שאלות תאורטי

להבדלים בין high order ו- first class, ראו [כאן](http://stackoverflow.com/questions/10141124/any-difference-between-first-class-function-and-high-order-function)

עבודה 3 2017

א. **פונקציה מסדר גבוה** (high order function) ניתן להחזיר מפונקציה ולהעביר לפונקציה כארגומנט - **לא נכון.**

ב. בפייתון 3 משתמשים בהצהרה nonlocal ע"מ לעדכן קשירה של משתנה במסגרת הגלובלית - **לא נכון.**

ההצהרה nonlocal תשנה אותנו לעדכן קשירה של משתנה במסגרת כלשהי מעל המסגרת הנוכחית ועד המסגרת הגלובלית, לא כולל. על המפרש לחפש את המשתנה בסביבה הקרובה ביותר בעל אותו שם.

ג. בשפות עם lexical scoping ניתן לממש טיפוסי נתונים חדשים שהם mutable תוך שימוש בפונקציות והשמה לא לוקאלית (non local). - **נכון**

כפי שראינו בהרצאה 6, ישנם טיפוסים שהם immutable – כלומר לאחר שהוגדרו, לא ניתן

לשנות את ערכם ) tuples, strings למשל(.

קונספט ה- lexical scoping )המתאפשר בפייתון באמצעות מודל הסביבות( מאפשר מעבר

בין מספר סביבות, ושינוי קשירות גם בסביבות מחוץ לסביבה המקומית.

כך אפשר להגדיר טיפוסים חדשים, ובאמצעות מניפולציות )"מוטציות", בלשון ההרצאה(

לשנות את ערכם לאחר הגדרתם.

בהרצאות אנו עובדים עם mutable rlist , אשר מקבלת "רשימה" ריקה המוגדרת מקומית,

ובאמצעות הפונקציות push, pop - בהן מתייחסים אל הרשימה בצורה לא-לוקאלית –

לשנות את ערך הרשימה.

מועד א' 2016

א. **message passing-**  זאת שיטה למימוש טיפוסי נתונים שלא ניתנים לשינוי

( immutable ) – **לא נכון**

ע"י שיטה זו ניתן לממש גם טיפוסים שכן ניתנים לשינוי. למשל טיפוס המכיל רשימה של מילים, ושיטה המקבלת מילה חדשה ומוסיפה אותם לרשימה הקיימת באמצעות append ולא באמצעות דריסת הרשימה הקיימת.

ב. ב- Python 3 קיימת הצהרה **local** ומשתמשים בה על מנת לעדכן קשירה של משתנה

במסגרת לוקאלית- **לא נכון**

לא קיימת הצהרה local בפייטון 3 ובנוסף לכך על מנת לעדכן קשירה של אותה מסגרת לוקאלית אין צורך בפקודה כלשהי כי זו פעולת השמה.

ג. במפרש של מחשבון ) calculator ) שכתבנו בכיתה קיימת רקורסיה הדדית בין פונקציות **calc\_eval** ו- **calc\_apply – לא נכון**

כי אין בינהם רקורסיה הדדית. Calc eval שולחת קריאה לפונקציה calc apply אך זו לא קוראת לה בחזרה.

ד. **conventional interface** זה עיקרון הפרדה בין מימוש של טיפוס נתונים לבין שימוש

בו. **לא נכון**

זהו עיקרון של עבודה משותפת על ערך מסויים כמו pipeline המבצע מס' פעולות על קלט ומחזיר בסופו של דבר את הפלט הרצוי.

ה. **data abstraction** זה עיקרון הפרדה בין מימוש של טיפוס נתונים לבין שימוש בו. **נכון** זה עיקרון שבו אנו ממשים את הטיפוס ואת כל ההגדרות החלות על סוג טיפוס מסויים.

מועד ב' 2016:

לגבי כל טענה – קבעו האם נכונה או לא ונמקו:

א. מטרת memoization היא לייעל חישוב רקורסיבי ע"י הקטנת ניצול זיכרון על חשבון זמן ריצה. **לא נכון**.  
בדיוק להיפך - memo' מייעלת חישוב רקורסיבי בכך שהיא משתמשת ביותר זיכרון פיזי כדי להקטין את זמן הריצה.

ב. ב- 3 Python משתמשים בהצהרה nonlocal על מנת לעדכן תוכן של מבנה נתונים (למשל מילון - dictionary) בסביבת כוללת (לא כולל הגלובלית). **נכון.**הצהרה על משתנה כ-nonlocal מגדירה למפרש שעליו לחפש בסביבות "בדרך" אל הסביבה הגלובלית (לא כולל) את המשתנה המדובר, ואם המתכנת יבחר לעדכן את תוכנו, אזי תוכנו ישתנה גם בסביבה בה הוגדר.

