

קורס שפות תכנות (10211)

הנדון : דו"ח פרויקט – שפות תכנות
מחלקה : הנדסת תוכנה

פרטי המגשים :

- עידן נוישול 207813635
- ספיר גילני 322358284

תוכן עניינים

| | | |
|---------|---|----|
| 3..... | הקדמה | 1. |
| 3..... | BNF – תיאור השפה והדקדוק | 2. |
| 6..... | איך להריץ – מדריך קצר | 3. |
| 6..... | תיאור חלקי המערכת | 4. |
| 7..... | 4.1 תיאור Use Case Diagram של ה-Interpreter | |
| 7..... | 4.2 תיאור ה-Lexer | |
| 9..... | 4.3 תיאור ה-Parser | |
| 10..... | 4.4 תיאור ה-Interpreter | |
| 11..... | 5. Design Decisions | 5. |
| 11..... | 5.1 תיאור חלקי המערכת והחלטות העיצוב | |
| 11..... | 5.2 הנחות ומגבלות של השפה | |
| 12..... | Cheat Sheet | 6. |

1. הקדמה

המערכת מממשת interpreter עבור שפת התכנות שלנו. הקבצים הנמצאים ב-Git מחלוקים באופן הבא:

- PartA – תיקייה המכילה את קבצי ה-Interpreter.
 - קובץ GrammerText – המכיל את ה-BNF של השפה
 - קובץ ExampleTest.lambda – קובץ להפעלת הפקודות ב-File Mode של התוכנית.
 - קובץ Testing.py – יחידת בדיקות ל-Interpreter המשתמש ב-unittest.
- PartB – תיקייה המכילה את קבצי הפיתוח של חלק ב' של הפרויקט.
 - קובץ README.md – עבור הגיט.
 - assets – תמונות עבור ה-README.md.
 - קובץ pdf – מסמך תיעוד לפרויקט.

2. BNF – תיאור השפה והדקדוק

תיאור מפורט של השפה – עם דוגמאות לפקודות תקינות.

הפונקציה הראשית ב-BNF המתארת את הפקודות המתאימות בשפה נקראת <language_command>.

```
<language_command> ::= <func_def> | <call_func> | <comment> | <printed_note>
| <lambda> | <comp_expression> | <TT_EXIT>
```

```
<func_def> ::= <TT_FUNC> <func_name> <TT_FUNC> <args> <TT_FUNC_SIGN>
<language_command>
```

```
<call_func> ::= <TT_CALL_FUNC> <func_name> <nested_func>+ |
<TT_CALL_FUNC> <func_name> <nested_func>+
```

```
<arg_value> ::= <arg_value> <TT_COMMA> <atom> | <atom>
```

```
<nested_func> ::= <TT_FUNC_LBRACKET> <arg_value> <TT_FUNC_RBRACKET> |
<TT_FUNC_LBRACKET> <TT_FUNC_RBRACKET>
```

```
<atom> ::= <INT> | <TT_STRING> | <bool>
```

```
<comment> ::= <TT_COMMENT> <text>*
```

```
<text> ::= <TT_STRING> | <INT> | <TT_STRING> <text> | <INT> <text>
```

```
<printed_note> ::= <TT_PRINTED_NOTE> <text>*
```

<lambda> ::= <TT_LLAMBDA> <arg_name> <TT_LAMBDA_SIGN>

<language_command> <TT_RLAMBDA> <nested_args>+

<nested_args> ::= <TT_LPAREN> <arg_value> <TT_RPAREN> | <TT_LPAREN>
<TT_RPAREN>

<comp_expression> ::= <TT_NOT> <comp_expression> | <second_expression>
<AND_OR> <second_expression> | <second_expression>

<second_expression> ::= <first_expression> <BOOL_OPS> <first_expression> |
<first_expression> | <second_expression> <BOOL_OPS> <second_expression>

<first_expression> ::= <factor> <PLUS_MINUS> <factor> | <factor> |
<first_expression> <PLUS_MINUS> <first_expression>

<factor> ::= <INT> | <bool> | <TT_STRING> | <lambda> | <TT_LPAREN>
<comp_expression> <TT_RPAREN> | <factor> <MUL_DIV_MOD> <factor> |
<call_func> | <lambda>

