LoLCode 1337

V 0.3

Programmers Guide and Reference

the_piper@web.de

2013-11-15

Table of Contents

1 Introduction

lolcode 1337 is based on the ideas of Adam Lindsay.

His original website is/was here: http://lolcode.com/about

Lolcode 1337 (speak lolcode leet) is not even in pre-alpha-stadium, but the first test programs are working already, even a tiny debugger is already included.

Lolcode 1337 is a standalone program which has a compiler, a virtual machine and a debugger.

You can run lolcode source code out of the box or compile it and run it with the virtual machine.

You can also create a runnable hexdump, a plain ASCII text you can send via email to other people, which can just run it.

You can even protect a binary or a hex dump with a password, so only people who know the password can run your programs.

LoLCode 1337 is not case sensitive, but i prefer to write it in UPPER case, and so are all examples in this document.

2 How to use this document

I would suggest that you browse through the chapters and have a look what is explained where.

Then, if you have at least an idea how to use lolcode, try the examples which come with the distribution or have a look at the code sniplets in this document and try them out.

If you have ideas or suggestions or bug reports or just a rant or just want to say thank you, use the LoLCode 1337 forum, which is here:

http://lolcode1337.freeforums.org/index.php?sid=7b36aa5d934069cbacba38d47286083f

3 Program Structure

A LoLCode 1337 program starts with a main section named HAI and ends with the command KTHXBB.

No variable definitions are allowed in the HAI section.

```
HAI
   here your lolcode
KTHXBB
```

So use the HAI section to do some output with VISIBLE and then call a subroutine.

The HAI section MUST be the last section in the source code.

Subroutines start with the LULZ command and end with the BBHF command.

```
LULZ subroutinename [PARAMETERS]
   here your lolcode
BBHF
```

To call a subroutine type a @ before the subroutine name

```
@subroutine  [PARAMETERS]
```

Example:

```
LULZ testsub param1$
   VISIBLE „param1 ="
   VISIBLE! Param1
BBHF
...
@testsub „here is parameter1"
```

You can put code into copybooks and reuse it in different programs. This is done with the CAN HAZ command.

```
CAN HAZ COPYBOOK1?
```

Replaces this line with the content of COPYBOOK1.cpy.

4 Running LoLCode 1337 programs

The suffix of the source code of a LoLCode 1337 program is **`.lol`**

The suffix of a compiled source code, a binary, is **`.lul`**

The suffix of a compiled hex dump is **`.lil`**
You can take this hex dump, which is nothing more than a plain ASCII text and send it via email to other people, they can save it with the suffix .lil and run it with lolcode at once.

The suffix of a compile list of a LoLCode 1337 program is **`.lst`**, this is needed by the debugger or just for looking at the whole source code including copybooks.

To run the source code, suffix .lol, type
`lolcode programname`
The suffix .lol is the default, so no need to type it.

To create a binary, if you want not to give away the source code in plain text, but share your programs with other ppl, type
`lolcode programname.lol -b`
which creates then programname.lul, the binary itself.

To create a compile list, which is needed to use the debugger, add the -l (listing) option or -ll (long listing) option. The debugger can use both of them.

Example:
`lolcode programname.lol -b -ll`

To use the debugger, compile listing needed, -l or -ll option, type
`lolcode programname.lol -dbg`

To run a program and produce a program trace at runtime type
`lolcode programname.lol -t`

To get rid of the start and end banners of a run of a LoLCode 1337 program, which show start and end time of a program run and some more informations, use the -nobanners option

`lolcode programname.lol -nobanners`

5 Banners

LoLCode, by default, displays a banner at program start and again at program end with informations about which program, .lol, .lul, .lil, was run, starting time, termination time, and, if in case of errors, some informations about the error itself.

If you don't want those banners to be displayed, use the **-nobanners** option.

