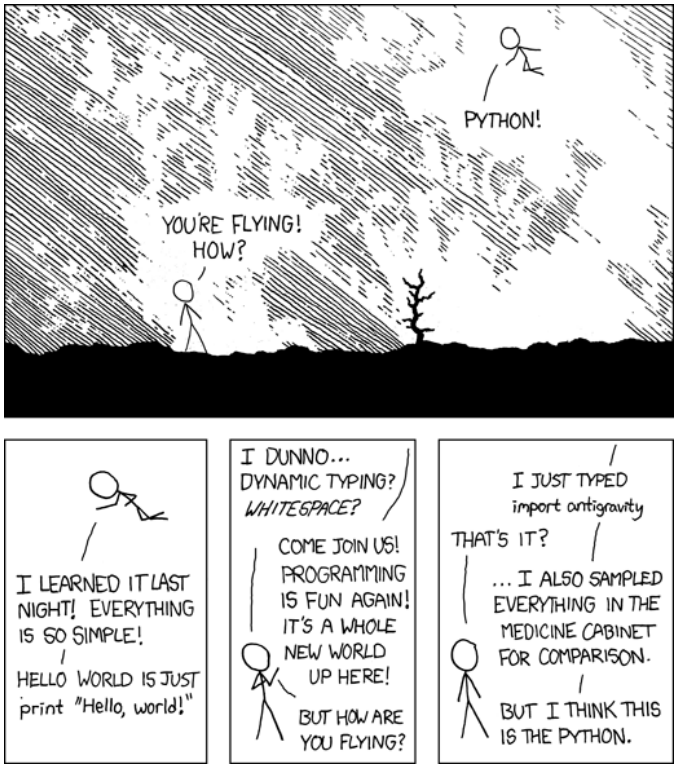
הטכניון – מכון טכנולוגי לישראל

הפקולטה להנדסת חשמל, המעבדה למערכות תוכנה מרושתות

ניסוי בפייתון



גרסה 1.2: נובמבר 2012, איתי אייל, אלכס שרמן

הניסוי מתקיים בבניין פישבך, חדר 375 (במסדרון למאייר).
המאחר ביותר מ-15 דקות לא יורשה לבצע את הניסוי.

המעבדה למערכות תוכנה מרושתות, הפקולטה להנדסת חשמל
אתר: http://softlab.technion.ac.il
טלפון: 04-829-4634
דואל: ilana@ee.technion.ac.il

תוכן העניינים

כל הערה והמלצה לשיפור תתקבל בברכה – מנושאים חסרים או לא ברורים ועד שגיאות הקלדה.
אנא העבירו הערותיכם למנחה הניסוי.

תודה, צוות המעבדה.

⚠ בטיחות ⚠

כללי

הנחיות הבטיחות מובאות לידיעת הסטודנטים כאמצעי למניעת תאונות בעת ביצוע ניסויים ופעילות במעבדה למערכות תוכנה. מטרתן להפנות תשומת לב לסיכונים הכרוכים בפעילויות המעבדה, כדי למנוע סבל לאדם ונזק לציוד.
אנא קראו הנחיות אלו בעיון ופעלו בהתאם להן.

מסגרת הבטיחות במעבדה

- אין לקיים ניסויים במעבדה ללא קבלת ציון עובר בקורס הבטיחות של מעבדות ההתמחות באלקטרוניקה (שהינו מקצוע קדם למעבדה זו).
- לפני התחלת הניסויים יש להתייצב בפני מדריך הקבוצה לקבלת תדריך ראשוני הנחיות בטיחות.
- אין לקיים ניסויים במעבדה ללא השגחת מדריך ללא אישור מראש.
- מדריך הקבוצה אחראי להסדרים בתחום פעילותך במעבדה; נהג על פי הוראותיו.

עשה ואל תעשה

- יש לידע את המדריך או את צוות המעבדה על מצב מסוכן וליקויים במעבדה או בסביבתה הקרובה.
- לא תיעשה במזיד ובלי סיבה סבירה, פעולה העלולה לסכן את הנוכחים במעבדה.
- אסור להשתמש לרעה בכל אמצעי או התקן שסופק או הותקן במעבדה.
- היאבקות, קטטה והשתטות אסורים. מעשי קונדס מעוררים לפעמים צחוק אך הם עלולים לגרום לתאונה.
- אין להשתמש בתוך המעבדה בסמים או במשקאות אלכוהוליים, או להיות תחת השפעתם.
- אין לעשן במעבדה ואין להכניס דברי מאכל או משקה.
- בסיום העבודה יש להשאיר את השולחן נקי ומסודר.
- בניסיון לחלץ דפים תקועים במדפסת - שים לב לחלקים חמים!

בטיחות חשמל

- בחלק משולחנות המעבדה מותקנים בתי תקע ("שקעים") אשר ציוד המעבדה מוזן מהם. אין להפעיל ציוד המוזן מבית תקע פגום.
- אין להשתמש בציוד המוזן דרך פתילים ("כבלים גמישים") אשר הבידוד שלהם פגום או אשר התקע שלהם אינו מחוזק כראוי.
- אסור לתקן או לפרק ציוד חשמלי כולל החלפת נתיכים המותקנים בתוך הציוד; יש להשאיר זאת לטפול הגורם המוסמך.
- אין לגעת בארון החשמל המרכזי, אלא בעת חירום וזאת - לצורך ניתוק המפסק הראשי.

מפסקי לחיצה לשעת חירום

- במעבדה ישנם מפסקים ראשיים להפסקת אספקת החשמל. זהה את מקומם.
- בעת חירום יש להפעיל מפסקי החשמל הראשיים.

בטיחות אש, החייאה ועזרה ראשונה

- במעבדה ממוקמים מטפי כיבוי אש זהה את מקומם.
- אין להפעיל את המטפים, אלא בעת חירום ובמידה והמדריכים וגורמים מקצועיים אחרים במעבדה אינם יכולים לפעול.

יציאות חירום

- בארוע חירום הדורש פינוי, כגון שריפה, יש להתפנות מיד מהמעבדה.

דיווח בעת אירוע חירום

- יש לדווח **מידית** למדריך ולאיש סגל המעבדה.
- המדריך או איש סגל המעבדה ידווחו מיידית לקצין הביטחון בטלפון; 2740, 2222.
  במידה ואין הם יכולים לעשות כך, ידווח אחד הסטודנטים לקצין הביטחון.
- לפי הוראת קצין הביטחון, או כאשר אין יכולת לדווח לקצין הביטחון, יש לדווח, לפי הצורך:
  משטרה 100,
  מגן דוד אדום 101,
  מכבי אש 102,
  גורמי בטיחות ו/או ביטחון אחרים.
  בנוסף לכך יש לדווח ליחידת סגן המנמ"פ לעניני בטיחות; 3033, 2146/7.
- בהמשך, יש לדווח לאחראי משק ותחזוקה; 4776
- לסיום, יש לדווח ל:
  מהנדס המעבדה (טל. 4635)
  בעת הצורך ניתן להודיע במקום למהנדס המעבדה לטכנאי המעבדה.

מבוא

שפת התכנות פייתון הופיעה בשנת 1991, ומאז יציאת גרסה 2 שלה בשנת 2000 היא צוברת פופולריות רבה בזכות הקלות בה ניתן לכתוב בעזרתה תוכניות מורכבות. השפה מאפשרת פיתוח מהיר של סקריפטים, סימולציות וגם מערכות שלמות.

