תרגיל 8: טיפוסי נתונים מופשטים (**ADT**)

תאריך פרסום: חלק א':27.12.21

תאריך פרסום חלק ב': 1.1.22

התרגיל יפורסם בשני חלקים בכדי להדגיש את החשיבות של Abstraction ו-Encapsulation - ההפרדה בין ממשק לבין המימוש שלו. בחלק הראשון תתבקשו לממש מספק ממשקים של טיפוסי נתונים מופשטים שבהם תשתמשו בחלק השני שיפורסם מאוחר יותר.

תאריך הגשה (שני החלקים ביחד): 13.1.22 בשעה 23:59

מתרגל אחראי: אוריאל פרידמן

משקל תרגיל (שני החלקים ביחד): 4 נקודות

הנחיות כלליות

קראו בעיון את ההנחיות והעבודה. לא יתקבלו ערעורים על טעויות שחרגו מההנחיות.

1. העבודה תבוצע ותוגש ביחידים.
2. מומלץ לקרוא את העבודה כולה לפני שאתם ניגשים לפתרון.
3. עליכם להוריד את קבצי הקוד שמסופק עם התרגיל ולהשלים את הקוד החסר. יש לממש את הפתרון לתרגיל אך ורק באזורים שהוגדרו לשם כך בקובץ!
4. <u>כתיבת קוד קריא</u>:
   4.1. השתמשו בשמות משתנים משמעותיים. שימוש בשמות לא משמעותיים עשוי לגרור לפגיעה בציון.
   4.2. כתבו תיעוד (הערות) שמסביר את הקוד שלכם. יש לכתוב תיעוד docstring בכל פונקציה כפי שנלמד בכיתה.
   4.3. אין לכתוב הערות בעברית! עבודה שתכיל טקסט בשפה שאינה אנגלית (או פייתון) תקבל <u>ציון אפס</u> ללא אפשרות ערעור.
5. אין להשתמש בחבילות או במודולים חיצוניים <u>מלבד</u> מה שהוגדר בתרגיל! אם יש ספק ניתן לשאול בפורום המתאים (ראו סעיף 10).
6. יש לכתוב קוד אך ורק באזורים שהוגדרו לשם כך!
7. <u>הנחות על הקלט</u>:
   7.1. בכל שאלה יוגדר מה הקלט שהקוד מקבל וניתן להניח כי הקלט שנבדוק מקיים את התנאים הללו. אין להניח הנחות נוספות על הקלט מעבר למה שהוגדר.

7.2. בכל שאלה סיפקנו עבורכם דוגמאות לקלט והפלט הרצוי עבורו. עליכם לערוך בדיקות נוספות לקוד שמימשתם ולא להסתמך על דוגמאות אלו בלבד.

8. בדיקת העבודה:

8.1. העבודה תיבדק באופן אוטומטי ולכן על הפלטים להיות זהים לפלטים שמוגדר בתרגיל.

8.2. טרם ההגשה יש לעבור על המסמך assignments_checklist שנמצא במודל.

8.3. מערכת הבדיקות קוראת לפונקציות שהוגדרו בתרגיל בצורה אוטומטית. אין לשנות את חתימות הפונקציות. חריגה מההנחיות תגרור ציון אפס.

9. העתקות:

9.1. אל תעתיקו!

9.2. העתקת קוד (משנים קודמות, מחברים או מהאינטרנט) אסורה בהחלט. בפרט אין להעביר קוד בין סטודנטים. צוות הקורס ישתמש בכלים אוטומטיים וידניים כדי לזהות העתקות. תלמיד שייתפס בהעתקה יועמד בפני ועדת משמעת (העונש המינימלי לפי תקנון האוניברסיטה הוא כישלון בקורס).

9.3. אנא קראו בעיון את המסמך שהכנו בנושא: https://moodle2.bgu.ac.il/moodle/mod/resource/view.php?id=1922556

10. שאלות על העבודה:

10.1. שאלות בנוגע לעבודה ישאלו בפורום שאלות לתרגיל במודל או בשעות הקבלה של המתרגלת האחראית בלבד .