ג. במפרש של מחשבון (calculator) שכתבנו בכיתה, הפונקציה **calc\_val** מממשת **רקורסית עץ** ע"מ להעריך את הארגומנטים של הביטוי המוערך. **נכון.**כל ביטוי כולל פעולה ושני אופרנדים. כל אחד מהאופרנדים יכול להיות פעולה בעצמו, ועל כן יש לממש רקורסית עץ, בה כל קודקוד הוא אופרטור ורק העלים הם מספרים של ממש.

ד. **פולימורפיזם בהורשה** ממומש בעזרת ממשק משותף בין הטיפוסים. **לא נכון**.  
ממומש בעזרת הסינטקס הבא:

class child(parent):

ה. פונקציות גנריות (**generic functions**) המוגדרות על פרמטרים מטיפוסים זרים (**parametric polymorphism**) ממומשות ע"י הגדרת ממשק משותף בין הטיפוסים של הארגומנטים.**נכון??**

מועד א' 2015:

א. פונקציה מסדר גבוה (high-order function) ניתן להחזיר מפונקציה ולהעביר לפונקציה

כארגומנט. **נכון**

כי פונקציה מסדר גבוה ניתן להחזיר פונקציה ולשלוח פונקציה כארגומנט.

--לפי מצגת 3: ההגדרה הנתונה בשאלה למעשה צריכה להיות מנומקת כך: בפייתון כל פונקציה או משתנה (אלמנט) הם first class elements, וככאלה, ניתן להעביר אותם כארגומנטים או להחזיר ואתם מפונקציות.

ב. ב- Python 3 משתמשים בהצהרה **nonlocal** על מנת לעדכן קשירה של משתנה

במסגרת גלובאלית. **לא נכון**

על מנת לעדכן משתנה שנמצא בסביבה גלובאלית נשתמש בפקודה global.

ג. במפרש של מחשבון ) calculator ) שכתבנו בכיתה פונקציית eval מזמנת פונקצית apply על מנת להעריך את הביטוי המתקבל מהצבת ערכי ארגומנטים לפרמטרים פורמליים של הפונקציה המופעלת.**נכון**

פונקציית calc eval מזמנת את פונקציית calc apply על מנת להעריך את הביטוי המתקבל מהצבת ערכי הארגומנט לפרמטרים פורמליים של הפונקצייה המופעלת. לאחר מכן פונקציית calc apply מציבה את האופרטורים הפורמליים ומחזירה חישוב של הביטוי המוערך.

ד. נתונה פונקציה A שמפעילה את פונקציה B . לפי מודל הסביבות Environment Model

ו- Lexical Scoping הפעלת פונקציה B יוצרת מסגרת חדשה שמצביעה על סביבת

הגדרה של A . **לא נכון**

בכל הפעלת פונקציה יוצרת סביבה(מסגרת) חדשה ואם פונקציה B אינה מוגדרת בפונקציה A אז הפעלת הפונקציה B מתוך A לא תפתח סביבה שתצביע לA אלא למקום בו הוגדרה.

ה. בשפות עם Lexical Scoping ניתן לממש טיפוסי נתונים חדשים שהם mutable תוך

שימוש בפונקציות והשמה לא לוקאלית nonlocal. **טענה נכונה**

ניתן לממש טיפוסי נתונים חדשים שהם mutable כי דרך nonlocal נוכל לגשת למשתנים ולשנות אותם.

מועד ב' 2015:

1. מטרת **memoization** היא לייעל חישוב רקורסיבי ע"י הפיכת רקורסיה לרקורסית זנב**. לא נכון.**   
   מטרת memorization היא לייעל חישוב רקורסיבי ע"י שמירת ערכי הרקורסיה על מנת להפחית זמן ריצה.
2. נתונה פונקציה A שמפעילה את פונקציה B . לפי מודל הסביבות( Environment Model ) ו- Lexical Scoping הפעלת פונקציה A יוצרת מסגרת חדשה שמצביעה על סביבת הגדרה של B**. לא נכון.**   
   הפונקציה המופעלת A תצביע לסביבה בה הוגדרה (אין בכלל קשר ל-B).
3. ב- Python 3 משתמשים בהצהרה **global** על מנת לעדכן קשירה של משתנה במסגרת גלובאלית. **נכון.**   
    global משמשת לעדכן משתנה במסגרת גלובאלית, אם לא נשתמש בגלובל אז הוא ישנה את המשתנה שנמצא במסגרת המקומית ואם לא קיים אז זו שגיאה.
4. במפרש של מחשבון (calculator) שכתבנו בכיתה ניתן להגיש כמה ביטוים בבת אחת, אך לקבל ערך של ביטוי האחרון בלבד**. לא נכון.**ניתן להגיש ביטוי אחד ולקבל ביטוי אחד מוערך (אם נותר משהו, כל דבר, לאחר הסוגר הימני הראשון - נקבל שגיאת סינטקס).
5. בשפות עם **Lexical Scoping** כל הפעלה של אותה פונקציה תקבל סביבה שונה אם פונקציות שונות מפעילות אותה (למשל, הפעלת פונקציה A מפונקציה B, והפעלת אותה פונקציה A מפונקציה C) **נכון.**  
   הפעלה של פונקציה תמיד תיצור סביבה חדשה והיא תצביע לסביבת הגדרתה.

