**פייתון - Python**

שפת תכנות דינמית
נכתבה בקוד פתוח - חינמית, מערך תמיכה גדול באינטרנט, הרבה ספריות שמספקות קוד שאפשר להשתמש בו כדי לבנות תוכניות מורכבות יותר.

**כתיבת הקוד:**
ביצוע פעולות פשוטות (ב-interpreter) -

פעולות חשבוניות - הפעולות הבסיסיות מבוצעות ע״י הסימנים המוכרים /,*,-,+, חזקה נכתבת ע״י ** דוגמה: 2+2, 16/4, 3**7

כתיבת מחרוזת תווים - נעשית ע״י גרש בודד או גרשיים - 'string', "string". הדפסה של המחרוזת נעשית ע״י פקודת print. דוגמה: print "hello world"

אחת התכונות הבסיסיות שהופכות מחשבון למחשב היא השימוש בזיכרון. לשם שימוש בזיכרון המחשב אנו מגדירים משתנים, כלומר מקצים קטע מזכרון המחשב באופן שמאפשר לשמור ערכים ולהשתמש בהם.
הסימן = מהווה סימן לשמור ערך לזיכרון (בשונה ממתמטיקה)
דוגמה: הפקודה x=17 מגדירה מקום בזיכרון, שנותנים לו את השם ״x״ ובתוך המקום הזה כותבים את הערך 17.
שימו לב כי פייתון רגיש לשינויי אות גדולה / אות קטנה - x ו-X אינם אותו משתנה.
דוגמה נוספת של הכנסת ערך למשתנה: y='hello'
עכשיו המשתנה שומר בזיכרון מחרוזת, אוסף של תווים. כלומר יש סוגים שונים של משתנים, כאשר עבורנו תספיק החלוקה הבסיסית הבאה: משתנים מספריים (טבעיים - 1,5,132, רציפים - 1.234, ועוד), מחרוזות (טקסט) או ערכים לוגיים.

ערך לוגי (אמת/שקר) - מתקבל ע״י הסימן == לשיוויון, או <,>,=<,=> (קטן / גדול / גדול או שווה / קטן או שווה). דוגמה: 0==1 ייתן False, 2<17 ייתן True

**כתיבת סקריפט או תוכנית**
תוכנית או סקריפט מהווה אוסף של פקודות שכולם יתבצעו בסדר בו כתוב הקובץ (כמו רצף קריאה). כדי לאפשר לתוכנית מחשב לתת תוצאה לבעיות מורכבות יש צורך לאפשר פיצול של הבעיה. לשם כך משתמשים במשפטי בקרה. ישנם שני סוגים מרכזיים של הוראות. הראשון הוא משפט בקרה שמבצע קטע קוד בתנאי (if). השני מבצע חזרה על אותו קטע קוד מספר פעמים (for/while loops).

**פקודות בקרה**
**משפטי תנאי - if** - לפעמים נרצה לבצע פעולה **אם** תנאי כלשהו מתקיים. התנאי שניתן צריך להיות ערך לוגי, כלומר - לקבל ערך אמת או שקר. לדוגמה: אם x אינו שלילי, הצג את x:

```
if x>0:
        print x
```

ניתן לכתוב תנאי מסובך יותר, ע״י שימוש ב-elif (״אחרת, אם…״) ו-else (״אחרת…״):

```
if x>0:
        print x
elif x<0:
        print abs(x)
else:
        print 'x is not positive and not negative'
```

**לולאות - for, while** - לפעמים נרצה לבצע פעולה **כל עוד** תנאי מתקיים. לשם כך משתמשים בלולאות.

for - לולאת "עבור". לדוגמה: עבור x שלם מ-1 עד 5, הצג את x בריבוע (דוגמה לקוד בהמשך העמוד).

while - לולאת "כל עוד". לדוגמה: כל עוד x<10, הדפס את הערך של 2*x. חשוב - במקרה כזה נרצה לשים פקודה בתוך הלולאה שתקדם את המשתנה עליו מוגדר התנאי, אחרת הלולאה לא תפסיק לרוץ! למשל, בדוגמה לעיל נרצה לכתוב בתוך הלולאה את הפקודה x=x+1, כלומר - בכל ריצה של הלולאה נגדיל את ערך המשתנה x באחד. בנוסף, יש להגדיר ערך התחלתי למשתנה עליו רצה הלולאה, למשל - x=0. הלולאה תיכתב באופן הבא:

```
x=0
while x<10:
        print 2*x
        x=x+1
```

בכל פעם שניתן לעבוד עם לולאת for נעדיף עבודה כזו על פני while, על מנת להימנע מהמצב שבו הלולאה תרוץ לנצח.
ההיררכיה בתוך הלולאה נעשית ע"י הזחה (אינדנטציה) - באמצעות מקש Tab.

**מושג הפונקציה**
פונקציה היא קטע קוד, שמקורו במושג הפונקציה המתמטית. הפונקציה מאפשרת לכתוב קטע קוד המקבל קלט כללי, ולהשתמש בו הרבה פעמים עבור קלטים שונים. לדוגמה, אם רוצים לחשב exp(x) עבור ערכים שונים. אבל הגדרה זו נכונה לכל קטע קוד לא רק לחישוב פונקציה מתמטית.
בשלב זה של הקורס לא נכתוב פונקציות משל עצמנו, אלא נשתמש בפונקציות הקיימות בספריות.

**עבודה עם ספריות:**
בפייתון מגוון רחב של ספריות שמיועדות לשימושים רבים ושונים. בקורס אנחנו נשתמש בעיקר בשתי ספריות שנועדו לתחום המתמטיקה והמדעים בשם NumPy ו-Matplotlib.
הספריה NumPy (קיצור ל-Numerical Python) מאפשרת עבודה עם משתנים מתמטיים (כמו - וקטורים, מטריצות) ועם פונקציות מתמטיות (כמו - סינוס, חזקה, כפל וקטורי)
הספריה Matplotlib נועדה לאפשר המחשה גרפית - שרטוט פונקציה, ציור כיווני זרימה של מערכת, מפות גובה ועוד.

