התרגיל 6: Abstract data type

תאריך פרסום: 8.1.24

תאריך הגשה: 28.1.24 בשעה 23:59

מתרגלת אחראית: אסראא נסאסרה

משקל תרגיל: 4 נקודות

הנחיות לתרגיל:

1) אין לייבא ספריות, חיצוניות או מובנות (למעט functools, random ו-copy).
2) שלד המחלקות של תרגיל בית זה יינתן לכם (ראו פירוט למטה). תיקיית המטלה תכיל בנוסף לקובץ המטלה עצמו (ה-pdf) את רשימת הקבצים הבאים:

הקבצים הבאים עבור שאלה 1: Cell.py, CellUpdateError.py, Board.py, Dice.py, Player.py, ו-Game.py.
הקבצים הבאים עבור שאלה 2: Task.py, Worker.py, TaskScheduler.py, ו-id_generator.py.

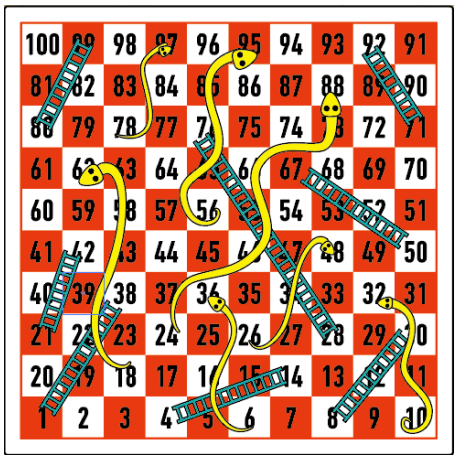
לרשותכם קבצי דוגמאות ההרצה הבאים:
לשאלה 1: Cell.txt, Board.txt, Dice.txt, Player.txt, ו-Game.txt.
לשאלה 2: Task.txt, Worker.txt ו- TaskScheduler.txt.

בנוסף למחלקות שאתם נדרשים לממש, יסופק לכם קובץ ADTs.py אשר יכיל מימוש של מבני הנתונים העומדים לרשותכם (Queue, ו-Stack).

3) מומלץ לקרוא את כל העבודה **לפני** תחילת המימוש. אנו ממליצים להבין היטב את ארכיטקטורת המערכת ולתכנן אותה על דף נייר לפני תחילת כתיבת הקוד.
4) **בתרגיל זה עליכם לבדוק תקינות קלט רק כאשר יש דרישה ברורה בשאלה**. בהיעדר דרישה זו הניחו כי הקלט תקין. במידה והקלט לא תקין עליכם לזרוק חריגה מתאימה בהתאם לדרישת השאלה\הסעיף. **אינכם נדרשים לכתוב docstring בעבודה זו.**
5) מותר להוסיף שדות למחלקות שתממשו (למעט מבני הנתונים שיש לכם ב-ADTs.py – אותם אין לשנות), ומותר (מומלץ!) להוסיף שיטות עזר למחלקות שתממשו.

שאלה 1:

בשאלה זו תממשו מערכת המדמה את משחק סולמות ונחשים. משחק הסולמות והנחשים מורכב מלוח שכל משבצת בו ממוספרת משמאל-לימין-לשמאל ומלמטה-למעלה (ראו איור) ומשתתפים בו שני שחקנים או יותר. בכל תור, כל שחקן מטיל קובייה ומתקדם לפי המספר שהורתה הקובייה. אם השחקן מגיע לרגלי "סולם", הוא מטפס לראש הסולם. אם השחקן מגיע לראש של "נחש", הוא גולש לזנב. אם שחקן מגיע למשבצת שעליה נמצא שחקן שני, השחקן השני נשלח חזרה למשבצת הראשונה בלוח ("1"). המשחק מסתיים כאשר אחד השחקנים מגיע למשבצת האחרונה ("100") בלוח או עובר אותה.



סעיף א'- המחלקה Cell:

השלימו את המימוש של המחלקה משבצת (**Cell**) המייצגת משבצת בלוח של משחק סולמות ונחשים. כל משבצת בלוח יכולה להכיל עד שני מסלולים אפשריים: מסלול ישיר המוביל למשבצת הבאה ברצף על פי עמדתה בלוח ומסלול נוסף שיכול להוביל למשבצת אחרת בלוח – קדימה בלוח (כמו סולם) או אחורה (כמו נחש).

