




# LOLCCODE

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)

 <https://www.facebook.com/tutorialspointindia>

 <https://twitter.com/tutorialspoint>

## About the Tutorial

---

LOLCODE is an esoteric programming language inspired by the funny things on the Internet. LOLCODE is designed to test the boundaries of programming language design.

This tutorial provides a basic level understanding of the LOLCODE programming language.

## Audience

---

This tutorial is meant for people who want to explore beyond general boring programming syntax. Readers of this tutorial can learn the programming language in simple and easy ways.

This tutorial will also be helpful for all those developers who want to learn the basics of LOLCODE.

## Prerequisites

---

The tutorial assumes that the readers have a knowhow about programming languages. If you have worked on any other programming language, it will be easier for you to learn LOLCODE.

## Copyright & Disclaimer

---

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com)

## Table of Contents

---

|  |           |
|--|-----------|
| About the Tutorial .....   | i         |
| Audience.....  | i         |
| Prerequisites .....  | i         |
| Copyright & Disclaimer .....                                     | i         |
| Table of Contents .....  | ii        |
| <b>1. LOLCODE – INTRODUCTION AND ENVIRONMENT SETUP .....</b>     | <b>1</b>  |
| Setting Up the Local Environment .....                           | 1         |
| Installation on Windows.....                                     | 1         |
| Executing Script Online with Tutorialspoint - codingground ..... | 1         |
| <b>2. LOLCODE – SYNTAX.....</b>                                  | <b>2</b>  |
| Constructs .....   | 2         |
| Whitespace .....   | 3         |
| Comments .....   | 4         |
| File Creation.....   | 4         |
| <b>3. LOLCODE – VARIABLES .....</b>                              | <b>5</b>  |
| Scope of Variables .....   | 5         |
| Naming Conventions .....   | 5         |
| Declaration and Assignment of Variables.....                     | 7         |
| <b>4. LOLCODE – TYPES .....</b>                                  | <b>9</b>  |
| Types .....  | 9         |
| Untyped (NOOB) .....   | 9         |
| Booleans (TROOFS).....   | 10        |
| Numerical Types (NUMBR) .....                                    | 11        |
| Strings (YARN).....  | 11        |
| BUKKIT .....   | 12        |
| <b>5. LOLCODE – OPERATORS.....</b>                               | <b>13</b> |

|   |           |
|---|-----------|
| Operators .....   | 13        |
| Comparison.....   | 15        |
| Concatenation of Values.....                                    | 16        |
| Type Casting.....   | 17        |
| <b>6. LOLCODE – INPUT/OUTPUT .....</b>                          | <b>18</b> |
| I/O from Terminal.....  | 18        |
| <b>7. LOLCODE – STATEMENTS AND FLOW CONTROL .....</b>           | <b>20</b> |
| Expression Statements.....                                      | 20        |
| Assignment Statements .....                                     | 20        |
| Conditional Statements .....                                    | 20        |
| Case Statements.....  | 22        |
| <b>8. LOLCODE – LOOPS.....</b>                                  | <b>23</b> |
| <b>9. LOLCODE – Functions .....</b>                             | <b>25</b> |
| Definition of a Function .....                                  | 25        |
| Returning Value from a Function.....                            | 25        |
| Calling Functions.....  | 26        |
| <b>10. LOLCODE - EXCEPTION HANDLING .....</b>                   | <b>28</b> |
| <b>11. LOLCODE – SOME MORE EXAMPLES .....</b>                   | <b>29</b> |
| Example 1: Program to Calculate the Power of a Number.....      | 29        |
| Example 2: Program to Make an Array .....                       | 30        |
| Example 3: Program to Calculate the Factorial of a Number ..... | 31        |
| Example 4: Program to Design a Calculator.....                  | 31        |

# 1. LOLCODE – INTRODUCTION AND ENVIRONMENT SETUP

LOLCODE is an esoteric programming language inspired by the funny things on the Internet. It is designed to test the boundaries of programming language design.

This chapter will make you familiar with setting up the local environment for LOLCODE, installing it on Windows, and executing its script online at [TutorialsPoint – codingground](https://www.tutorialspoint.com/codingground/).

## Setting Up the Local Environment

---

The LOLCODE interpreter is written in C Language. It interprets the code written in LOLCODE language on multiple platforms. The LOLCODE interpreter is known as lci, which stands for LOLCODE Interpreter.

Please note that LOLCODE officially supports direct installation of interpreter for MAC operating Systems only. To install LOLCODE in your operating system, you need to follow the steps given below:

- Press Command+Space, and type **Terminal** and press **enter/return** key.
- Run in Terminal app
- `$ git clone https://github.com/justinmeza/lci.git`
- `$ cd lci`
- `$ cmake .`
- `$ make && make install`

## Installation on Windows

---

If you need to install LOLCODE on Windows operating system, please take these steps:

- First add MinGW and Python to your environment variables path. To do this, right click on **My Computer**, choose **Properties**, then select **Advanced system settings**. Select **Environment Variables**. In this box, select the **PATH** variable and then click **Edit**.
- Now, add ";C:\MinGW\bin;C:\Python32" to the end of that path.
- Next, open the **Command Prompt** and navigate to the project directory using the "cd" command, for example.
- Run the script install.py.

