ת.ז.: _____ מס' מחברת: _____

מבחן בקורס CS 1001.py

סמסטר א' התשע"ב, מועד א'

תאריך: 2.3.2012

מרצה: פרופ' בני שור

מתרגל: רני הוד

מומלץ לקרוא את כל ההנחיות והשאלות בתחילת המבחן, לפני תחילת כתיבת התשובות.

- משך הבחינה שלוש שעות.
- חומר עזר מותר: שני דפי A4, כתובים משני הצדדים.
- במבחן חמש שאלות שלחלקן סעיפי משנה. כדי לקבל ציון 100 בבחינה יש לענות נכונה על כל השאלות. ניקוד כל סעיף מצוין לידו. אין בהכרח קשר בין ניקוד הסעיף ובין קושיו.
  - את התשובות לשאלות 1 ו-2 יש לסמן בטופס המבחן. לכל סעיף יש להקיף בעיגול אות בודדת (א', ב', ג' או ד'). בשאלה 1 יש להוסיף הסבר קצר במסגרת המצורפת ובשאלה 2 אין להוסיף הסבר כלל.
  - את התשובות לשאלות 3, 4 ו-5 יש לרשום במחברת, כל אחת בדף נפרד. יש לענות תשובות ברורות ותמציתיות. תשובות מסורבלות או לא ניתנות לקריאה ע"י הבודקים יזכו לניקוד חלקי בלבד.
- על כל סעיף מתוך השאלות 3, 4 או 5 ניתן לענות *"אינני יודע/ת"* כתשובה; על סעיף זה יינתנו 20% מהנקודות. במקרה זה אין להוסיף שום הסבר.
- בפתרון סעיף בשאלה מותר להשתמש בתוצאות הסעיפים הקודמים, גם אם לא פתרתם אותם.

בהצלחה!

לשימוש משרדי בלבד			
שאלה 1			
שאלה 2			
שאלה 3			
שאלה 4			
שאלה 5			

שאלה 1 (10 נק' לכל סעיף; סה"כ 30 נק')

עבור מספר שלם $n \geq 2$ נגדיר n-מבוך כמטריצה (אובייקט Matrix) $n \times n$ שכל תא בה הוא ריק (ערכו 0) או מלא (ערכו 1).
מסלול פנוי במבוך מתחיל בתא (0,0) – הפינה השמאלית העליונה – ובכל צעד מתקדם לתא ריק סמוך. מותר לזוז ימינה, שמאלה, למעלה או למטה, אך אסור לזוז באלכסון או לצאת מגבולות המבוך. ניתן להניח שהתא (0,0) תמיד ריק.
התא (i,j) נקרא *נגיש* אם יש מסלול פנוי שמסתיים בו; המבוך נקרא *פתיר* אם הפינה הימנית התחתונה (n-1,n-1) נגישה.

דוגמא: המבוך $\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$ הוא פתיר ולעומתו המבוך $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ הוא בלתי־פתיר.

להלן "תכנית מסטר" קצרה לפתרון הבעיה *'בהנתן מבוך, האם הוא פתיר?'* ושלוש הצעות למימוש הפונקציה reachable.

```
def solvable(maze):
    n,n = maze.dim()
    return reachable(maze, n-1, n-1)
```

סמנו את התשובה המתאימה ביותר לכל אחת משלוש ההצעות, והוסיפו הסבר מילולי באורך שתי שורות לכל היותר. אם טענתכם מסתמכת על דוגמא או דוגמאות קלט ספציפיות, רישמו אותה/ן במקום המתאים. הסתפקו בדוגמאות עבורן $2 \leq n \leq 5$.

(א) התכנית לעולם לא עוצרת.

(ב) יש קלטים עבורם התכנית עוצרת ויש קלטים עבורם התכנית לא עוצרת.

(ג) התכנית תמיד עוצרת; יש קלטים עבורם התכנית מחזירה תשובה נכונה ויש קלטים עבורם התשובה שגויה.