A banner at program start.

```
************************************************************
* This is lolcode 1337  V 0.2                              *
*                                                          *
*                  Program started                         *
*                                                          *
*                      t t t . l o l                       *
*                                                          *
*                at 2013.05.10 15:44:28                    *
*                                                          *
*                  at machine piperhome                    *
*                                                          *
************************************************************
```

You can see

- the version of lolcode
- the program which was loaded,
- the start date and time
- the machine name from the environment vairable LOLCODEMACHINE.

A banner at program termination without errors:

```
**************************************************************
*                                                            *
*                    Program terminated                      *
*                                                            *
*                        t t t . l o l                       *
*                                                            *
*                  at 2013.05.10 15:43:05                    *
*                                                            *
*                    at machine piperhome                    *
*                                                            *
*        Time used 00:00:00                                  *
*        Errorcode=00                                        *
*                                                            *
**************************************************************
```

You can see

- the full program name which was loaded
- date and time when the program run ended
- the machine name taken from **`$LOLCODEMACHINE`**
- the time used by this program run, this one was pretty fast :)
- the internal error code, which is zero

A banner at program termination with error (blank lines deleted).

```
************************************************************
*                   Program terminated                     *
*                       t t t . l o l                      *
*                at 2013.05.10 15:44:28                    *
*                  at machine piperhome                    *
*       Time used 00:00:00                                 *
*       Errorcode=01                                       *
*       Program pointer at error time: 0065                *
*       Error message: FILE NOT FOUND                      *
* 00000055 (000065)                                       *
* 00000056 (000065)      IMP "isnotthere.txt" AS "test"    *
*            E R R O R   T E R M I N A T I O N            *
*            E R R O R   T E R M I N A T I O N            *
*            E R R O R   T E R M I N A T I O N            *
************************************************************
```

You can see

- the full program name which was loaded
- date and time when the program run ended
- the machine name taken from **$LOLCODEMACHINE**
- the time used by this program run
- the internal error code and message
- the program pointer where the error occured, see *.lst file

and, if an actual *.lst file is available, the line(s) of the source code where the error occured.

In this example you can see that the error code/message is FILE NOT FOUND and the error occured at an IMP command which wants to import a file „isnotthere.txt“, which is obviously not there.

There you go :)

6 The compile listing - *.lst file

To get a compile listing, specify the option -l (listing) or -ll (long listing) like

```
lolcode test.lol -ll
```

`-l` creates only a listing of the source code with line numbers and program pointers.

Line numbers are obviously the first numbes in a line, the program pointer is the second number in a line followed by the source code.

```
primes.lol                          2013.05.10 23:36:06  Page 0001
------------------------------------------------------------------
00000000 (000000) ****************************************
00000001 (000000) *                                      *
00000002 (000000) * primes.lol    Version 1 2013-03-02 *
00000003 (000000) *                                      *
00000004 (000000) * Calculates the first prime numbers *
00000005 (000000) *                                      *
00000006 (000000) ****************************************
00000007 (000000) LULZ primes
00000008 (000002) BTW - - - - - - - - - -
00000009 (000002)    I HAS A A%
00000010 (000004)    I HAS A B%
00000011 (000006)    I HAS A C%
00000012 (000008)    I HAS A D%
00000013 (000010)    I HAS A T%
00000014 (000012)    I HAS A T2%
00000015 (000014)    I HAS A I%
00000016 (000016)    I HAS A F%
00000017 (000018) BTW - - - - - - - - - -
00000018 (000018)    LOL T R 1
00000019 (000025)    VISIBLE! "1"
00000020 (000029)    IM IN YR LOOP
00000021 (000029)       UP T
00000022 (000032)       UP T
00000023 (000035)       IZ T BIGGER THAN 100?
00000024 (000047)       YARLY
00000025 (000047)          ENUF
```

-ll creates an additional byte dump of the binary code which is run by the virtual machine.

```
                                * * * BYTE-DUMP * * *
00000000 [056] x'38' c[8]
00000001 [000] x'00' c[.]
00000002 [048] x'30' c[0]
00000003 [037] x'25' c[%]
00000004 [048] x'30' c[0]
00000005 [037] x'25' c[%]
00000006 [048] x'30' c[0]
00000007 [037] x'25' c[%]
00000008 [048] x'30' c[0]
00000009 [037] x'25' c[%]
```

This can be used for debugging purposes, if you are interested how the virtual machine is working.