מועד א 2014:

1. פונקציה מסדר גבוה (**high-order function**) לא ניתן להחזיר מפונקציה ולהעביר לפונקציה כארגומנט**. לא נכון.** פונקציה מסדר גבוה ניתן להחזיר מפונקציה ולהעביר לפונקציה כארגומנט.  
   -- תיקון אפשרי: בפייתון, כל אלמנט הוא first order element, וככזה ניתן להעביר אותו אותו לפונקציה כארגומנט או להחזיר אותו. ראו מצגת 3.
2. ב- **Python** לא ניתן להשתמש בערכו של משתנה שמוגדר במסגרת מעל המסגרת
3. הנוכחית ללא שימוש בהצהרה **nonlocal לא נכון.**   
   עדיין ניתן להשתמש הצהרה global אשר הולכת למסגרת הגלובלית - המסגרת העליונה ביותר.
4. זמן ריצה (סיבוכיות) של חישוב עצרת כרקורסיית זנב יותר טוב מאשר חישוב עצרת כרקורסיה רגילה (לינארית). **לא נכון.**רקורסיית זנב חוסכת בניצול זיכרון (בחלק מהקומפיילרים)ן אך בכל מקרה לא בזמן ריצה. זמן הריצה נשאר זהה בשתי השיטות.
5. לפי מודל הסביבות (**Environment Model**) הפעלת פונקציה יוצרת קשירה חדשה לשם של הפונקציה במסגרת שמרחיבה את הסביבה הנוכחית. **לא נכון.** הצהרה חסרת משמעות. הפעלת פונקציה לא יוצרת קשירה חדשה אבל כן יוצרת מסגרת חדשה המרחיבה את הסביבה הנוכחית.
6. במערכת אובייקטים שבנינו בכיתה (של **Shmython**) לא ניתן להוסיף פונקציות למאפיינים של מופע (**instance** **attributes**) **. לא נכון.** ניתן להוסיף פונקציה למאפיינים של מופע ע"י הוספת ערך למילון ומימוש של הפונקציה. (למשל, (obj ['set'] ('function\_name',function מתחת להגדרת המילון obj.

מועד ב 2014:

א.בשפות עם Lexical Scoping ניתן לממש טיפוסי נתונים חדשים שהם mutable תוך שימוש בפונקציות בלבד (ללא שימוש בדברים נוספים). **לא נכון**.  
 חסרה הצהרת nonlocal למימוש טיפוס mutable. --

ב. ב- **Python** לא ניתן לשנות ערך של משתנה שמוגדר במסגרת מעל המסגרת הנוכחית ללא שימוש בהצהרה **nonlocal.** **נכון**.  
 על מנת לשנות ערך של משתנה שמוגדר במסגרת מעל המסגרת הנוכחית חייב להשתמש בהצהרה nonlocal. --מה עם GLOBAL? תלוי אם זה "ישירה מעל" או "כלשהי מעל".

ג. בשפות בעלי **Lexical Scoping** לכל פונקציה בסדר גבוה (**high-order**) שמגדירה ומחזירה פונקציות חדשות, יש גישה למשתנים לוקאליים של אותן פונקציות חדשות. **לא נכון**.  
 כיוון שפונקציות בסדר גבוה תחפש את המשתנים במסגרת שלה או במסגרות עליוניות ולא במסגרות התחתונות שיפתחו.

ד.לפי מנגנון של הפשטת נתונים (**data abstraction**) מתכנת חייב לראות מימוש של טיפוסי נתונים על מנת להשתמש בהם בתוכנית שלו **לא נכון.**   
בdata abstraction יש "מחסומים" שבגללם מתכנת שלא תכנת את הקוד לא יוכל לדעת כיצד מומשו אך ניתן יהיה להשתמש בהם. (כמו למשל איך ממומשת str.lower או isnumeric)

ה. בהינתן a,b = X,Y. ו-a.func(2). מה יהיה הפלט של b.func(2). **לא נכון.**   
תלוי מה יהיה בפונקציה b אם היא קיימת בכלל.