10.2. אין לפנות במייל לבודקת התרגילים או למתרגלים אחרים בנוגע לעבודות הגשה. מיילים בנושאים אלו לא יקבלו מענה.

10.3. לפני ששואלים שאלה בפורום יש לוודא שהשאלה לא נשאלה קודם!

10.4. אנו מעודדים סטודנטים לענות על שאלות של סטודנטים אחרים. המתרגלת האחראית תאשר שתשובה כזו נכונה.

11. הגשת העבודה:

11.1. עליכם להוריד את הקבצים מתיקיית "תרגיל בית 8" מהמודל. התיקייה תכיל תיקייה נוספת ובה קבצי העבודה וקובץ הוראות. עליכם למלא את הפתרון במקום המתאים ובהתאם להוראות התרגיל.

11.2. שימו לב: בנוסף לקבצי העבודה מצורף קובץ בשם get_id.py. עליכם למלא במקום המתאים בקובץ את תעודת הזהות שלכם. הגשה שלא תכיל את הקובץ הנ"ל עם תעודת הזהות הנכונה לא תיבדק ותקבל ציון אפס!

11.3. את העבודה יש להגיש באמצעות תיבת ההגשה הייעודית במודל.

11.4. פורמט ההגשה הוא להלן:

11.4.1. את התיקייה בשם hw8 שהורדתם מתיקיית העבודה במודל, ובה נמצאים קבצים הקוד שלכם, יש לכווץ לפורמט zip (קבצים אחרים, כגון קבצי rar, יקבלו ציון אפס).

11.4.2. השם של התיקייה המכווצת יהיה תעודת הזהות שלכם.

11.4.3. העלו את התיקייה המכווצת לתיבת ההגשה של העבודה.

דגשים

1. נא לקרוא את **כל העבודה לפני** שאתם מתחילים לכתוב את הקוד.
2. בכל העבודה אין ליצור שיטות שלא הונחיתם ליצור במפורש בקוד.
3. בכל העבודה אין ליצור כלל שדות (לא פרטיים, לא ציבוריים וגם לא סטטיים) שלא נאמר לכם במפורש ליצור.
4. בכל העבודה אין להגדיר שדות מסוג שונה ממה שנאמר לכם במפורש להגדיר.
5. בכל מקום בו אתם משתמשים במבנה נתונים קיים עליכם להניח ש**אינכם** יודעים כיצד מבנה הנתונים ממומש ועליכם להשתמש רק ב - API של אותו מבנה בכדי לבצע את הפונקציונליות הנדרשת מכם.
6. העבודה כתובה בסדר בו עליכם לכתוב את הקוד, מומלץ בחום לכתוב את הקוד לפי סדר הסעיפים בעבודה.
7. בכל העבודה נאסר עליכם להשתמש במבני הנתונים הקיימים בשפת פייתון (גם כאלו שלא למדנו) כגון: list, dict, set וכו', ניתן להשתמש במתודות של str בשיטות שבהן נדרשת החזרת מחרוזת או שמקבלים מחרוזת כפרמטר.
8. בכל סעיף בעבודה ניתן להעזר במבני נתונים שמימשתם בסעיפים קודמים בעבודה ורק אם נכתב במפורש שהדבר מותר.
1. נא לקרוא את **כל העבודה לפני** שאתם מתחילים לכתוב את הקוד. הסעיף הזה נכתב שוב וזה לא בטעות.

הקדמה

תזכורת – מכיוון שהמחשב הינו מכשיר אלקטרוני - פיזיקלית הוא יכול להכיל רק שני מצבים שונים: יש מתח (נהוג לסמן כ 1) או אין מתח (נהוג לסמן כ 0). יחידת זיכרון אטומית במחשב נקראת ביט (bit) או סיבית בעברית. סיבית הינה יחידה המייצגת מספר קטן של אפשרויות ולכן הזיכרון בפועל מחולק לבייטים (Bytes) – או בתים בעברית - שמכילים כל אחד 8 סיביות. בכיתה למדנו על ייצוג מספרים שלמים וחיוביים בבסיס בינארי (ראו נושא 5# בקורס).