## Executing Script Online with TutorialsPoint -codingground

---

To execute your scripts easily and swiftly, use the [codingground](https://www.tutorialspoint.com/codingground/) platform provided by TutorialsPoint. For this, go to the following link to execute your scripts online: [https://www.tutorialspoint.com/execute\\_loocode\\_online.php](https://www.tutorialspoint.com/execute_loocode_online.php)

## 2. LOLCODE – SYNTAX

LOLCODE has a different syntax compared to other programming languages, however, it is easier to remember. This chapter gives you the basic syntax of LOLCODE.

### Constructs

---

The LOLCODE constructs are slang words. The following table shows the alphabetical list of constructs implemented so far:

| LOLCODE Construct  | Purpose/ Usage   |
|--|--|
| <b>BTW</b>   | It starts a single line comment.   |
| <b>DOWN &lt;variable&gt;!!&lt;times&gt;</b>                            | This corresponds to <code>variable = variable - times</code> . Note that "times" is a wut-only language extension.   |
| <b>GIMMEH &lt;variable&gt;</b>   | This represents the input statement.   |
| <b>GTFO</b>  | This is similar to <b>break</b> in other languages and provides a way to break out of a loop.  |
| <b>HAI</b>   | This corresponds to <b>main ()</b> function in other languages. It is the program entry point in LOLCODE.  |
| <b>HEREZ &lt;label&gt;</b>   | This is another wut-only language extension and declares a label for use with SHOO   |
| <b>I HAS A &lt;type&gt; &lt;variable&gt;</b>                           | This declares a variable of said type.<br><br>There are three built-in types in LOLCODE:<br>NUMBAH (int)<br>DECINUMBAH (double)<br>WORDZ (std::string)<br><br>Note that types are a wut-only language extension. |
| <b>IM IN YR LOOP</b>   | This starts an infinite loop. The only way to exit the loop is using GTFO. Corresponds to <code>for(;;)</code> in other languages  |
| <b>IZ &lt;expr1&gt; &lt;operator&gt; &lt;expr2&gt;?:<br/>structure</b> | This is similar to if operator in other languages. Operator is one of: BIGGER THAN, SMALLER THAN, SAEM AS. Note that the ? at the end is optional.   |
| <b>KTHX</b>  | It ends a block. Corresponds to <code>}</code>   |
| <b>KTHXBAI</b>   | This ends a program  |

|   |  |
|---|--|
| <b>NOWAI</b>                              | This corresponds to else   |
| <b>PURR &lt;expr&gt;</b>                  | This prints argument on screen, followed by a newline. It is a wut-only language extension.      |
| <b>RELSE</b>                              | This corresponds to <b>else (if)</b>   |
| <b>SHOO</b>                               | This is another wut-only language extension, that corresponds to <b>goto</b> (the horror!)       |
| <b>UP &lt;variable&gt;!!&lt;times&gt;</b> | This corresponds to variables = variable + times. Here "times" is a wut-only language extension. |
| <b>VISIBLE &lt;expr&gt;</b>               | This prints the argument on screen. Note that this does not print a newline.                     |
| <b>YARLY</b>                              | This denotes the start of the "true" conditional block   |

Some examples of slang terms in LOLCODE are:

- HAI is hi
- KTHXBYE is okay, thanks, bye
- BTW is by the way
- OBTW is oh, by the way
- TLDR is too long; didn't read

## Whitespace

---

In most programming languages, keywords or tokens may not have spaces between them. However, in some languages, spaces are used in tokens to differentiate them.

### Comma

The comma behaves like a newline keyword in most languages, for example, `\n` in Java and C. You can write many commands in a single line in LOLCODE, provided that you separate them using a comma (,).

### Three Periods (...)

The three periods (...) enables you to combine multiple lines of code into a single line or a single command by including (...) at the end of the line. This makes the compiler to treat the content of the next line as the content of previous line only. Infinite lines of code can be written together as a single command, as long as each line is ended with three periods.

A comment is terminated by a newline. Please note that the line continuation (...) and (,) after the comment (BTW) are ignored by the lci.

## Comments

---

Single line comments are written followed by the BTW keyword. They may occur anywhere inside a program body: it can be at the first line of program, in between the program, in between some line, or at the end of a program.

All of these are valid single line comments:

```
I HAS A VAL ITZ 19          BTW VAL = 19
I HAS A VAL ITZ 19,        BTW VAL = 19

I HAS A VAL ITZ 14
BTW VAR = 14
```

In LOLCODE, multiple line comments are written followed by OBTW and they are ended with TLDR.

This is a valid multi-line comment:

```
I HAS A VAL ITZ 51
    OBTW this is a comment
        No it's a two line comment
        Oops no.. it has many lines here
    TLDR
```

## File Creation

---

A LOLCODE program begins with HAI keyword and it should end with KTHXBYE. As LOLCODE uses shorthand language HAI basically stands for **Hi** and KTHXBYE can be remembered as "**Ok, thanks, bye**".

### Example

```
HAI 1.2
I HAS A NAME
VISIBLE "NAME::"!
GIMMEH NAME
VISIBLE "tutorialsPoint " NAME "!
KTHXBYE
```



## 3. LOLCODE – VARIABLES

As in any other programming language, LOLCODE allows you to define variables of various types. This chapter will make you familiar with working with variables in LOLCODE.

