בו הרכיב המדומה שלילי ולרווחים בין המספרים לסימן ה '+'.
* הערה: הבדיקה של __repr__ היא ע"י הפונקציה str() כי המתודה __str__ הדיפולטית של כל אובייקט קוראת פשוט ל- __repr__.
דוגמא:
```
>>> str(z)
1 + 2i
```

5. ממשו את המתודה __eq__: מתודה זו בודקת האם שני מרוכבים שווים. המתודה מקבלת שני מרוכבים (self, other) ומחזירה True אם ורק אם שני המרוכבים שווים. אחרת המתודה מחזירה False. יש לבצע בדיקת תקינות טיפוסי המשתנים.
דוגמא:
```
>>> z == ComplexNum(1,1)
False
>>> z == ComplexNum(1,2)
True
```
6. ממשו את אופרטור החיבור ע"י מימוש המתודה __add__: המתודה מקבלת שני מרוכבים (self, other) ומחזירה מרוכב חדש שהוא סכום המרוכבים מהקלט. שימו לב שחיבור מרוכבים הוא לפי רכיבים – למשל, הרכיב הממשי של המרוכב החדש הוא סכום הרכיבים הממשיים של מרוכבי הקלט.
דוגמא:
```
>>> str(z)
1 + 2i
>>> str(z + ComplexNum(1,-3))
2 - 1i
```
7. ממשו את אופרטור החיסור ע"י מימוש המתודות __neg__ ו-__sub__: מכיוון שחיסור הוא חיבור בשלילי, ממשו תחילה את המתודה __neg__ שמחזירה את השלילי למרוכב זה, כלומר, המרוכב שהסכום איתו ייתן 0. לאחר מכן ממשו את המתודה __sub__ בעזרת המתודות __neg__ ו-__add__.
דוגמא:

```
>>> str(z.neg())
-1 - 2i
>>> str(z – ComplexNum(4,3))
-3 - 1i
```

8. ממשו את המתודה __mul__ שמקבלת מרוכב נוסף כמשתנה קלט ומחשבת את המכפלה. חישוב המכפלה של שני מרוכבים מחזיר מרוכב לפי הנוסחא הבאה:
(x + yi)(u + vi) = (xu – yv) + (xv + yu)i.
אם ערך הקלט אינו מרוכב, עליכם להרים שגיאה (raise exception) מסוג TypeError עם הודעת שגיאה:
Complex multiplication only defined for Complex Numbers.
דוגמא:

```
>>> str(z * z)
-3 + 4i
>>> str(z * ComplexNum(2, 3))
-4 + 7i
>>> z * 2
...
TypeError: Complex multiplication only defined for Complex Numbers
```

9. ממשו את המתודה conjugate שמחשבת את הצמוד המרוכב. תזכורת:
z* = (a + bi)* = a - bi
דוגמא:

```
>>> z.conjugate()
1 - 2i
```

10. ממשו את המתודה abs אשר מחשבת את גודל המספר המרוכב: ממשו את המתודה בעזרת המתודה conjugate והפונקציה sqrt מהמודול math. המתודה תחזיר את השורש של המכפלה של המרוכב עם הצמוד שלו (לפי הנוסחא של $|z| = \sqrt{z \cdot z*}$ ולא תקבל משתני קלט מעבר ל-self.
דוגמא:

```
>>> z.abs()
2.2361
```

שאלה 2 – הורשה (32 נק')

בשאלה זו ניתן להשתמש בפונקציות type, is בלבד המוגדרות בפייטון.

א. עליכם לממש את פונק' isInstancePPL(object1, classInfo). הפונק' תקבל אובייקט object1 ותחזיר true אם האובייקט יורש מהמחלקה שהיא classInfo (אין הגבלה על רמת ההיררכיה). שימוש לב ש-object1 הינו טיפוס מסוג **אובייקט** ו-classInfo הינו טיפוס מסוג **מחלקה**. יש צורך לבדוק טיפוסי המשתנים.

ב. עליכם לממש את פונק' numInstancePPL(object1, classInfo). הפונק' תקבל טיפוס מסוג אובייקט object1 ותחזיר את **רמת ההיררכיה** (מספר האבות שהם מסוג המחלקה בהיררכיה) שהטיפוס object1 יורש מהמחלקה שהאב שלה הוא classInfo. שימוש לב ש-object1 הינו טיפוס מסוג **אובייקט** ו-classInfo הינו טיפוס מסוג **מחלקה**. במידה ואין הורשה אפשרית בין האובייקט לאחד האחד שהוא מסוג classInfo, יוחזר 0. במידה ואובייקט הוא מופע של המחלקה באופן ישיר, יוחזר 1.