ספר הניסוי הזה מהווה קורס מבוא לפייתון. הוא נועד לתת כלים ראשונים לשימוש בשפה, תוך הישענות על ספרי עזר או מקורות באינטרנט. רשימה של מקורות כאלה נמצאת בפרק הביבליוגרפיה. חלקם, בפרט האתר המומלץ `Dive into python`, יכולים לשמש תחליף לספר זה עבור סטודנטים המעוניינים לרכוש ידע מעמיק יותר של השפה. לאלה המעוניינים לכתוב תוכניות מורכבות, מומלץ להתחיל בלימוד תכנות מונחה עצמים, חריגות ותכנות מקבילי בפייתון – נושאים אשר לא מכוסים בספר זה בשל מגבלות מקום.

הרקע הנדרש לקריאת חוברת זו: ידע בסיסי בתכנות ובתכנות מונחה עצמים (`C++/Java`).

השאלות בגוף החוברת מהוות את השאלות של הדו"ח המכין לניסוי המעבדה.
שימו לב: לא ניתן לבצע את הניסוי ללא הגשת דו"ח מכין.
את הדו"ח יש להגיש במערכת `LABADMIN`.

הוראות הפעלה

הפעלה אינטראקטיבית (Interactive Shell)

ניתן להשתמש בפייתון באופן אינטראקטיבי – לכתוב פקודות לביצוע מיידי, ולקבל בחזרה את הערכים המוחזרים. לשם כך, יש להפעיל את פייתון על ידי כתיבת הפקודה python מתוך קונסולה במחשב בו התוכנה מותקנת. (הוראות התקנה באתר www.python.org). התוכנה מופעלת ועל המסך מופיע ה-interactive shell. בתוך ה-shell ניתן לכתוב פקודות פייתון (בדוגמא להלן נכתבה הפקודה "print "Hello world) והתוצאה מודפסת מיד (בדוגמא להלן מודפסות המלים Hello world).

```
~$ python
Python 2.6.4 (r264:75706, Dec  7 2009, 18:45:15)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello world"
Hello world
>>>
```

ליציאה, יש להקיש את צירוף המקשים Ctrl+D.

הרצה משורת פקודה

על מנת להריץ תוכניות הנכתבות בשפת פייתון, כותבים את התוכנית בקובץ, לדוגמא הקובץ הבא בעל השם hello.py:

```
Print "Hello world from a file."
```

ואז, מקונסולה, מספקים את שם הקובץ כפרמטר ל-python:

```
~$ python hello.py
Hello world from a file.
~$
```

תוכנת python מריצה את הקוד מהקובץ hello.py, והתוצאה נכתבת למסך. עם סיום הריצה, הפיקוד חוזר לידי מערכת ההפעלה.

פרמטרים משורת הפקודה זמינים בתוכנית פייתון ברשימה בשם sys.argv. האיבר הראשון ברשימה הוא שמה של התוכנית והאיברים הבאים הם הפרמטרים. כדי להשתמש במשתנה זה יש לייבא את המודול sys. כיצד עושים זאת, וכיצד מטפלים ברשימות נלמד בהמשך.

Eclipse

כדי לכתוב תוכניות גדולות, נוח להשתמש בסביבת עבודה משולבת המספקת כלי debugging נוחים.
סביבה מומלצת היא Eclipse (www.eclipse.org) עם התוסף Pydev (pydev.org).
הוראות וקבצים להתקנה, כמו גם הוראות שימוש וקבצי עזרה ניתן למצוא באתרים הנ"ל.

דוגמא ראשונה

להלן תוכנית פייתון:

```
1. import re
2.
3. if __name__ == "__main__":
4.
5.     file = open("stam.txt")
6.     lines = file.readlines()
7.     file.close()
8.
9.     lines = [line.upper() for line in lines]
10.    for line in lines:
11.        inParentheses = re.findall("\(.*?\)", line)[0]
12.        if inParentheses:
13.            print inParentheses
```

נפתח בהערה מפתיעה על הסינטקס של פייתון. בלוקים בפייתון מסומנים על ידי הזחה (indentation), ולא על ידי הקפתם בסוגריים מסולסלים כמו ב-C וב-Java. המשפט לו כפוף הבלוק מסתיים בנקודתיים, כפי שניתן לראות בשורות 3, 10, 12. שימו לב – בפייתון אין צורך ב-; בסוף שורה. זה לא מזיק. זה גם לא מועיל.

ביאור התוכנית:

1: ייבוא פונקציות של ביטויים רגולריים (regular expressions) בהן נשתמש בהמשך (שורה 11).
3: הסבר על משמעות שורה זו יגיע בפרק העוסק במודולים. בשלב זה ניתן להתעלם ממנה.
5: פתח את הקובץ stam.txt ושייך אותו למשתנה file.
6: קרא את שורות הקובץ לתוך מערך של מחרוזות בשם lines, כל שורה במחרוזת משלה.
7: סגור את הקובץ.
9: עבור כל שורה במערך lines, צור מחרוזת עם אותו תוכן אך אותיות גדולות (capitals). את אוסף המחרוזות הנוצר הכנס למערך lines.
10: לכל אחת מהשורות ב-lines
11: חפש את הביטוי הראשון בסוגריים. כאן אנו משתמשים בביטוי רגולרי המתאר "ביטוי בסוגריים". לשם כך אנחנו נעזרים בספריה re אותה ייבאנו בשורה 1.
12: אם יש ביטוי כזה,
13: הדפס אותו.

למשל, אם הקובץ stam.txt הינו:

```
(Hello), my name is (Inigo Montoya).
It's a small (world) after all.
```

הפלט של התוכנית הנ"ל יהיה:

```
(HELLO)
(WORLD)
```

פונקציות

פונקציה בפייתון מגדירים בעזרת המלה השמורה def. כל הפונקציות בפייתון מחזירות ערך, None אם לא נאמר אחרת. מספר הפרמטרים אינו בהכרח קבוע. ניתן לקבוע שלחלק מהפרמטרים יש ברירת מחדל. בדוגמא להלן ניתן לוותר על הפרמטר השני אשר מקבל כברירת מחדל את הערך 2.
בתחילת כל פונקציה ניתן (ורצוי, אחרת נפגעים הקריאות קריאות הקוד והציון בתרגיל) לרשום תיעוד. התיעוד נרשם בין שלשת גרשיים ונקרא docstring. התיעוד זמין בזמן ריצה כשדה של הפונקציה (בדוגמא להלן multiply.__doc__), וכמו כן מופיע כ-tooltip כאשר הסמן עובר מעל שם הפונקציה ב-eclipse.

```
def multiply(x, times=2):
    """Multiply a variable
    Multiplies the variable x times times.

    Returns the multiplication result."""

    return x * times
```

טיפוסי הנתונים נקבעים בזמן הריצה. עבור הפונקציה הנ"ל, למשל, הטיפוס של x והטיפוס של הערך המוחזר אינם קבועים מראש:

```
>>> multiply(3)
6
>>> multiply("ni ", 9)
'ni ni ni ni ni ni ni ni ni '
```

מודולים

פייתון מציע מגוון ספריות עם פונקציות ואובייקטים לצרכים רבים ומגוונים – טיפול במדיה, תקשורת, מבני נתונים ועוד. ספריות מיובאות בעזרת הפונקציה המובנית import. בדוגמא הראשונה לעיל ייבאנו את ספריית הביטויים הרגולריים re.