### Scope of Variables

---

The scope of a variable is local to the function or to the program block, i.e. a variable defined in one scope cannot be called in any other scope of the same program. Variables are accessible only after they are declared.

Please note that there is no global scope of variables in LOLCODE.

### Naming Conventions

---

Variable names are usually called identifiers. Here are some of the conventions for naming variables in LOLCODE:

- Variable identifiers may be in all CAPITAL or lowercase letters (or a mixture of the two).
- They can only begin with a letter and then may be followed by other letters, numbers, and underscores.
- LOLCODE does not allow use of spaces, dashes, or other symbols while naming a variable.
- Variable identifiers are case sensitive.

Here are some of the rules for valid and invalid names for variables in LOLCODE:

The name should always begin with an alphabet. For example, **name**, **Name** are valid.

- The name of a variable cannot begin with a digit. For example, **2var** is invalid.
- The name of a variable cannot begin with a special character.
- A variable can contain `_` or a digit anywhere inside its name, except at the starting index. For example, **name2\_m** is a valid name.

Some examples of valid names in LOLCODE are shown below:

```
HAI 1.2
I HAS A food ITZ "111.00033"
I HAS A food2 ITZ "111"
I HAS A fo_od ITZ "1"
VISIBLE food
```

```
VISIBLE food2
VISIBLE fo_od
KTHXBYE
```

All the declaration statements in the above code are valid and will produce the following output when executed:

```
sh-
4.3$ lci main.lo
111.00033
111
1
```

Some examples of invalid statements and their output are given below:

### Example 1

```
HAI 1.2
I HAS A 2food ITZ "111.00033"
KTHXBYE
```

The above code will give the following output when you execute it:

```
sh-
4.3$ lci main.lo
Line 2: Expected: identifier; Got: int(2).
```

### Example 2

```
HAI 1.2
I HAS A _food ITZ "111.00033"
KTHXBYE
```

The above code will give the following output when you execute it:

```
sh-
4.3$ lci main.lo
Line 2: Unrecognized sequence at: _food ITZ "111.00033".
```

### Example 3

```
HAI 1.2
I HAS A f$ood ITZ "111.00033"
KTHXBYE
```

The above code will give the following output when you execute it:

```
sh-
4.3$ lci main.lo
Line 2: Unrecognized sequence at: $ood ITZ "111.00033".
```

## Declaration and Assignment of Variables

To **declare** a variable, LOLCODE provides a keyword "I HAS A" which is followed by the variable name. You can find below the syntax for declaring a variable.

```
I HAS A VAR    BTW VAR is empty now, You can use any name instead of var
```

To **assign** the variable a value in the same statement, you can then follow the variable name with "ITZ" and then give the value you want to assign. Use the following syntax to assign a value to a variable:

```
<variable> R <expression>
```

### Example

```
VAR R "Green"          BTW VAR is now a YARN and equals "Green"
VAR R 30                BTW VAR is now a NUMBR and equals 30
```

You can also **declare** and **assign** variables at the same time using the following syntax:

```
I HAS A VAR ITZ VALUE
```

### Example

```
I HAS A NAME ITS "TUTORIALS POINT"
```

### Example

```
HAI 1.2
BTW this is how we declare variables
I HAS A food
I HAS A bird
```

```

BTW this is how we assign variables
food R 1
bird R 5
BTW this is how initialize variables
I HAS A biz ITZ "OMG!"
VISIBLE food
VISIBLE biz
VISIBLE bird
KTHXBYE

```

The above program shows the declaration of variables and prints them. The output is:

```

sh-
4.3$ lci main.lo
1
OMG!
5

```

## Type Casting

To convert a value of one type to another type, we use type casting. Casting a NUMBAR to a NUMBR truncates the decimal portion of the floating point number. Casting a NUMBAR to a YARN (by printing it, for example), truncates the output to a default 2 decimal places.

## Example

```

HAI 1.2
I HAS A food ITZ "111.00033"
VISIBLE food
BTW this is how we do type casting
MAEK food A NUMBAR
VISIBLE food
KTHXBYE

```

The above line of code will produce the following output:

```

sh-4.3$ lci main.lo
111.00033
111.00033

```

All the variables declared in a LOLCODE program are local variables and there is no global scope in this language for any variable.

## 4. LOLCODE – TYPES

LOLCODE is designed to test the boundaries of the programming language design. It is an esoteric programming language inspired by the funny things on the Internet. This chapter gives you an understanding of LOLCODE types.

### Types

---

Currently, the variable types in LOLCODE are:

- strings (YARN)
- integers (NUMBR)
- floats (NUMBAR)
- and booleans (TROOF)
- Arrays (BUKKIT)

In LOLCODE the variable type is handled dynamically by the compiler. If a variable does not have an initial value, it is called untyped (known as NOOB in LOLCODE).