(ד) התכנית תמיד עוצרת ומחזירה תשובה נכונה.

.1

```
EMPTY, FULL, REACHABLE = 0, 1, 2
def reachable1(maze, i, j):
    maze[0,0] = REACHABLE                    # by assumption
    n,n = maze.dim()
    for steps in range(n*n):
        for a in range(n):
            for b in range(n):
                if maze[i,j] == FULL: continue # not reachable anyway
                right = b < n-1 and maze[a,b+1] == REACHABLE
                left  = b > 0   and maze[a,b-1] == REACHABLE
                down  = a < n-1 and maze[a+1,b] == REACHABLE
                up    = a > 0   and maze[a-1,b] == REACHABLE
                if right or left or down or up:
                    maze[a,b] = REACHABLE
    return maze[i,j] == REACHABLE
```

התשובה היא א / ב / ג / ד כי _____

$\begin{bmatrix} 0 & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix}$ $\begin{bmatrix} 0 & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix}$

.2

```
EMPTY, FULL, REACHABLE = 0, 1, 2
def reachable2(maze, i, j):
    maze[0,0] = REACHABLE                          # by assumption
    n,n = maze.dim()
    for steps in range(2*n):
        for a in range(n):
            for b in range(n):
                if maze[a,b] == FULL: continue # not reachable anyway
                right = b < n-1 and maze[a,b+1] == REACHABLE
                left  = b > 0   and maze[a,b-1] == REACHABLE
                down  = a < n-1 and maze[a+1,b] == REACHABLE
                up    = a > 0   and maze[a-1,b] == REACHABLE
                if right or left or down or up:
                    maze[a,b] = REACHABLE
    return maze[i,j] == REACHABLE
```

התשובה היא א / ב / ג / ד כי _____

$$\begin{bmatrix} 0 & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix} \qquad \begin{bmatrix} 0 & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix}$$

.3

```
EMPTY, FULL = 0,1
def reachable3(maze, i, j):
    if i==j==0:
        return True       # by assumption
    if maze[i,j] == FULL:
        return False      # not reachable anyway
    n,n = maze.dim()
    if   j < n-1 and reachable3(maze, i, j+1): return True # right
    elif j > 0   and reachable3(maze, i, j-1): return True # left
    elif i < n-1 and reachable3(maze, i+1, j): return True # down
    elif i > 0   and reachable3(maze, i-1, j): return True # up
    else:
        return False
```

התשובה היא א / ב / ג / ד כי _____

$$\begin{bmatrix} 0 & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix} \qquad \begin{bmatrix} 0 & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ \end{bmatrix}$$

שאלה 2 (10 נק')

להלן פונקציית קידוד עבור קוד חדש בשם Silly code.

```
def silly_encode(x,y):
    z = (x+y)%2       # parity
    return (x,y,z)*3  # 3-fold repetition
```

<u>תזכורת</u>: קוד $E : \{0,1\}^k \to \{0,1\}^n$ עם מרחק מינימלי d נקרא קוד מטיפוס $[n,k,d]$.

השלימו את המשפט הבא:

- Silly code הוא קוד מטיפוס $[n = \_\_\_\_, k = \_\_\_\_, d = \_\_\_\_]$.

שאלה 3 (10 נק' לכל סעיף; סה"כ 20 נק')

תמורה באורך n היא איברי range(n) מסודרים בסדר כלשהו. להלן מימוש בסיסי של מחלקת תמורות בפייתון.

```
class Permutation:
    def __init__(self, perm):
        assert isinstance(perm, list)
        n = len(perm)
        assert sorted(perm) == list(range(n))
        self._perm = perm[:]
    def __getitem__(self, i):
        return self._perm[i]
    def __len__(self):
        return len(self._perm)
    def __repr__(self):
        return "Permutation({})".format(self._perm)
```

1. התמורות σ, τ נקראות *הופכיות* אם מתקיים $i = \sigma[\tau[i]]$ לכל i מתוך range(n).