7 Option summary

[NONE] - load, compile and run the source code as it is, suffix .lol, or the binary, suffix .lul, or a hexdump, suffix .lil

-b – create a **`binary`**, suffix .lul, from the source code, suffix lol
-l – create a compile **`listing`** including all copybooks
-ll - create a compile **`listing`** including all copybooks and a byte dump
-t - display a program **`trace`** at run time
-dbg - run the program in the **`debugger`**, compile listing needed
-d - write a core **`dump`** to stderr, when the EEKS! Command is executed
-v - print the **`version`** of lolcode
-hex - create a runnable **`hex dump,`** simple ASCII text
-p - protect a binary or a hex dump with a **`password`**. Of course, you must specify the password too when you want to run this protected program
-wae - **`Wait At End`** for a keypress, mainly for Windows, the dosbox does not disappear, so you can read the output of your program

8 Environment variables

LoLCode 1337 uses the following environment variables:

LOLCODEMACHINE

This is used to identify the machine LoLCode runs on. For example, if you run a development machine and a productive machine, you can specify **`LOLCODEMACHINE="development"`** on the development machine and **`LOLCODE="productive"`** on the productive machine, and LoLCode will display the machine name in the banner shown when a program starts to run.

This way, if you are going to save the output of LoLCode programs like

```
lolcode program.lol -d >output.txt 2>dump.txt
```

you can always see in the output.txt on which machine the program was run.

Specify **`LOLCODEMACHINE`** on the machine itself and not in the shellscript you use to start the program, just to avoid trouble when you transport a shell script from the development machine to another machine.

9 Comments

A comment in LoLCode 1337 can be written in two different ways.

First, the BTW command:

BTW this is a comment

Second, an asteric in column 1

* this too

Your choice which way you prefer :)

10 Declaring variables

To declare variables use the I HAS A command, like:

```
I HAS A IDX%
I HAS A IIDX%
I HAS A IIIDX%
I HAS A LIMIT%
I HAS A NAME$
```

The type of the variable, numeric integer or string is determined by the character added to the variable name.

% - percentage sign, it's a numeric integer variable
$ - dollar sighn, it's a string

When you use the variable in your programs, you don't need to type the % or $, because LoLCode knows already the variable type from the definition with I HAS A.

Example:

```
I HAS A IDX%
LOL IDX R 0
```

Local and Global variables

Local variables are declared inside a subroutine.

They can only be accessed in this subroutine.

Global variables are declared outside of a subroutine.

They can be accessed from everywhere, inside every subroutine.

Example:

```
I HAS A GLOBALVARIABLE%
LULZ SUB1
       I HAS A LOCALVARIABLE%
BBHF
HAI
       @SUB1
KTHXBB
```

11 Declaring Arrays

You can't. Use instead the internal database.

This is intentional, LoLCode should be different than most other programming languages :)

For example, if you want to store a list, use the category as the list name, the key is the index and thats it.

If you want a two dimensional array, create a more clever key than just a simple index.

```
LOL KEY R „1 7“
LOL KEY R „37 225“
```

Or 3 dimensions

```
LOL KEY R „1 7 9“
LOL KEY R „37 225 18“
```

And so on.

Have a look at the database and string commands, how to do this.

12 The Debugger

To use the debugger, you must run a program with the **`-dbg`** option.