<TT_FUNC> ::= <whitespace>* "\$" <whitespace>*

<func_name> ::= <whitespace>* <TT_STRING> <whitespace>*

<TT_LPAREN> ::= <whitespace>* "(" <whitespace>*

<TT_RPAREN> ::= <whitespace>* ")" <whitespace>*

<args> ::= <TT_LPAREN> <arg_name> <TT_RPAREN> | <TT_LPAREN>
<TT_RPAREN>

<arg_name> ::= <TT_STRING> <TT_COMMA> <TT_STRING> | <TT_STRING>

<TT_FUNC_SIGN> ::= <whitespace>* "=" <whitespace>*

<TT_LLAMBDA> ::= <whitespace>* "[" <whitespace>*

<TT_RLAMBDA> ::= <whitespace>* "]" <whitespace>*

<TT_LAMBDA_SIGN> ::= <whitespace>* ":" <whitespace>*

<TT_CALL_FUNC> ::= <whitespace>* "@" <whitespace>*

<TT_FUNC_LBRACKET> ::= <whitespace>* "{" <whitespace>*

<TT_FUNC_RBRACKET> ::= <whitespace>* "}" <whitespace>*

<AND_OR> ::= <TT_AND> | <TT_OR>

<TT_AND> ::= <whitespace>* "&&" <whitespace>*

<TT_OR> ::= <whitespace>* "||" <whitespace>*

<TT_STRING> ::= <whitespace>* <letters> <whitespace>*

<letters> ::= <small_letter> <TT_STRING> | <large_letter> <TT_STRING> |
<small_letter> | <large_letter>

<small_letter> ::= [a-z]+

<large_letter> ::= [A-Z]+

<TRUE> ::= <whitespace>* "True" <whitespace>*

<FALSE> ::= <whitespace>* "False" <whitespace>*

<bool> ::= <TRUE> | <FALSE>

<TT_COMMA> ::= <whitespace>* "," <whitespace>*

<TT_NOT> ::= <whitespace>* "!" <whitespace>*

<TT_COMMENT> ::= <whitespace>* "#"

<TT_PRINTED_NOTE> ::= <whitespace>* "###"

<TT_EXIT> ::= <whitespace>* "EXIT" <whitespace>*

<INT> ::= "-" <INT> | <number> | <float>

<float> ::= <number> "." <number>

<number> ::= <non_digit_zero> <digits>* | <zero>

<digits> ::= <zero> | <non_digit_zero>

<non_digit_zero> ::= [1-9]+

<zero> ::= "0"

<PLUS_MINUS> ::= <TT_PLUS> | <TT_MINUS>

<TT_PLUS> ::= <whitespace>* "+" <whitespace>*

<TT_MINUS> ::= <whitespace>* "-" <whitespace>*

<MUL_DIV_MOD> ::= <TT_MUL> | <TT_DIV> | <TT_MODULO>

<TT_MUL> ::= <whitespace>* "*" <whitespace>*

<TT_DIV> ::= <whitespace>* "/" <whitespace>*

<TT_MODULO> ::= <whitespace>* "%" <whitespace>*

<BOOL_OPS> ::= <EE> | <NE> | <GT> | <GTE> | <LT> | <LTE>

<EE> ::= <whitespace>* "==" <whitespace>*

<NE> ::= <whitespace>* "!=" <whitespace>*

<GT> ::= <whitespace>* ">" <whitespace>*

<LT> ::= <whitespace>* "<" <whitespace>*

<GTE> ::= <whitespace>* ">=" <whitespace>*

<LTE> ::= <whitespace>* "<=" <whitespace>*

<whitespace> ::= " " | "\t"

