Mython - My Python Interpreter

משימה 1 - המבנה הכללי

במשימה זו נכיר את הפונקציה הראשית של האינטרפרטר, ולמעשה את האינטרצקיה עם המשתמש.
הוסיפו את הקבצים של המשימה ל-VS.

1.1

בקובץ interpreter.cpp ממומשת הפונקציה main עם הלוגיקה הבאה :

1. בהרצה של התכנית תחילה מוצגת השורה:
Mython - My Python Interpreter version 1.0 by [**YOUR NAME**]

2. לאחר מכן מתחילה **לולאה אינסופית** המאפשרת למשתמש להזין את שורות הקוד שלו. בתחילת השורה מוצגת המחרוזת " <<<" (ללא ירידת שורה לאחר מכן) כדי לתת למשתמש אינדיקציה שעליו להקליד. את הטקסט שמזין המשתמש אנו קולטים לתוך אובייקט מסוג std::string (באמצעות הפונקציה std::getline, קראו עליה אם אתם לא מכירים), ואת המחרוזת שולחים כפרמטר לפונקציה הסטטית parseString של המחלקה Parser שמוגדרת בקובץ parser.h. כרגע המימוש של הפונקציה הזו (שמופיע בקובץ parser.cpp) הוא פשוט הדפסה של המחרוזת שהתקבלה (במידה והאורך שלה אינו 0) - בהמשך אתם תשנו את המימוש.

א. קמפלו והריצו את התכנית (קראו לקובץ ההרצה Mython), ודאו (עם המדריך) שהיא עובדת כפי שצריך.

ב. שנו את הערך של הקבוע YOUR_NAME המוגדר בתחילת הקובץ interpreter.cpp למחרוזת עם שמכם.

ג. התכנית אמורה לסיים את ריצתה כאשר המשתמש מזין את הפקודה quit() . הריצו את התכנית ובדקו שזה עובד.

1.2

אחד הדברים החשובים בפייתון זה ההקפדה על הזחה נכונה. עדכנו את הפונקציה parseString שתבדוק אם המחרוזת שהתקבלה מתחילה ברווח. אם כן - עליה לזרוק אקספשן מסוג IndentationException (כבר ממומשת), אשר יורשת מ-InterperterException. בלולאה הראשית (בפונקציה main) עליכם לקרוא לparseString בתוך בלוק של try ובמקרה שנזרק האקספשן הנ"ל עליכם לתפוס אותו ולהדפיס את הודעת השגיאה שמחזירה הפונקציה what שלו.

משימה 2 - טיפוסים

האינטרפטר שלנו צריך לתמוך בשלב זה בארבעה סוגי משתנים:

Integer - מספר שלם - חיובי שלילי או אפס, בבסיס עשרוני

Boolean - ערך בוליאני - True /False

String - מחרוזת - אוסף של תווים תחומים בין שני גרשיים או בין שני גרש בודד

Void - ערך ריק

2.1 הוספת אקספשיין

בקובץ SyntaxException.h הוסיפו מחלקה חדשה בשם SyntaxException היורשת את InterpreterException . על הפונקציה what של מחלקה זו להחזיר את המחרוזת:
"SyntaxError: invalid syntax"

2.2 תשתית

נתונים לכם הקבצים המתאימים, צרפו אותם לפרויקט והגדירו את המחלקות הבאות:

1. המחלקה Type, אשר מייצגת טיפוס כללי
2. המחלקות Integer, Boolean, Void אשר יורשות את Type
3. המחלקה האבסטרקטית Sequence אשר יורשת את Type, אשר מייצגת טיפוס שהוא רצף של טיפוס אחר
4. המחלקה String אשר יורשת את Sequence (מחרוזת היא רצף משום שהיא למעשה רצף של תווים)

יש לצרף גם את קבצי ה-CPP המתאימים שם עליכם לממש את הפונקציות של המחלקות.

המחלקה Type

למחלקה Type יש שדה שנקרא: _isTemp (שתפקידו לציין האם הסוג נוצר באופן זמני או לא). ערך ב"מ של השדה הוא False.
במחלקה ימומשו ה-getter וה-setter עבור השדה הזה.