ניתן לייצר ספריות חדשות על ידי שמירת קובץ פייתון עם פונקציות ואובייקטים וביצוע import לקובץ זה. בעת ביצוע import יבוצעו כל הפקודות הרשומות בקובץ. ואולם, לעתים נרצה שהקובץ יריץ פקודות מסויימות רק אם הוא נקרא ישירות (ק' על ידי הרצת python filename ולא על ידי import). לשם כך ניתן לבדוק את תוכנו של המשתנה __name__. תוכנו של משתנה זה הינו המחרוזת "__main__" אם ורק אם הקובץ נקרא ישירות.
מומלץ תמיד להקדים לפקודות העיקריות של הקובץ (ולא לפני הפונקציות) שאילתה הבודקת האם הוא נקרא ישירות. כך נעשה בדוגמא הראשונה בחוברת. הרגל זה יבטיח שתוכלו להשתמש בכל קובץ כספרייה, אם תחפצו בכך.

אובייקטים

בתכנות מונחה עצמים, מידע ופעולות מצורפים יחד. איחוד כזה של מידע ופעולות נקרא אובייקט (עצם). לכל אובייקט יש תכונות (משתנים) ויש מתודות (פונקציות). בפייתון, כל דבר נשמר כאובייקט במובן של תכנות מונחה עצמים. זה כולל את כל סוגי המשתנים לעיל, מודולים (להם עושים import) וגם פונקציות (עליהן נלמד בהמשך).
לתכונות ולפעולות של אובייקט קוראים על פי שמם אותו רושמים אחרי שם האובייקט ונקודה. בדוגמא לעיל הופעלה המתודה close של האובייקט file כדי לסגור את הקובץ.

כדי להבין את ההתנהגות של פייתון, יש להבדיל בין אובייקט לבין שם של אובייקט. האובייקט הוא התכונות והמתודות, ושם האובייקט מייצג רפרנס (הפניה) לאובייקט. לפיכך, יכול להיות אובייקט אשר מפנים אליו מספר שמות. כמו כן, שם המצביע לאובייקט אחד, יכול בשלב מאוחר יותר בתוכנית להצביע לאובייקט אחר מסוג מספר, פונקציה או רשימה:

```
>>> a = 7                          # a is the number 7
>>> a
7
>>> a = len                        # a is the function len
>>> len("goodbye")
7
>>> a("goodbye")
7
>>> a = "goodbye"                  # a is the string goodbye
>>> a
'goodbye'
```

באופן מפתיע, גם הערכים עצמם הינם עצמים. המספר המרוכב `4+3j`, לדוגמא, הוא אובייקט עם תכונות וניתן לבצע עליו פעולות. כאשר כותבים `4+3j` המשמעות היא מצביע לאובייקט שזה ערכו. לפיכך נרשום ונקבל:

```
>>> (4+3j).conjugate()
(4-3j)
```

משמעותם של אופרטורים נקבעת על פי האובייקטים עליהם הם פועלים. למשל האופרטור + המופעל על מספרים משמעו סכום, אך כשהוא מופעל על מחרוזות משמעו שרשור מחרוזות.

```
>>> 5 + 5
10
>>> "5" + "5"
'55'
```

האובייקטים בפייתון מתחלקים לשני סוגים – אובייקטים הניתנים לשינוי (`mutable`) ואובייקטים שאינם ניתנים לשינוי (`immutable`). טיפוס הרשימה שנכיר בהמשך הוא דוגמא לאובייקט הניתן לשינוי – אפשר להוסיף ולהסיר איברים מרשימה. טיפוס המחרוזת, שגם אותו נכיר בהמשך, הוא דוגמא לטיפוס שאינו ניתן לשינוי. אם רוצים לערוך את המחרוזת אליה מפנה משתנה מסויים, צריך למעשה לשנות את ההפנייה של המשתנה למחרוזת חדשה.
בשל האמור לעיל, נשים לב לתכונה מעניינת של העברת פרמטרים לפונקציה. הפרמטרים המתקבלים הם רפרנסים לאובייקטים. הפונקציה יכולה לשנות את האובייקטים שהועברו אליה (בהנחה שהם `mutable`), כיוון שהיא מקבלת רפרנס אליהם. מצד שני, היא אינה יכולה לשנות את היעד אליו מצביע הרפרנס. בדוגמא שלהלן שתי פונקציות (המוגדרות בעזרת המלה `def`). הראשונה משנה את האובייקט, והשנייה משנה את הרפרנס.

```
def funcAddElement(lst):
    return lst.append("Added from within func")

def funcChangeRef(lst):
    lst = [2, 3, 6]

if __name__ == "__main__":

    x = [1, 2, 5]

    funcAddElement(x)
    print x

    funcChangeRef(x)
    print x
```

בפלט שלהלן ניתן לראות שלפונקציה השנייה לא הייתה השפעה. הפלט הוא בשני המקרים סדרה בת 4 איברים. הפונקציה הראשונה הוסיפה את האיבר הרביעי לאובייקט x אשר נקרא בתוך המחרוזת בשם lst. הפונקציה השניה קיבלה ב-lst רפרנס ל-x, אבל החליפה אותו ברפרנס לרשימה אחרת. האובייקט x לא השתנה.

```
[1, 2, 5, 'Added from within func']
[1, 2, 5, 'Added from within func']
```

הפונקציה dir

ניתן לקבל את רשימת המתודות של אובייקט בעזרת הפונקציה המובנית dir. עבור האובייקט x יש לרשום

```
dir(x)
```

נסו זאת כבר עתה עבור מחרוזות ומספרים.

הפונקציה help

ניתן לקבל עזרה מפורטת עבור כל אובייקט בעזרת הפונקציה help:

```
help(x)
```

משתנים והדפסתם, הצבה

בפייתון מספר מבני נתונים מובנים – מבנים פשוטים לאחסון מספרים ומחרוזות, ומבנים מורכבים לאחסון אוספי משתנים. לפני שנתחיל בסקירת סוגי המשתנים, להלן מספר פרטים כלליים. כדוגמא נשתמש בינתיים במספרים שלמים (Integer), הדומים לשלמים בשפות אחרות דוגמת C ו-Java.

- משתנים נוצרים ברגע שמציבים להם ערך. לא מכריזים עליהם מראש כמו ב-C. למשל, ליצירת משתנה בשם x עם הערך 5, משתמשים בפקודה להצבת הערך:

```
>>> x = 5
```

- ניתן להציב כמה משתנים יחד. ליצירת המשתנים y ו-z בעלי ערכים 6 ו-7 בהתאמה:

```
>>> y, z = 6, 7
```

- ניתן להדפיס את ערכו של משתנה בעזרת הפקודה print. לאחר הפעלת הפקודות לעיל, ניתן להדפיס את המשתנים:

```
>>> print x
5
>>> print y
6
>>> print z
7
```

מספרים לסוגיהם

קיימים 5 טיפוסים בסיסיים של מספרים. כל מספר בפייתון שייך לאחד הטיפוסים. אופן כתיבת המספר מסביר לשפה מה טיפוסו. בטבלא להלן מפורטים הטיפוסים השונים.

טיפוס	תיאור	דוגמאות
שלמים (Integer)	מספרים שלמים. כמו משתני long של שפת C.	3, -100, 0
שלמים ארוכים (Long Integer)	שלמים ללא מגבלת גודל.	עם L בסוף: 12345678901234567890L
נקודה צפה (Floating Point)	משתני נקודה צפה, כמו Double של C	עם נקודה ו-e: 3.1415, 12e-5, 5.4e123
בסיסים 8 ו-16 (Octal, Hex)	כשמם	קידומת אפס לבסיס 8, ואפס-איקס ל-16: 01357, 0x1F
מרוכבים	כשמם	j או J לציון החלק המרוכב: 3+4j, 2.5+1.25j, 9J
True/False	כשמם. למעשה False=0, True=1	רושמים True או False. דוגמאות בהמשך, בסעיף שאילתות.

