פתרון הבחינה במבוא מורחב למדעי המחשב בשפת פייתון

0368.1105.07

סמסטר ב', מועד א' ,תשע"ב

11/07/2012

אוהד ברזילי, אמיר רובינשטיין

הוראות (נא לקרוא!)

- משך הבחינה **שלוש שעות**, חלקו את זמנכם ביעילות.
- בבחינה 14 עמודים והיא כוללת 5 שאלות (מרובות סעיפים)
- חומר עזר מותר: 2 דפי A4 כתובים משני צידיהם.
- מחברת הבחינה נועדה לרישומי טיוטה. תשובות במחברת הבחינה לא תיבדקנה. יש לענות על כל השאלות בטופס זה במקום המיועד לכך. המקום המיועד מספיק לתשובות מלאות, ויותר, ואולם במידת הצורך ניתן לכתוב בגב טופס הבחינה. יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תיפסל.
- יש למלא מספר סידורי (מס' מחברת) ומספר ת.ז על כל דף של טופס הבחינה.

- ניתן להניח לאורך הבחינה שכל החבילות הדרושות יובאו, ואין צורך לכתוב שורות `import`.
- במקומות בהם תתבקשו לכתוב מתודה או פונקציה, ניתן לכתוב גם פונקציות עזר, אלא אם צוין במפורש אחרת.

לשימוש הבודקים:

שאלה	א	ב	ג	ד	ה	סה"כ
1						
2						
3						
4						
5						

מס' סידורי:_____ ת.ז.:_____

שאלה 1 (25 נקודות)

להלן חלק מהגדרת המחלקה **Point** שראינו בכיתה, המייצגת נקודה במישור:

```
class Point:
  """ Represents a point (x, y) in the plane. """

  def __init__(self, x=0, y=0):
    self.x = x
    self.y = y

  def __eq__(self, other):
    return isinstance(other, Point) and \
       self.x == other.x and self.y == other.y
```

א. (5 נק') ממשו את המתודה **distance(self, other)** של המחלקה **Point**, המחזירה את המרחק בין הנקודה **self** לנקודה **other**.

```
def distance(self, other):

        return math.sqrt((self.x-other.x)**2+(self.y-other.y)**2)
```

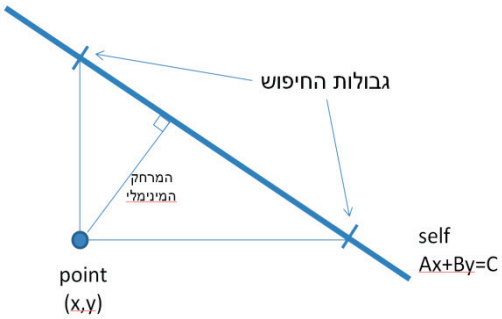
מס' סידורי:_____ ת.ז.:_____

להלן חלק מהגדרת המחלקה **Line** שראינו בכיתה, המייצגת ישר במישור, אשר הוספנו לה שתי מתודות חדשות (**intersect_x** ו- **intersect_y**):

```
class Line:
    """ Represents a straight line in the plane, using the equation
         ax + by = c """
    def __init__(self,a=0,b=0,c=0):
        assert a!=0 or b!=0
        self.a=a
        self.b=b
        self.c=c

    def __eq__(self,other):
        assert isinstance(self,Line) and isinstance(other,Line)
        return self.a*other.b == other.a*self.b and \
               self.b*other.c == other.b*self.c and \
               self.a*other.c == other.a*self.c

    def is_horizontal(self):
        return self.a==0

    def is_vertical(self):
        return self.b==0

    # returns the point on the line self which has X value of x
    def intersect_x(self, x):
        return Point(x, (self.c - self.a * x) / self.b)

    # returns the point on the line self which has Y value of y
    def intersect_y(self, y):
        return Point((self.c - self.b * y) / self.a ,y)
```

ב. (15 נקודות) עליכם לממש את המתודה **`distance(self, point, epsilon=10**(-6))`** של המחלקה **`Line`**, המחזירה את המרחק בין הישר **`self`** לנקודה **`point`**, בעזרת קרוב איטרטיבי נומרי.