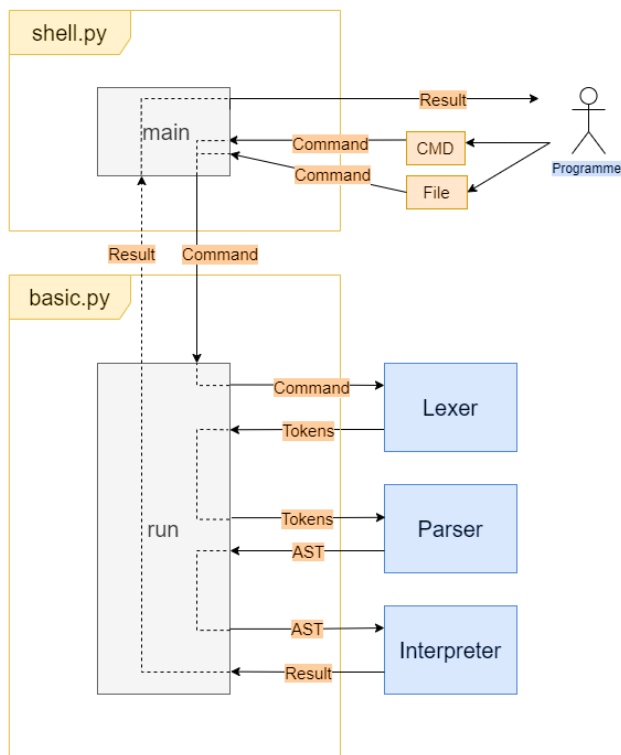
3. איך להריץ – מדריך קצר

- להרצת התוכנית בטרמינל ב-Interpreter Mode :
 - python shell.py
- ליציאה ממצב Interpreter Mode :
 - EXIT
- להרצת התוכנית בטרמינל ב-File Mode :
 - python shell.py filename.lambda
- להרצת בדיקות התוכנית בטרמינל :
 - python Testing.py

4. תיאור חלקי המערכת

4.1 תיאור Use Case Diagram של ה-Interpreter

שרטוט המתאר את ה-Interpreter של שפת התכנות שלנו:



המתכנת המשתמש בשפה שלנו ומריץ פקודות ב-Interpreter יכול להריץ בשני אופנים: פקודה-פקודה ב-Command Line או כתוכנית שלמה בקובץ נפרד המסתיים ב-lambda*. המכיל את הפקודות הדרושות.

קובץ ה-shell.py הוא התוכנית המפעילה את ה-Interpreter ומעבירה את הפקודות המגיעות מהמתכנת לפונקציית run() שבקובץ ה-basic.py. הפקודה עוברת את שלבי ה-Lexer, ה-Parser, ולבסוף ה-Interpreter שיוסברו בסעיפים הבאים, עד אשר היא מופעלת ומוחזר התוצאה חזרה למשתמש.

4.2 תיאור ה-Lexer

השלב הראשון בתהליך, בו מבוצע ניתוח מילולי ורצף התווים שהוכנסו על ידי המתכנת מתורגמים ל-token-ים המוכרים על ידי השפה. הסיווג בשפה שלנו וה-token-ים בהם השפה תומכת הם:

| הגדרת פונקציות | פעולות השוואה | פעולות בוליאניות | ביטויים אריתמטיים |
|----------------|---------------|------------------------|-------------------|
| FUNC (\$) | EQ (=) | AND (&&) | INT |
| FUNC_NAME | EE (==) | OR () | FLOAT (.) |
| FUNC_ARGS | NE (!=) | NOT (!) | PLUS (+) |
| FUNC_SIGN (=>) | LT (<) | BOOL ('True', 'False') | MINUS (-) |
| FUNC_BODY | GT (>) | | MUL (*) |
| | LTE (<=) | | DIV (/) |
| | GTE (>=) | | MODULO (%) |
| | | | LPAREN (' |
| | | | RPAREN '') |

| אחר | Lambda | קריאה לפונקציות |
|-------------------|-----------------|------------------|
| STRING | LLAMBDA '[' | CALL_FUNC (@) |
| COMMA (,) | RLAMBDA '] | FUNC_LBRACKET '{ |
| COMMENT (#) | LAMBDA_SIGN (:) | FUNC_RBRACKET '} |
| PRINTED_NOTE (##) | | |
| EOF | | |
| EXIT ('EXIT') | | |

מספר הערות:

- השפה תומכת בסימנים (המתוארים מעלה), באותיות a-z,A-Z, במספרים חיוביים ושלילים.
- ה-token הנקרא STRING הוא עבור כל רצף אותיות עוקבות. מכיוון שהשפה לא תומכת במחרוזות, המשמעות כאן היא עבור שמות של פונקציות, שמות של משתנים מקומיים בפונקצייה או ב-Lambda, הערות או הערות להדפסה, ושמות שמורים כמו True, False, EXIT.
- הפקודה EXIT נועדה לשימוש ב-Interpreter mode להרצת פקודות אחת אחרי השנייה ב-command line ועבור יציאה מהתוכנית ניתן לכתוב EXIT.
- ה-token הנקרא EOF נועד עבור סימון של סוף הפקודה.

