**BNF :**

<program> ::= <statement>

<statement> ::= <function>

| <function\_call>

| <condition>

<function> ::= "defun" "{" < identifier> ":" (<parameter\_list>, ) "}" <statement>

<function\_name> ::= "'" <identifier> "'" "," <identifier>

<parameter\_list> ::= "(" <parameter> {"," <parameter>} ")"

<parameter> ::= <identifier>

<function\_call> ::= <identifier> "(" <arguments> ")"

<arguments> ::= <statement> {"," <statement>}

<condition> ::= <expr> [<comparison\_op> <expr>]

| <expr> [<boolean\_op> <expr>]

<expr> ::= <term> {("+" | "-") <term>}

<term> ::= <factor> {("\*" | "/" | "%") <factor>}

<factor> ::= <number>

| <bool>

| <unary\_op> <statement>

| "(" <statement> ")"

| <parameter>

| <function\_call>

<number> ::= <digit> {<digit>}

<bool> ::= "true" | "false"

<boolean\_op> ::= "&&" | "||"

<comparison\_op> ::= "==" | "!=" | ">" | "<" | ">=" | "<="

<unary\_op> ::= "!" | "not"

<identifier> ::= <letter> {<letter> | <digit>}

<letter> ::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z"

<digit> ::= "0" | "1" | ... | "9"

חלק b - נמצא בקובץ. partB.py

**מתורגמן:**

**מבוא :**

פרויקט זה כולל פיתוח של מתורגמן המסוגל לעבד פעולות אריתמטיות וקריאות לפונקציות. המתורגמן יכול לנתח לחשב/ להעריך ביטויים מתמטיים שונים הכוללים חיבור, חיסור, כפל , חילוק וכו׳ , שימוש בסוגריים ובסדר פעולות חשבון. בנוסף , הוא תומך בממשק אינטראקטיבי ובפונקציות המוגדרות על ידי המשתמש עם היכולת להפעיל פונקציות אלה על ידי הארגומנטים שצוינו. בעוד המתורגמן היה אמור לכלול תמיכה ברקורסיה , תכונות אלו לא יושמו.

**מטרות ויעדים:**

המטרה העיקרית של הפרויקט הייתה להשיג הבנה מעמיקה של יצירת מתורגמן ואת המבנים הבסיסיים של שפת התכנות. המטרה הייתה ליצור מתורגמן פשוט אך פונקציונלי שיוכל להתמודד עם פעולות אריתמטיות והגדרת פונקציות.

**מבנה:**

המתורגמן נבנה על ידי שלושה מרכיבים עיקריים :

1. Lexer
2. Parser
3. Interpreter

ה – lexer בו הוגדרו מחרוזות הקלט שניתנים לזיהוי על ידי המורגמן.

ה – parser בונה עץ תחביר מאותם מחרוזות שהוגדרו.

ה- interpreter מבצע את ה״ביקורים״ בעץ שנוצר ומחשב את התוצאה הסופית.

**טיפול בשגיאות:**

המתורגמן מטפל בשגיאות שבהן לא ניתן לחלק באפס ובשגיאות של אופרנדים לא מוגדרים בשפה.

כאשר מתגלה שגיאה השגיאה תודפס למסך.

**אתגרים במהלך הפרויקט :**

כפי שצוין במבוא, המתורגמן שלנו לא תומך ברקורסיות, וזוהי הייתה אחת הבעיות ואחד האתגרים בו נתקלנו במהלך כתיבת הקוד. הבנו, שכאשר הקוד תומך בפונקציות הוא אמור לתמוך אוטומטית ברקורסיה הרי היא זאת אמורה לקרוא לפונקציה – שיודעת לחשב את התוצאה. לצערנו לא הצלחנו להתגבר על אתגר זה.

**הסבר שימוש במתורגמן:**

**פעולות אריתמטיות :**

* ניתן לבצע זאת על ידי : ‘number’ ‘operator’ ‘number’

והמתורגמן יחשב את התוצאה המתאימה.

**דוגמא 1 :**

3+4

Result : 7

**דוגמא 2 :**

(3 + 4) \* (-2 - 1)

Result : -21

**פונקציות :**

1. ניתן להגדיר פונקציות בשפה על ידי:

* יש לשים לב שלאחר כתיבת הפרמטר בסוגריים נוסיף ״ , ״

defun {'name': 'function\_name', 'arguments': (‘parameter’ ,)} parameter+ 1

1. לאחר הגדרת הפונקציה יודפס למסך : Result :None
2. כדי לקרוא לפונקציה נבצע זאת כך :

'function\_name’(‘number’ ,)

1. יתבצע החישוב . Result : ‘result’

**דוגמא:**

defun {'name': 'Add', 'arguments': (x,)} x+1

Add(1,)

Result : 2

בשביל לבצע פעולות בוליאניות עם משתנים , נגדיר שוב פונקציה ורק אז יתבצע החישוב.

למשל , נרצה לבדוק האם : (x==0) && (y==0) , ולכן נגדיר פונקציה שתבדוק את החישוב :

defun {'name': 'checkEQ', 'arguments': (x,y,)} (x==0) && (y==0)

checkEQ(0,0,)

Result : true

במידה ונרצה לקרוא לאותה הפונקציה עם משתנים אחרים נוכל לעשות זאת על ידי:

checkEQ(2,0,)

Result : false