לא להתבלבל – משתנים שלמים רגילים בפייתון שקולים למשתנים שלמים מטיפוס long של C. בנוסף לכך, יש בפייתון משתנים שלמים בעלי אורך בלתי מוגבל (פרט למגבלה של כמות הזיכרון הפנוי במחשב) המכונים Long Integer.

פייתון מבצע פעולות חשבון על פי סדר פעולות כצפוי.

<u>שאלה 1</u>

הפעילו פייתון באופן אינטראקטיבי, ובצעו את החישובים להלן. שרטטו טבלא ורשמו לכל חישוב (1) את הביטוי שכתבתם, שימו לב שלעתים אתם נדרשים לסוג משתנה ולסוג הצבה מסויימים, (2) את התוצאה שחישב המחשב, (3) ואת הטיפוס של התוצאה.

1. שלוש וחצי (נקודה צפה) ועוד ארבע (שלם).
2. 15_8 (אוקטאלי) ועוד 255_{16} (הקסדצימאלי).
3. 999999999 (תשע תשיעיות) כפול עשר.
4. (1+2j) * 2
5. 1 + 2j * 2

מחרוזות

טיפוס המחרוזת מאכסן רצף של תווים. מחרוזת מוקפת ב-' או ב-", על פי בחירתכם. ניתן לרשום מחרוזת בת מספר שורות בין שלישיית גרשיים """ לעוד שלישיית גרשיים. במקרה זה, מעבר שורה נשמר למחרוזת. ניתן להציב מחרוזות לתוך משתנים בעזרת =. מחרוזות מכילות תווים רגילים, כמו גם escape codes דוגמת \n לסוף שורה ו-\t לטאב.

ניתן להדפיס מחרוזות בעזרת הפונקציה המובנית print. המחרוזות מודפסות כאשר ה-escape codes מתורגמים. בהפעלה אינטראקטיבית, ניתן לרשום את המחרוזת בשורת הפקודה, אז היא מודפסת כמו בקוד, בין ' וללא תרגום של ה-escape codes.

```
>>> S = "Hello\nWorld"
>>> S
'Hello\nWorld'
>>> print S
Hello
World
```

הפונקציה המובנית `len` מחזירה את אורכה של מחרוזת. היא מחזירה גם את אורכן של רשימות ושל `tuples` עליהן נלמד בהמשך.
ניתן לקבל קטע מחרוזת על ידי רישום טווח אינדקסים בתוך סוגריים מרובעים. מספר יחיד בסוגריים מרובעים מחזיר את התו בעל אינדקס זה. מספרו של התו הראשון 0. טווח תוים מופרד על ידי :, למשל `S[x:y]` מחזיר תת מחרוזת של `S`, מאינדקס `x` ועד אינדקס `y`, לא כולל את `y`. אם `x` או `y` לא נכתבים, מוחזרת המחרוזת מההתחלה או עד הסוף, בהתאמה.

```
>>> S = 'grail'
>>> S[2]
'a'
>>> S[1:]
'rail'
>>> len(S)
5
>>> S[2:5]
'ail'
```

ניתן לשרשר מחרוזות באמצעות +:

```
>>> S1 = 'Holy'
>>> S2 = ' Grail'
>>> S3 = S1 + S2
>>> print S3
Holy Grail
```

טיפוס המחרוזת בפייתון אינו בר שינוי (הוא `immutable`), כלומר לא ניתן לשנות מחרוזת, למרות שניתן כמובן להציב במשתנה ערך חדש.
זה לא יעבוד:

```
>>> S = "grail"
>>> S[4] = "n"
תקלה!
```

זה כן, כי אנחנו מציבים אובייקט מחרוזת חדש לתוך השם `S`:

```
>>> S = "grail"
>>> S = S[0:4] + "n"
>>> print S
```

שאלה 2:

1. מה השגיאה המתקבלת במקרה הקודם בו ניסינו לשנות את המחרוזת?
2. מה התוצאה שידפיס print בדוגמא האחרונה?
3. מה קורה אם רושמים מספר שלילי בטווח? בידקו.

str.format()

כדי לשלב טיפוסי נתונים ומשתנים בתוך מחרוזת ניתן להשתמש במתודה format של המחלקה String. שימו לב שבדוגמאות להלן אנחנו משתמשים במתודות של מחלקת String ישירות מתוך מחרוזת:

```
>>> print "{0} Sir {1}".format("Brave", "Robin")
Brave Sir Robin

>>> print "{1} Sir {0}".format("Robin", "Brave")
Brave Sir Robin

>>> print "{adj} Sir {name}".format(adj="Brave", name="Robin")
Brave Sir Robin

>>> print "{four} shalt thou not count, neither count thou {two}, excepting
that thou then proceed to {three:.3f}.".format(four=0x4, three=3.0, two=02)

4 shalt thou not count, neither count thou 2, excepting that thou then
proceed to 3.000.
```

עד לאחרונה, פירמוט מחרוזות נעשה באופן דומה ל-printf של C בעזרת האופרטור % כדלקמן:

```
>>> print "%s Sir %s" % ("Brave", "Robin")
Brave Sir Robin
```

סינטקס זה צפוי לעבור מן העולם, אך אתם עשויים לפגוש בו בינתיים בתוכניות פייתון ישנות.

קבלת קלט מהמשתמש

על מנת לקבל קלט מהמשתמש במהלך ריצת פייתון לתוך מחרוזת, ניתן להשתמש בפונקציה המובנית raw_input. בדקו באינטרנט כיצד משתמשים בפונקציה הזאת – תזדקקו לה בשעת הניסוי.

רשימות

רשימות (Lists) הן טיפוס המחזיק אוסף אובייקטים אליהם ניתן להגיע לפי אינדקס.
האובייקטים ברשימה יכולים להיות מכל טיפוס, בפרט ניתן להחזיק באותה מחרוזת אובייקטים מטיפוסים שונים, כולל רשימות.
כמו במחרוזות, ניתן לצרף רשימות בעזרת +, לבדוק את אורכן בעזרת len, ולקבל תת רשימה על ידי רישום טווח בתוך סוגריים מרובעים.
רשימה מוגדרת על ידי רישום אוסף אובייקטים מופרדים בפסיקים בתוך סוגריים מרובעים.

```
>>> count = [1, 2, 3, "O'Clock"]
>>> count
[1, 2, 3, "O'Clock"]
>>> len(count)
4
>>> count[2:]
[3, "O'Clock"]
>>> count + [4, "O'Clock"]
[1, 2, 3, "O'Clock", 4, "O'Clock"]
```

ניתן לשנות את האובייקטים שברשימה באופן דינאמי – רשימה היא mutable, בניגוד למחרוזות.

```
>>> count
[1, 2, 3, "O'Clock"]
>>> count[0] = 2
>>> count
[2, 2, 3, "O'Clock"]
>>> count[1:] = [4, 6, 16]
>>> count
[2, 4, 6, 16]
```

ניתן גם לשנות את אורכה של רשימה באופן דינאמי. לשם כך מפעילים מתודות של רשימות. להפעלת מתודה, מוסיפים נקודה אחרי שם הרשימה ורושמים את שם המתודה. בדוגמא להלן, המתודה append מוסיפה איבר בסוף הרשימה, והפקודה pop שולפת איבר מהרשימה:

```
>>> count = [1, 2, 3, "O'Clock"]
>>> count.append(4)
>>> count
[1, 2, 3, "O'Clock", 4]
>>> count.pop(2)
3
>>> count
[1, 2, "O'Clock", 4]
```

שימו לב – למרות שגודלן של רשימות הוא דינאמי, לא ניתן לשלוף או להציב מחוץ לתחומי הרשימה. זה לא יעבוד:

```
>>> count
[1, 2, "O'Clock", 4]
>>> count[5] = 9
תקלה!
```

שאלה 3

מה השגיאה המתקבלת בדוגמא האחרונה?

