תרגיל 11 – מבני נתונים ואיטרטורים

בתרגיל הנ"ל תתבקשו לממש תוכנית לניהול משימות.

הערות כלליות על התרגיל:

- התרגיל אומנם נראה ארוך אך זאת רק מראית עין ☺

רקע מקדים 1:

מערכות רבות צריכות לבצע מספר רב של פעולות בו זמנית, אך האם כל פעולה היא בחשיבות זהה? ככל הנראה שלא. למשל, במצב בו אירגון גדול נשאר עם מעט נייר לצורכי הדפסה תהיה חשיבות רבה יותר (כנראה) לצרכי ההדפסה של המנכ"לית לעומת צרכי ההדפסה של אחד העובדים הפשוטים. דוגמא נוספת יכולה להיות ניהול של לוח משמרות, הבקשות למשמרות של עובדים ותיקים כנראה ייקבלו עדיפות על אלו של העובדים החדשים.

בתרגיל זה אנחנו נממש מערכת לניהול משימות עבור המחשב, מערכת כזו יכולה להיות מוטמעת למשל בשרתי אינטרנט או שרתי ניהול של דואר אלקטרוני.

המטרה של התרגיל הינה היכרות שלכם עם יצירה של מבני נתונים ומימוש של API מוגדר.

רקע מקדים 2:

אנחנו נממש את התור כתור רשימה מקושרת, רשימה מקושרת הינה מבנה נתונים המורכב מאיברים המקושרים זה לזה – האיברים הללו נקראים Nodes. סדר הרשימה יהיה מהעדיפות הגבוהה ביותר לנמוכה ביותר. **הערה כללית:** מבנה הרשימה אינו המבנה העדיף ביותר לבנייה של תור, מבנים כאלו תכירו בקורס מבני נתונים.

אנחנו נמדל את האיברים ברשימה כתור מחלקה בשם Node אשר תכיל שני משתנים, המשימה לביצוע והאיבר (Node) הבא ברשימה.

ה-API של משימה (כלשהי) ניתנת כאן:

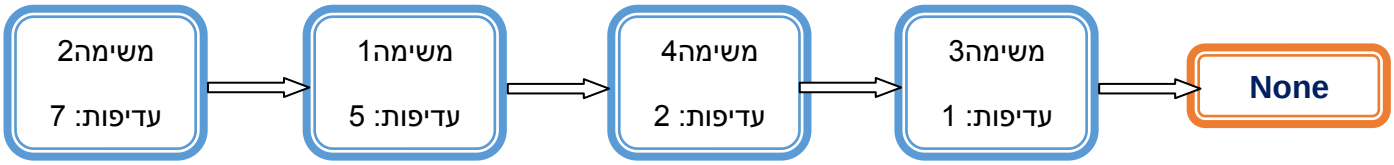
- **get_priority(self)**: פונקציה זו תחזיר את החשיבות של המשימה.
- **set_priority(self, new_priority):** פונקציה זו תשנה את העדיפות של המשימה הנ"ל.
- **execute(self)**: פונקציה זו תבצע את המשימה.
- **__str__(self):** פונקציה זו מייצגת את המשימה כתור מחרוזת.
- **__eq__(self):** פונקציה זו תחזיר האם המשימה זהה למשימה אחרת.

אנחנו נספק לכם מחלקת משימה אחת לדוגמא (**StringTask**), אבל אל תניחו שאלו המשימות היחידות שנבדוק אתכם עלפיהם (מכיוון שהגדרנו רק API יכולות להיות מספר רב של משימות לבדיקה).

למשל אם יהיו לנו את המשימות והעדיפויות הבאות בפורמט של (משימה,עדיפות):

(משימה1, 5) , (משימה2, 7) , (משימה3, 1) , (משימה4, 2)

התור שלנו ייצטרך להראות ככה (**שימו לב** שהאיבר האחרון בתור מצביע על **None**):



משימות התרגיל

הערה 1: כל מחלקה שתיצרו בתרגיל צריכה להיות מוגדרת בקובץ עם אותו שם לפי הכללים הבאים:

1. כל האותיות צריכות להיות קטנות.
2. שתי מילים שונות צריכות להיות מופרדות בקו תחתון, '_'.

