ת.ז.: _____ מס' מחברת: _____

מבחן בקורס CS 1001.py

סמסטר א' התשע"ב, מועד ב'

תאריך: 30.3.2012

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. בשאלה 1 יש למלא את שמות התמונות ובשאלה 2 יש להקיף בעיגול אות בודדת (א', ב', ג' או ד'). 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!

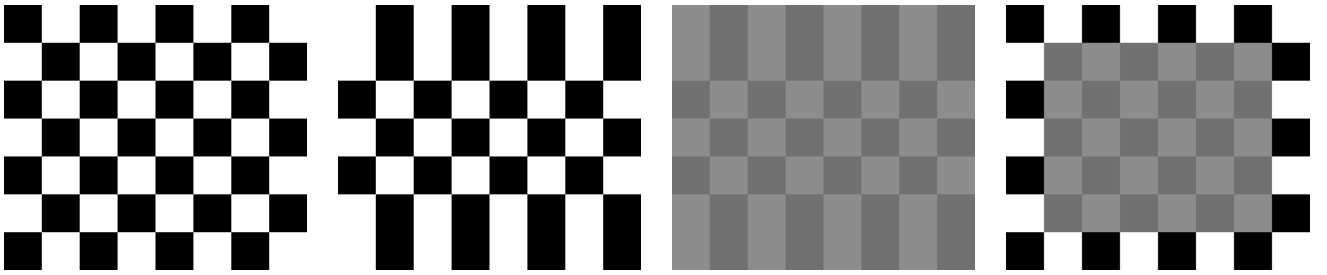
לשימוש משרדי בלבד			
שאלה 1			
שאלה 2			
שאלה 3			
שאלה 4			
שאלה 5			

שאלה 1 (10 נק')

להלן קטע קוד המגדיר פונקציות לתיקון לוקאלי של תמונות ומפעילן על תמונה אחת.

```
def mean(W):
  return sum(W)//len(W)
def median(W):
  return sorted(W)[len(W)//2]

def local_correct(M, window_func):
  n,m = M.dim()
  result=M[:,:]
  for i in range(1,n-1):
    for j in range(1,m-1):
      W = sum(M[i-1:i+2,j-1:j+2].rows,[]) # get 3×3 window as a 9-element list
      result[i,j] = window_func(W)        # apply window function
  return result

def wraparound(M, window_func):
  n,m = M.dim()
  temp = Matrix(2*n,2*m)
  temp[:n,:m] = temp[n:,:m] = temp[:n,m:] = temp[n:,m:] = M
  L = local_correct(temp, window_func) # locally correct a ×2 blown-up matrix
  result = L[:n,:m]                    # copy wrapped-around values to place
  result[:1,:] = L[n:n+1,:m]
  result[:,:1] = L[:n,m:m+1]
  result[0,0]  = L[n,m]
  return result

original_img = Matrix(7,8)
for i in range(7):
  for j in range(8):
    if (i+j)%2:
      original_img[i,j] = 255

img1 = local_correct(original_img, mean)
img2 = local_correct(original_img, median)
img3 = wraparound(original_img, mean)
img4 = wraparound(original_img, median)
```

לפניכם התמונות `img1,img2,img3,img4` בסדר כלשהו. רישמו מתחת לכל תמונה את שמה.



שאלה 2 (10 נק')

נתונה מטריצת תמונה `M` בגודל 1000×1000 פיקסלים.[1] נייצג אותה בשתי דרכים – שרשור השורות `R` ושרשור העמודות `C`:

```
R = [M[0,0], M[0,1], ..., M[0,999], M[1,0], M[1,1], ..., M[999,998], M[999,999]]
C = [M[0,0], M[1,0], ..., M[999,0], M[0,1], M[1,1], ..., M[998,999], M[999,999]]
```

הן `R` והן `C` ניתנים לייצוג בצורה נאיבית ע"י בדיוק $N = 8 \cdot 1000^2$ ביטים; נסמן ב-NR את מספר הביטים הדרוש לייצוג `R` כשמשתמשים בדחיסת Lempel–Ziv וב-NC את מספר הביטים הדרוש לייצוג `C` כשמשתמשים בדחיסת Lempel–Ziv.

בחרו את האפשרות הנכונה.

(א) לכל תמונה מתקיים $NC = NR < N$.

(ב) לכל תמונה מתקיים $\frac{1}{2}(NC + NR) < N$ אך יש תמונות שעבורן $NC < NR$ ויש כאלו שעבורן $NR < NC$.

(ג) לכל תמונה מתקיים $NC < N < NR$ או $NR < N < NC$.

(ד) תשובות א'-ג' אינן נכונות.

שאלה 3 (20 נק')

רשימה של מספרים נקראת *כמעט ממוינת* אם היא נוצרה ע"י הוספה של איבר לרשימה ממוינת בסדר עולה, במקום שאינו המקום הנכון ע"פ סדר המיון. האיבר שאינו במקומו נקרא *חריג*.

כיתבו פונקציה יעילה ככל הניתן בשם `search_almost_sorted` שמקבלת כקלט רשימת מספרים ממשיים שונים `L` כמעט ממוינת ומספר נוסף `x`, ומחזירה `True` אם `x` איבר של `L` ו-`False` אם לאו. נתון ש-`x` עצמו אינו חריג וש-`L` אינה ריקה.

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

<u>דוגמא:</u>

```
>>> L = [12, 21, 36, 43, 52, 3, 65, 75, 88, 91, 100]
>>> search_almost_sorted(L, 21)
True
>>> search_almost_sorted(L, 77)
False
```

[1] כרגיל, כל פיקסל הוא מספר שלם בין 0 ל-255.