כל המחלקות היורשות את Type צריכות לתמוך בפונקציות הבאות:

- **בנאי**

  בנאי של כל אחד מהטיפוסים מקבל את הטיפוס אותו הוא מייצג (למשל Integer יקבל int).

- **isPrintable**

מחזירה ערך בוליאני אם ניתן להדפיס את האובייקט. יש להחזיר true בכל המחלקות פרט לVoid , שם יש להחזיר false.

- **toString**

מחזירה מחרוזת (מסוג std::string) אשר מייצגת את ערך הטיפוס – הסבר על הייצוג של כל טיפוס בהמשך.

שתי הפונקציות לא משנות את האובייקט הקורא להן, לכן עליהן להיות מוגדרות כ const.

המחלקות Type ו Sequence הן מחלקות אבסטרקטיות , ולא צריך להופיע בהן מימוש עבור פונקציות אלו.

2.3 בדיקת טיפוס

בקובץ parser.h יש הצהרה על הפונקציה הסטטית getType שמקבלת מחרוזת ומחזירה מצביע לטיפוס שהמחרוזת הזו מייצגת. עליכם לממש את הפונקציה הזו כך שתבדוק אם המחרוזת מייצגת ערך של טיפוס נתמך כלשהו (מספר שלם / ביטוי בוליאני / מחרוזת) אם כן, תחזיר אובייקט מתאים (Integer \ Boolean \ String) עם הערך הנ"ל, כך שהשדה שמציין האם זהו טיפוס זמני יהיה "אמת".
יש לקצץ מהמחרוזת רווחים בהתחלה ובסוף.
במידה והמחרוזת לא מייצגת אף ערך נתמך יש להחזיר null.

כדי לבדוק אם המחרוזת מייצגת ערך כלשהו, יש לבדוק את הדברים הבאים:

<u>מספר שלם</u>: כל התווים במחרוזת הם ספרות (0-9), פרט לספרה הראשונה (הכי שמאלית) שיכולה להיות מינוס ('-'). זה בסדר אם יש אפסים מובילים (בניגוד לפייתון האמיתי, אנחנו פשוט נתעלם מאפסים מובילים, ולא נחשיב את זה בתור מספר בבסיס 8).
כאשר יוצרים אובייקט מסוג Integer יש לשמור בתוכו את הערך בתור מספר אמיתי (מטיפוס int) ולא בתור מחרוזת. הפונקציה toString צריכה להחזיר ייצוג של המספר הזה בתור מחרוזת (מסוג std::string) ללא אפסים מובילים.

<u>ביטוי בוליאני:</u> המחרוזת "True" או המחרוזת "False". כאשר יוצרים אובייקט מסוג Boolean אפשר לשמור את הערך שהתקבל בכל צורה שתרצו (אם כי עדיף בתור משתנה מסוג bool - זה עשוי לעזור בהמשך). הפונקציה toString צריכה להחזיר ייצוג של הערך בתור מחרוזת (מסוג std::string) - כלומר "True" או "False".

<u>מחרוזת</u>: ערך חוקי יהיה כל מחרוזת שהתו הראשון והאחרון שלה שניהם גרשיים (") ואף תו אחר במחרוזת הוא לא גרשיים. או לחילופין - כל מחרוזת שהתו הראשון והאחרון שלה שניהם גרש בודד (') , ואף תו אחר במחרוזת הוא לא גרש בודד.
במקרה כזה יש ליצור אובייקט מסוג String (שימו לב! לא std::string אלא String שיורש את Type) ולשמור את המחרוזת. הפונקציה toString צריכה להחזיר מחרוזת (מסוג std::string) עם כל תווי המחרוזת המקורית, כאשר התו הראשון והאחרון הם גרש בודד

(גם אם במחרוזת המקורית הם היו גרשיים). במידה ובתוך המחרוזת מופיע התו גרש בודד ('), התו הראשון והאחרון צריכים להיות גרשיים (").
אין צורך לתמוך בתווים מיוחדים כמו בפייתון (escaped characters).