The syntax for declaring and using different types in LOLCODE is shown below:

#### To create a variable of any data type

```
I HAS A <VARIABLE> ITZ A <DATA TYPE>
```

#### To create a variable and assign a value to it

```
I HAS A <VARIABLE> ITZ <EXPRESSION>
```

#### To assign a value to an already created data type

```
<VARIABLE> R <EXPRESSION>
```

### Untyped (NOOB)

---

The untyped data type (known as NOOB), cannot be converted into any other type except into a TROOF data type. The implicit casting of a NOOB into TROOF makes the variable FAIL. After that any operation on a NOOB results in an error.

Explicit casts of a NOOB data type (i.e. the types that are uninitialized and do not have any initial value) variable results to zero values for all other types.

To define an untyped variable, just declare a variable and assign a value as shown in this example:

```
HAI 1.2
I HAS A VAR3
VAR3 R "ANYVALUE"
VISIBLE VAR3
BTW Or declare in same line
I HAS A VAR4 ITZ 44
VISIBLE VAR4
KTHXBYE
```

When you run the above program, you will find the following result:

```
sh-
4.3$ lci main.lo
ANYVALUE
44
```

## Booleans (TROOFS)

In LOLCODE, the Boolean values are of two types. BOOLEAN generally have two values- true and false. But, in LOLCODE, the Boolean is known as TROOF, and the true/ false values are known as WIN/FAIL respectively. All the uninitialized values like an empty string (""), or an empty array will all cast to FAIL. All other initialized values evaluate to WIN.

### Example

```
HAI 1.2
I HAS A VAR3 ITZ A TROOF
VAR3 R "FAIL"
    VISIBLE VAR3
KTHXBYE
```

You can see the following output when you execute the above code:

```
sh-4.3$ lci main.lo
FAIL
```

## Numerical Types (NUMBR)

In LOLCODE, a NUMBR stands for an integer. Any sequence of digits is considered as a NUMBR, unless it has a decimal appearing anywhere in between the sequence. To make any number negative, it may be preceded by a hyphen (-) which signifies a negative number.

### Example

```
HAI 1.2
I HAS A VAR3 ITZ A NUMBR
    VISIBLE VAR3
KTHXBYE
```

The above code shows you the following result when you run it:

```
sh-
4.3$ lci main.lo
0
```

Similar to NUMBR, LOLCODE has another data type, which represents a decimal or a float in many programming languages. In LOLCODE, a NUMBAR is a float containing one decimal point. Casting a NUMBAR to a NUMBR truncates the decimal portion of the floating point number and returns it as a NUMBR, without any decimal.

## Strings (YARN)

In LOLCODE, value containing strings, i.e. string literals (YARN) should start and end with double quotation marks ("").

Anything may be written inside the string, like space, comma, full stop, exclamation or any other symbol. A string where any single quote is missing may cause an error. Colons are used as escape characters in LOLCODE, and any value following a colon gets a special meaning.

- :) - A closing bracket following a colon represents a newline (\n)
- :> - A closing angle bracket following a colon represents a tab (\t)
- :o - A 'o' character following a colon represents a bell (beep) (\g)
- :'" - A " following a colon represents a literal double quote (")
- :: - A colon following a colon represents a single literal colon (:)

### Example

```
HAI 1.2
I HAS A VAR3 ITZ A YARN
VAR3 R "XYZ"
```

```
VISIBLE VAR3
KTHXBYE
```

The code given above produces the following output upon execution:

```
sh-
4.3$ lci main.lo
XYZ
```

## BUKKIT

This type represents an array. It has named slots, which can contain either variables or functions. A BUKKIT can be declared in the following way:

```
BTW declaration of the BUKKIT
I HAS A [object] ITZ A BUKKIT BTW creating a variable in a slots
[object] HAS A [var] ITZ [value] BTW creating a function inside the BUKKIT
HOW IZ [object] [function name] (YR [argument1] (AN YR [argument2] (AN YR
[argument3] ...)))
[function code]
IF U SAY SO
```

A function inside a BUKKIT may also access variables and other functions of the BUKKIT by using ME'Z [var] or ME IZ [function name] (YR [argument1] (AN YR [argument2] (AN YR [argument3] ...))) MKAY.

## Example

```
HAI 1.2
  I HAS A VAR6 ITZ A BUKKIT
  BTW DECLARING AN ARRAY
  VAR6 HAS A VAR7 ITZ "DOGE"
  BTW VAR7 IS A STRING VARIABLE THAT IS INSERTED INTO ARRAY VAR6
  VISIBLE VAR6'Z VAR7
  BTW GET THE ELEMENT OF ARRAY
KTHXBYE
```

This is the output you will find when you run the code given above:

```
sh-
4.3$ lci main.lo
DOGE
```



## 5. LOLCODE – OPERATORS

Operators play an important role to perform various operations on variables. This chapter brings you various operators in LOLCODE and their usage.

### Operators

---

Mathematical operators depend on a prefix notation i.e. the notation that comes before the operand. When all the operators have known number of arguments or operands, then no grouping markers are necessary. In cases where operators don't have fixed arguments or operands, the operation is closed with MKAY.