עבודה זו תעסוק במבני נתונים ותתבסס על ייצוג מספרים חיוביים ושלמים בבסיס בינארי.

חלק א'

חוליית בית (ByteNode)

המחלקה חוליית בית (ByteNode) היא מחלקה mutable ומוגדרת על ידי השדות הבאים:

- **byte** – משתנה מסוג מחרוזת באורך שמונה תווים, כל תו הוא או '0' או '1'.
- **next** – מצביע על ByteNode אחר או על None.

ממשו את המחלקה ByteNode:

```
def __init__(self, byte):
```

בנאי המאתחל את שדה ה-byte בארגומנט byte ואת שדה ה-next ל-None.

קלט:

- **byte [Object]**, עבור קלט מסוג לא תקין יש לזרוק חריגה מסוג TypeError עם הודעה אינפורמטיבית כרצונכם, עבור קלט עם ערכים לא תקינים (סוג תקין לא באורך המתאים או שעם תווים שאינם '0' או '1') יש לזרוק חריגה מסוג ValueError.

לדוגמה, בהרצת שורת הקוד הבאה:

```
bn = ByteNode(10000000)
```

תיזרק חריגה מסוג TypeError בעוד שעבור הרצת שורות הקוד הבאות (כל אחת בנפרד):

```
bn = ByteNode('100000001')
bn = ByteNode('10020001')
```

תיזרק חריגה מסוג ValueError.

```
def get_byte(self):
```

השיטה תחזיר את השדה byte.

פלט:

- **byte [str]**.

לדוגמה, עבור הרצת הקוד הבא:

```
bn = ByteNode('10011000')
print(bn.get_byte())
print(type(bn.get_byte()))
```

יודפס:

```
10011000
<class 'str'>
```

```
def get_next(self):
```

השיטה תחזיר את ערך השדה next.

<u>פלט:</u>

- **[ByteNode / None] `next`**.

לדוגמה, עבור הרצת הקוד הבא:

```
bn = ByteNode('10011000')
print(bn.get_next())
```

יודפס:

```
None
```

```
def __repr__(self):
```

השיטה תחזיר מחרוזת המייצגת את ה-ByteNode בתצורה הבאה:

‘[byte]=>‘

<u>פלט:</u>

- **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט.

לדוגמה, עבור הרצת הקוד הבא:

```
bn = ByteNode('10011000')
print(bn)
```

יודפס:

```
[10011000]=>
```

חלק ב'

כפי שבתוך הבית מיקום של כל ביט משמעותי כך גם כאשר מייצגים מספר שלם בעזרת מספר בתים – הסדר שלהם חשוב - ככל שבית נמצא משמאל בייצוג כך הוא מקודד חזקות גבוהות יותר של 2.
הבית השמאלי ביותר נקרא הבית ה-MSB (Most Significant Byte) והבית הימני ביותר נקרא הבית ה-LSB (Less Significant Byte). לדוגמא המספר 256 ייוצג באמצעות 2 בתים (אדום - MSB , כחול – LSB):

0000000100000000

בעוד שבשפת Python גודל המספר בזיכרון המחשב יכול להיות דינמי בכדי לאפשר זיכרון בהתאם למספר הבתים הנדרש לייצוג המספר, בשפות תוכנה רבות נקבע מספר קבוע של בתים בהתאם לסוג המשתנה, מאחר ומימוש של מספרים מעל הגודל הקבוע אינו אפשרי בשפות אלו - נממש את המספרים באמצעות רשימה מקושרת של נתונים.