כאמור, רשימות יכולות להכיל תת רשימות. לדוגמא:

```
>>> count = [[1, 2, 3], "O'Clock"]
>>> count
[[1, 2, 3], "O'Clock"]
>>> count[0]
[1, 2, 3]
>>> count[0][1]
2
```

שאלה 4

רשמו סדרת פקודות כדלקמן: (1) הצבת מטריצה 3x3 בעלת הערכים 1-9 במשתנה M; (2) הצגת המטריצה; (3) החלפת האיבר האמצעי במלה "Robin"; (4) הצגת התוצאה.

למחלקה מחרוזת מתודה split המחלקת את המחרוזת לחלקים. המתודה מחלקת את המחרוזת לפי רווחים ומחזירה את התשובה ברשימה. המתודה יכולה לקבל כפרמטר מחרוזת אחרת לפיה תבצע את החלוקה במקום רווח:

```
>>> "a;b c;d".split()
['a;b', 'c;d']
>>> "a;b c;d".split(';')
['a', 'b c', 'd']
```

למחלקה מחרוזת מתודה המאפשרת לשרשר רשימת מחרוזות למחרוזת אחת כאשר המחרוזת עבורה מופעל ה-join מפרידה בין אברי הרשימה:

```
>>> "--".join(["one", "two", "three", "four"])
'one--two--three—four'
```

מילונים

מילון הוא מערך אסוציאטיבי, כלומר מיפוי ממפתחות לערכים. השימוש במילון דומה לשימוש ברשימה, אך הגישה לאובייקטים היא על פי המפתח שלהם ולא על פי אינדקס.
מילון מוגדר על ידי רישום בסוגריים מסולסלים של זוגות אובייקטים (מפתח וערך) מופרדים בנקודתיים, כאשר בין כל שני זוגות מפריד פסיק:

```
>>> car = {'Year' : 2005, 'Maker' : 'Skoda', 'Model' : 'Fabia'}
>>> car['Year']
2005
>>> car['Model']
'Fabia'
>>> car['Year'] += 1
>>> car['Year']
2006
```

ניתן להוסיף מפתחות וערכים למילון באופן ישיר על ידי הצבה:

```
>>> car
{'Model': 'Fabia', 'Maker': 'Skoda', 'Year': 2006}
>>> car['Color'] = 'Gray'
>>> car
{'Color': 'Gray', 'Model': 'Fabia', 'Maker': 'Skoda', 'Year': 2006}
```

שאלה 5

בנו רשימה המכילה שני מילונים, כל אחד בעבור מכונית אחרת, עם השדות הנ"ל.

ניתן לקבל את רשימת המפתחות במילון באמצעות המתודה keys:

```
>>> car.keys()
['Color', 'Model', 'Maker', 'Year']
```

N-יות (Tuples)

N-יה דומה לרשימה, אך אינה ניתנת לשינוי (immutable). היא יכולה להיות מוקפת בסוגריים רגילים (ולא מרובעים כמו רשימה), או ללא סוגריים:

```
>>> count = (1, 2, 3, "O'Clock")
>>> count
(1, 2, 3, "O'Clock")
>>> len(count)
4
>>> count + (4, "O'Clock")
(1, 2, 3, "O'Clock", 4, "O'Clock")
>>> count[2] = 5
```

תקלה!

שאלה 6

מה השגיאה המתקבלת בדוגמא האחרונה?

ניתן להציב למספר משתנים בו זמנית בעזרת N-יות:

```
>>> x, y, z = 1, 2, 3
>>> print "{X} + {Y} + {Z} = {sum}".format(X=x, Y=y, Z=z, sum=x+y+z)
```

פונקציה יכולה להחזיר N-יה כדי להחזיר מספר ערכים.

תנאים ולולאות

תנאים

רגיל

המבנה של שאילתה:

```
if <test1>:
    <statements1>
elif <test2>:
    <statements2>
elif <test3>:
    <statements3>
else:
    <statement4>
```

התנאים test1-3 בנויים כמו ב-C, וה-statements1-4 הם אוסף שורות לביצוע. שים לב שהכפיפות של שורות statement לתנאי שלפניהן מבוטאת על ידי הזחה.

דוגמא:

```
x = 15
if x < 0:
    print "x is negative."
elif x < 10:
    print "x is a small number."
elif x < 20:
    print "mmm..."
    print "Maybe x is not so small."
else:
    print "x is large!"
```

שאלה 7

מה תדפיס התוכנית הנ"ל? ומה אם נציב בשורה הראשונה 100 במקום 15?

תנאים (מה שכינינו קודם test) הם למעשה הערכים true/false, כאשר אפס נחשב false וכל מספר אחר true. גם משתנים (לא ריקים) נחשבים true. ניתן לצרף מספר תנאים בעזרת אופרטורים:

- a and b מחזיר true אם"ם גם a וגם b הם true.
- a or b מחזיר false אם"ם גם a וגם b הם false.
- not a מחזיר true אם"ם a הוא false.

ניתן להשוות מספרים בעזרת אי שוויונים, כך 3<5 יחזיר true ואילו 5==3 יחזיר false. באופן דומה ניתן להשוות גם מחרוזות (לפי סדר א"ב), ולמעשה כל זוג אובייקטים עליהם מוגדר סדר.

ביטוי השאילתה המשולש

קיימת דרך נוספת לכתוב שאילתות באופן מוטמע בהצבה. הביטוי

```
A = Y if X else Z
```

שקול לקוד הבא:

```
if X:
    A = Y
else:
    A = Z
```

לולאות

while

לולאת while נראית כך:

```
while <test>:
    <statements1>
else:
    <statements2>
<more code>
```

הפקודות statements1 מבוצעות בלולאה כל עוד תנאי test הוא אמת. עד כאן – כמו C.
הלולאה יכולה להישבר בשתי דרכים: אם כאשר הלולאה מגיעה לשורת ה-while, התנאי test הינו false, מבוצעות שורות statements2, ואחר כך שורות more code. אם לעומת זאת בתוך statements1 ישנה פקודת break, הלולאה מפסיקה ומדלגים ישירות אל more code.

דוגמא:

```
x = range(7)
while x:                    # While x is not empty
    if x[0] > 4:
        break
    x = x[1:]
else:
    print "All values are not greater than 4."
print "Done."
```

שאלה 8

כמה שאלות:

1. מה מדפיסה התוכנית הנ"ל?
2. מה מדפיסה התוכנית אם בשורה הראשונה נחליף את 7 ב-4?
3. מה מדפיסה התוכנית אם בשורה הראשונה נחליף את 7 ב-5?
4. מה עושה הפונקציה range?

for

לולאת for נראית כך:

```
for <target> in <object>:
    <statements1>
else:
    <statements2>
```

הפקודות statements1 מבוצעות בלולאה, כאשר בכל סיבוב, target מקבל ערך אחר. הערכים של target נקבעים על ידי object. למשל, אם object הוא רשימה, אז target יקבל את ערכי הרשימה בזה אחר זה:

```
 for x in [1, 2, 3, 4, 5, 6, 7]:
    if x%2 == 0:
        print x, " is even."
    else:
        print x, " is odd."
```

שאלה 9

מה מדפיסה התוכנית לעיל?