An MKAY may not be used if it coincides with the end of the statement. In such cases, the EOL keyword should be used. To use unary mathematical operators, use the following syntax:

```
<operator> <expression>
```

The AN keyword can optionally be used to separate arguments, and apply a single operation on more than one operand, so a binary operator expression has the following syntax:

```
<operator> <expression1> AN <expression2>
```

Any expression containing an operator with infinite number of arguments can be expressed with the following syntax:

```
<operator> <expression1> [[AN <expression2>] AN <expression3> ...] MKAY
```

### Math

Following are the basic mathematical operations in LOLCODE:

|                        |                                   |
|------------------------|-----------------------------------|
| SUM OF <a> AN <b>      | BTW This is a plus + operator     |
| DIFF OF <a> AN <n>     | BTW This is a minus - operator    |
| PRODUKT OF <a> AN <n>  | BTW This is a multiply operator * |
| QUOSHUNT OF <a> AN <n> | BTW This is a divide operator     |
| MOD OF <a> AN <n>      | BTW This is a modulo operator     |
| BIGGR OF <a> AN <n>    | BTW This is a max operator        |
| SMALLR OF <a> AN <n>   | BTW This is a min operator        |

<a> and <b> can each be unique expressions in the above, so mathematical operators can be nested and grouped indefinitely.

Math is performed considering arguments as integer math in the presence of two NUMBRs, but if either of the expressions is NUMBAR, then operations are considered as floating point operations.

### Example

```
HAI 1.2
  I HAS A m ITZ 4
  I HAS A n ITZ 2
VISIBLE SUM OF m AN n      BTW +
VISIBLE DIFF OF m AN n     BTW -
VISIBLE PRODUKT OF m AN n BTW *
VISIBLE QUOSHUNT OF m AN n BTW /
VISIBLE MOD OF m AN n      BTW modulo
VISIBLE BIGGR OF m AN n    BTW max
VISIBLE SMALLR OF m AN n   BTW min
KTHXBYE
```

The above code will produce the following output when you run it:

```
sh-
4.3$ lci main.lo

6

2

8

2

0

4

2
```

### Important Points:

Consider the following important points related to working with mathematical operators in LOLCODE:

- If one or both arguments in an expression are YARN, they are treated as NUMBARs.

- If any of the arguments cannot be safely casted internally to a numerical type, then it fails with an error.

## Boolean

Boolean operators are applied on those values that may be true or false. Boolean operators working on TROOFs are as following:

|                            |   |
|----------------------------|---|
| BOTH OF <m> AN <n>         | BTW its and operation: WIN if m=WIN and n=WIN     |
| EITHER OF <m> AN <n>       | BTW its or operation: FAIL iff m=FAIL, n=FAIL     |
| WON OF <m> AN <n>          | BTW its xor operation: FAIL if m=n                |
| NOT <m>                    | BTW its an unary negation: WIN if m=FAIL          |
| ALL OF <m> AN <n> ... MKAY | BTW it will take infinite arguments and apply AND |
| ANY OF <m> AN <n> ... MKAY | BTW it will take infinite arguments and apply OR. |

Please note that <m> and <n> in the expression syntax above are automatically cast as TROOF values if they are not already TROOF Values.

## Comparison

When you want to compare two or more operands in LOLCODE, you can do so in any of the following methods:

### Method 1

You can compare two binary operands using equality operators. The syntax is shown below:

|                      |   |
|----------------------|---|
| BOTH SAEM <m> AN <n> | BTW this will return WIN if m is equal to n     |
| DIFFRINT <m> AN <n>  | BTW this will return WIN if m is not equal to n |

### Method 2

You can compare if both the values are of NUMBRs type. Remember that if either of the values are NUMBARs, then they are compared as floating point values.

### Method 3

You can also perform comparison using the minimum and maximum operators. The syntax is shown below:

|                                       |
|---------------------------------------|
| BOTH SAEM <m> AN BIGGR OF <m> AN <n>  |
| BOTH SAEM <m> AN SMALLR OF <m> AN <n> |
| DIFFRINT <m> AN SMALLR OF <m> AN <n>  |
| DIFFRINT <m> AN BIGGR OF <m> AN <n>   |

## Example

```
HAI 1.2
  I HAS A VAR11 ITZ 7
  BOTH SAEM VAR11 SMALLR OF VAR11 AN 8, O RLY?
    YA RLY
      VISIBLE "TRUE"
    NO WAI
      VISIBLE "FALSE"
  OIC
KTHXBYE
```

You can see the following output when you execute the given code:

```
sh-
4.3$ lci main.lo

TRUE
```

## Concatenation of Values

LOLCODE allows you to explicitly concatenate infinite number of YARNs using the SMOOSH...MKAY operator. For concatenation, multiple arguments can be separated with the **AN** operator.

## Example

```
HAI 1.2
  I HAS A VAR1 ITZ A YARN
  VAR1 R "TRUE"
  I HAS A VAR2 ITZ A YARN
  VAR2 R "ANOTHER TRUE"
  I HAS A VAR3 ITZ A YARN
  VAR3 R "ONE MORE TRUE"
  VISIBLE SMOOSH VAR1 " " VAR3 " " VAR2 MKAY
KTHXBYE
```

The above given code will produce the following result upon execution:

```
sh-
4.3$ lci main.lo

TRUE ONE MORE TRUE ANOTHER TRUE
```

## Type Casting

Operators that work on specific types implicitly cast or convert the values of one type to other type safely. If the value cannot be safely converted to other type, then it results in an error.