מספר בינארי (LinkedListBinaryNum)

המחלקה תממש ייצוג של מספר שלם וחיובי גדול באמצעות רשימה מקושרת של חוליות מסוג (ByteNode), במימוש של LinkedListBinaryNum, ראש הרשימה יהיה ה-MSB בית בעוד האיבר האחרון ברשימה יהיה ה-LSB בית, מימוש של המספר 256 באמצעות המחלקה LinkedListBinaryNum יתואר כרשימה מקושרת של חוליות:

[00000001]=>[00000000]=>None

ברשימה המייצגת את המספר 0 תהיה חוליה אחת בראש הרשימה שערכה הוא '00000000', עבור כל מספר גדול מאפס הרשימה תכיל את המינימום האפשרי של בתים הנדרשים לייצוג המספר – לא יהיה MSB המכיל אפסים בלבד (שוב, למעט עבור הייצוג של המספר 0).
המחלקה מספר בינארי (LinkedListBinaryNum) מוגדרת על ידי השדות הבאים:

- **head** – ByteNode המייצג את ראש הרשימה.
- **size** – מספר הבתים הדרושים לייצוג המספר.

<u>ממשו את המחלקה LinkedListBinaryNum:</u>

```
def __init__(self, num):
```

בנאי היוצר LinkedListBinaryNum המייצג את הקלט num.

<u>קלט:</u>

- **num [int]**, מספר שלם חיובי בייצוג עשרוני. עבור קלט לא שלם יש לזרוק חריגה מסוג TypeError, עבור קלט לא חיובי יש לזרוק חריגה מסוג ValueError, על החריגות להכיל הודעות אינפורמטיביות כרצונכם.

def add_MSB(self, byte):

שיטה המוסיפה חוליית ByteNode כ-MSB ב-LinkedListBinaryNum.

קלט:

- **byte [str]**, מחרוזת בגודל 8 המכילה תווים שהם '0' או '1' בלבד.

def __repr__(self):

שיטה המחזירה מחרוזת המייצגת את המספר בייצוג בינארי (משמשת בעיקר עבור מפתחים המשתמשים במחלקה). השיטה תחזיר מחרוזת המפרטת את כמות החוליות וסדר ההצבעה שלהן לפי סידור החוליות, עבור רשימות עם חוליה אחת תודפס המילה Byte ולא המילה Bytes.

פלט:

- **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט.

לדוגמה, בהרצת קטע הקוד הבא:

```
bn1 = LinkedListBinaryNum(0)
print(bn1.__repr__())
bn1 = LinkedListBinaryNum(255)
print(bn1.__repr__())
bn1 = LinkedListBinaryNum(4294967296)
print(bn1.__repr__())
```

יודפס:

```
LinkedListBinaryNum with 1 Bytes, Bytes map: [00000000]=>None
LinkedListBinaryNum with 1 Byte, Bytes map: [11111111]=>None
LinkedListBinaryNum with 5 Bytes, Bytes map:
[00000001]=>[00000000]=>[00000000]=>[00000000]=>[00000000]=>None
```

def __str__(self):

שיטה המחזירה מחרוזת מייצגת לצורכי הדפסה (משמשת בעיקר עבור משתמש הקצה – לא מפתח). כל בית יופרד באמצעות התו '|' , שיופיע בתחילת ובסיום המחרוזת.

פלט:

- **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט.

לדוגמה, בהרצת קטע הקוד הבא:

```
bn1 = LinkedListBinaryNum(0)
print(bn1)
bn1 = LinkedListBinaryNum(255)
print(bn1)
```

```
bn1 = LinkedListBinaryNum(4294967296)
print(bn1)
```

יודפס:

```
|00000000|
|11111111|
|00000001|00000000|00000000|00000000|00000000|
```

def __len__(self):

שיטה המחזירה את מספר החוליות ברשימה המקושרת.

<u>פלט:</u>

- **[int]** – מספר הבתים לייצוג המספר.