כמו ב-while, הפקודות שב-else יבוצעו אם הלולאה הסתיימה בשלום ולא הופסקה על ידי break.

List Comprehension

שפת פייתון מאפשרת לטפל ברשימות איברים אחד-אחד באמצעות List Comprehension. כלי זה מורה לפייתון לבצע פעולה כלשהי על כל איבר ברשימה. התוצאה היא רשימה ובה התוצאות של הפעולה על כל אחד מאברי הרשימה המקורית.

בקוד להלן משתמשים בלולאת for כדי לטפל בכל אחד מאברי הרשימה L:

```
L = [1, 2, 3, 4, 5]
newL = [None, None, None, None, None]
for i in range(len(L)):
    newL[i] *= 2
```

שימוש ב-list comprehension נותן אותה תוצאה:

```
>>> L = [1, 2, 3, 4, 5]
>>> newL = [x * 2 for x in L]                    # List Comprehension
>>> print newL
[2, 4, 6, 8, 10]
```

למעשה מצד ימין של השוויון (בשורה השנייה) נוצרת רשימה חדשה אשר מוצבת ל-newL. כל איבר ברשימה החדשה נוצר על פי איבר מהרשימה L על ידי הכפלתו פי 2.

List Filtering

ניתן להשתמש במנגנון זה כדי לסנן אברי רשימה על ידי הוספת שאילתה כדלקמן:

```
>>> L = [1, 2, 3, 4, 5]
>>> newL = [x - 3 for x in L if x >= 3]
[0, 1, 2]
```

קבצים

פתיחה, קריאה וכתיבה

לפתיחת קובץ משתמשים בפונקציה המובנית open אשר מקבלת את שם הקובץ ואת אופן הגישה (קריאה/כתיבה) ומחזירה אובייקט לגישה לקובץ. לסגירת קובץ משתמשים במתודה close שלו:

פתיחת קובץ לקריאה:	`f = open("filename", "r")`
קריאה היא ברירת המחדל, אם לא כותבים "r" או "w":	`f = open("filename")`
פתיחת קובץ לכתיבה:	`f = open("filename", "w")`
סגירת קובץ:	`f.close()`

ארבע מתודות שימושיות לקריאת קובץ:

קריאת כל הקובץ למחרוזת יחידה:	`bigString = f.read()`
קריאת N בתים למחרוזת:	`smallString = f.read(N)`
קריאת שורה אחת:	`line = f.readline()`
קריאת השורות לרשימת מחרוזות:	`lines = f.readlines()`

ולכתיבת לקובץ:

כתיבת מחרוזת יחידה:	`f.write(aString)`
כתיבת רשימת מחרוזות:	`f.writelines(aList)`

כדוגמא, נניח שקיים הקובץ stam.txt הבא:

```
First line!
Line 2.
Line number three.
```

התוכנית הבאה מציגה אותו על המסך:

```
>>> txt = open("stam.txt")
>>> toPrint = txt.read()
>>> print(toPrint)
First line!
Line 2.
Line number three.
```

התבוננו בפקודות הבאות:

```
>>> txt = open("stam.txt")
>>> lines = txt.readlines()
>>> lines = [line.upper() for line in lines]
>>> txt.close()
>>> txt = open("upStam.txt", "w")
>>> txt.writelines(lines)
>>> txt.close()
```

שאלה 10

הסבירו מה עושה כל שורה בתוכנית הנ"ל. מה מכיל הקובץ upStam.txt עם סיום ריצתה?

תכנות פונקציונאלי

לפייתון מספר כלים המקבלים השראה מפרדיגמת התכנות הפונקציונאלי. כאלה הם שלוש הפונקציות הבאות לטיפול ברשימות. כולן מקבלות כפרמטרים פונקציה ורשימה והן מפעילות את הפונקציה על כל אברי הרשימה בזה אחר זה.

עבור שתי הפונקציות הראשונות הפונקציה בפרמטר מקבלת פרמטר יחיד.
הפונקציה `filter` מחזירה את רשימת האיברים עבורם הפונקציה מחזירה ערך אמת.
הפונקציה `map` מחזירה את רשימת התוצאות שמחזירה הפונקציה.
עבור הפונקציה `reduce`, הפונקציה בפרמטר צריכה לקבל שני פרמטרים. `reduce` מפעילה את הפונקציה על האיבר הראשון והשני, אחר כך את מפעילה אותה שוב על התוצאה ועל האיבר השלישי; אחר כך על התוצאה החדשה והאיבר הרביעי וחוזר חלילה. היא מחזירה את התוצאה.

פייתון מאפשר להגדיר פונקציה ללא שם בעזרת המבנה למבדה. פונקציה כזאת נכתבת כך:

```
lambda x: x * 2
```

כמובן שאין טעם לכתוב שורה כזאת, כיוון שלא ניתן לקרוא לפונקציה בלי שם. אפשר לתת לפונקציה שם ואז לקרוא לה (כמו פונקציה רגילה):

```
>>> mult = lambda x: x * 2
>>> mult(5)
10
```

חזרה לפונקציות המטפלות ברשימות. בדוגמאות להלן נשתמש בלמבדה במקום להכריז על פונקציות מראש.

```
>>> filter(lambda x: True if x>5 else None, [2, 4, 6, 8, 10])
[6, 8, 10]
>>> map(lambda x: x if x>5 else 5+x, [2, 4, 6, 8, 10])
[7, 9, 6, 8, 10]
>>> reduce(lambda x, y: x+y, [2, 4, 6, 8, 10])
30
```

ביטויים רגולריים

מבוא

ביטויים רגולריים (regular expressions, regexp או regex בקיצור) הם מחרוזות המתארות טקסט. לכל ביטוי רגולרי RE ולכל טקסט ניתן לקבוע האם הטקסט מתאים ל-RE.

דוגמא טריוויאלית – הביטוי הרגולרי **three** מתאר את המחרוזת "three" ורק אותה.
ביטויים רגולריים משמשים למשל לחיפוש טקסט. בטקסט הבא מסומנות כל התת מחרוזות המתוארות על ידי **three**:

```
And the Lord spake, saying, "First shalt thou take out the Holy Pin. Then
shalt thou count to three, no more, no less. Three shall be the number thou
shalt count, and the number of the counting shall be three. Four shalt thou
not count, neither count thou two, excepting that thou then proceed to three.
Five is right out. Once the number three, being the third number, be reached,
then lobbest thou thy Holy Hand Grenade of Antioch towards thy foe, who,
being naughty in my sight, shall snuff it.
```

ביטוי רגולרי יחיד יכול לתאר גם מחרוזות שונות. למשל, הביטוי **[tT]hree** מתאר את המחרוזת "three" וגם את המחרוזת "Three" – אוסף תווים בין סוגריים מרובעים משמעו "אחד מבין התווים הללו". בין סוגריים מרובעים ניתן גם לרשום תחומי תווים, כך ש-[a-z] משמעו אחת האותיות מ-a עד z. הביטוי **t[a-z][a-z][a-z]** יתאים לכל המחרוזות שמתחילות באות t ואחריה 3 אותיות כלשהן. בטקסט הנ"ל:

```
And the Lord spake, saying, "First shalt thou take out the Holy Pin. Then
shalt thou count to three, no more, no less. Three shall be the number thou
shalt count, and the number of the counting shall be three. Four shalt thou
not count, neither count thou two, excepting that thou then proceed to three.
Five is right out. Once the number three, being the third number, be reached,
then lobbest thou thy Holy Hand Grenade of Antioch towards thy foe, who,
being naughty in my sight, shall snuff it.
```