An expression's value may be explicitly casted or converted to some other type with the binary MAEK operator. The syntax of MAEK Operator is:

```
MAEK <expression> A <type>
```

where, <type> can be one of TROOF, YARN, NUMBR, NUMBAR, or NOOB.

To explicitly cast a variable to some other type, a normal assignment statement with the MAEK operator can be used, or a casting assignment statement may be used as follows:

```
<Any_variable> IS NOW A <type>      BTW this code will be equal to
<Any_variable> R MAEK <variable> A <type>
```

### Example

```
HAI 1.2
I HAS A food ITZ "111.00033"
VISIBLE food
BTW this is how we do type casting
MAEK food A NUMBAR
VISIBLE food
KTHXBYE
```

The above code will produce the following output:

```
sh-4.3$ lci main.lo
111.00033
```

## 6. LOLCODE – INPUT/OUTPUT

This chapter will explain you how to input a value through LOLCODE terminal and how to output it onto the terminal.

### IO from Terminal

---

You can use the keyword `VISIBLE` to print something in LOLCODE. `VISIBLE` is a function which can take an infinite number of characters as input, and prints them all at once by internally concatenating them, and converting them to strings or YARN.

The `VISIBLE` function ends or terminates by a delimiter, which is either a line end or a comma.

The output is automatically terminated by the compiler with a carriage return. If the final token is terminated with an exclamation symbol (!), then the carriage returned is overridden by this symbol.

```
VISIBLE <any_expression> [<any_expression> ...][!]
```

Please note that in LOLCODE, currently there is no defined standard for printing some data to a file.

To take some input from the user, the keyword used is `GIMMEH`. It is a function which can take any number of variables as input. It takes YARN as the input and stores the value in any given variable.

```
GIMMEH <any_variable>
```

### Example

```
HAI 1.2
  I HAS A VAR ITZ A YARN BTW DECLARE A VARIABLE FOR LATER USE
  VISIBLE "TYPE SOMETHING AND ENTER"
  GIMMEH VAR BTW GET INPUT (STRING) INTO VARIABLE
  VISIBLE VAR
KTHXBYE
```

When this code is run, it will ask you to enter a number and then prints the number back in the next line automatically. When you run this code, it will print the following output:

```
sh-
4.3$ lci main.lo

TYPE SOMETHING AND ENTER
```

67

67

# 7. LOLCODE – STATEMENTS AND FLOW CONTROL

LOLCODE allows you to control the flow of program through various statements. This chapter explains different types of statements available in LOLCODE.

## Expression Statements

---

An expression without any assignment, i.e. simply calling a mathematical operation or any function, is a legal statement in LOLCODE. Once the expression is evaluated, its final value is placed in the temporary variable IT. The value of IT remains in the local scope, and exists until the next time it is replaced with an expression.

## Assignment Statements

---

Assignment statements are used to assign the output of any expression to a given variable. They are generally of the form:

```
<any_variable> <assignment operator> <any expression>
```

Please note that, you can use a variable in the expression, even before it is being assigned.

## Conditional Statements

---

### If-Then Statements

The if-then statement is a very simple operation working on the IT variable. It is similar to if-else statements in other programming languages like C and Java.

There are four keywords to apply the if-then statements.

- O RLY?
- YA RLY
- NO WAI
- OIC

The general form is:

```
<any_ expression>
O RLY?
  YA RLY
    <code to execute if above condition is true>
  NO WAI
    <code to execute in this block>
```



```
OIC
```

All of the above statements can be written in the same line separated by commas like:

```
BOTH SAEM NAMES AN "Name", O RLY?
  YA RLY, VISIBLE "My name is ABCD"
  NO WAI, VISIBLE "Your name is ABCD"
OIC
```

While using the if-then statements, an optional MEBBE <any expression> may be used between the YA RLY and NO WAI blocks.

If the <any expression> following MEBBE is True (WIN), then that block is executed. Otherwise, if that expression is false, the block is skipped until the next MEBBE, NO WAI, or OIC statements.

### Example

```
<any expression>
O RLY?
  YA RLY
    <code to be executed if true>
  MEBBE <expression>
    <code to be executed mebbe is true>
  MEBBE <expression>
    <code to be executed mebbe is true>
NO WAI
  <code to be executed if above are false>
OIC
```

### Example

```
BOTH SAEM NAMES AN "NAME"
O RLY?
  YA RLY, VISIBLE "YOUR NAME IS ABCD"
  MEBBE BOTH SAEM ANIMAL AN "OUR NAME IS ABCD"
    VISIBLE "NO ABCD"
OIC
```

## Case Statements

In LOLCODE, the keyword 'WTF?' is similar to switch in many other languages. The keyword WTF? takes IT as the expression value for comparison. To use WTF, a comparison block is opened by OMG which should be a literal, and not an expression.

Please remember that each literal must be unique, similar to the case in other languages.

The OMG block must be terminated by a GTFO statement. If an OMG block is not terminated by a GTFO, then the next OMG block is executed till GTFO is reached.