לדוגמה, בהרצת קטע הקוד הבא:

```
bn1 = LinkedListBinaryNum(4294967296)
print(len(bn1))
```

יודפס:

```
5
```

def __getitem__(self, item):

שיטה המקבלת אינדקס ומחזירה את הבית המתאים. עבור סוג לא תקין יש לזרוק חריגת TypeError ועבור אינדקס לא תקין (עבור X חוליות אינדקס לא תקין יהיה X כולל ומעלה או X-1- כולל ומטה) יש לזרוק חריגת IndexError, לדוגמה, בהרצת קטע הקוד הבא:

```
bn1 = LinkedListBinaryNum(654321)
print(bn1)
print('Iterations', end = ':')
for i in bn1:
    print(i, end=',')
print('\nbn1[0]:', bn1[0])
print('bn1[-1]:', bn1[-1])
```

יודפס:

```
|00001001|11011111|10001111|
Iterations:00001001,11011111,10001111,
bn1[0]: 00001001
bn1[-1]: 11110001
```

יחסי סדר:

על המחלקה לתמוך בכל ששת האופרטורים המייצגים יחס סדר, עליכם להשתמש ב- total ordering.

עבור כל האופרטורים:

קלט:

- **`other` [Object]**, חיובי ושלם, עבור קלט מסוג לא תקין יש לזרוק חריגה מסוג TypeError.

לדוגמה, בהרצת קטע הקוד הבא:

```
bn1 = LinkedListBinaryNum(4294967296)
bn2 = LinkedListBinaryNum(4294967297)
print(bn1 == bn2)
print(bn1 != bn2)
print(bn1 < bn2)
print(bn1 >= bn2)
print(bn1 > bn2)
print(bn1 <= bn2)
```

יודפס:

```
False
True
True
False
False
True
```

אופרטורים אריתמטיים:

- **בסעיף זה עליכם לבצע חיבור או חיסור בינארי (אין להמיר את המספרים הבינאריים למספרים בבסיס 10 ואז לבצע חיבור ולהחזיר למספר בינארי).**
- **ניתן להשתמש במחרוזות.**

```
def __add__(self, other):
```

אופרטור חיבור המחזיר אובייקט חדש מסוג LinkedListBinaryNum. אם הקלט אינו מסוג LinkedListBinaryNum או מספר שלם יש לזרוק חריגה מסוג TypeError, אם הקלט הוא מספר שלם שלילי יש לזרוק חריגה מסוג ValueError.

קלט:

- **`other` [Object]**, עבור קלט מסוג לא תקין יש לזרוק חריגה מסוג TypeError, עבור קלט שלם שלילי יש לזרוק חריגה מסוג ValueError.

```
def __radd__(self, other):
```

אופרטור חיבור מימין.

קלט:

- **`other` [int]**, ניתן להניח ש-other תמיד יהיה שלם וחיובי.

לדוגמה, בהרצת קטע הקוד הבא:

```
bn1 = LinkedListBinaryNum(1000000)
print(bn1)
bn2 = LinkedListBinaryNum(2000000)
print(bn2)
bn3 = bn1 + bn2
print(bn3)
print(LinkedListBinaryNum(2000000)  + 1000000)
print(1000000 + LinkedListBinaryNum(2000000))
```

יודפס:

```
|00001111|01000010|01000000|
|00011110|10000100|10000000|
|00101101|11000110|11000000|
|00101101|11000110|11000000|
|00101101|11000110|11000000|
```

```
def __sub__(self, other):
```

אופרטור חיסור המחזיר אובייקט חדש מסוג LinkedListBinaryNum. אם הקלט אינו מסוג LinkedListBinaryNum או מספר שלם יש לזרוק חריגה מסוג TypeError, אם הקלט הוא מספר שלם שלילי יש לזרוק חריגה מסוג ValueError, אם ערכו של other גדול מערכו של self יש לזרוק חריגה מסוג ValueError.

קלט:

- **`other` [Object]**.

לדוגמה עבור הקוד הבא:

```
print(LinkedListBinaryNum(4294967297)-1)
print(LinkedListBinaryNum(510)-LinkedListBinaryNum(256))
```

יודפס:

```
|00000001|00000000|00000000|00000000|00000000|
|11111110|
```

חלק ג'

חוליה מקושרת דו כיוונית - DoublyLinkedNode

בחלק זה תממשו חוליה של רשימה מקושרת שמצביעה לשני הכיוונים: קדימה ואחורה. המצביע הנוסף מעלה את סיבוכיות הזיכרון אך חוסך בזמן ריצה במקרים שבהם יש להתקדם אחורה ברשימה.