<u>תזכורת מתמטית</u>: המרחק בין נקודה לישר הוא המינימום מבין כל המרחקים בין הנקודה ובין הנקודות שנמצאות על הישר.

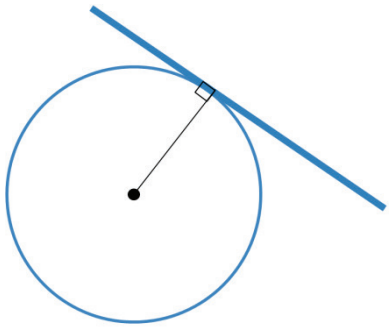


<u>הדרכה</u>: אם הישר הנתון מקביל לאחד הצירים יש לחשב את המרחק באופן אנליטי (חישוב ישיר). אחרת, יש לחפש נקודה על הישר שעליה שהמרחק בינה ובין הנקודה **`point`** הוא המינימלי. גבולות החיפוש יקבעו לפי היטלי ה-x וה-y של הנקודה על הישר (ראו איור). עליכם לדגום נקודות על הישר באופן יעיל, עד שתגיעו לנקודה שהמרחק בינה ובין הנקודה הנתונה **`point`** מינימלי עד כדי **`epsilon`**.

```
def distance(self, point, epsilon=10**(-6)):
    if self.is_horizontal():
        return abs(self.c/self.b - point.y)

    if self.is_vertical():
        return abs(self.c/self.a - point.x)

    left_point = self.intersect_x(point.x)
    right_point = self.intersect_y(point.y)

    distance_left = point.distance(left_point)
    distance_right = point.distance(right_point)

    while(abs(distance_right - distance_left) > epsilon):
        middle_point = self.intersect_x((left_point.x + right_point.x)/2)
        if distance_left > distance_right:
            left_point = middle_point
            distance_left = point.distance(left_point)
        else:
            right_point = middle_point
            distance_right = point.distance(right_point)

    middle_point = self.intersect_x((left_point.x + right_point.x)/2)
    return point.distance(middle_point)
```

להלן חלק מהגדרת המחלקה **Circle** שראינו בכיתה, המייצגת מעגל במישור:

```
class Circle:
    """ Represents a circle with a center and a radius """
    def __init__(self, center=Point(0,0), radius=0):
        assert isinstance(center,Point) and isinstance(radius, (float,int))
        assert radius >= 0
        self.center = center
        self.radius = radius

    def area(self):
        return math.pi*self.radius**2
```



ג. (5 נקודות) ממשו את המתודה הבולאנית **isTangent(self, line, epsilon=10**(-6))** המחזירה האם הישר הנתון, **line**, משיק למעגל **self**. אם המימוש כולל השוואת מספרים בנקודה צפה, ניתן להניח כי מספרים אשר ההפרש בינהם קטן מ-**epsilon** הם שווים.

<u>תזכורת מתמטית</u>: משיק למעגל מאונך לרדיוס המעגל בנקודת ההשקה.

```
def isTangent(self, line, epsilon=10**(-6)):
    return abs(line.distance(self.center)-self.radius) < epsilon
```

שאלה 2 (40 נקודות)

בשאלה זו נעסוק בזרמים (streams) של תווים (characters), ונייצגם באופן דחוס.

רצף של תווים זהים נקרא RUN.

Run Length Encoding (RLE) הוא מנגנון דחיסה פשוט שבו RUN מיוצג ע"י זוג (tuple) הכולל את התו ואת מספר הפעמים שהוא מופיע ברצף. ניתן לייצג זרם של תווים ע"י זרם של זוגות כאלו, כאשר כל זוג מייצג RUN. נקרא לזרם כזה ייצוג ה- RLE של הזרם המקורי.