ג. עליכם לממש את פונק' isSubclassPPL(class, classInfo). הפונק' תקבל טיפוס מסוג מחלקה class ותחזיר true אם הטיפוס class יורש מהמחלקה שהיא classInfo. שימוש לב ש-class הינו טיפוס מסוג **מחלקה** ו-classInfo הינו טיפוס מסוג **מחלקה**. יש צורך לבדוק טיפוסי המשתנים.

ד. עליכם לממש את פונק' numSubclassPPL(class, classInfo). הפונק' תקבל טיפוס מסוג מחלקה class ותחזיר את **רמת ההיררכיה** (מספר האבות בהיררכיה) שהטיפוס class יורש מהמחלקה שהיא classInfo. שימוש לב ש-class הינו טיפוס מסוג מחלקה ו-classInfo הינו טיפוס מסוג מחלקה. במידה ואין הורשה בין המחלקות, יוחזר 0. במידה וקיימת הורשה עצמית, יוחזר 1.

בהינתן כי קיימת מחלקה X (class X) ומחלקה Y שיורשת מ-X (class Y(X)), להלן דוגמאות הרצה:

(יתכן כי ישנה מחלקה Z שיורשת ממחלקה Y וכו'...)

```
>>> x = X()
>>> y = Y()
>>> isInstancePPL(x, X)
True
>>> isInstancePPL(x, Y)
False
>>> isInstancePPL(y, X)
True
>>> isInstancePPL(y, Y)
True

>>> isSubclassPPL(X, X)
True
>>> isSubclassPPL(X, Y)
False
>>> isSubclassPPL(Y, X)
True
>>> numSubclassPPL(Y, X)
2
>>> isSubclassPPL(Y, Y)
True
>>> numSubclassPPL(Y, Y)
```

```
1

>>> isSubclassPPL(type(x), X)
True
>>> isSubclassPPL(type(x), Y)
False
>>> isSubclassPPL(type(y), X)
True
>>> isSubclassPPL(type(y), Y)
True

>>> isSubclassPPL(x.__class__, X)
True
>>> isSubclassPPL(x.__class__, Y)
False
>>> isSubclassPPL(y.__class__, X)
True
>>> isSubclassPPL(y.__class__, Y)
True
```

שאלה 3 – High order Functions (32 נק')

א. עליכם לממש באופן היעיל ביותר את הפונק' count_if (הפועלת בדומה לאקסל) המקבלת רשימה lst ופונק' func ותחזיר את מספר הפעמים שכל איבר ברשימה קיים את הפונק' func (החזרת כמות הפעמים שהוחזר הערך true בהפעלת הפונקציה הבוליאנית על כל איבר). יש לבצע בדיקות מתאימות.

דוגמא:

```
>>> count_if([1,0,8], lambda x: x>2)
1
>>> count_if([1,1,8], lambda x: x=1)
2
```

ב. עליכם לממש באופן היעיל ביותר את הפונק' for_all המקבלת רשימה lst, פונק' apply ופונק' pred. הפונק' תחזיר ערך בוליאני, האם לאחר הפעלת הפונק' apply על כל איבר **בנפרד** ברשימה יתקיים התנאי pred. יש לבצע בדיקות מתאימות.
דוגמא:

```
>>> for_all([1,0,8], lambda x: x*2, lambda x: x>0)
False
>>> for_all([1,1,8], lambda x: x, lambda x: x>0)
True
```

ג. עליכם לממש באופן היעיל ביותר את הפונק' for_all_red המקבלת רשימה lst, פונק' apply ופונק' pred. הפונק' תחזיר ערך בוליאני, האם לאחר הפעלת הפונק' apply על כל האיברים **ביחד** ברשימה יתקיים התנאי pred. יש לבצע בדיקות מתאימות.
דוגמא:

```
>>> for_all_red ([1,0,8], lambda x, y: x*y, lambda x: x>0)
False
```

```
>>> for_all_red ([1,1,8], lambda x, y: x*y, lambda x: x>7)
True
```

ד. עליכם לממש באופן היעיל ביותר את הפונק' there_exists המקבלת רשימה lst, מס' n ופונק' pred. הפונק' תחזיר ערך בוליאני האם קיימים לפחות n איברים ברשימה המקיימים את התנאי pred. יש לבצע בדיקות מתאימות.

בהצלחה! ☺