המחלקה משבצת (**Cell**) מכילה את השדות הבאים:

- שדה position – שדה מטיפוס int המכיל את עמדת המשבצת בלוח.
- שדה cell_type – שדה מטיפוס מחרוזת המייצג את סוג המשבצת. השדה יאותחל עם אחד הערכים הבאים: 'L' כאשר המשבצת היא סולם (ladder), 'S' כאשר המשבצת היא נחש (snake) או 'R' כאשר המשבצת רגילה (regular).
- שדה next – שדה המצביע על המשבצת בעמדה הישירה הבאה.
- שדה leap – שדה המצביע על משבצת **נוספת** (השונה מ-next) בלוח אליה אפשר לגשת מהמשבצת הנוכחית – קדימה בלוח (כאשר המשבצת הנוכחית היא סולם) או אחורה (כאשר המשבצת הנוכחית היא נחש).

1) ממשו את בנאי המחלקה אשר מקבל את עמדת המשבצת position וסוג המשבצת cell_type ומאתחל מופע של משבצת. מופע חדש של משבצת אינו מצביע על משבצות אחרות בעת אתחולו.

אם הארגומנטים אותם מקבל הבנאי אינם מהטיפוס המתאים או שערכם אינו תקין, עליכם לזרוק את החריגה TypeError, או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה. בהעדר הארגומנט cell_type יש לאתחל אותו עם הערך "R".

```
def __init__(self, position, cell_type="R")
```

2) ממשו את השיטה update_next אשר מקבלת את הארגומנט next_cell מופע של משבצת (**Cell**) ומטרתה לעדכן את השדה next של המשבצת הנוכחית.
אם הארגומנט אינו מהטיפוס המתאים עליכם לזרוק את החריגה CellUpdateError (ראו סעיף ב'), בצירוף הודעה מתאימה. מאחר והשדה next מצביע על המשבצת בעמדה הישירה הבאה, עליכם לבדוק שהארגומנט next_cell מכיל משבצת שהעמדה שלה היא העמדה העוקבת במדויק לזו של המשבצת הנוכחית. אם דרישה זו אינה מתקיימת עבור הארגומנט next_cell, עליכם לזרוק את החריגה ValueError, בצירוף הודעה מתאימה.

אין לאפשר עדכון חוזר של שדה זה- במקרה כזה עליכם לזרוק את החריגה CellUpdateError, בצירוף הודעה מתאימה.

```
def update_next(self, next_cell):
```

3) ממשו את השיטה update_leap אשר מקבלת מופע של משבצת (**Cell**) ומעדכנת את השדה leap של המשבצת. הארגומנט leap חייב להיות משבצת בעלת עמדה קודמת אם סוג המשבצת הוא 'S', ומשבצת בעלת עמדה באה אם סוג המשבצת הוא 'L'. אם המשבצת היא 'R' אז אין לאפשר עדכון של השדה leap.

אם הארגומנט אינו מהטיפוס המתאים עליכם לזרוק את החריגה CellUpdateError (ראו סעיף ב'), בצירוף הודעה מתאימה. אם ערכו של הארגומנט אינו תקין עליכם לזרוק את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה.
אם סוג המשבצת הנוכחית הוא 'R' זרקו חריגה את החריגה CellUpdateError בצירוף הודעה מתאימה.
אין לאפשר עדכון חוזר של השדה leap- במקרה כזה עליכם לזרוק את החריגה CellUpdateError, בצירוף הודעה מתאימה.

```
def update_leap(self, leap):
```

4) עליכם לדרוס את האופרטור __repr__ כך שכעת הוא יחזיר תיאור למשבצת לפי הפורמט הבא:

f"{position}:{cell_type}->{leap.position or ''}"
כאשר שחור מסמן את התווים הקבועים ואדום את ערכי השדות המתאימים (אם המשבצת אינה מסוג R יש להדפיס את המשבצת הנוספת אליה אפשר להגיע מהמשבצת הנוכחית - השדה leap-, אחרת מחרוזת ריקה).
דוגמאות הרצה: ראו את הקובץ Cell.txt

סעיף ב' – המחלקה CellUpdateError:

ממשו את החריגה **CellUpdateError** אשר תיזרק כאשר מנסים לעדכן בצורה לא תקינה את השדה **next** או את השדה **leap** למופע משבצת (**Cell**).
בחריגה זו עליכם לדרוס את השיטה **__str__** של המחלקה. על השיטה להחזיר את המחרוזת הבאה המייצגת תיאור של החריגה:
"Invalid attempt to modify the next or leap attribute!"