המחלקה חוליה מקושרת דו כיוונית (DoublyLinkedNode) מוגדרת על ידי השדות הבאים:

- **data** – Object המייצג את המידע בחוליה.
- **next** – המייצג את המצביע לחוליה הבאה או None אם זו החוליה האחרונה.
- **prev** – המייצג את המצביע לחוליה הקודמת או None אם זו החוליה הראשונה.

שימו לב: אם חוליה A היא החוליה הבאה של חוליה B אז חוליה B היא החוליה הקודמת של חוליה A ולהפך.

ממשו את המחלקה DoublyLinkedNode:

```
def __init__(self, data):
```

בנאי היוצר DoublyLinkedNode המכיל את הערך data.

קלט:

- **data [Object]**, הערך יכול להיות מכל סוג.

```
def get_data(self):
```

שיטה המחזירה את הערך בחוליה.

פלט:

- **[Object]**.

```
def get_next(self):
```

שיטה המחזירה מצביע לחוליה הבאה.

פלט:

- **[None / DoublyLinkedNode]**.

```
def get_prev(self):
```

שיטה המחזירה מצביע לחוליה הקודמת.

פלט:

- **[None / DoublyLinkedNode]**.

```
def set_next(self, next):
```

שיטה המעדכנת את החוליה הבאה.

קלט:

- **next [DoublyLinkedNode]**.

```
def set_prev(self, prev):
```

שיטה המעדכנת את החוליה הקודמת.

קלט:

- **prev [DoublyLinkedNode]**.

```
def __repr__(self):
```

שיטה המחזירה מחרוזת המייצגת את החוליה עם ציון תחילת החוליה במחרוזת '=>[' וציון סוף החוליה באמצעות המחרוזת '<=]' כאשר בתוך יהיה ייצוג המידע.

פלט:

- **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט.

לדוגמה, בהרצת קטע הקוד הבא:

לדוגמה, בהרצת קטע הקוד הבא:

```
dln1 = DoublyLinkedNode('1st')
print(dln1)
print(dln1.get_data())
dln2 = DoublyLinkedNode('2nd')
dln1.set_next(dln2)
dln2.set_prev(dln1)
print(dln1.get_prev(), dln1.get_next())
print(dln2.get_prev(), dln2.get_next())
```

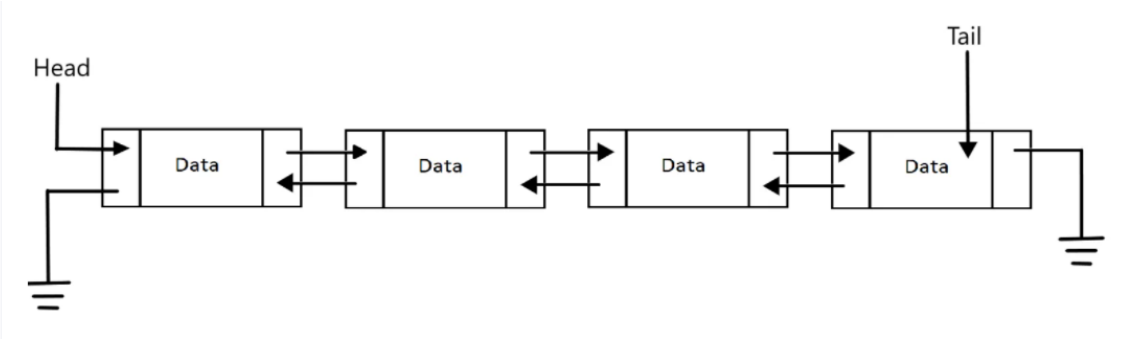
יודפס:

```
=>[1st]<=
1st
None =>[2nd]<=
=>[1st]<= None
```

חלק ד'

רשימה מקושרת דו כיוונית - DoublyLinkedList

רשימה מקושרת דו כיוונית היא רשימה מקושרת הבנויה מחוליות דו כיוניות ומכילה מצביע על ראש החוליה (head) ומצביע לזנב החוליה (tail). כל חוליה ברשימה מצביעה גם על החוליה הבאה אחריה (אלא אם היא האחרונה ואז היא מצביעה על None) וגם על הקודמת לה (אלא אם היא הראשונה ואז היא מצביעה על None).