שאלה 4 (10 נק' לכל סעיף; סה"כ 30 נק')

יהופץ וצפניה משחקים *אבן נייר ומספריים*. סיבוב אחד של המשחק נראה כך: שני המשתתפים שולפים בו־זמנית סימן מתוך הקבוצה $\{R, P, S\}$; אם שני הסימנים שווים, הסיבוב נגמר בתיקו ואף אחד לא מקבל נקודה; אחרת, המנצח בסיבוב נקבע ע"פ היחסים $P < S, S < R, R < P$ והוא מקבל נקודה.

1. סדרת הסימנים שישלוף כל משתתף מיוצרת ע"י איטרטור. כיתבו פונקציה בשם play_rps שמקבלת כקלט שני איטרטורים, אחד עבור יהופץ ואחד עבור צפניה, ומחזירה את הניקוד בסוף המשחק כזוג סדור. ניתן להניח שהאיטרטורים מייצגים סדרות באורך זהה.

   <u>דוגמא:</u>

```
>>> R,P,S = "rock","paper","scissors"
>>> Y = iter([P, P, R, R, S, S, P, P, R, S, S])
>>> Z = iter([R, P, S, P, R, S, P, S, R, S, R])
>>> play_rps(Y, Z)
(2, 4)
```

2. לאחר מספר סיבובי משחק שם לב יהופץ שבשני סיבובים עוקבים צפניה לעולם אינו מוציא את אותו הסימן. הוא חקר, בדק וגילה שבמקום להתייצב למשחק בעצמו, שלח צפניה רובוט בדמותו שמשחק לפי התכנית הבאה:

```
def robo_tzfania_player():
  x = random.choice({R, P, S}) # select each symbol with equal probability
  while True:
    yield x
    x = random.choice({R, P, S} - {x})
```

   הסבירו כיצד יהופץ יכול לנצל את המידע לטובתו: כיתבו פונקציה בשם yehopatz_player שמקבלת כקלט רשימה ובה מהלך המשחק עד כה (כלומר, האיבר ה-k ברשימה הוא זוג הסימנים שנשלפו בסיבוב ה-k), ומחזירה את הסימן הבא שישלוף יהופץ. מטרתו של יהופץ היא לנצח כמה שיותר ולהפסיד כמה שפחות. הפונקציה יכולה כמובן להסתייע בבחירה אקראית. הסבירו ראשית את פעולת הפונקציה במילים וממשו בקוד פייתון.

3. נתחו משחק אבן נייר ומספריים בן 21 סיבובים בין yehopatz_player שכתבתם בסעיף הקודם לרובו־צפניה. מהי התוצאה הטובה ביותר ליהופץ? מהי התוצאה הגרועה ביותר ליהופץ? מהי התוצאה האופיינית?

שאלה 5 (10 נק' לכל סעיף; סה"כ 30 נק')

פולינום (מעל $\mathbb{R}$) ממעלה n הוא פונקציה המאופיינת ע"י $n+1$ פרמטרים $a_0, a_1, \ldots, a_n$, שנקראים מקדמי הפולינום.[2] הפולינום ממפה את x ל-$a_0 + a_1 x + \cdots + a_n x^n$. להלן מימוש בסיסי של מחלקת פולינומים בפייתון.

```
class Polynomial:
  def __init__(self, coeffs):
    assert isinstance(coeffs, list)
    assert (not coeffs) or coeffs[-1]!=0 # top coefficient must be non-zero
    self._coeffs = coeffs[:]
  def degree(self):
    return len(self._coeffs)-1
  def __repr__(self):
    return "Polynomial({})".format(self._coeffs)
```

1. להלן שני מימושים של המתודה `eval`, שמחשבת את ערך הפולינום `self` בנקודה נתונה `x`. אחד מהם נכון ואחד שגוי. תקנו את השגיאה, הסבירו כל אחד מהמימושים (אחרי התיקון, כמובן) וקיבעו מי מהם יעיל יותר.

```
class Polynomial:
  ...
  def eval1(self, x):
    result = 0
    for c in self._coeffs:
      result = result*c+x
    return result

  def eval2(self, x):
    return sum(c*x**i for i,c in enumerate(self._coeffs))
```

2. כיתבו מתודה בשם `__add__` שמחזירה את סכום הפולינומים `self` ו-`other`, כאובייקט מסוג `Polynomial`. אפשר להניח ש-`other` הוא מסוג `Polynomial`.

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

```
>>> f = Polynomial([4, -5, 1])          # x^2 - 5x + 4
>>> g = Polynomial([3, 0, -2, 2])       # 2x^3 - 2x^2 + 3
>>> f+g
Polynomial([7, -5, -1, 2])              # 2x^3 - x^2 - 5x + 7
```

3. כיתבו מתודה בשם `derivative` שמחזירה את הנגזרת של הפולינום `self`, כאובייקט מסוג `Polynomial`.

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

```
>>> f = Polynomial([4, -5, 1])          # x^2 - 5x + 4
>>> f.derivative()
Polynomial([2, -5])                     # 2x - 5
```

[2] כדי לוודא שהמעלה מוגדרת היטב, נניח $a_n \neq 0$, אלא אם הפולינום הוא זהותית אפס, כלומר מיוצג בפייתון כ-`Polynomial([])`. במקרה כזה אנחנו נאמר שמעלתו היא -1, למרות שמתמטיקאים אמיתיים היו אומרים שמעלתו היא $-\infty$.