סעיף ג' – המחלקה Board:

השלימו את המימוש של המחלקה **Board** המייצגת את הלוח של המשחק. המחלקה לוח מכילה את השדות הבאים:

- שדה פרטי size מטיפוס int, מספר טבעי הגדול שווה ל-25, השדה size מייצג את גודל הלוח של המשחק.
- שדה פרטי grid המייצג את הלוח עצמו. השדה grid הוא מטיפוס Cell והוא מצביע על המשבצת הראשונה של הלוח.

1) ממשו את בנאי המחלקה **Board** אשר מקבל את שלושת הארגומנטים:
   - board_width מספר טבעי הגדול מ-4 המייצג את רוחב הלוח, ערך ברירת מחדל הוא 5.
   - board_height מספר טבעי הגדול או שווה ל-board_width ומייצג את גובה הלוח, ערך ברירת מחדל הוא 5.
   - snakes_ladders מילון המכיל שני מפתחות: המפתח 'L' ממופה למילון כאשר המילון הפנימי ממפה כל עמדה מסוג 'L', סולם, לעמדה leap שלה. המפתח 'S' ממופה למילון כאשר המילון הפנימי ממפה כל עמדה מסוג 'S', נחש, לעמדה leap שלה. ערכו של המילון הממופה למפתחות "S" ו-"L" לא יכול להיות מילון ריק. בנוסף, על המילון הממופה למפתחות "S" ו-"L" למפות בצורה נכונה את עמדות הסולמות והנחשים לעמדות היעד המתאימות. כלומר משבצת סולם ומשבצת נחש אינן מפנות לאותה משבצת רגילה, אינן מפנות למשבצות שאינן מוגדרות כמשבצות רגילות ואינן מפנות לעמדה לא מתאימה (מבחינת הסדר).

   ערך ברירת המחדל של ארגומנט זה הוא: {'L': {3: 9, 5: 10}, 'S': {20: 4, 16: 2, 18: 10}}.

על הבנאי לאתחל את השדה grid -לוח המשחק- לפי הארגומנטים המתאימים. הבנאי מאתחל את לוח המשחק ומפזר בתוכו את המשבצות של נחשים וסולמות לפי הארגומנט snakes_ladders.
אם הארגומנטים שהבנאי מקבל (board_width, board_height, ו-snakes_ladders) אינם מהטיפוס המתאים או שערכם אינו תקין עליכם לזרוק את החריגה TypeError או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה.

```
def __init__(self, board_width=5, board_height=5, snakes_ladders= {'L':{3:9, 5:11, 6:24},'S':{20:4, 16:2, 18:10}}):
```

2) על המימוש שלכם למחלקה Board לתמוך באיטרציה בעזרת לולאת for. כלומר עליכם לממש את __iter__ ו-__next__ כך שיאפשרו מעבר על המשבצות של הלוח לפי הסדר של עמדותיהן (אין חשיבות לשדה leap בסעיף זה).
3) עליכם לממש את השיטה __len__ אשר תחזיר מספר טבעי המייצג את גודל הלוח.
5) ממשו את השיטה get_grid. שיטה זו אינה מקבלות קלט ומטרתה להחזיר את ערכי השדה הפרטי grid של מופע מהמחלקה **Board**. להלן החתימה של השיטה:

```
def get_grid(self):
```

אין צורך לדרוס את האופרטור __repr__. אתם תקבלו בקובץ Board.py את המימוש שלו אשר יחזיר מחרוזת המייצגת את הלוח.

דוגמאות הרצה: ראו את הקובץ Board.txt

סעיף ד' – המחלקה Dice:

השלימו את המימוש של המחלקה **Dice** המייצגת קובייה. למחלקה יש שיטה יחידה ואין שדות.

1) ממשו את השיטה roll השייכת למחלקה Dice. השיטה אינה מקבלת ארגומנטים והיא מדמה את זריקת קובייה הוגנת. השיטה מחזירה מספר טבעי בין 1 ל-6.

```
def roll(self):
```

דוגמאות הרצה: ראו את הקובץ Dice.txt

סעיף ה' – המחלקה Player:

השלימו את המימוש של המחלקה **Player** אשר מייצגת שחקן במשחק. למחלקה **Player** יש את 4 השדות הבאים:

- השדה name שם השחקן, מחרוזת לא ריקה המורכבת אך ורק מאותיות אלפביתיות.
- השדה position מופע של Cell, המייצג את המשבצת בה נמצא השחקן בכל רגע נתון. השדה מאותחל ב-None.
- השדה board מופע של Board המייצג את לוח המשחק.
- השדה num_turns המייצג את מספר הסיבובים שכל שחקן שיחק טרם סיום המשחק. השדה num_turns מאותחל בערך ההתחלתי 0.