על מנת לבדוק התאמת המחרוזת לכל אחד מהטיפוסים ניתן להשתמש בפונקציות העזר הסטטיות הקיימות במחלקה Helper.

2.4 פרסור

בשלב הזה נעדכן את הפונקציה הסטטית parseString (שבקובץ parser.cpp) כך שהאינטרפרטר שלנו יתמוך רק בטיפוסים החוקיים שלנו. (יש להוסיף בדיקות למה שכבר בדקנו בסעיפים הקודמים).
יש לעדכן את הפונקציה כך שהיא לא תדפיס יותר את ערך המחרוזת שהיא קיבלה.

תחילה הסירו רווחים מסוף המחרוזת (אתם יכולים להשתמש בפונקציית עזר קיימת בשביל זה), וקראו לפונקציה שכתבתם בסעיף הקודם כדי לקבל את הטיפוס המתאים למחרוזת זו.
אם אכן קיים טיפוס מתאים (כלומר חוזר ערך שאינו null) יש להחזירו לפונקציה הראשית.

בלולאה הראשית יש לבדוק האם האובייקט שחזר ניתן להדפסה, ואם כן יש להדפיס את הערך המוחזר מהפונקציה toString שלו.
במידה והטיפוס שחזר הוא טיפוס זמני, יש לשחרר את הזיכרון שלו.

במידה והמחרוזת לא מתאימה לאף אחד משלושת הטיפוסים על הפונקציה לזרוק אקספשיין מסוג SyntaxException. יש לתפוס את האקספשיין בלולאה הראשית, ולהדפיס את הערך המוחזר מהפונקציה what.

משימה 3 - משתנים

3.1 מנגנון למיפוי שמות משתנים לאובייקטים

כדי לתמוך בפעולות הבאות שתתבקשו לממש, תצטרכו ליצור מנגנון אשר יודע למפות שם משתנה (בתור מחרוזת) לאובייקט המכיל את הערך של המשתנה. קראו באינטרנט על std::unordered_map והשתמשו ב-container זה כדי לממש את המנגנון.
הוסיפו במחלקה Parser משתנה סטטי בשם _variables מסוג unordered_map שממפה מחרוזות למצביעים לאובייקטים מסוג Type.

3.1.1 שאלת בונוס תיאורטית : הסבירו למדריך מדוע std::unordered_map עדיף על std::map במקרה הזה

3.2 בדיקה אם שם משתנה הוא חוקי

בקובץ parser.h יש הצהרה על הפונקציה הסטטית הבוליאנית isLegalVarName, שבהינתן מחרוזת מחזירה האם המחרוזת היא שם משתנה חוקי בפייתון.
שם משתנה יהיה חוקי אם ורק אם:

- הוא מורכב מאותיות באנגלית, ספרות, והתו '_' (underscore) בלבד.
- האות הראשונה אינה ספרה.

ממשו פונקציה זו בקובץ parser.cpp. היעזרו בפונקציות העזר הממומשות שם.

3.3 עוד אקספשיינים

הוסיפו בקובץ המתאים מחלקה חדשה שיורשת את InterpreterException:

1. מחלקה בשם NameErrorException, שמחזיקה שדה מסוג מחרוזת בשם _name.
   הפונקציה what שלה צריכה להחזיר את המחרוזת:
   "NameError : name '**[_name]**' is not defined"
   (במקום [_name] צריך להופיע הערך של השדה)

3.4 פרסור פעולות השמה

האינטרפרטר שלכם צריך לתמוך בהשמה למשתנים. הצהרה על משתנה תהיה באופן הבא:
variable_name = value
בקובץ parser.h יש הצהרה על הפונקציה הסטטית makeAssignment שמקבלת מחרוזת, ומחזירה ערך בוליאני. במידה והמחרוזת מייצגת פעולת השמה, על הפונקציה **לבצע את פעולת ההשמה** (מיד נגדיר איך) ולהחזיר true. אם המחרוזת לא מייצגת פעולת השמה, על הפונקציה להחזיר false.