תווים

הרכיב הקטן ביותר של ביטוי רגולרי הוא תו יחיד. מדובר בתווי הא"ב האנגלי (גדולות וקטנות), מספרים ותווים אחרים דוגמת <, > ו-&. ואולם, לתווים מסויימים יש שימושים מיוחדים בפייתון כפי שנראה בהמשך. כדי לחפש תווים אלה בביטוי רגולרי, יש להקדים להם \ המסמן שהכוונה לתו עצמו ולא לשימוש המיוחד שלו. סוגר מרובע שמאלי וגם '–' שהוצגו במבוא הם דוגמא לכך. הביטוי **t\[a\-z]** יתאים לטקסט t[a-z] (ורק לו).
אחד עשר התווים המיוחדים הדורשים \ לפניהם:

```
[ \ ^ $ . | ? * + ( )
```

ישנם תווים אשר אינם בני הדפסה אשר גם הם נרשמים בעזרת \ לפניהם. שלושה שימושיים במיוחד הם:

\t	Tab
\r	Carriage return
\n	Line feed

שימו לב שביוניקס (ובלינוקס) שורה מסתיימת ב-\n ואילו בחלונות שורה מסתיימת ב-\r\n.

לחובבי שפות שאינן אנגלית, ניתן לרשום תווי יוניקוד כך: \uFFFF. למשל התו א' בעל יוניקוד 0x05D0 ירשם \u05D0.

קבוצות תווים

קבוצות התווים כבר הוצגו במבוא של פרק זה. קבוצות תווים מסומנות בין סוגריים מרובעים. במקום בו רשומה קבוצת תווים יכול להתאים כל אחד מהתווים בקבוצה. לא נחזור כאן על הדוגמא מהמבוא.

בין הסוגריים המרובעים של קבוצת תווים, התווים המיוחדים אינם התווים המיוחדים הרגילים (ראו לעיל) אלא הארבעה הבאים:
`] \ ^ -`

ניתן להשתמש בקבוצת תווים לשלילה בעזרת הסימן ^ אחרי הסוגר המרובע השמאלי. הביטוי **`[^x]`** משמעו "תו שאינו x". הביטוי **`x[^y]z`** יתאים למחרוזות `xaz` ו-`xbz` אך לא ל-`xyz`.

למספר קבוצות תווים סימונים מקוצרים ואלה הן:

סימון	תיאור	רישום מלא
`\d`	ספרה	`[0-9]`
`\w`	תו אלפאנומרי	`[A-Za-z0-9_]`
`\s`	תו רווח – רווח, טאב ומעבר שורה	`[ \t\r\n]`
`.`	כל תו פרט לסוף שורה	`[^\r\n]`

שימוש באות גדולה במקום קטנה משמעו שלילת הקבוצה, למשל `\D` משמעו כל תו שאינו ספרה.

ניתן להשתמש בסימון מקוצר כזה בפני עצמו, או בתוך סוגריים מרובעים, כלומר `\d\d` משמעו שתי ספרות ברצף, ו-`[\dA-F]` משמעו ספרה או אחת האותיות `A-F` (כלומר ספרה הקסדצימאלית אחת).

עוגנים

עוגנים מתארים את המיקום שבין תווים.

הסימן `\b` מסמן גבול מלה – לפני תחילתה או אחרי סופה. אם המלה היא בתחילת מחרוזת, יקדם לה `\b` ואם היא בסוף מחרוזת יהיה אחריה `\b`. בתוך מחרוזת, ה-`\b` נמצא בין המלה לבין הרווחים שלפני ואחריה.
הסימן `\B` הוא השלילה של `\b`, כלומר הוא מתאים בכל מקום בו `\b` לא מתאים.
לדוגמא, נתבונן במחרוזת הבאה:
`This is an island`

בטבלא הבאה רשומים ביטויים רגולריים שונים ותת המחרוזות להן הם מתאימים במחרוזת:

`is`	`Th`**`is`** **`is`** `an` **`is`**`land`
`\bis\b`	`This` **`is`** `an island`
`\Bis`	`Th`**`is`** `is an island`
`is\b`	`Th`**`is`** **`is`** `an island`

<u>שאלה 11</u>

איזה תת מחרוזות מתאימות לביטויים הבאים עבור המחרוזת הנ"ל: (רשמו טבלא כמו הטבלא לעיל)

1. an
2. \ban
3. \Ban

הסימנים ^ ו-$ (שלא בתוך סוגריים מרובעים של קבוצת תווים) מסמנים את תחילתה ואת סופה של המחרוזת בהתאמה. הביטוי x^ יתאים רק ל-x שבתחילת המחרוזת והביטוי x$ יתאים רק ל-x שבסוף המחרוזת.

היררכיית ביטויים רגולריים

כל ביטוי רגולרי מורכב באופן היררכי מתת-ביטויים רגולריים. ביטוי רגולרי מכיל רצף ביטויים רגולריים והוא מתאים למחרוזת כאשר הביטויים מתאימים בזה אחר זה. כל תו הינו ביטוי רגולרי וביטוי רגולרי בן מספר תוים מורכב מביטויים רגולריים בני תו יחיד.

אופרטורים מיוחדים מרחיבים את הכוח של ההיררכיה. כדוגמא ראשונה, נתבונן באופרטור |, שמשמעו "או", כלומר או הביטוי הרגולרי שמשמאלו, או הביטוי הרוגלרי שמימינו. דוגמא:

th.|\b..th	cut down **the** mightiest tree in **the** forest **with**... a herring

ניתן לעטוף תת ביטוי רגולרי בסוגריים. דוגמא:

the mightiest|the forest	cut down **the mightiest** tree in **the forest** with... a herring
the (mightiest|forest)	cut down **the mightiest** tree in **the forest** with... a herring

חזרות

האופרטורים הבאים משמעם חזרה מספר מוגדר של פעמים על הביטוי הרגולרי שלפניהם. שימו לב שכל תו הוא ביטוי רגולרי, לכן יש להקפיד להקיף בסוגריים את כל הביטוי שרוצים שיחזור.

אופרטור	מספר חזרות	
?	0-1	ביטוי אופציונאלי
*	0-∞	מספר כלשהו של חזרות (אולי אפס)
+	1-∞	לפחות חזרה אחת
{m,M}	m-M	
{m,}	m-∞	
{,M}	0-M	

דוגמאות:

[1-9]\d*	מספר שלם (ללא אפס בהתחלה)
[1-9]\d{3}	מספר שלם 4 ספרתי
\b\d{1,8}-\d\b	מספר זהות עם ספרת ביקורת מופרדת על ידי מקף

האופרטורים של חזרה הינם greedy, כלומר הביטוי המתאים הוא זה עם המספר המקסימאלי של חזרות. דוגמא – נרצה לזהות ביטויים המוקפים סוגריים:

\(.+\)	0***(1+2)*(3+4)**

אך במקום זאת נקבל את כל הביטוי המודגש, כיוון שה-+ ניסה למצוא כמה שיותר חזרות לפני ה-(. הפתרון – הפיכת החזרה ל-lazy, כך שהחזרה תנסה להתאים למספר המינימאלי של פעמים. עושים זאת על ידי הוספת ? אחרי אופרטור החזרה (זה עובד גם על *, {} ו-?). עם הדוגמא הקודמת:

\(.+?\)	0***(1+2)*(3+4)**

שאלה 12

רשמו ביטויים רגולריים למציאת המחרוזות הבאות:

1. מספר טלפון עם או בלי קידומת, כאשר מקף מפריד בין הקידומת לטלפון, וללא מקפים נוספים, כלומר 04-1234567 או 1234567.
2. כתובת אינטרנט (URL) המתחילה ב-http://, ואחריו שניים או יותר שמות מופרדים בנקודות, כאשר כל שם מורכב מאותיות abc בלבד.