<u>עובדה</u>: לכל תמורה יש בדיוק תמורה הופכית אחת.

כיתבו מתודה בשם inverse שמחזירה את התמורה ההופכית ל-self.

הסבירו במילים, ממשו בקוד פייתון וציינו מהי סיבוכיות זמן הריצה (בסימון $O(\cdots)$, כפונקציה של אורך התמורה n).

<u>דוגמא</u>:

```
>>> p = Permutation([4, 2, 0, 6, 1, 5, 3])
>>> p.inverse()
Permutation([2, 4, 1, 6, 0, 5, 3])
```

2. *הסדר* של איבר i ביחס לתמורה π הוא ה-k החיובי המינימלי המקיים $\pi^k[i] = \overbrace{\pi[\pi[\cdots[\pi}^{k}[i]]\cdots]] = i$.

<u>עובדה</u>: הסדר של איבר הוא תמיד בין 1 ל-n.

כיתבו מתודה יעילה ככל הניתן בשם order שמקבלת כקלט את התמורה ומספר i מתוך range(n), ומחזירה את הסדר של i ביחס לתמורה self.

הסבירו במילים, ממשו בקוד פייתון וציינו מהי סיבוכיות זמן הריצה (בסימון $O(\cdots)$, כפונקציה של אורך התמורה n).

<u>דוגמא</u>:

```
>>> p = Permutation([4, 2, 0, 6, 1, 5, 3])
>>> [p.order(i) for i in range(7)]
[4, 4, 4, 2, 4, 1, 2]
```

שאלה 4 (20 נק')

רשימה של מספרים נקראת *עולה-יורדת* אם היא מהווה שרשור של רשימה ממוינת בסדר עולה ממש עם רשימה ממוינת בסדר יורד ממש.

כיתבו פונקציה יעילה ככל הניתן בשם `max_updown` שמקבלת כקלט רשימת מספרים ממשיים שונים `L` שהיא עולה-יורדת, ומחזירה את האיבר המקסימלי בה.

הסבירו ראשית את האלגוריתם במילים, כיתבו אותו בקוד פייתון ונתחו את זמן הריצה במקרה הגרוע ביותר (בסימון $O(\cdots)$, כפונקציה של אורך הרשימה n).

דוגמא:

```
>>> L = [12, 21, 36, 43, 52, 65, 75, 88, 83, 47, 25, 6]
>>> max_updown(L)
88
```

שאלה 5 (10 נק' לכל סעיף; סה"כ 20 נק')

בכל אחד מהסעיפים שלפניכם עליכם לכתוב פונקציה שמקבלת סדרת קלט אינסופית ומחזירה סדרת פלט אינסופית. סדרות אלו מיוצגות ע"י איטרטורים ועל כן הפונקציה שאתם כותבים היא גנרטור. מותר להשתמש בכמות קבועה של זכרון נוסף; בפרט, אי אפשר לשמור את כל הקלט שראינו עד כה.[1]

הסבירו בקצרה את דרך הפעולה של הפונקציות שכתבתם.

1. האיבר ה-k בסדרת הפלט הוא ממוצע חשבוני של האיברים ה-$k, k+1, k+2, k+3$ בסדרת הקלט. קיראו לפונקציה בשם `average_four`.
2. האיבר ה-k בסדרת הפלט הוא ממוצע חשבוני של $k+1$ האיברים הראשונים בסדרת הקלט. קיראו לפונקציה בשם `average_prefix`.

דוגמאות:

```
>>> L = [1, 5, 3, -12, 99, 0, -5, ...]
>>> list(average_four(iter(L)))
[-0.75, 23.75, 22.5, 20.5]                # e.g., 23.75 = (5+3-12+99)/4
>>> list(average_prefix(iter(L)))
[1, 3, 3, -0.75, 19.2, 16, 13, ...]       # e.g., 19.2 = (1+5+3-12+99)/5
```

[1] פתרון שמשתמש בכמות לא קבועה של זכרון יזכה לניקוד חלקי.