לדוגמא: המחלקה Class צריכה להיות בקובץ בשם class.py ומהחלקה MyLongClass צריכה להיות בקובץ בשם my_long_class.py .

הערה 2: אנחנו מספקים לכם אובייקט משימה לדוגמא (בהמשך) וקובץ לבדיקה, תוצאות הבדיקות נמצאות בתוך ה-docstring של הבדיקה.

הערה 3: אתם **לא** יכולים להניח שכל המשימות יהיו עם עדיפות שונה (כלומר יכולות להיות שתי משימות, או יותר, עם עדיפות זהה).

משימה 1 – יצירה של איבר ברשימה

שם המחלקה: **Node**

תוכן המחלקה: המחלקה הזו תייצג לנו את איברי התור, לכל אחד האיברים (כמו שהוסבר למעלה) יש שני משתנים: (1) המשימה שהוא מייצג ו-(2) האיבר הבא בתור.

ה-**API** שחושפת המחלקה:

- **__init__(self, task, next = None)** - פונקציה זו תאתחל את האיבר בתור, אם לא נאמר אחרת היא תשמור שהאיבר הבא ברשימה הינו **None**.
- **get_priority(self)** - פונקציה זו תחזיר את העדיפות של המשימה שהמחלקה עוטפת.
- **set_priotiry(self, new_priority)** - פונקציה זו תשנה את העדיפות של המשימה שהמחלקה עוטפת אל הערך new_priority.
- **get_task(self)** - פונקציה זו תחזיר את המשימה.
- **get_next(self)** - פונקציה זו תחזיר את האיבר הבא אחרינו (האיבר שאנחנו מצביעים עליו).
- **set_next(self, next_node)** - פונקציה זו תשנה את האיבר הבא שאנחנו מצביעים עליו (מהאיבר הנוכחי) אל next_node.
- **has_next(self)** - פונקציה זו תחזיר האם יש איבר אחרינו, כלומר תחזיר אמת אמ"מ האיבר שאנחנו מצביעים עליו אינו **None**.

הערה 1: שמות משתנים פרטיים צריכים להתחיל בתו קו-תחתון ("_")

הערה 2: אל תשכחו לרשום דוקסטרינג.

משימה 2 – יצירת תור קדימויות

שם המחלקה: **PriorityQueue**

תוכן המחלקה: המחלקה הזו תכיל את כל המשימות שלנו, במהלך התרגיל נרחיב אותה ואת השימוש בה.

כרגע הפונקציה היחידה שצריכה להיות במחלקה (ושאתם צריכים לממש) הינה פונקציית האתחול הבאה:

__init__(self, tasks = [])

שימו לב שאנחנו מקבלים רשימה של משימות אפשריות להתחיל איתן את התור, אם הרשימה אינה ריקה אנחנו נרצה להוסיף את כל המשימות האלו לתור שלנו בעזרת פונקציית append שנתאר בהמשך.

הערה 1: אין בעיה להשתמש במשתני עזר בפונקציות המחלקה.

הערה 2: האיבר שבראש התור בהתחלה צריך להיות **None**. אם הפונקציה נקראת עם משימות בתוכה אז הפונקציה append (המוסיפה איברים חדשים לתור) תצטרך לדאוג לכך שהאיבר שבראש התור יהיה איבר אמיתי (ולא **None**).

משימה 3 – הוספת איבר לתור

אנחנו נרצה את היכולת להוסיף איבר לתור, בשביל זה תצטרכו לממש את הפונקציה הבאה:

enque(self, task)

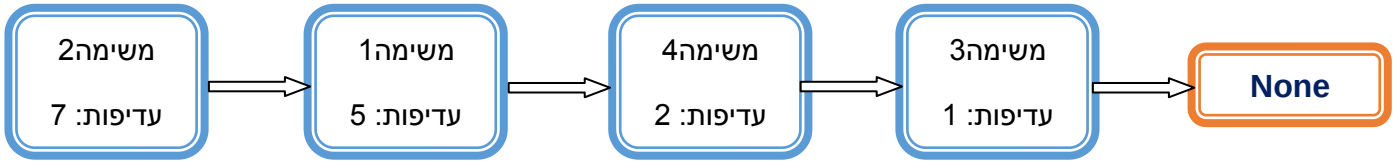
הפונקציה מקבלת כקלט משימה (תזכרו שלמשימה יש עדיפות) ומוסיפה אותה אל התור. ישנן שתי דרכים לממש את הפונקציה הנ"ל:

1. הוספת המשימה אל ראש התור, ללא התחשבות בעדיפות שלה. דרך זו תתן לנו סיבוכיות זמן הכנסה של O(1) - אך בשביל להוציא איבר מהתור נצטרך לעבור על כולו (בכדי למצוא את המשימה עם העדיפות הגבוהה ביותר) בסיבוכיות של O(n).
2. הוספת המשימה אל המיקום המתאים לה בתור, כלומר עם התחשבות בעדיפות שלה. במצב זה סיבוכיות ההכנסה וההוצאה מתהפכות (לעומת האופציה הקודמת).

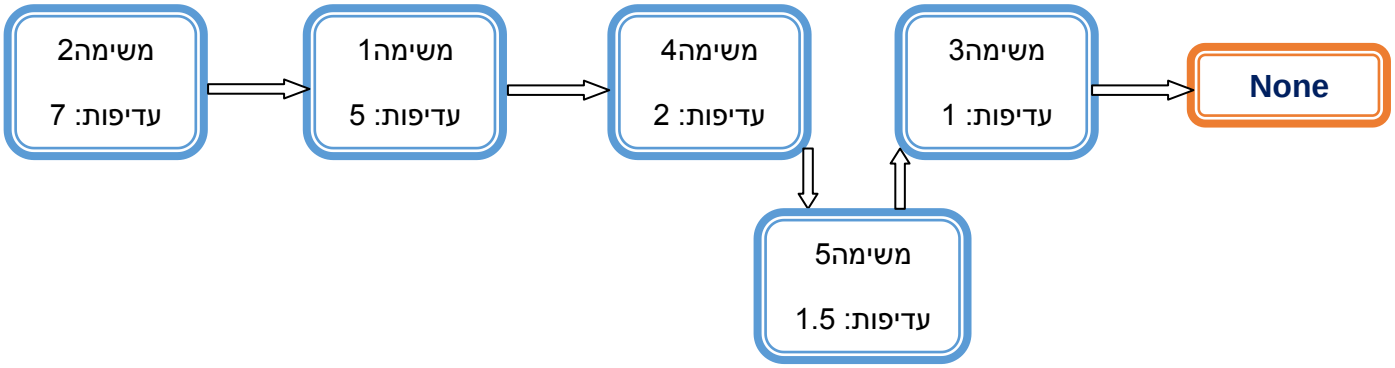
בתרגיל שלנו **אתם נדרשים לממש את הדרך השנייה**.

דוגמא:

אם היה לנו את התור (שבתחילת התרגיל):



והיינו מקבלים את המשימה החדשה (משימה5, 1.5), נסתכל איך היה נראה התור שלנו עבור הדרך השנייה (**לאחר** ביצוע ההכנסה):



היינו צריכים לגלות שמיקום המשימה החדשה הינו בין משימה4 למשימה2 ולשנות את התור בהתאם.

<u>הערה 1:</u> אם יש שתי משימות (או יותר) עם אותו תור המשימה החדשה צריכה להיות אחרונה מבין כל המשימות בעלות אותה עדיפות. כלומר אם נכניס משימה חדשה עם עדיפות 7, למשל את (משימה6, 7) אז משימה 6 צריכה לבוא בין משימה2 למשימה1.

<u>משימה 4 – החזרה, ללא הוצאה, של המשימה שבראש התור</u>

עליכם לממש את הפונקציה:

peek(self)

פונקציה זו תחזיר את המשימה שבראש התור (ללא הוצאתה).

<u>הערה:</u> שימו לב שקריאות חוזרות לפונקציה הזו צריכות להחזיר את אותה משימה (כל עוד לא שינינו את התור).

<u>משימה 5 – החזרה, עם הוצאה, של האיבר שבראש התור</u>

עליכם לממש את הפונקציה:

deque(self)

פונקציה זו תחזיר את המשימה שבראש התור, אך בשונה מהסעיף הקודם הפונקציה תוציא את המשימה מן התור. אם התור **<u>ריק</u>** על הפונקציה להחזיר **None**.