<u>כדי לאפשר שימוש נוח בספריות אלו נתחיל תמיד את העבודה בשתי הפקודות הבאות</u>:

```
import numpy as np
import matplotlib.pyplot as plt
```

(גם ב-interpreter וגם ב-editor). פקודת ה-import מיידעת את המחשב שאנו מתכוונים לבצע שימוש בספריות אלו. הקיצורים np ו-plt נועדו לאפשר גישה נוחה - לאחר ביצועהפקודות הנ"ל, נוכל לגשת לפונקציות מ-numpy ע"י הקיצור np.Name ולפונקציות מ-matplotlib ע"י הקיצור plt.Name.

**פונקציות שימושיות ב-NumPy** (נשתמש בכתיב המקוצר np.[…])
np.ones() / np.zeros() - יוצרות וקטור או מטריצה של אפסים / אחדים בגודל הרצוי. לדוגמה:
np.zeros(5) ייתן וקטור של חמישה אפסים, np.ones((2,3)) ייתן מטריצה בגודל 2x3 של אחדים.
np.arange(start,end,step) - יוצרת וקטור של ערכים מ-start עד end, לא כולל הערך של end, במרווחים של step. שימושי ללולאות for, כמו הדוגמה שניתנה למעלה:

```
for x in np.arange(1,6,1):
        print x**2
```

np.power - חישוב חזקה - שימושית עבור ערכי חזקה לא שלמים. לדוגמה: np.power(4,1.5) ייתן 8
np.exp(),np.cos(),np.sin() - פונקציות מתמטיות - סינוס, קוסינוס, אקספוננט
np.size() - מחזירה את כמות הערכים שיש בוקטור / מטריצה. בקורס נשתמש בפונקציה עבור וקטורים בלבד. עבור וקטור x, np.size(x) ייתן את אורך הוקטור.

**הערה - על וקטורים / מטריצות בתכנות**: בעוד אנו כמתמטיקאים רגילים לחשוב על וקטורים ומטריצות כיישויות מתמטיות, כאשר מדובר על וקטור או על מטריצה בכתיבת תוכנה הכוונה היא

לרוב לשורה (וקטור) או לטבלה (מטריצה) של ערכים, אשר כולם מאותו סוג משתנה (למשל - מספרים עשרוניים). זה מאפשר לבצע את אותה פעולה על כמות גדולה של ערכים בצורה מקוצרת. לדוגמה, הפקודה

```
x=np.arange(1,5,1)
```

תתן לנו את הוקטור הבא:

```
x=(1,2,3,4)
```

(אם נסתכל ב-Variable explorer ב-Spyder, נראה שכתובה לפני הוקטור המילה array, שפירושה מערך. זהו המינוח המקובל לוקטורים ומטריצות במדעי המחשב.)
כעת נוכל לבצע פעולות חשבון עם הוקטור:

```
y=x+1
```

ייתן וקטור y בו לכל ערך ב-x נוסף הערך 1.

```
z=x+y
```

ייתן וקטור z בו כל ערך הינו הסכום של הערכים במיקום המתאים של x ושל y.
ניתן לבצע גם פעולות חשבון מורכבות יותר:

```
np.sin(x), x**2, …
```

כל פעולה תבוצע על כל אחד מערכי הוקטור בנפרד.
בנוסף, מכיוון ש-NumPy הינה ספריה מדעית, קיימות פונקציות המאפשרות לבצע חשבון וקטורי כפי שאנו מכירים ממתמטיקה, על וקטורים כפי שהוגדרו לעיל. בקורס לא נידרש לפונקציות אלו… :) מי שמעוניין בכל זאת ללמוד יותר מוזמן לשלוח מייל למיכל.

**פונקציות שימושיות ב-matplotlib** (נשתמש בכתיב המקוצר […].plt)
plt.plot(x,y) - מציג גרף דו-מימדי של y מול x. x,y יכולה להיות נקודה או שני וקטורים באותו האורך, לדוגמה: x=np.arange(0,10,0.1) ו-y=np.sin(x)
plt.hold('on') - פוקד על הגרף "להמתין" (to hold) - מאפשר הצגה של ערכים נוספים על אותו גרף. ללא פקודה זו כל פקודת plt.plot תאפס את התצוגה!
plt.grid('on') - יציג רשת קואורדינטות על גבי הגרף
plt.xlabel('x') / plt.ylabel('y') - נותן תוויות שם לצירים - x ו-y
plt.title - נותן כותרת לגרף (בדומה לתוויות של הצירים, את הכותרת כותבים בסוגריים בין גרשיים)
plt.legend - נותנת מקרא לנתונים שהוצגו בגרף. סדר הערכים במקרא הינו הסדר בו הנתונים הוצגו בגרף. אופן הכתיבה הינו plt.legend(['data1','data2',…]).

דוגמה: ציור הפונקציה y=cos(x) בתחום הערכים [0,2pi] יבוצע באופן הבא (שימו לב - הערך של פיי נלקח מתוך הספריה המדעית NumPy, ולכן נכתב כ-np.pi):

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0,2*np.pi,0.01)
y = np.cos(x)

plt.plot(x,y)
```

ונוכל להשתמש בשאר הפקודות לעיל על מנת להוסיף תוויות שם לצירים, כותרת לגרף, ועוד.
במסמך ההתקנה והעבודה עם פייתון מופיעות גם הוראות כיצד נוכל לשמור את הגרף שקיבלנו לקובץ.