An actual compile listing is needed, because the debugger will use that to print the lines of the source code while running/debugging the program. (**`-l`** or **`-ll`** option)

Debugger commands:

help / h / ?	- display the help message
[ENTER KEY]	- step line by line through the program
U / UNTIL	- run the program to the end of the current subroutine
Z / ZOOM	- run the program and display all lines of executed source code
ZU / ZOOMUNTIL	- run the program and display all lines of executed source code to the end of the current subroutine
VDUMP	- dumps the actual variables from the varstack and their type and content
DBDUMP	- dumps the actual content of the internal data base
BREAK <line>	- sets a breakpoint at line <line>, like break 20
UNBREAK <line>	- removes a breakpoint from <line>
BREAKIF <condition>	- stops when a given condition is reached. This condition can only check INTEGER variables. Example: breakif idx = 3 Valid conditions are <, >, = and !=
CLEARBREAKIF	- removes all conditional breaks
BREAKS	- lists all breakpoints and conditional breaks
MONITOR <variable>	- displays the content of variables at every break
CLEARMONITOR	- stops monitoring variables

To leave the debugger type quit or press CTRL-c and LoLCode 1337 terminates.

A sample debugger session

We are going to debug this program:

```
************************************************
* untiltest.lol                    2013-04-13      *
* Tests the breakif command of the debugger        *
************************************************
LULZ looptest
BTW - - - - - - - - - -
   I HAS A IDX%
   I HAS A IIDX%
   I HAS A IIIDX%
   I HAS A LIMIT%
   I HAS A COUNT%
BTW - - - - - - - - - -
* full loop command with increment by 1
  VISIBLE! "********************"
   VISIBLE! "* Doing loop tests *"
   VISIBLE! "********************"
   LOL IDX R 0
   LOL LIMIT R 10
   IM IN YR LOOP UPPIN YR IDX TIL BOTH SAEM IDX AN 7
      VISIBLE IDX
      VISIBLE! "<--"
      LOL IIDX R 0
      IM IN YR ILOOP UPPIN YR IIDX TIL BOTH SAEM IIDX AN LIMIT
         VISIBLE  IIDX
         VISIBLE " "
      IM OUTTA YR ILOOP
      VISIBLE! ""
   IM OUTTA YR LOOP
BBHF
********************************************************************************
HAI
   CAN HAS STDIO?
   VISIBLE! "in untiltest.lol"
   VISIBLE! "================"
```

```
   @looptest
   VISIBLE! "End of untiltest.lol"
KTHXBB
```

We load this program with the following command

```
lolcode untiltest.lol -ll -dbg
```

so lolcode creates a long listing (**-ll)** and then switches to debug mode (**-dbg**).

Hit a few times ENTER, step line by line through the source code until you reach this line:

```
00000028 (000017)    I HAS A IIIDX%
dbg>
```

Then type vdump to see the variables and their contents, which do exist right now.

```
dbg>vdump
-------------- vdump -----------------
0) IDX                   Integer 0
1) IIDX                  Integer 0
-------------- end vdump -------------
dbg>
```

As you can see, we have right now an integer variable **IDX**, with the value 0, this is the one we want to set a conditional break on. Type

```
dbg>breakif idx = 3
Conditional break set.
00000027 (000017)
00000028 (000017)    I HAS A IIIDX%
dbg>
```

Now type **breaks** to see all breakpoints and conditional breaks.

```
dbg>breaks
idx = 3
dbg>
```

As we can see, we have only one, lolcode will stop, when the variable **IDX** has the value 3.

Now let's run the program until it stops, use the **continue** command, short **c.**

```
dbg>c
```

```
********************
* Doing loop tests *
********************
1<--
1 2 3 4 5 6 7 8 9
2<--
1 2 3 4 5 6 7 8 9
* * * conditional break (idx = 3) * * *
* * * conditional break (idx = 3) * * *
00000053 (000142)
00000054 (000142)       VISIBLE IDX
dbg>
```

which brings us here, with 2 messages about a conditional break.

Now lets see if the debugger is right and **IDX** is really 3, type **vdump.**

```
dbg>vdump
-------------- vdump -----------------
0) IDX                  Integer 3
1) IIDX                 Integer 10
2) IIIDX                Integer 0
3) LIMIT                Integer 10
4) COUNT                Integer 0
-------------- end vdump -------------
```

And yes, he is \o/

So let's finish this program run, type **c** to continue.

```
dbg>c
3
* * * conditional break (idx = 3) * * *
00000055 (000146)
00000056 (000146)       VISIBLE! "<--"
dbg>
```

Oops, another conditional break. Right, **IDX** is still 3, so lolcode stops again.