דוגמה: הזרם: 'a', 'a', 'a', 'a', 'b', 'b', 'a', 'b', 'b'...
מיוצג ב- RLE ע"י זרם זוגות: ('a', 4) ('b', 2) ('a', 1) ('b', 2)...

שימו לב שכל RUN מייצג את האורך של הרצף המקסימאלי של אותו תו, כלומר, למשל, הזרם [a a] מיוצג ע"י [(a 2)] ולא על ידי [(a 1) (a 1)]. במקרה של זרמים אינסופים אפשר להניח שאין בהם RUN אינסופי.

ניתן להיעזר בפונקציות הבאות:

```
def make_run (char, length):
    return (char, length)

def run_val (r) :
    return r[0]

def run_len (r):
    return r[1]
```

הערה: אנו משתמשים בדוגמאות בפונקציה בשם **display_stream_head**. פונקציה זו מקבלת זרם (סופי או אינסופי) ומספר **n** ומדפיסה את **n** האיברים הראשונים בזרם (אם יש **n** או יותר אברים, ואחרת את כל איברי הזרם).

א. (10 נקודות) ממשו את הפונקציה **RLE_encode**, אשר מקבלת כקלט זרם של תווים (סופי או אינסופי), ומחזירה את הזרם שמהווה את ייצוג ה- RLE שלו.

דוגמה:

```
>>> w = RLE_encode(iter(['a','a','a','a','b','b','a','b','b']))
>>> display_stream_head(w, 4)
('a', 4) ('b', 2) ('a', 1) ('b', 2)
```

מס' סידורי:_____ ת.ז.:_____

```
def RLE_encode(strm):

    current_char = next(s)

    try:
        counter = 1
        while True :
            next_char = next(s)
            if current_char == next_char:
                counter += 1
            else :
                yield (current_char, counter)
                counter = 1
                current_char = next_char

    except StopIteration:
        yield (current_char, counter)
```

ב. (10 נקודות) ממשו את הפונקציה **remove_stream_elem** המקבלת זרם RLE (סופי או אינסופי) ומספר **k**, ומחזירה זרם RLE המבוסס על זרם התווים המקורי שממנו הוסרו **k** התווים הראשונים.

שימו לב שאנו מתייחסים להסרת k איברים מהזרם המקורי (שממנו יוצר זרם ה RLE שהועבר כקלט), ולא מזרם ה- RLE שהתקבל כקלט.

דוגמה:

```
>>> w = RLE_encode (iter([ 'a','a','a','a','b','b','a','b','b']))
>>> display_stream_head (remove_stream_elem (w, 5), 3)
('b', 1) ('a', 1) ('b', 2)
```

כלומר, הפונקציה **remove_stream_elem** מקבלת כקלט את הזרם **w** (RLE Stream): **[(a 4) (b 2) (a 1) (b 2)]**, המייצג את הזרם: **[a a a a b b a b b]**. מזרם זה מסירים את 5 האיברים ראשונים, כלומר מקבלים **[b a b b]**, ואותו מקודדים בקידוד RLE, כדי לקבל את התוצאה הסופית.

```
def remove_stream_elem (rle_strm, k):
    counter = 0
    while True :
        next_element = next(rle_strm)
        counter += run_len(next_element)
        if counter > k :
            yield (run_val(next_element), counter-k)
            break
    while True :
        yield next(rle_strm)
```

פתרון אחר השתמש בהגדרת פונקציית עזר rle_decode:

```
def rle_decode(s):
    for element in s:
        for i in range(run_len(element)):
            yield run_val(element)

def remove_stream_elem(rle_strm, k):
    decoded_strm = rle_decode(rle_strm)
    for i in range(k):
        next(decoded_strm)
    return rle_encode(decoded_strm)
```

ג. (10 נקודות) זרם RLE הוא **ייצוג ביניים** של נתונים. כאשר ברצוננו לשמור את הנתונים לקובץ (או לשלוח אותם ברשת), אנו צריכים להמיר אותם ל**ייצוג בינארי** (אפסים ואחדות).