המחלקה רשימה מקושרת דו כיוונית (DoublyLinkedList) מוגדרת על ידי השדות הבאים:

- **head** – מצביע לחוליה הראשונה או None אם הרשימה ריקה.
- **tail** – מצביע לחוליה האחרונה או None אם הרשימה ריקה.
- **size** – מכיל את כמות החוליות ברשימה.

<u>ממשו את המחלקה DoublyLinkedList:</u>

```
def __init__(self):
```

בנאי היוצר רשימה DoublyLinkedList ריקה.

```
def __repr__(self):
```

שיטה המחזירה מחרוזת המייצגת את הרשימה המקושרת הדו כיוונית עם ציון ראש הרשימה (תחילת המחרוזת 'Head==>') וזנב הרשימה (סיום המחרוזת '<==Tail').

<u>פלט:</u>

- **[str]** - מחרוזת המייצגת את האובייקט לפי הפורמט המפורט.

לדוגמה, בהרצת קטע הקוד הבא עבור הרשימה הריקה הבאה:

```
dll = DoublyLinkedList()
print(dll)
```

יודפס:

```
Head==><==Tail
```

```
def get_head(self):
```

שיטה המחזירה את המצביע לחוליה הראשונה ברשימה.

פלט:

- **[DoublyLinkedNode / None]**.

```
def get_tail(self):
```

שיטה המחזירה את המצביע לחוליה האחרונה ברשימה.

פלט:

- **[DoublyLinkedNode / None]**.

```
def __len__(self):
```

שיטה המחזירה את מספר החוליות ברשימה המקושרת.

פלט:

- **[int]**.

```
def __is_empty__(self):
```

שיטה המחזירה האם הרשימה ריקה.

פלט:

- **[int]** True אם הרשימה אינה מכילה חוליות או False אם אחרת.

לדוגמה, בהרצת קטע הקוד הבא עבור הרשימה הריקה הבאה:

```
dll = DoublyLinkedList()
print(len(dll))
print(dll.is_empty())
print(dll.get_head())
print(dll.get_tail())
```

יודפס:

```
0
True
None
None
```

```
def add_at_start(self, data):
```

שיטה המוסיפה את data בתחילת הרשימה.

קלט:

- **data [Object]** הערך יכול להיות מכל סוג.

```
def remove_from_end(self):
```

- שיטה המוציאה ומחזירה את **המידע** מהחוליה האחרונה ברשימה או חריגה מסוג StopIteration אם הרשימה ריקה.

פלט:

- **[Object]** הערך יכול להיות מכל סוג.

לדוגמה, בהרצת קטע הקוד הבא עבור הרשימה הריקה הבאה:

```
dll = DoublyLinkedList()
dll.add_at_start('B')
dll.add_at_start(2)
dll.add_at_start('or not')
dll.add_at_start('B')
dll.add_at_start(2)
print(dll)
res = dll.remove_from_end()
print(res)
print(dll)
```

יודפס:

```
Head==>[2]<==>[B]<==>[or not]<==>[2]<==>[B]<==Tail
B
Head==>[2]<==>[B]<==>[or not]<==>[2]<==Tail
```

חלק ה'

בהרצאה למדנו על מבנה הנתונים תור וראינו מימוש באמצעות רשימה. בחלק זה נממש תור באמצעות רשימה מקושרת דו-כיוונית ועם פונקציונליות נוספת.