<u>הערה 1:</u> שימו לב שקריאות לפונקציה הזו אחת אחרי השנייה יוציאו את המשימות מהתור מהעדיפות הגבוהה ביותר לנמוכה ביותר.

<u>משימה 6 – החזרת ראש התור</u>

עליכם לממש את הפונקציה:

get_head(self)

פונקציה זו תחזיר, ולא תוציא מהתור, את ה-Node שמייצגת את ראש התור שלנו (הפונקציה תחזיר ערך מסוג Node).

<u>הערה 1:</u> שימו לב שבמשימה זו אתם צריכים להחזיר אובייקט מסוג Node, לעומת משימה 5 בה החזרנו אובייקט שעונה על ה-API של משימה (כפי שהוגדר בראש התרגיל). כלומר במשימה הקודמת החזרנו את המשימה שנמצאת בתוך האובייקט שהיינו מחזירים בקריאה לפונקציה הזו.

משימה 7 - שינוי עדיפות של משימה

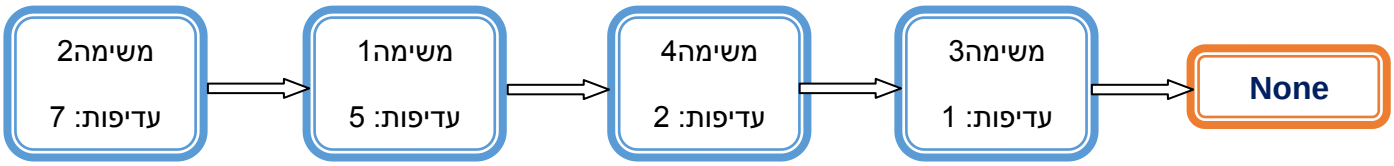
עליכם לממש את הפונקציה:

change_priority(self, old, new)

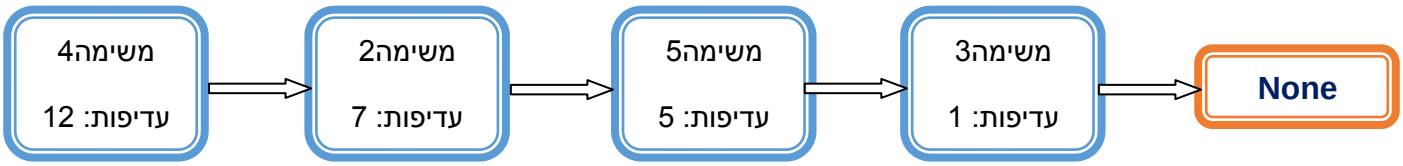
הפונקציה מקבל כקלט את העדיפות הישנה ואת העדיפות החדשה של משימה כלשהי. אם קיימת משימה כזו על הפונקציה לשנות את העדיפות של המשימה ולעדכן את התור בהתאם. **שימו לב!** אם אין משימה עם ערך השווה לעדיפות הישנה התור לא יישתנה.

דוגמא:

אם היה לנו את התור (שבתחילת התרגיל):



והיינו קוראים לפונקציה עם עדיפות ישנה=2 ועדיפות חדשה=12 על התור להשתנות בהתאם:



לעומת זאת, אם היינו קוראים לפונקציה עם עדיפות ישנה=12 ועדיפות חדשה=9 התור לא היה משתנה.

הערה 1: אם יש יותר ממשימה אחת עם העדיפות המבוקשת אתם נדרשים לשנות את הראשונה בתור.

משימה 8 – החזרת אורך התור

עליכם לממש את הפונקציה:

__len__(self)

פונקציה זו נועדה לשימוש במקרה שנרצה לדעת מה אורך התור. למשל,אם שם התור שלנו הוא my_queue ואורכו 5 אז קריאה ל- len(my_queue) תחזיר את הערך 5.

חשוב 1: אם התור ריק על הפונקציה להחזיר 0.

חשוב 2: סיבוכיות הריצה של הפונקציה צריכה להיות קבועה, כלומר לא תלויה בכמות האיברים שיש לי בתור (**רמז:** אורך הפונקציה לא צריך להיות יותר משורה אחת).