1) ממשו את בנאי המחלקה אשר מקבל את הארגומנט name משתנה מטיפוס מחרוזת ו-board מופע של Board ומאתחל את השדות של המחלקה. אם הארגומנט name אינו מהטיפוס המתאים או שערכו אינו תקין עליכם לזרוק את החריגה TypeError או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה. אם הארגומנט board אינו מהטיפוס המתאים זרקו את החריגה TypeError, בצירוף הודעה מתאימה.

```
def __init__(self, name, board):
```

2) ממשו את השיטה move אשר מקבלת roll מספר טבעי הגדול -0 וקטן מ-7 ומייצג את תוצאת הטלת קובייה הוגנת. על השיטה לקדם את העמדה בה נמצא השחקן בהתאם ל-roll. השיטה מחזירה True אם השחק הגיע למשבצת האחרונה ו-False אחרת.

```
def move(self, roll):
```

אם הארגומנט אינו מהטיפוס המתאים או שערכו אינו תקין, זרקו את החריגה TypeError או ValueError, בהתאמה.

3) עליכם לדרוס את האופרטור __repr__. השיטה תחזיר ייצוג לשחקן לפי הפורמט הבא:

f"Player(name={name}, position={position})"

כאשר שחור מסמן את התווים הקבועים ואדום את ערכי השדות המתאימים.

דוגמאות הרצה: ראו את הקובץ Player.txt

סעיף ו' – המחלקה Game:

השלימו את המימוש של המחלקה **Game** אשר מייצגת סיבוב שלם של משחק בין מספר שחקנים. למחלקה **Game** יש את 4 השדות הבאים:

- השדה הפרטי dice, מופע של Dice המייצג את קוביית המשחק.
- השדה הפרטי players מטיפוס רשימה אשר איבריה הם מופעים של המחלקה Player, players תכיל את השחקנים שהולכים לשחק במשחק.
- השדה הפרטי board מופע של לוח משחק (מטיפוס Board). שדה זה מאותחל רק פעם יחידה בתחילת המשחק.
- השדה winner שמאותחל עם None ומתעדכן ברגע סיום המשחק. משחק מסתיים כאשר שחקן מקבל מספר בהטלת קובייה המאפשרת לו לנחות במשבצת האחרונה. מספר המאפשר "לדלג" על המשבצת האחרונה ייחשב שסיום המשחק גם כן.

1) ממשו את בנאי המחלקה אשר מקבל את מימדי הלוח (board_height, board_width) ו-snakes_ladders שנועדו לטובת אתחול הלוח במשחק. הארגומנט snakes_ladders מילון מקונן הממפה את המשבצות מסוג סולמות ונחשים לשדה leap שלהן, (כפי שהוסבר בסעיף ג' - המחלקה Board). בנוסף לארגומנטים אלה, הבנאי מקבל את game_players רשימת המכילה את שמות השחקנים (ערך ברירת המחדל הוא [“Omri”, “Tal”]). הבנאי מאתחל לוח חדש ומייצר שחקנים חדשים למשחק בהתאם לארגומנטים שאותם הוא מקבל.

על התוכנית שלכם להמשיך לפעול עם ערכי ברירת מחדל במקרה שאחד או יותר מהקלטים שהוזנו לבנאי אינם תקינים.

```
def __init__(self,  board_width, board_height, snakes_ladders, game_players=[“Omri”, “Tal”]):
```

2) ממשו את האופרטור __repr__ אשר מחזיר את ייצוג המשחק לפי הפורמט הבא:

```
f"Game(players={players},\nboard={board}\n****winner={winner if winner else 'game is still going on….'}****)"
```

כאשר שחור מסמן את התווים הקבועים ואדום את ערכי השדות המתאימים.

בשתי השיטות הבאות מומלץ מאוד להדפיס חיווי על התקדמות המשחק בין השחקנים בכל סבב של הטלת קובייה. לדוגמאות, ראו את דוגמאות הרצה המצורפות בקובץ Game.txt. הדפסות אלה אינן חובה אך מומלצים בחום.