יש לקרוא לפונקציה הזו מתוך parseString (אחרי שבדקנו אם המחרוזת שהתקבלה מייצגת טיפוס מסויים והתברר שלא). שימו לב - הפונקציה עשוייה לזרוק אקספשיין - יש לתפוס אותו בפונקציה Main ולא בפונקציה parseString. במידה והפונקציה makeAssignment מחזירה True - הפונקציה parseString שקראה לה צריכה להחזיר מצביע לאובייקט מסוג Void, כך שהשדה שמציין האם הוא זמני הוא "אמת".

הפונקציה אמורה לעבוד כך:

1. בדיקה שהמחרוזת היא מהפורמט המתאים: שמופיע תו '=' בודד באמצע המחרוזת.
   יש להחזיר false אם לא.

2. פירוק המחרוזת לשתי מחרוזות: כל מה שמשמאל לאופרטור - יש לשמור בתור **שם המשתנה** (יש להסיר את כל הרווחים בין סוף השם לאופרטור); כל מה שמימין לאופרטור - יש לשמור בתור **ערך ההשמה** (יש להסיר את כל הרווחים בין האופרטור לערך ההשמה, ואת כל הרווחים בסוף המחרוזת).
3. יש לבדוק אם שם המשתנה חוקי (באמצעות הפונקציה מסעיף 3.2). במידה ולא יש לזרוק אקספשיין מסוג SyntaxException (שכתבתם בסעיף 3.3).
4. יש לבדוק אם ערך ההשמה הוא ערך חוקי ולהחזיר מצביע לאובייקט מסוג הטיפוס המתאים (באמצעות הפנקציה מסעיף 2.3). במידה והערך אינו חוקי יש לזרוק אקספשיין מסוג SyntaxException (שכתבתם בסעיף 2.2).
5. אם שם המשתנה חוקי וגם ערך ההשמה חוקי יש להוסיף למפה variables_ (שיצרתם בסעיף 3.1) אלמנט חדש שהמפתח שלו הוא שם המשתנה, והערך שלו הוא המצביע לאובייקט שנוצר עבור ערך ההשמה. לאחר מכן יש להחזיר true.

> במידה ושם המשתנה כבר קיים כמפתח ב- variables, זה בסדר - יש לשנות את הערך שלו לערך החדש שהוזן.

3.5 שיערוך של משתנים

האינטרפרטר שלנו צריך גם לתמוך באופציה הבאה: המשתמש יכול להזין שם של משתנה וללחוץ אנטר. במידה והמשתנה קיים - יודפס ערכו של המשתנה בשורה מתחת; במידה והמשתנה אינו קיים תודפס הודעת שגיאה כזאת:
בהנחה ששם המשתנה הוא count והוא אינו קיים

NameError : name ‘count’ is not defined

במידה וזהו שם משתנה לא תקני ולא ערך של טיפוס זוהי שגיאת תחביר.

ממשו את הפונקציה הסטטית getVariableValue (מוצהרת ב-parser.h) שמקבלת מחרוזת.
במידה והיא קיימת בתור מפתח ב-variables יש להחזיר את הערך שלה (מצביע לאובייקט מסוג Type); במידה והיא לא קיימת יש להחזיר null.

הוסיפו קריאה לפונקציה הזו בתוך parseString. מומלץ לעשות זאת לפני שבודקים אם המחרוזת שהתקבלה מייצגת טיפוס מסויים (עם הפונקציה getType) כיוון שזו בדיקה מהירה יותר. רק במידה והפונקציה החזירה null יש להמשיך עם הלוגיקה הקיימת של parseString

3.6 שחרור זכרון

יש להוסיף למחלקה Parser פונקציה סטטית אשר מנקה את הזיכרון של המשתנים שנוצרו.