חשוב 3: מותר לכם (ואף רצוי) לשנות את שאר הפונקציות במחלקה על מנת לעמוד בדרישות משימה זו.

משימה 9 – מימוש איטרטור לתור

עליכם לממש את הפונקציות __iter__(self) ו- __next__(self) (כפי שראינו בשיעור ובתרגולים) על מנת לייצר איטרטור למחלקת התור שלנו.

הערה: שימו לב שאנחנו ממשים **איטרטור** ולא גנרטור, על כן אתם נדרשים להשתמש ב-return ולא ב-yield בפונקציה __next__(self).

כלומר לאחר מימוש הפונקציות הנ"ל ובהנתן שקיים לנו אובייקט תור " my_queue " אשר בו יש כבר מספר מסויים של משימות, הקריאה הבאה צריכה להדפיס את כל משימות התור מהעדיפות הגבוהה לנמוכה:

```
for task in my_queue:
    print( task )
```

מה יקרה אם נקרא בדיוק לאותן שורות פעם שנייה? הסבירו בקובץ ה-README שלכם.

<u>משימה 10 – מימוש איטרטור לתור (נסיון שני)</u>

עליכם לממש את המחלקה **PriorityQueueIterator**, המחלקה תהיה אובייקט איטרטור לתור שלנו.

המחלקה תכיל 4 פונקיות: (1) פונקציית אתחול (שתקבל את התור שלנו) ; (2) פונקציית __iter__ ; (3) פונקציית __next__ ו-(4) פונקציית has_next.

הפונקצייה החדשה היחידה במחלקה הזו הינה הפונקציה has_next ומשמעותה האם יש איבר נוסף באיטרטור או לא (כלומר קריאה נוסף לפונקציה next תגרור שגיאת StopIteration).

כעת, קריאה לפונקציה __iter__ (מתוך המחלקה **PriorityQueue**) צריכה להחזיר מופע חדש של מחלקה זו. בצעו את אותו ניסוי כבמשימה הקודמת (משימה 9) ותארו מה ההבדל בין המימוש של משימה זו למשימה הנוכחית בקובץ ה-README.

<u>הערה 1:</u> מימוש נכון של המשימה הקודמת יגרור שהמשימה הזו תהיה כמעט טריוויאלית.

<u>הערה 2:</u> אל תמחקו את הפונקציה __next__ שכתבתם במשימה הקודמת מן המחלקה **PriorityQueue**.

<u>משימה 11 – הדפסת התור</u>

עליכם לממש את הפונקציה:

__str__(self)

פונקציה זו תחזיר מחרוזת המייצגת את התור. המחרוזת צריכה להיות בצורה זהה לייצוג רשימה כמחרוזת, כלומר: [איבר_אחרון ...,איבר_שני ,איבר_ראשון] – כאשר האיברים הינם ייצוג כל אחת מהמשימות כמחרוזת (שימו לב שכל משימה ממשת אף היא את הפונקציה __str__ גם כן).

<u>הערה 1:</u> התנהגות הפונקציה צריכה להיות זהה לפקודה הבאה:

```
str( list( my_queue ) )
```

אך **אסור** לכם לתרגם את התור my_queue לרשימה.

משימה 12 – חיבור של שני תורים שונים (והחזרה של תור חדש משולב)

עליכם לממש את הפונקציה:

__add__(self, other)

פונקציה זו תקבל כקלט תור נוסף ותחזיר תור חדש המכיל את איברי שני התורים, התור השמאלי בביטוי והתור הימיני בביטוי.

דוגמא:

הפונקציה __add__ תקרא על התור q1 בביטוי q1+q2.

הוסיפו הסבר לסיבוכיות זמן הריצה של פונקציה זו בקובץ ה-README שלכם.

משימה 13 – שוויון של שני תורים

עליכם לממש את הפונקציה:

__eq__(self, other)

פונקציה זו תקבל כקלט תור נוסף ותחזיר האם התור שלנו שווה לתור השני.

דוגמא:

הפונקציה __eq__ תקרא על התור q1 בביטוי q1==q2, תורים נחשבים כשווים אם המשימות שלהם שוות.

הוסיפו הסבר לסיבוכיות זמן הריצה של פונקציה זו בקובץ ה-README שלכם.