דוגמה להמרת פקודה לרשימת token-ים:

- הפקודה: 5*(3-1)
- [INT:5, MUL, LPAREN, INT:3, MINUS, INT:1, RPAREN]

4.3
תיאור ה-Parser

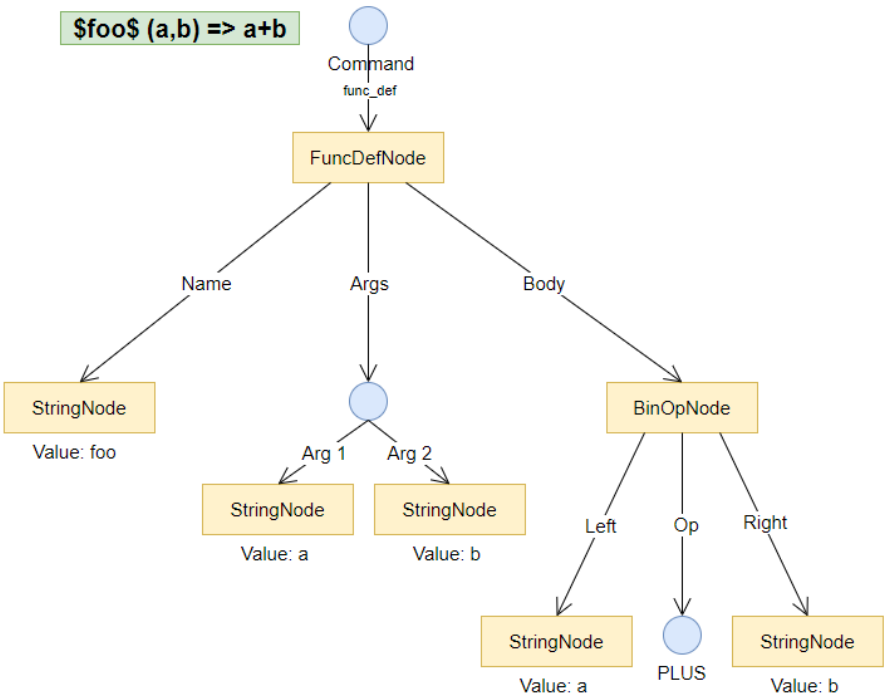
בשלב השני בתהליך, מבוצע ניתוח מחרוזות של הסמלים וה-token-ים וסיווגם לפי אובייקטים ומרכיבים המתאימים לקשר ביניהם. הפקודות בשפה שלנו מפורשים למספר סוגי פקודות והם:

- הגדרה של פונקציה
- קריאה לפונקציה
- הערות בקוד
- הערות מודפסות
- למבדות
- יציאה מה- interpreter
- ביטויים בוליאנים/אריתמטיים ושילובם

אובייקטים אותם ה- parser בונה הם:

- FuncDefNode
 - CallFuncNode
 - LambdaNode
 - CommentNode
 - PrintedNoteNode
 - StringNode
- NumberNode
 - BinOpNode
 - UnaryOpNode
 - BoolNode
 - ExitNode

דוגמה לבניית AST (Abstract Syntax Tree) על ידי ה-Parser עבור הגדרת פונקציה חדשה:



לאחר בניית ה-AST, ה-parser מחזיר אובייקט ParseResult שבמידה ופעולת הפרסור הצליחה חוזר האובייקט המתאים עם הערך המפורסר, אחרת, חוזרת שגיאה.