יש לקרוא לפונקציה הזאת במקום המתאים מה-main

3.7 העתקת משתנה

פעולות ההשמה שהתעסקנו איתן עד כה הן מהצורה הזו:

```
[variable name]  = [value]
Temp = 10
st = "Mython is Cool"
```

כעת נרצה לתמוך באפשרות שהערך המושם (מה שמצד ימין לסימן ההשמה) יהיה בעצמו שם של משתנה חוקי שקיים במערכת.
לדוגמא:

```
dst = src
```

נקרא לפעולה זו העתקה – אנחנו למעשה נקח את הערך שנמצא במשתנה src ונעתיק אותו למשתנה dst. הוסיפו תמיכה באפשרות זו לאינטרפרטר שלכם. שימו לב – יש לבצע העתקה עמוקה, כלומר לא למפות את שתי השמות לאותו מצביע (Type*) אלא ליצור אובייקט חדש מאותו סוג ש src מצביע עליו , ועם אותו ערך , ולמפות את dst למצביע החדש.

אם השם של src תקין כמשתנה אך טרם הוגדר, יש לזרוק חריגת NameErrorException.

משימה 4 - הוספת טיפוס

במשימה זו נוסף לאינטרפרטר שלנו תמיכה בטיפוס מסוג רשימה (List). ערך של טיפוס רשימה הוא למעשה אוסף של ערכים נתמכים (מספר שלם, ביטוי בוליאני, או מחרוזת).
לצורך הפשטות - בשלב זה רשימה לא יכולה להכיל רשימות אחרות.

הייצוג של ערך כזה אצלנו הוא מחרוזת שמתחילה ונגמרת בסוגריים מרובעים [ ] , כאשר ביניהם קיימות מחרוזות המייצגות טיפוסים, מופרדות בפסיק זו מזו.
לדוגמא:

```
>>> [ 1 , 2 , “hello world”]
>>> [True]
>>> [ ]
```

> כפי שניתן לראות, הרשימה יכולה להכיל ערכים מטיפוסים שונים, איבר בודד, או לא להכיל איברים כלל.

4.1 המחלקה List

הוסיפו בקובץ המתאים מחלקה בשם List היורשת את המחלקה האבסטרקטית Sequence.
כמובן שיש לממש את הפונקציות שהן pure virtual.

הפונקציה toString צריכה להחזיר מחרוזת המייצגת את הרשימה - המחרוזת צריכה להתחיל בסוגר מרובע שמאלי '[' וצמוד אליו האיבר הראשון ברשימה. לאחר כל איבר ברשימה (פרט לאחרון) מופיע פסיק ולאחריו רווח בודד. לאחר האיבר האחרון צריך להופיע סוגר מרובע ימני ']' ללא רווח ביניהם.
לדוגמא:

```
>>> [   1   ,2    ,    3      ]
[1, 2, 3]
```

שימו לב! על מנת לתמוך בפעולות מתקדמות בהמשך, על המחלקה להחזיק מצביעים לאובייקטים מסוג Type, ולא את המחרוזות המייצגות אותן כפי שהתקבלו מהמשתמש.

4.2 פרסור רשימה

עדכנו את הפונקציה הסטטית getType כך שתתמוך גם בזיהוי רשימה ויצירת אובייקט מתאים.

כדי להקל בשלב הזה - ניתן להניח שאם אחד או יותר מאיברי הרשימה הוא מחרוזת - היא לא מכילה את התו פסיק (' , ') בתוכה. בנוסף, כפי שכבר צויין, הרשימה לא יכולה להכיל בתוכה רשימות אחרות.

> אלו הם חלק מהדברים שהפרסר המאד פרימיטיבי שלנו מתקשה להתמודד איתם. בתור אתגר, לאחר שתסיימו את כל הסעיפים, נסו לחשוב איך לתכנן את הפרסר בצורה חכמה יותר כך שיוכל לתמוך גם במצבים אלו.

4.3 העתקת רשימות

וודאו תחילה שהמנגנון שמימשתם במשימה 3 מאפשר עכשיו גם השמת רשימה לתוך משתנה. יש רק הבדל חשוב בין רשימה לטיפוסים אחרים בפעולת ההשמה:
כאשר אנחנו מבצעים השמה של משתנה קיים לתוך משתנה חדש (העתקה, למעשה) אנחנו יוצרים העתק של הערך וממפים את שם המשתנה החדש לעותק החדש שיצרנו. ברשימה נרצה שזה יפעל אחרת - אנחנו לא ניצור העתק, אלא שם המשתנה החדש (dest) ימופה לאותו ערך אליו ממופה שם המשתנה הקיים (src). כלומר, במקרה זה – לא נרצה לבצע העתקה עמוקה!