לפניכם הפונקציה **inter_to_bin** הממירה זרם RLE מייצוג ביניים לזרם RLE בייצוג בינארי באופן הבא: עבור כל זוג (tuple) בזרם המקורי, היא מחזירה בזה אחר זה את הביטים שבייצוג הבינארי של התו (לפי הערך ה ascii שלו ב-7 ביטים), ומיד אחר כך את הביטים שבייצוג הבינארי של אורך הרצף. אורך הרצף המקסימלי מוגבל ל-31 תווים:

```
def inter_to_bin (lst , max_length = 2**5 -1):
    """ converts intermediate format compressed RLE steam
    to a stream of bits """
    width = math.ceil(math.log( max_length ,2))
    for run in lst :
      for bit in '{:07b}'.format(ord(run_val(run) )):
            yield bit
      for bit in '{num:0{w}b}'.format(num=run_len(run), w=width):
            yield bit
```

מהו ייצוג ה RLE הבינארי של המחרוזת "Gooooo!!!!!!1" , המתקבל מהפעלת הפקודות הבאות:

```
>>> RLE_inter_encode = RLE_encode(iter("Gooooo!!!!!!1"))
>>> RLE_bin_encode = inter_to_bin(RLE_inter_encode)
>>> "".join(bit for bit in RLE_bin_encode)
```

ההדפסה (ללא רווחים):

```
1000111 00001 1101111 00101 0100001 00110 0110001 00001
# 'G'     1     'o'     5     '!'     6     '1'     1
```

לנוחותכם מצורפת טבלת ASCII:

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

מס' סידורי:_____ ת.ז.:_____

ד. (5 נקודות) איזה קידוד ישיג תוצאות טובות יותר בדחיסת המחרוזת "1!!!!!!Goooooo"? קידוד RLE או **קידוד האפמן**, כאשר הקורפוס לבניית המילון הוא המחרוזת עצמה? נמקו בקצרה (2-3 משפטים). תשובות ללא נימוק לא תתקבלנה.

קידוד האפמן.

בקידוד האפמן התווים '!' ו- 'o' יקודדו בעזרת סיבית או שתיים (בהתאמה), והתווים 'G' ו- '1' בעזרת שלוש סיביות – סה"כ 22 סיביות.

לקידוד RLE מהסעיף הקודם דרושות 4*(5+7) = 48 סיביות

ה. (5 נקודות) לשאלתיאל קוואק לא ברור מדוע היה צורך להגביל את אורך הרצף המקסימלי (ל-31 תווים או לכל ערך אחר). הוא מציע לקודד את האורך של כל אחד מהרצפים בעזרת מספר הביטים אשר דרושים לייצוג אותו מספר. מה הבעיה בהצעתו של שאלתיאל?

אם הקידוד לא מגביל את אורך הרצף המקסימלי, אזי בעת תהליך הפענוח לא ניתן לדעת כמה סיביות מוקצות לקידוד האורך, ומתי מתחילות הסיביות המייצגות את התו הבא.

שאלה 3 (15 נקודות)

נתון קטע הקוד הבא (ב- IDLE), ואחריו כמה משפטים המתייחסים לקטע הקוד. עבור כל אחד מהם יש לציין האם הוא אמת או שקר. אם ציינתם שהמשפט הוא שקרי יש צורך לנמק.

```
>>> x = 1/2
>>> x
```

א. לאחר ביצוע 2 השורות יודפס למסך "**0.5**"

אמת

ב. ההדפסה למסך במקרה זה אינה מייצגת את הערך האמיתי ש-**x** מצביע אליו, אלא קרוב של הערך הזה, אשר עוגל לצורכי נוחות