4.4 תיאור ה-Interpreter

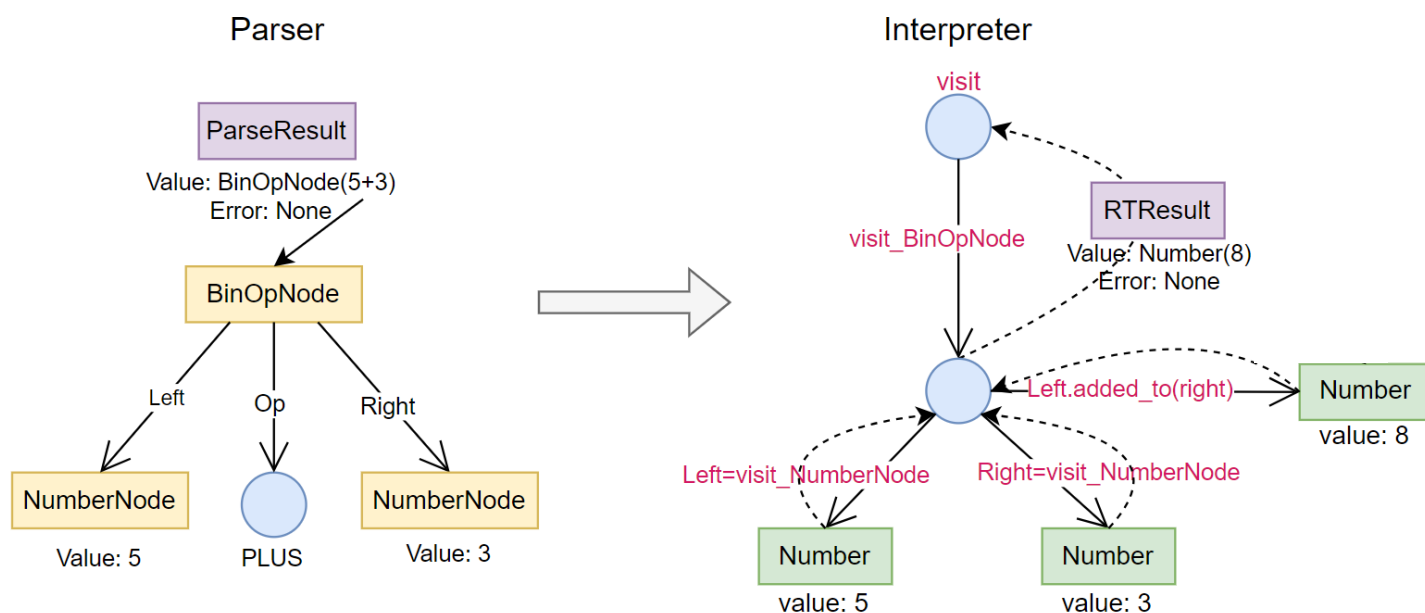
בשלב האחרון של התהליך, האובייקט שחזר מתוצאת הפרסור מגיע ל-Interpreter, לפונקציה הראשית הנקראת visit. פונקציה זו אחראית על הפעלת המתודות המקמפלות את האובייקט המתאים. לכל סוג של אובייקט קיימת פונקציה התואמת את שמה (visit_{ObjectName}). כך למשל, עבור האובייקט FuncDefNode, קיימת הפונקציה visit_FuncDefNode האחראית על קמפול הפקודה. בהתאם לכל אובייקט וכל סוג פקודה, ייבנה לבסוף האובייקט שיכיל את הערך המתאים ואת התוצאה הסופית.

האובייקטים שבהם ה-interpreter תומך הם:

- Number
- Bool
- Function
- Lambda

ה-interpreter מחזיר אובייקט RTResult שבמידה ופעולת הקמפול הצליחה, חוזר האובייקט המתאים עם תוצאת הקמפול, אחרת, חוזרת שגיאה.

דוגמה עבור הפקודה $5+3$:

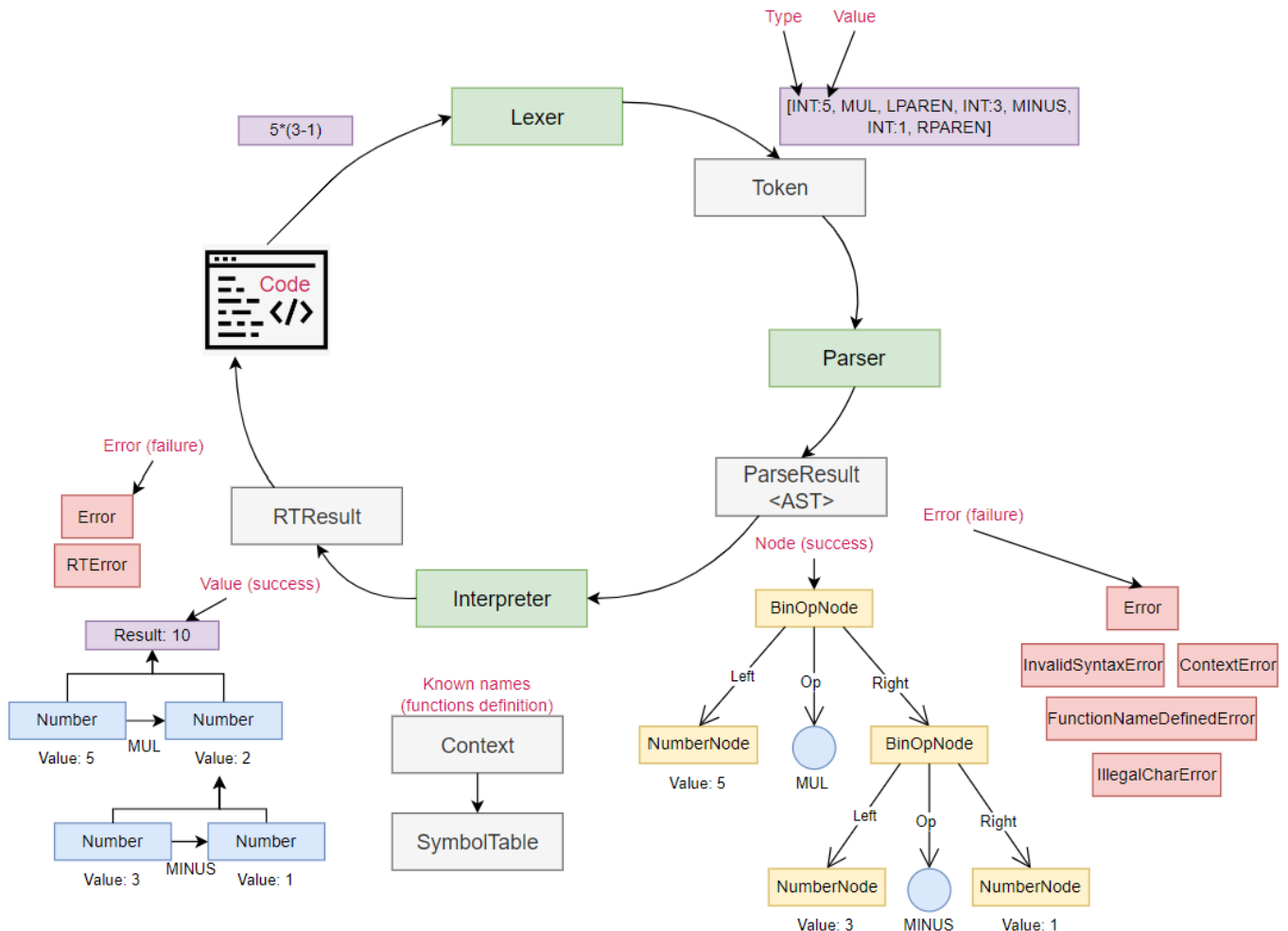


צד שמאל, זה תוצאת ה-parser המועברת לעיבוד אצל ה-interpreter בצד ימין. תוצאה המוחזרת זה אובייקט מסוג Number שערכו 8 בערך ה-value של ה-RTResult כאשר אין אף שגיאה. במידה וקורית שגיאה, מוחזר אובייקט Error וה-value הוא ריק.

5. Design Decisions

5.1 תיאור חלקי המערכת והחלטות העיצוב

שרטוט חלקי המערכת והמעברים:



5.2 הנחות ומגבלות של השפה

- להפעלת פונקציה בתוך פעולה בינארית יש להוסיף סוגריים עגולים
 - למשל: $(@a\{4\}) + 3$
- להפעלת למבדה פנימית בתוך למבדה יש להוסיף סוגריים עגולים,
 - למשל: $[x: ([y: y+4](2)) * x](3)$
- שמות של פונקציות ושל local arguments הם באותיות גדולות או קטנות ללא סימנים נוספים.

Cheat Sheet .6

| Data Types | | | |
|------------|------------------------------------|-----------|---------------|
| type | Set of values | operators | sample |
| Int | Integers | + - * / % | 1 -2 33 |
| Float | Floating point numbers | + - * / % | 3.14 -2.5 |
| Bool | Boolean values | && ! | True False |
| String | Function names and local arguments | | foo a b |

| Comparison | | | |
|------------|-----------------------|--------|--------|
| Op | meaning | True | False |
| == | equal | 2==2 | 2==3 |
| != | not equal | 3 != 2 | 2 != 2 |
| < | less than | 2 < 13 | 2 < 2 |
| <= | less than or equal | 2 <= 2 | 3 <= 2 |
| > | greater than | 13 > 2 | 2 > 13 |
| >= | greater than or equal | 3 >= 2 | 2 >= 3 |

| Lambda | Functions Declare |
| Lambda declaration Local arguments values `[x,y: x*5+y-2](10,12)` Local arguments variable Lambda body | Function declaration Local arguments variable `foo (a,b) => a+b` Function name Function sign Function body |
| Comments & Printed Notes | Functions Call |
| #for a comment ##for a printed note | Function Call Function name Local arguments values `@foo{ 3,4 }` |
| How To Run | Higher Order |
| File mode Shell.py ExampleTest.lambda | `bb () => aa (x) => 1+x` `@bb{}{2}` Arguments for bb Arguments for aa |
| Exit Interpreter Mode | |
| EXIT | |