מועד א' 2013:

1. פונקציה מסדר גבוה (**high-order function**) זוהי פונקציה מובנת (**built-in**) של שפה, שמקבלת פונקציות משתמש כארגומנטים. **לא נכון**.   
   זה לא בהכרח פונקציה מובנית של השפה, אלא כל פונקציה המקבלת פונקציות אחרות כארגומנטים, ו\או מחזירה פונקציות כארגומנטים.
2. ב- **Lexical Scoping** ניתן להחזיר פונקציה מפונקציה אחרת, אך הפונקציה המוחזרת תהיה ללא סביבה. **לא נכון.** פונקציה שמוחזרת היא תמיד עם סביבה.
3. מפרש של שפות פשוטות אינו חייב לממש רקורסיה הדדית בין **eval** ו- **apply** .**נכון**. הפעלת פונקציה מתבצעת בסביבה חדשה שמרחיבה את הסביבה הקיימת של הפונקציה המופעלת.כלומר מתבצעת ביחס לסביבה הנוכחית שמתחילה במסגרת חדשה.
4. **נכון**, לדוגמא מפרש של מחשבון שכתבנו בכיתה : eval קורא לapply אך apply לא קורא לeval.
5. ניתן באופן כללי לקחת פונקציה רקורסיבית המבצעת חישובים חוזרים על אותם פרמטרים ולשפר את זמן הריצה שלה ללא שינוי קוד הפונקציה (בשפה עם פונקציות שהן אובייקטים מסדר ראשון) **.נכון**.   
   memorization משפרת את זמן הריצה ע"י שמירת ערכים חוזרים במילון וכך שואבת יותר זיכרון למעשה. חשוב מאוד להדגיש כאן שיש צורך של השפה להיות מסדר ראשון. בשפות שאינן כאלה, עדיין נוכל לממש memo, אך נהיה חייבים לשנות את הקוד של הפונקציה אותה נרצה לשפר.

מועד' ב' 2013:

1. ב-python 3 משתמשים בהצהרה **global** ע"מ לעדכן קשירה של משתנה במסגרת X מעל המסגרת הנוכחית (כאשר X איננה בהכרח המסגרת הגלובלית) **לא נכון**.  
    בהצהרה global משתמשים כדי לעדכן משתנה שנמצא במסגרת גלובאלית בלבד.
2. אובייקטים מסדר ראשון (first order object\element) ניתן להחזיר מפונקציה ולהעביר לפונקציה כארגומנט.  **נכון**.  
    בפייטון ניתן להעביר כל אובייקט מסדר ראשון כארגומנט לפונקציה ולהחזיר אותו. למשל פונקציה הוא גם אובייקט מסדר ראשון וניתן להעביר אותה ולהחזיר אותה מפונקציה אחרת. (למעשה, זה שלא שניתן להעביר כל אובייקט מסדר ראשון, אלא כל אובייקט הוא מסדר ראשון. יחי ההבדל הקטן.)
3. בשפות עם **lexical scoping** ניתן לממש טיפוסי נתונים חדשים שהם **immutable** בשימוש בפונקציות בלבד (הטריק בשאלה: האם אפשר להסתדר בלי **nonlocal**) **נכון**,  
    ניתן לממש טיפוסי נתונים חדשים שהם immutable בשימוש של פונקציות בלבד. אם נרצה לממש mutable נצטרך להשתמש גם בהצהרת nonlocal.  
   הערה: אני דווקא לא מסכים עם זה. כדי לממש פונקציות אשר משנות את מצב האובייקט הקיים, עלינו להשתמש בדברים שיראו ככה: X[index] = value. בטיפוסים שהם immutable (נגיד tuple) אי אפשר לעשות דבר כזה. חייבים לדרוס את הטיפוס הישן, ובשביל זה צריך השמה, ובשביל זה צריך nonlocal.
4. לפי מודל הסביבות (**Environment Model**) הגדרת פונקציה חדשה מוסיפה קשירה לסביבה הגלובלית. **לא נכון.** הגדרת פונקציה אכן מוסיפה קשירה חדשה אך לא דווקא למסגרת גלובלית.
5. מפרש של שפות תכנות כמו **python** מזמן את פונקצית eval מפונקצית apply ע"מ להעריך את הביטוי המתקבל מהצבת ערכי ארגומנטים לפרמטרים פורמליים של הפונקציה המופעלת. **נכון**.  
    המפרש של פייטון מפעיל רקורסיה הדדית בין eval לapply.