שקר – מכיוון ש 1/2 ניתן לייצוג בבסיס 2 ללא כל שגיאה – אין שגיאה גם בערך הנשמר בזיכרון

ג. הטיפוס של המשתנה **x** הוא **Fraction**

שקר – מספרים שאינם שלמים מיוצגים ע"י float

ד. אם מריצים את הקוד שלמעלה ב Python 3 על Windows 7 בגרסת 64 ביט, אזי לצורך שמירת ערכו של **x**, יוקצה בזיכרון מקום של 64 סיביות בפורמט IEEE-754

אמת

ה. **int(x)==0** הוא **True**

אמת

שאלה 4 (10 נקודות)

חברת ההזנק "לחישות פזיזות" מתמחה בייצור מבני נתונים המבוססים על פונקציות גיבוב (hash functions) פורצות דרך. חוו דעתכם על 2 הפיתוחים האחרונים שלהם:

```
def hash1(element) :
    return id(element)

def hash2(element) :
    return 42
```

האם הפונקציה **hash1** היא פונקציית hash **חוקית**?

אם לא, הסבירו בקצרה (עד 3 משפטים) ותנו דוגמא למקרה שבו לא תעבוד כהלכה.

אם עניתם כן, אין צורך לנמק.

הפונקציה אינה חוקית –

על פונקציית hash להקצות לשני ערכים שווים את אותו ערך hash. ואולם מכיוון שבפייתון ניתן לאחסן שני ערכים שווים במקומות שונים בזכרון (למשל מספרים גדולים), שני ערכים אלו יקבלו ערך hash שונה זה מזה.

האם הפונקציה **hash2** היא פונקציית hash **חוקית**?

אם לא, הסבירו בקצרה (עד 3 משפטים) ותנו דוגמא למקרה שבו לא תעבוד כהלכה.

אם עניתם כן, אין צורך לנמק.

הפונקציה חוקית.

פונקציות hash נבדלות זו מזו (בין השאר) במידת הפיזור של הערכים, הפונקציה hash2 היא דוגמא קיצונית לפונקציה אשר מפזרת את הערכים באופן גרוע במיוחד (אם כי חוקי...)

שאלה 5 (10 נקודות)

נתון קטע הקוד הבא (המבוסס על הקוד שראינו בהרצאה) בתוספת השורות **single_black_pixel**:

```
from PIL import Image

def neighbours(pix,  width, height, x, y, nx=1, ny=1):
    nlst = []
    for yy in range(max(y-ny, 0), min(y+ny+1, height)):
        for xx in range(max(x-nx, 0), min(x+nx+1, width)):
            nlst.append(pix[xx, yy])

    return nlst

def morphological(im, operator = min, nx = 5, ny = 5):
    width, height = im.size
    out_im = Image.new('L',(width, height), 'white')
    in_pix = im.load()
    out_pix = out_im.load()

    for y in range(height):
        for x in range(width):
            nlst = neighbours(in_pix, width, height, x, y, nx, ny)
            out_pix[x, y] = operator(nlst)
    return out_im

def erosion(im, nx = 5, ny = 5):
     return morphological(im, min, nx, ny)

def single_black_pixel():
    out_im = Image.new('L',(256, 256), 'white')
    out_pix = out_im.load()
    out_pix[127, 127] = 0
    return out_im
```

בקטע הקוד הבא, מהו הערך המינימאלי שאותו יש לרשום במקום סימני השאלה (???) כדי שתוצג למסך תמונה שחורה לחלוטין?

```
>>> pic = single_black_pixel()
>>> for i in range(???):          # what should be written here?
        pic = erosion(pic)
>>> pic.show()
```

א. 1

ב. 26

ג. 32

ד. 128

ה. התמונה שתוצג למסך לא תהיה שחורה לחלוטין לכל ערך שהוא