3) ממשו את השיטה play_turn שמדמה סבב אחד של משחק בו כל שחקן מטיל את הקובייה פעם אחת. השיטה אינה מקבלת קלט אך מעדכנת את המשבצות בהן נמצא כל שחקן בהתאם לתוצאת הטלת הקובייה שלו. אם שחקן אחד מגיע לאותה משבצת שיש בה שחקן אחר, השחקן שהגיע למשבצת מוקדם יותר חוזר למשבצת הראשונה בלוח. אין לבצע סבב נוסף כאשר כבר הוכרז מנצח במשחק.

```
def play_turn(self):
```

4) ממשו את השיטה run_game המדמה משחק שלם. סיבוב שלם של משחק מסתיים כאשר אחד מהשחקנים מגיע למשבצת האחרונה בלוח. אם מדובר במשחק שכבר התחיל, על השיטה לדמות המשך של המשחק ועד לקבלת מנצח.

בסוף המשחק החזירו את השחק המנצח ואת מספר הסבבים שהוא היה צריך לשחק (כטאפל).

```
def run_game(self):
```

5) ממשו את השיטה solve אשר מחזירה רשימה המכילה את המשבצות שיש לעבור בהן על מנת לנצח במסלול הקצר ביותר בין המשבצת הראשונה למשבצת האחרונה (ראו דוגמה בהמשך).

```
def solve(self):
```

למשל עבור הלוח הבא:

36	35	34	33	32	31
25	26	27	28	29	30
24	23	22	21	20	19
13	14	15	16	17	18
12	11	10	9	8	7
1	2	3	4	5	6

בלוח זה רשימת המשבצות שיש לעבור בהן על מנת לנצח במסלול הקצר ביותר האפשרי הן:
[1,2,3,4,5,6,7,8,9,10,33,34,35,36].
שימו לב כי המסלול הקצר **אינו מתייחס למשבצות בהן השחקן נוחת בהכרח**. המסלול הקצר עבור הלוח בדוגמה מתקבל ע"י קבלת ההטלות הבאות: 6, 4, ואז 5. בהתאם להטלות אלה אין נחיתה ב-9 או 5 לעומת שכן אנו עוברים בהן במסלול.

דוגמאות הרצה למחלקה זו: ראו את הקובץ Game.txt

שאלה 2:

בשאלה זו תממשו מערכת לניהול משימות מבוססת על עקרונות תכנות מונחה-עצמים ושימוש במבני נתונים אבסטרקטיים. המערכת כוללת תעדוף משימות, שיוך כישורים הנדרשים למשימה, והתאמת עובדים למטלות על בסיס כישורים וזמינות. מערכת זו מתוארת באמצעות המחלקות הבאות: Task המייצגת משימה, Worker המייצגת עובד עם כישורים וזמינות ו-TaskScheduler המנהלת את המשימות והעובדים.

עליכם לממש את המערכת כך שהיא תומכת בהוספת משימות, עדכון משימות, ושיוך משימות לעובדים. על המערכת לשמור על סדר עדיפויות ברור עבור המשימות, תוך מתן דגש לפרמטרים הבאים: דחיפות (urgency), חשיבות (importance), וזמן חלון (time window). בנוסף, על המערכת לוודא כי הכישורים הנדרשים למשימה תואמים לכישורי וזמינות העובדים. למשל, משימה יכולה לדרוש "יכולת שימוש באקסל" עם ניסיון של 5 שנים ועל מנת לשייך אותה לעובד על העובד להיות עם הכישור "יכולת שימוש באקסל" באותה דרגת ניסיון או יותר.

סעיף א' – המחלקה Task:

השלימו את המימוש של המחלקה **Task** המייצגת משימה. למופע של המחלקה **Task** יש ששה שדות:

- task_id שדה פרטי המייצג את המזהה הייחודי לכל משימה, מספר טבעי.
- describtion תיאור המשימה, מחרוזת לא ריקה.
- urgency דירוג דחיפות המשימה מספר טבעי גדול מ-0.
- importance דירוג חשיבות המשימה מספר טבעי גדול מ-0.
- time_window חלון זמן לביצוע המשימה, טווח של זמנים (זמן התחלה וזמן סיום כמספרים שלמים), כטופל. חלון זמן של משימה חייבת להתחיל ולהסתיים בשעה עגולה. טווחי הזמן יכולים להיות בין השעות 0 ל-23 בלבד כאשר זמן ההתחלה תמיד קטן מזמן הסיום.
- needed_skills מילון לא ריק, מייצג את הכישורים הנדרשים לביצוע המשימה (מילון הממפה כל כישור, מחרוזת לא ריקה לרמת הניסיון הנדרשת ממנו לביצוע המשימה, מספר טבעי הגדול מ-0). למשל עבור משימה מסוימת נדרשים הכישורים הבאים: "יכולת תיאום פגישה" עם ניסיון של 5 ו-"יכולת שימוש באקסל" עם ניסיון של 6.