בדקו את זה באינטרפרטר אמיתי של פייתון:
משתנה רגיל:

```
>>> x = 2
>>> y = x
>>> x += 1
>>> y
2
```

לעומת רשימה:

```
>>> x = [1,2,3]
>>> y = x
>>> x.append(4)
>>> y
[1, 2, 3, 4]
```

עדכנו את הפונקציה makeAssignment (מסעיף 3.3) כך שתתמוך בהתנהגות זו (כרגע לא נוכל לבדוק זאת דרך האינטרפטר עצמו, כי אין לנו עוד פעולות שמשנות רשימות, אבל אולי נרצה בהמשך. דרך שבה ניתן לבדוק שזה עובד היא בעזרת ה-דיבאגר של VS).

משימה 5 - פעולות על משתנים

* בחלק זה אתם אחראים על כל התכנון והאלגוריתם. אין הנחיות כיצד לממש את הפונקציונאליות.

5.1 הפונקציה type

האינטרפרטר שלנו צריך לתמוך באפשרות לבדוק מה טיפוס המשתנה.
ניתן להניח שהפונקציה מקבלת ארגומנט אחד. (כלומר מי שרושם את הפקודה באינטרפרטר מפעיל אותה עם ארגומנט אחד בלבד)
זה צריך לעבוד באופן הבא:

```
>>> a = 5
>>> type(a)
<type  'int'>
>>> a = True
>>> type(a)
<type  'bool'>
>>> a = "hello"
>>> type(a)
<type  'str'>
>>> type(3)
<type 'int'>
>>> type([1])
<type 'list'>
>>> type(b)
NameError: name 'b' is not defined
```

5.2 הפונקציה del

מאפשרת למחוק משתנים מאוסף המשתנים המוגדרים שלנו. לדוגמא:

```
>>> x = "hello"
>>> x
'hello'
>>> del x
>>> x
NameError: name 'x' is not defined.
```

> הפונקציה עובדת על כל סוגי המשתנים.

5.3 הפונקציה len

רלוונטית רק למשתנים מסוג sequence - מחזירה את מספר האיברים ברשימה/מספר התווים במחרוזת (לא כולל המרכאות התוחמות את המחרוזת) בתור ערך מספרי. הפונקציה

parseString צריכה להחזיר במקרה כזה מצביע לאובייקט מסוג Integer שהערך שלו הוא גודל הרשימה/מחרוזת.

דוגמה:

```
>>> x = “hello”
>>> len(x)
5
>>> y = 7
>>> len(7)
TypeError: object of type ‘int’ has no len() .
>>> len
<built-in function len>
>>> type(len(x))
<type  ‘int’ >
```

משימה 6 - המקום שלכם להתפרע

אם סיימתם את כל שאר המשימות במהלך הסדנא - **כל הכבוד! זה היה ממש מהר!** התייעצו עם המדריך מה עוד כדאי להוסיף.

אם לא סיימתם הכל במהלך המפגש בכתה, מאד מומלץ להמשיך לעבוד על זה בבית.

אם אהבתם את הסדנא ואתם רוצים ללמוד עוד על אינטרפרטרים/קומפיילרים או אפילו לבנות אינטרפרטר/קומפיילר שלם בעצמכם לשפה לפי בחירתכם (ושוב אני אומר - כדאי להתחיל עם שפה מצומצמת יותר, פייתון זו שפה קצת מורכבת בשביל התחלה) - **מעולה! אתם עושים בחירה מצויינת**, גם תלמדו מזה הרבה, זה גם מאד מרשים בראיונות (לצבא, ואפילו בראיונות עבודה לחברות הייטק) וזה גם ממש כיף.
מומלץ לחפש קצת באינטרנט, יש ספרים שלמים שעוסקים בעניין, ויש בלוגים אשר מיועדים למתחילים , וכמובן אתם מוזמנים להתייעץ עם המדריכים.

בהצלחה!