כן: http://name1.name2.name3.name4.com

כן: http://name1.name2

לא: http://name1..name2

לא: http://name1

לא: name1.name2.name3

רפרנס לביטוי קודם

בקובצי html התגים … משמשים לסימון טקסט אשר ייראה מודגש (bold). תג דומה עם התו u במקום ה-b מסמן טקסט עם קו תחתי.

דוגמא: הטקטס הבא:

```
cut down <u>the mightiest tree <b>in the forest</b></u> with... <u>a
herring</u>
```

ייראה בדפדפן אינטרנט כך:

cut down the mightiest tree **in the forest** with... a herring

כעת נרצה לרשום ביטוי רגולרי אשר יימצא ביטויים המוקפים בתגים. לשם כך, עלינו לחפש תחילה תג פותח ואחריו תג סוגר. ניסוי ראשון ולא מוצלח:

<\w>.+</\w>	cut down **<u>the mightiest tree in the forest</u> with... <u>a</u> herring**

תוצאה לא טובה – קיבלנו ביטוי שמתחיל ב-<u> ונגמר ב-.

עלינו לדרוש שהאות בתג הסוגר תהיה אותה אות של התג הפותח. לשם כך נשתמש ב-backward reference. כבר ראינו בפרק על היררכיית הביטויים הרגולריים כי ניתן להקיף תת ביטוי רגולרי בסוגריים. תת ביטויים המוקפים סוגריים ממוספרים באופן אוטומטי 1, 2, … . ניתן לרשום \x כדי לדרוש מופע נוסף של ביטוי אשר התאים לסוגריים מספר x. לפיכך הביטוי הדרוש לנו עבור המקרה הנ"ל אמור להיות:

<(\w)>.+</\1>	cut down **<u>the mightiest tree in the forest</u> with... <u>a</u> herring**

אולם גם הוא אינו מתאים. בתרגיל להלן תפתרו את הבעיה.

שאלה 13

1. השתמשו בכלים שבידכם כעת כדי לרשום את הביטוי הנכון למציאת טקסט המוקף בתגים מתאימים. עבור הדוגמא לעיל התוצאה אמורה להיות (3 התאמות, כל אחת מסומנת בקו תחתי):
```
cut down <u>the mightiest tree <b>in the forest</b></u> with...
<u>a</u> <b>herring</b>
```
2. בידנו מסד נתונים עם רשומות שקריות. נרצה לנצל את הידע החדש שלנו בביטויים רגולריים כדי לזהות רשומות חשודות.
   2.1. רשמו ביטוי לזיהוי כתובת אימייל בה המחרוזות לפני ואחרי הכרוכית (שטרודל) זהות.
   2.2. רשמו ביטוי לזיהוי שם מלא (ראשון + שני + משפחה) ששלושת חלקיו זהים.
   2.3. רשמו ביטוי לזיהוי שם מלא (ראשון + שני + משפחה) בו שם המשפחה זהה לשם הראשון או לשני.

ביטויים רגולריים בפייתון

כדי להשתמש בביטויים רגולריים בפייתון עלינו לייבא את המודול `re` בעזרת הפקודה `import re`. כאן נתמקד בשתי מתודות בלבד – `findall` המוצאת ביטוי, ו-`sub` המחליפה ביטויים במחרוזת. נלמד גם שני טריקים שקצת יקלו עלינו את החיים.

מחרוזות גולמיות

ראשית נציין כלי המקל במעט על קריאת ביטויים רגולריים בפייתון – מחרוזות גולמיות (`raw`). בעת הצבה למחרוזת, פייתון מתרגם באופן אוטומטי escape codes. כך `\t` הופך לטאב, `\n` לשורה חדשה ו-`\\` ל-`\`. פייתון מאפשר (באופן כללי, בלי קשר לביטויים רגולריים) להציב ערך למחרוזת ללא התרגום הזה. כדי לסמן לפייתון לעשות זאת, יש להקדים `r` למחרוזת כך:

```
pattern = r'\bword\b'
```

במקום `pattern = '\\bword\\b'`.

חיפוש ביטויים

המתודה `re.findall(pattern, text)` מחפשת את כל ההתאמות של הביטוי הרגולרי `pattern` בטקסט `text` ומחזירה אותן כרשימת מחרוזות. אם הביטוי מכיל קבוצות (מוקפות בסוגריים), מוחזרת רשימה אשר כל איבר שלה הוא `tuple`.

```
>>> print re.findall(r'(\d+)\D+(\d+)', "12a3 bc 3de4")
[('12', '3'), ('3', '4')]
```

החלפת ביטויים

המתודה `re.sub(search, replace, text)` מחפשת בטקסט `text` התאמות לביטוי `search` ומחליפה אותן במחרוזת `replace`. אם המחרוזת `replace` מכילה escape codes מהסוג `\x`, הם מוחלפים בהתאמה ה-`x` של הביטוי.

```
>>> re.sub(r'(\D*)(\d+)(\D*)(\d+)', r'\1\2\3\2', "ab 12 cd 34")
'ab 12 cd 12'
```

רישום מפורט של ביטויים רגולריים

לא קל לרשום ביטויים רגולריים, אך הרבה יותר קשר לקרוא אותם. כדי להקל על שני הקשיים, ניתן לרשום ביטוי רגולרי באופן מפורט תוך שילוב הערות.
לדוגמא, הביטוי הבא משמש לזיהוי מספרי טלפון ישראלים:

```
^((972-?)|0)\d{1,2}(-?\d){7}$
```

נרשום אותו כביטוי מלא כך:

```
pattern = """
    ^                       # Line start
    ((972-?)|0)             # Either 972 (maybe with -) or 0
    \d{1,2}                 # Area code 1-2 digits (e.g. 2 or 54)
    (-?\d){7}               # 7 digits, maybe with some -'s
    $                       # Line end
"""
```

כאשר רושמים ביטוי רגולרי באופן כזה, יש להורות לפייתון להתעלם מטאבים, מעברי שורה, רווחים והערות. יש להעזר ב-escape codes אם רוצים לכלול רווחים בביטוי. כדי שפייתון ידע שהביטוי במשתנה `pattern` של פונקציות `re` הוא ביטוי מלא, יש להציב את הדגל `re.VERBOSE` כפרמטר אחרון כך:

```
>>> re.search(pattern, '972591234567', re.VERBOSE)
```

ביבליוגרפיה

1. www.python.org
האתר הרשמי. הורדת התוכנה, קישורים ופרסומות.
2. www.eclipse.org, pydev.org
סביבת העבודה אקליפס והתוסף פיידב לתכנות פייתון. הורדות והוראות.
3. docs.python.org
מדריכים, רפרנסים וכיו"ב. מקיף ביותר.
4. Mark Lutz, *Learning Python*, Oreilly, 2009
פייתון כשפה ראשונה.
5. http://diveintopython.org
מדריך פייתון מהיר, מעט מעמיק יותר מחוברת זו. **מומלץ**.
6. http://www.regular-expressions.info/tutorial.html
קורס ביטויים רגולריים. **מומלץ**.
7. http://regexpal.com
כלי ווב לבדיקת ביטויים רגולריים.
8. http://xkcd.com/353
תמונת השער.