1) ממשו את בנאי המחלקה אשר מקבל את הארגומנטים: task_id, description, urgency, importance, time_window ו-needed_skills . על הבנאי לאתחל את השדות לפי הארגומנט המתאים.

אם הארגומנטים אותם מקבל הבנאי אינם מהטיפוס המתאים או שערכם אינו תקין, עליכם לזרוק את החריגה TypeError או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה.

```
def __init__(self, task_id, description, urgency, importance, time_window, needed_skills):
```

2) ממשו את השיטה get_task_id אשר מחזירה את המזהה הייחודי של המשימה:

```
def get_task_id(self):
```

3) דרסו את האופרטור __repr__ כך שיחזיר מחרוזת המייצגת את המופע בפורמט הבא:

f"Task ID: {task_id}, Description: {description}, Urgency: {urgency}, Importance: {importance},

Time Window: {time_window}, Needed skills: { needed_skills }"

כאשר שחור מסמן את התווים הקבועים ואדום את ערכי השדות המתאימים (אין צורך לרדת שורה- ראו דוגמאות הרצה).

דוגמאות הרצה: ראו את הקובץ task.txt

סעיף ב' – המחלקה Worker:

השלימו את המימוש של המחלקה **Worker** המייצגת עובד. למחלקה יש את השדות הבאים:

- worker_id שדה פרטי המייצג מזהה ייחודי לכל עובד, מספר טבעי.
- name שם העובד, מחרוזת לא ריקה.
- skills שדה פרטי, מילון לא ריק המייצג את הכישורים, שבו כל מפתח הוא שם משאב מטיפוס מחרוזת הממופה לרמת הכישורים באותו משאב (מספר טבעי גדול מ-0).
- availability שדה פרטי, רשימה של טווחי זמן (כל טווח הוא טופל של זמן התחלה וסיום כמספרים). טווחי הזמן יכולים להיות בין 0 ל-23 בלבד כאשר זמן ההתחלה תמיד קטן מזמן הסיום. הניחו כי אין חפיפה בין חלונות הזמן.
- salary שדה פרטי המייצג את משכורת העובד, מספר ממשי הגדול מ-0.

1) ממשו את בנאי המחלקה אשר מקבל את הארגומנטים name, skills, availability, ו-salary ומאתחל את השדות של המחלקה. אם אחד הארגומנטים name, skills, availability, או salary אינו מהטיפוס המתאים או שערכם אינו תקין, על הבנאי לזרוק את החריגות TypeError או ValueError בהתאמה.

```
def __init__(self, worker_id, name, skills, availability, salary):
```

2) דרסו את האופרטור __repr__ כך שיחזיר מחרוזת המייצגת את העובד לפי הפורמט הבא:

f"Worker ID: {worker_id}, Name: {name}, Skills: {skills}, Availability: {availability}, Salary: {salary}"

כאשר שחור מסמן את התווים הקבועים ואדום את ערכי השדות המתאימים (אין צורך בשורה חדשה).

3) ממשו את שיטה update_skills אשר מקבלת את הכישורים החדשים שיש להוסיף אותם לעובד. הארגומנט new_skills הוא רשימה המורכבת ממחרוזות לא ריקות כאשר כל מחרוזת היא כישור שיש להוסיף אותו לכישורי העובד. יש לאתחל כל כישור חדש המתווסף לכישורי עובד בניסיון של 1. כאשר אחד הכישרונות בקלט נמצא בכישרונות העובד יש להוסיף 1 לניסיון הנוכחי שלו בכישרון. אם הארגומנט new_skills אינו מהטיפוס המתאים או שערכיו אינם תקינים עליכם לזרוק את החריגה TypeError או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה.

```
def update_skills(self, new_skills):
```

4) ממשו את השיטה update_salary אשר מקבלת את תוספת השכר שיש להוסיף אותו לשכר הנוכחי של העובד. הארגומנט salary הוא מספר הגדול מ-0. אם הארגומנט additional_salary אינו מהטיפוס המתאים או שערכו לא תקין, עליכם לזרוק את החריגה TypeError או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה.

```
def update_salary(self, additional_salary):
```

