**🌟 Custom Human-Readable Programming Language**

A clean, natural-language-inspired language designed for readability and clarity, resembling English prose.

**🧠 Language Philosophy**

* Use **natural English syntax** like let, be, is greater than, define function, etc.
* Make programs feel like **pseudocode**, but executable.
* Encourage clarity, readability, and accessibility for non-programmers.

**✅ Core Language Features**

| **Category** | **Feature Description** |
| --- | --- |
| ✅ Variables | English-like declaration: let x be 5; |
| ✅ Arithmetic | Supports +, -, \*, / |
| ✅ Strings | Double-quoted string support with escape sequences: "Hello\nWorld" |
| ✅ Conditionals | if, then, else, with rich comparisons like is greater than |
| ✅ Loops | repeat X times, repeat until, while loops |
| ✅ Functions | define function NAME with PARAMS, returns TYPE, end function |
| ✅ Boolean logic | and, or, not with true, false literals |
| ✅ Input/Output | ask, print, run |
| ✅ Type system | Basic types: number, text, boolean |
| ✅ Pattern matching | match ... case ... endmatch for conditional branching |
| ✅ Error handling | try ... catch ... endtry |
| ✅ Comments | Inline: # comment |

**🔤 Keywords**

| **Keyword** | **Purpose** | **Token** |
| --- | --- | --- |
| let | Declare variable | T\_LET |
| be | Assign value | T\_BE |
| set | Alternative assignment | T\_SET |
| is | Used in comparisons | T\_IS |
| null | Null literal | T\_NULL |
| if, then, else | Conditionals | T\_IF, T\_THEN, T\_ELSE |
| while, repeat, until, do | Loops | T\_WHILE, T\_REPEAT, etc. |
| define, function, end, with, returns | Functions | T\_DEFINE, T\_FUNCTION, T\_END, etc. |
| return, give | Return from function | T\_RETURN |
| and, or, not | Logical operators | T\_AND, T\_OR, T\_NOT |
| true, false | Boolean literals | T\_TRUE, T\_FALSE |
| ask, print, run | Input/output/system calls | T\_ASK, T\_PRINT, T\_RUN |
| match, case, endmatch | Pattern matching | T\_MATCH, T\_CASE, T\_END\_MATCH |
| try, catch, endtry | Error handling | T\_TRY, T\_CATCH, T\_END\_TRY |
| into | For ask ... into ... | T\_INTO |
| number, text, boolean | Types | T\_TYPE\_NUM, T\_TYPE\_TEXT, T\_TYPE\_BOOL |

**🔎 Comparison Phrases**

These are **multi-word tokens** Lex recognizes as single units:

| **Phrase** | **Meaning** | **Token** |
| --- | --- | --- |
| is equal to | Equality | T\_EQ |
| is not | Inequality | T\_NEQ |
| is greater than | Greater than | T\_GT |
| is less than | Less than | T\_LT |

**🧱 Symbols & Punctuation**

| **Symbol** | **Meaning** | **Token** |
| --- | --- | --- |
| + | Addition | T\_PLUS |
| - | Subtraction | T\_MINUS |
| \* | Multiplication | T\_MUL |
| / | Division | T\_DIV |
| (, ) | Grouping | T\_LPAREN, T\_RPAREN |
| {, } | Code blocks | T\_LBRACE, T\_RBRACE |
| [, ] | List indexing | T\_LBRACKET, T\_RBRACKET |
| ; | Statement terminator | T\_SEMI |
| , | Separator | T\_COMMA |
| : | Used in match/case, objects | T\_COLON |

**🔢 Literals and Identifiers**

| **Type** | **Example** | **Token** |
| --- | --- | --- |
| String | "Hello" | T\_STRING |
| Integer | 42 | T\_NUM |
| Real number | 3.14 | T\_RNUM |
| Identifier | myVar, age | T\_ID |

**🧾 Comments**

* Any line starting with # is ignored
* Example:
* let name be "Alice"; # This is a comment

**🧪 Sample Program**

define function greet with name

let msg be "Hello, ";

if name is not null then

print msg;

print name;

else

print "Guest";

end function