If none of the literals evaluate as true, then default case is signified by OMGWTF.

```

WTF?
  OMG <any value to compare>
    <code block to execute if expression is satisfied>
  OMG <any value to compare>
    <code block to execute if expression is satisfied>
  OMGWTF
    <code block to execute as a default case>
OIC
NAME, WTF?
  OMG "A"
    VISIBLE "ABCD"
  GTFO
  OMG "E"
    VISIBLE "EFGH"
  GTFO
  OMGWTF
    VISIBLE "ZYXW"
OIC

```

The output results of the above code will be:

"E":

```
EFGH
```

## 8. LOLCODE – LOOPS

Loops are used in programming languages to execute a set of statements multiple times. For example, if you want to print the digit 5 for five times, then instead of writing the **VISIBLE "5"** statement five times, you can run a loop with single **VISIBLE "5"** statement for five times.

Simple loops are represented with `IM IN YR <label>` and `IM OUTTA YR <label>`. Loops defined in this way are infinite loops and they should be terminated with a `GTFO` break statement.

Iteration loops have the following structure:

```
IM IN YR <label> <any_operation> YR <any_variable> [TIL|WILE <expression>]
    <code block to execute inside the loop multiple times>
IM OUTTA YR <label>
```

Please note that inside the function body, `UPPIN` (increment by one), `NERFIN` (decrement by one), or any unary function can be used.

The `TIL` keyword calculates the expression as a TROOF: if it evaluates as `FAIL`, the loop continues once more, if it evaluates as `WIN`, then the loop execution stops, and continues after the matching `IM OUTTA YR` statement.

The `WILE` keyword is the opposite of `TIL` keyword, if the expression is `WIN`, execution continues, otherwise the loop exits.

### Example

```
HAI 1.2
I HAS A VAR ITZ 0
IM IN YR LOOPY UPPIN YR VAR TIL BOTH SAEM VAR AN 10
    VISIBLE SUM OF VAR AN 1
IM OUTTA YR LOOPY
KTHXBYE
```

When the above code is compiled on any LOLCODE compiler, or on our online codingground, this will produce the following output.

```
sh-
4.3$ lci main.lo

1

2
```

|    |  |
|----|--|
| 3  |  |
| 4  |  |
| 5  |  |
| 6  |  |
| 7  |  |
| 8  |  |
| 9  |  |
| 10 |  |

## 9. LOLCODE – Functions

Functions are useful in programming because they reduce time and effort for writing the code again and again. A well written function code offers high reusability. This chapter explains you how to write and work with functions in LOLCODE.

### Definition of a Function

---

A function is a set of statements that are executed all at once by calling that function. In LOLCODE, a function's definition starts with the keyword "HOW IZ I" and the closing keyword is "IF U SAY SO".

The syntax for writing a function in LOLCODE is:

```
HOW IZ I <function name> [YR <parameter/argument> [AN YR <other _arguments..>
...]]
    <code block to execute / Set of statements to execute>
IF U SAY SO
```

### Important Points:

Consider the following important points when you are defining a LOLCODE function:

- In LOLCODE, the function can accept only a certain fixed number of arguments as an input.
- The arguments or parameters, are the identifiers that become a variable to the function.
- Functions in LOLCODE can't access any other values other than the values passed to them as arguments.

### Returning Value from a Function

---

Return in coding means something that is given back. In programming, a function can return some value to the program when its execution is completed. In LOLCODE, functions return varying values as explained below:

- **FOUND YR <any\_expression>** returns the value of the expression when function block is executed completely.
- **GTFO** returns no value (NOOB), which is similar to **return 0** in other programming languages like C and Java.
- If no other return statement is found, then **IF U SAY SO** is executed and the value in the IT variable is returned.

## Calling Functions

A function is defined in the body of program and is later called for execution. A function which accepts a given number of arguments is called as shown below:

```
I IZ <function_name> [YR <expression_One> [AN YR <expression_Two> [AN YR
<expression_Three> ... ]]] MKAY
```

While calling a function, the expression is formed by the function name, followed by the number of arguments that the function will accept. These arguments can be simple variables or any expressions. If a function accepts any expression instead of a simple value, then the expressions' values are calculated before the function is called.

Please remember that the number of arguments a function will accept, should be defined in the definition of the function.

### Example

```
HAI
HOW DUZ I MAINUMBA
  I HAS A NUMBA
  GIMMEH NUMBA
  FOUND YR NUMBA
IF U SAY SO
  VISIBLE MAINUMBA
KTHXBYE
```

When you run the above code, it will ask for an input, and then when you submit the input, you'll see the same as the result. For example, if we enter 55, it will print 55.

### Example

```
HAI 1.2
HOW IZ I MULTIPLY YR FIRSTOPERANT AN YR SECONDOPERANT
  FOUND YR PRODUKT OF FIRSTOPERANT AN SECONDOPERANT
  IF U SAY SO
  VISIBLE I IZ MULTIPLY YR 2 AN YR 3
KTHXBYE
```

The above function that performs multiplication of input operands will print the following output when you run it:

```
sh-
4.3$ lci main.lo

6
```

## Example

```
HAI 1.2
I HAS A STRINGARRAY ITZ A BUKKIT
  STRINGARRAY HAS A VAR17 ITZ "OBJECT1"
  STRINGARRAY HAS A VAR18 ITZ "OBJECT2"
HOW IZ STRINGARRAY ACCESS YR VARIABLE
  FOUND YR STRINGARRAY'Z SRS VARIABLE
IF U SAY SO
  I HAS A STRING ITZ "VAR17"
  VISIBLE STRINGARRAY IZ ACCESS YR STRING MKAY
KTHXBYE
```

The output that the above code will produce is:

```
sh-
4.3$ lci main.lo
OBJECT1
```

## 10. LOLCODE - EXCEPTION HANDLING

Exception handling is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained. LOLCODE does not have a lot of support for exception handling like other programming Languages. Similar to the Try-Catch block in other languages, LOLCODE has the PLZ-block.