5) ממשו את השיטה update_availability אשר מקבלת new_availability טאפל המייצג חלון זמן חדש שיש להוסיף או להרחיב בו אחד החלונות הקיימים של העובד. חלון זמן לביצוע המשימה, הוא טווח של זמנים (זמן התחלה וזמן סיום כמספרים שלמים). חלון זמן של משימה חייב להתחיל ולהסתיים בשעה עגולה. טווחי הזמן יכולים להיות בין השעות 0 ל-23 בלבד כאשר זמן ההתחלה תמיד קטן מזמן הסיום. אם חלון הזמן אינו בחפיפה עם חלון זמן הקיים אצל העובד, יש להוסיף אותו לרשימה. אם חלון הזמן כן נמצא בחפיפה אש יש להרחיב את החלון הקיים. למשל אם הארגומנט הוא (12,14) וחלון הזמן היחידי הקיים אצל העובד הוא (13,15), החלון הסופי אשר אמור להיות בשדה availability הוא (12,15). אם הארגומנט new_availability אינו מהטיפוס המתאים או שערכיו אינם תקינים, עליכם לזרוק את החריגה TypeError או את החריגה ValueError, בהתאמה, בצירוף הודעה מתאימה.

הניחו כי כל חלון זמן בארגומנט new_avialabilty יכול להיות בחפיפה עם חלון אחד לכל היותר בשדה availability.

```
def update_availability (self, new_availability):
```

6) ממשו get_salary, get_availability, get_worker_id, ו-get_skills. שיטות אלה אינן מקבלות קלט ומטרתן להחזיר את ערכי השדות הפרטיים של מופעי המחלקה **Worker**. להלן החתימות של השיטות.

```
def get_salary(self):
```

```
def get_availability(self):
```

```
def get_worker_id(self):
```

```
def get_skills(self):
```

<u>דוגמאות הרצה- ראו את הקובץ Worker.txt</u>

שימו לב כי בסעיף הבא **אין להשתמש במבני הנתונים המובנים בפייתון** (כמו רשימה, מילון וכדומה).

סעיף ג'- TaskScheduler:

השלימו את המימוש של מחלקת המערכת לניהול המשימות **TaskScheduler**. המחלקה **TaskScheduler** מיוצגת ע"י **השדות הפרטיים** הבאים:

- tasks, מבנה נתונים המאפשר אחסון המשימות וביצוען לפי הקריטריונים הבאים דחיפות> חשיבות> חלון זמן. בעת אתחול המערכת מבנה הנתונים מאותחל ריק.
- workers, מבנה נתונים המאפשר אחסון מופעי העובדים. בעת אתחול המערכת מבנה הנתונים מאותחל ריק. עובדים מאוחסנים לפי סדר קבלתם לעבודה – כלומר לפי סדר ההכנסה שלהם FIFO.
- completed_tasks, מבנה נתונים המאפשר אחסון של המשימות אשר בוצעו בעבר. בעת אתחול המערכת מבנה הנתונים מאותחל ריק.

1) ממשו את בנאי המחלקה אשר מאתחל את השדות של המחלקה.

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

2) ממשו את השיטה add_task המקבלת משימה חדשה ומוסיפה אותה למערכת תוך כדי שמירה על מבנה הנתונים tasks ממוין לפי הקריטריונים הבאים:
   - דחיפות – משימה דחופה יותר תוקדם למשימה דחופה פחות.
   - חשיבות – אם שתי משימות בעלות אותה דחיפות, המשימה החשובה יותר תוקדם למשימה חשובה פחות.
   - חלון זמן – אם הדחיפות והחשיבות שוות, יש למיין לפי נקודת ההתחלה של חלון הזמן (משימה עם נקודת התחלה מוקדמת תוקדם למשימה עם נקודת התחלה מאוחרת). במידה ונקודת ההתחלה זהה, יש למיין לפי נקודת הסיום.

לטובת שמירה על המיון של מבנה הנתונים tasks בכל רגע נתון, אתם רשאים להשתמש בזיכרון של O(n). אם הארגומנט task אינו מהטיפוס המתאים זרקו את החריגה TypeError בצירוף הודעה מתאימה.

```
def add_task(self, task):
```

שימו לב כי לבצע שינויים על השדות הפרטיים בשני הסעיפים הבאים (3 ו-4).