תור - DoublyLinkedListQueue

המחלקה תור (Queue) מוגדרת על ידי השדה הבא **בלבד**:

- **data** – שדה מסוג רשימה מקושרת דו כיוונית DoublyLinkedList, יש להשתמש ב API שלה בלבד.

שלד המחלקה יסופק לכם, אין לכתוב שיטות נוספות ובכל שיטה במחלקה בה לא אושר במפורש - אין ליצור מבני נתונים נוספים – גם לא רשימה מקושרת דו כיוונית נוספת (מלבד זו שמוגדרת בבנאי).

ממשו את המחלקה DoublyLinkedListQueue:

```
def __init__(self):
```

בנאי (ממומש)

```
def enqueue(self, val):
```

הוספת איבר לתור.

קלט:

val [Object], הערך יכול להיות מכל סוג.

```
def __len__(self):
```

שיטה המחזירה את מספר האברים בתור.

פלט:

- **[int]** – מספר המייצג את כמות האברים בתור.

```
def __repr__(self):
```

שיטה המחזירה מחרוזת המייצגת את התור. כל זוג נתונים יופרד בפסיק, לא יהיה פסיק לפני הנתון הראשון ואחרי הנתון האחרון, המחרוזת תתחיל ב- '['<=Newest' ותסתיים ב- 'Oldest=>']'.

פלט:

- **[str]** – מחרוזת המייצגת את התור כפי שהוגדר.

לדוגמה, בהרצת קטע הקוד הבא:

```
q = DoublyLinkedListQueue()
print(q)
print(len(q))
q.enqueue('you')
q.enqueue(4)
q.enqueue('queue')
print(q)
print(len(q))
```

יודפס:

```
Newest=>[]<=Oldest
0
Newest=>[queue,4,you]<=Oldest
3
```

```
def is_empty(self):
```

שיטה המחזירה ערך בוליאני המציין האם התור ריק.

<u>פלט:</u>

- **[bool]** – ערך בוליאני: True אם אורך התור הוא 0 או False אם אחרת.

```
def dequeue(self):
```

הוצאת והחזרת איבר מתור או זריקת חריגה מסוג StopIteration אם התור ריק.

<u>פלט:</u>

- **[Object]** הערך יכול להיות מכל סוג שהוא.

לדוגמה, בהרצת קטע הקוד הבא:

```
q = DoublyLinkedListQueue()
print(q.is_empty())
q.enqueue('you')
q.enqueue(4)
q.enqueue('queue')
print('Dequeue:')
print(q)
print(q.dequeue())
```

```
print(q)
print(q.is_empty())
```

יודפס:

```
True
Dequeue:
Newest=>[queue,4,you]<=Oldest
you
Newest=>[queue,4]<=Oldest
False
```

מימוש איטרטור

סדר האיברים ייקבע לפי סדר ההכנסה לתור.

```
def __iter__(self):
```

שיטה המחזירה איטרטור למחלקה.

בשיטה זו ובשיטה זו בלבד (ובמחלקה זו בלבד) ניתן להוסיף שדה למחלקה.

```
def __next__(self):
```

שיטה המחזירה את ה**מידע** של החוליה הבאה באיטרציה.

לדוגמה, בהרצת קטע הקוד הבא:

```
q = DoublyLinkedListQueue()
q.enqueue('u')
q.enqueue(4)
q.enqueue('Queue')
print('For loop:')
for i in q:
    print(i)
print('Iterations:')
it = iter(q)
print(next(it))
print(next(it))
print(next(it))
#print(next(it)) #StopIteration
```

יודפס:

For loop:

u

4

Queue

Iterations:

u

4

Queue

חלק ו'

ניהול מספרים שלמים - שימוש במבני נתונים קיימים

חברת המעבדים "CPU for you" מעוניינת לנתח את תמונת הזיכרון של המחשב בכדי לתכנן מעבדים יעילים יותר. ראש צוות הפיתוח ביקש משלושה מהנדסים לכתוב מחלקה עם הפונקציונאליות שתפורט בהמשך....

יפורסם במוצא"ש, ה 1.1.22

בהצלחה!! אוריאל.