For example, if you want to open a file that may or may not exist, use:

```
PLZ OPEN FILE "filename.TXT"?  
  AWSUM THX  
    VISIBLE FILE  
  O NOES  
    INVISIBLE "ERROR!"  
KTHX
```

The code that may cause an exception is written in the PLZ block, and the exception is handled in the O NOES block. Here, the INVISIBLE keyword sends an inner message to the debugger.

Please note that as LOLCODE is not maintained regularly, there are no more updates available for LOLCODE exception handling and many other features.



# 11. LOLCODE – SOME MORE EXAMPLES

The previous chapters explained you the programming in LOLCODE. In this chapter, you will learn some examples that lets you code at an advanced level in LOLCODE.

## Example 1: Program to Calculate the Power of a Number

In this example, you will find the code to calculate the power of an input number. For example, 2 raised to power 4 is equal to 16.

```
HAI 1.2
HOW IZ I POWERTWO YR NUM

    BTW RETURN 1 IF 2 TO POWER OF 0
    BOTH SAEM NUM AN 0, 0 RLY?
    YA RLY, FOUND YR 1
    OIC

    BTW CALCULATE 2 TO POWER OF NUM
    I HAS A INDEX ITZ 0
    I HAS A TOTAL ITZ 1
    IM IN YR LOOP UPPIN YR INDEX TIL BOTH SAEM INDEX AN NUM
    TOTAL R PRODUKT OF TOTAL AN 2
    IM OUTTA YR LOOP

    FOUND YR TOTAL
    IF U SAY SO

    BTW OUTPUT: 8
    VISIBLE I IZ POWERTWO YR 4 MKAY
KTHXBYE
```

The above code will print the following output once it runs successfully:

```
sh-
4.3$ lci main.lo

16
```

## Example 2: Program to Make an Array

This example shows the code for making an array with five elements and each element with value 10.

```
HAI 1.3
  OBTW
      CREATES A ONE DIMENSIONAL ARRAY WITH N ELEMENTS, EACH IS A 0
  TLDR

  HOW IZ I MAKEMATRIX YR N
      I HAS A MATRIX ITZ A BUKKIT
      IM IN YR LOOP UPPIN YR INDEX TIL BOTH SAEM INDEX N
          MATRIX HAS A SRS INDEX ITZ 10
      IM OUTTA YR LOOP

      FOUND YR MATRIX
  IF U SAY SO

  I HAS A N ITZ 5
  I HAS A MATRIX ITZ A BUKKIT
  MATRIX R I IZ MAKEMATRIX YR N MKAY

  BTW PRINTS THE CONTENTS OF THE ARRAY
  IM IN YR LOOP UPPIN YR INDEX TIL BOTH SAEM INDEX N
      VISIBLE MATRIX'Z SRS INDEX
  IM OUTTA YR LOOP

KTHXBYE
```

You can see the following output when you execute the above code:

```
sh-4.3$ lci main.lo
10
10
10
10
10
```

## Example 3: Program to Calculate the Factorial of a Number

This program shows the code to calculate the factorial of an input number.

```
HAI 1.3
    HOW IZ I FACTORIAL YR N
    BOTH SAEM N AN 0
    O RLY?
        YA RLY, FOUND YR 1
    NO WAI
        FOUND YR PRODUKT OF N AN I IZ FACTORIAL YR DIFF OF N AN 1
MKAY
    OIC
    IF U SAY SO

    VISIBLE I IZ FACTORIAL YR 6 MKAY
KTHXBYE
```

The above program prints the factorial of the number 6 and you can see the output as shown below:

```
sh-
4.3$ lci main.lo

720
```

## Example 4: Program to Design a Calculator

You can design a calculator to perform basic math operations using LOLCODE programming. Observe the code given below:

```
HAI 1.2

    I HAS A V1
    I HAS A V2
    I HAS A CHOICE
    VISIBLE "VALUE1"
    GIMMEH V1
    VISIBLE "VALUE2"
    GIMMEH V2 VISIBLE "Choose Operation? + - * /"
    GIMMEH CHOICE    CHOICE, WTF?
    OMG "+"
```

```

        VISIBLE SUM OF V1 AN V2

        GTF0
    OMG "-"
        VISIBLE DIFF OF V1 AN V2

        GTF0
    OMG "*"
        VISIBLE PRODUKT OF V1 AN V2

        GTF0
    OMG "/"
        VISIBLE QUOSHUNT OF V1 AN V2

        GTF0
    OMGWTF
        VISIBLE "CHOOSE SOME OPERATION"

    OIC

KTHXBYE

```

When we execute the above program with following input:

```

3
4
+

```

Upon execution, the above program will generate the following output:

```

VALUE1
VALUE2
Choose Operation? + - * /
7

```