3) ממשו את השיטה completed_task_gen אשר מחזירה גנרטור. על הגנרטור לאפשר מעבר על המשימות שכבר בוצעו במערכת מהמשימה שבוצעה לאחרונה להכי ותיקה, תחת אילוץ זמן של O(1).

```
def completed_tasks_gen(self):
```

4) ממשו את השיטה workers_gen אשר מחזירה גנרטור. על הגנרטור לאפשר מעבר על העובדים במערכת לפי סדר קבלתם לעבודה מהכי ותיק להכי חדש, תחת אילוץ זמן של O(1).

```
def workers_gen(self):
```

5) ממשו את השיטה add_worker. השיטה מקבלת עובד מופע של המחלקה worker ומוסיפה אותו למבנה הנתונים אשר מאחסן את העובדים במערכת. עובד חדש מתווסף לסוף מבנה הנתונים בשדה workers. אם הארגומנט worker אינו מהטיפוס המתאים זרקו את החריגה TypeError בצירוף הודעה מתאימה.

```
def add_worker(self, worker):
```

6) ממשו את השיטה allocate_task אשר מנסה לשייך את המשימה בעלת העדיפות הגבוהה ביותר לעובד זמין עם הכישורים המתאימים שהמשימה דורשת. אם אין אפשרות לשייך את המשימה לעובד כזה במערכת, המשימה מוחזרת למבנה הנתונים tasks. בעת שמשימה תשויך לעובד, המשימה עוברת למבנה הנתונים המאחסן את המשימות לאחר ביצוען, כלומר לשדה completed_tasks. אם אין במבנה הנתונים משימות השיטה לא מבצעת כלום.

```
def allocate_task(self):
```

7) ממשו את השיטה update_task אשר מעדכנת משימה קיימת לפי המזהה הייחודי שלה. את העדכון יש לבצע לפי המילון **kwargs אשר מכיל את השדות של המופע task שיש לעדכן כאשר הם ממופים לערכיהם החדשים. קראו על זה פה .

אם הארגומנט task_id אינו מהטיפוס המתאים, או שאינו קיים במערכת זרקו את החריגה TypeError או ValueError, בהתאמה, בצירוף הודעה מתאימה.

```
def update_task(self, task_id, **kwargs):
```

8) ממשו את השיטה undo_task אשר מאפשרת להחזיר את המשימה האחרונה שבוצעה למבנה הנתונים המאוחסן בשדה tasks בזמן של O(1). אם אין משימות כאלה השיטה מדפיסה את ההודעה הבאה:

"No tasks to undo."

```
def undo_task(self):
```

9) ממשו את השיטה promote_senior אשר מקדמת את העובד הוותיק ביותר למשרה גבוהה יותר. קידום של עובד למשרה גבוהה מתבטא בעלייה בשכר שלו בסכום של 1000₪. בנוסף לכך, העובד מקבל חוזה חדש לעבודה. כאשר עובד מקבל חוזה חדש לעבודה הוא הופך להיות עובד "חדש".

```
def promote_senior(self):
```

10) ממשו את השיטה yearly_update אשר מעדכנת את הניסיון של כל עובד בכל כישרונותיו ב-1. על השיטה לשמור על הסדר של העובדים בו הם התקבלו לעבודה במבנה הנתונים של המערכת. לסעיף זה עומד לרשותכם זיכרון נוסף של O(1).

```
def yearly_update(self):
```

11) ממשו את השיטה peek_task אשר מחזירה את המשימה בעלת העדיפות הגבוהה ביותר מבלי להסיר אותה ממבנה הנתונים של המערכת. אם אין משימה כזו השיטה מדפיסה את ההודעה הבאה:

"No tasks in waiting tasks."

```
def peek_task(self):
```

12) ממשו את השיטה get_tasks. שיטה זו אינה מקבלת קלט ומטרתה להחזיר את ערך השדה tasks בו יש את מבנה הנתונים המאחסן את המשימות שעוד לא שויכו לעובד במערכת.

```
def get_tasks(self):
```

דוגמאות הרצה: ראו את הקובץ TaskScheduler.txt

שימו לב כי סעיף זה אינו קשור לסעיפים הקודמים.

סעיף ד:

ממשו **פונקציה** המקבלת n מספר טבעי (גדול מ-0) המייצג כמות של המופעים שיש לייצר עבורם מזהה אוטומטי (מ-0 עד n לא כולל n). הפונקציה מחזירה גנרטור אשר יאפשר מעבר על ערכי המזהה האוטומטי.

```
def id_generator(n):
```