הערות לסיכום

הערה 1: המימוש של התור שלנו בצורה של רשימה הוא לא הכי יעיל שניתן, אבל על כל זאת ועוד בקורס דאסט סמסטר הבא.

הערה 2: התחילו את התרגיל מוקדם.

הערה 3: הערות חשובות יופיעו בראש פורום התרגיל.

קובץ ה-README

אל תשכחו להוסיף לקובץ:

1. תשובה לשאלה שנשאלה במשימה 9.
2. תשובה לשאלה שנשאלה במשימה 10.
3. תשובה לשאלה שנשאלה במשימה 12.
4. תשובה לשאלה שנשאלה במשימה 13.

נהלי הגשה

הלינק להגשה של התרגיל הינו תחת ex11.

בתרגיל זה עליכם להגיש את הקבצים הבאים:

1. node.py – בקובץ זה תממשו את מחלקת ה-Node.
2. priority_queue.py – בקובץ זה תממשו את התור כפי שמתואר בתרגיל.
3. priority_queue_iterator.py – בקובץ זה תממשו את האיטרטור לתור כפי שמתואר במשימה 10.
4. README (על פי פורמט ה-README לדוגמא שיש באתר הקורס, ועל פי ההנחיות לכתיבת README המפורטות בקובץ נהלי הקורס).

יש ליצור קובץ tar הנקרא ex11.tar המכיל בדיוק את ארבעת הקבצים הנ"ל (בנוסף לקבצים הנוספים שייצרתם – אם הם קיימים), בעזרת פקודת ה-shell הבאה:

tar cvf ex11.tar node.py priority_queue.py priority_queue_iterator.py README, אם קיימים עוד קבצים שייצרתם הוסיפו את השמות שלהם אחרי קובץ ה-README.

הערה: מומלץ לבדוק את קובץ ה-tar שיצרתם על ידי העתקת התוכן שלו לתיקייה נפרדת ופתיחתו (extract) בעזרת ביצוע הפקודה: tar xvf ex11.tar, ולאחר מכן יש לבדוק באמצעות הפקודה ls -h שכל הקבצים הדרושים קיימים שם ולא ריקים.

סקריפט קדם-הגשה (Pre submit script):

זהו סקריפט לבדיקה בסיסית של קבצי ההגשה של התרגיל. על מנת להריץ את הסקריפט לתרגיל 7 הריצו את השורה הבאה ב-shell:

~/intro2cs/bin/presubmit/ex11 ex11.tar

הסקריפט מייצר הודעת הצלחה במקרה של מעבר כל הבדיקות הבסיסיות והודעות שגיאה רלוונטיות במקרה של כישלון בחלק מהבדיקות.

שימו לב, סקריפט קדם ההגשה נועד לוודא רק תקינות בסיסית, ומעבר של בדיקות הסקריפט לא מבטיח את תקינותה של התוכנית! עליכם לוודא בעצמכם שהתוכנית שלכם פועלת כפי שדרוש.

הגשת קובץ tar

עליכם להגיש את הקובץ ex11.tar בקישור ההגשה של תרגיל 11! לאחר הגשת התרגיל, ניתן ומומלץ להוריד אותו ולוודא כי הקבצים המוגשים הם אלו שהתכוונתם להגיש וכי הקוד עובד על פי ציפיותיכם.

לאחר שאתם מגישים את התרגיל באתר הקורס, תוך מספר שניות יווצר הקובץ submission.pdf . עליכם לבדוק שהכל תקין בקובץ submission.pdf! ואם יש בעיה כלשהי בקבצים שלכם שבאה לידי ביטוי בקובץ ה-pdf עליכם לתקן אותה, גם אם לא נאמר שום דבר בפירוש בתרגיל לגבי זה, כך שקובץ ה-pdf שמיוצר מהתרגיל שלכם יהיה תקין לגמרי. וודאו שאין שורות ארוכות מדי בקוד שלכם שנחתכות בקובץ ה-pdf, ושהקובץ מסודר ורואים בו את הכל בצורה טובה וברורה. זכרו את ההגבלה של 79 תווים לכל היותר בשורה (כולל הערות).

בהצלחה ☺!