Remove all conditional breaks with **clearbreakif.**

```
dbg>clearbreakif
```

```
All breakif's cleared.
00000055 (000146)
00000056 (000146)       VISIBLE! "<--"
dbg>
```

And NOW we can continue the program run by typing **c** again, and thats it.

13 Protecting a binary/hex dump with a password

For whatever reasons you don't want others to know what a lolcode program is or not.

For example, if you create a runnable hex dump and send it via email to a friend, you don't want that any organizations, who scan emails, can find out, that this really is a lolcode program.

You can easily identify a lolcode binary or hexdump, because it begins with an uncrypted magic number, followed by the compile timestamp.

Or you just don't want everybody, who has access to your computer, to execute your lolcode programs.

So you can create a binary or a runnable hex dump and protect it with a password.

Then the magic number and compile timestamp isn't readable anymore and no one who doesn't know the password can execute this program with lolcode.

How to do that?

Easy.

To create a password protected binary or hex dump, add the -p option, like -pMyPassWord.

Example:

```
lolcode myprog.lol -b -pMyPassWord
```

to create a password protected binary. To run it, type

```
lolcode myprog.lul -pMyPassWord
```

The same for a hex dump:

```
lolcode myprog.lol -hex -pMyPassWord
```

and

```
lolcode myprog.lil -pMyPassWord
```

Easy, isn't it? :)

14 The internal data base

LoLCode 1337 has no arrays, but instead an internal data base where you can store data with an assigned key and a category.

Example:
```
ST0R DATA WITH KEY IN CATEGORY
ST0R "data2" WITH "key2" IN "testcat"
```

Reminder: ST0R contains the number zero „0“, not the letter „O“.

So you can separate stuff in different categories and there separate it with unique keys and retrieve it with the GIEF command.

Example:
```
GIEF RDATA WITH RKEY FROM RCATEGORY
```

which fills the string variable RDATA with the data identified by RKEY and RCATEGORY.

The data, key and category the internal data base can handle are strings and string variables.

You can import a whole text file into the data base with the IMP command, the category will then be the file name, for example, and the key is the line number of every line of text.

Example:
```
IMP "names.txt" AS "file"
```

Which will import the file „names.txt“ into the category „file“.

To see the whole content of the data base for testing purposes, use the DBDUMP command, which works too when in debugging mode (**`-dbg`**).

15 String commands

$CAT is used to concatenate strings.

Example:
```
I HAS A FIELD$
LOL FIELD R "ONE"
LOL FIELD R $CAT " TWO"
```

Declares a string variable named FIELD, assignes the value „ONE“ to it, then concatenates the value „ TWO“, so the result is „ONE TWO“.

%2$ converts an integer into a string.

Example:
```
LOL FIELD R %2$ 47
```

Copies the string „47“ into the string variable FIELD.

$2% converts a string into an integer value. If the string doesn't contain an integer number, the result is zero.

Example:
```
I HAS A NUMFIELD %
LOL NUMFIELD R $2% „23“
```

16 Loop commands

Loop commands are:

```
IM IN YR LOOP [LOOPNAME]
     your lolcode here
IM OUTTA YR LOOP [LOOPNAME]
```

This is an infinite loop, you can leave it with the ENUF command.

Example:

```
I HAS A I%
I HAS A T%
I HAS A T2%
I HAS A B%
LOL B R 2
IM IN YR LOOP2
   UP I
   LOL T2 R T OVAR B
   IZ I BIGGER THAN T2?
   YARLY
      ENUF
   KTHX
IM OUTTA YR LOOP2
```

Next one:

```
IM IN YR LOOP UPPIN YR IDX TIL BOTH SAEM IDX AN 7
     your lolcode here
IM OUTTA YR LOOP
```

This is a loop which increments a loop variable up to the given limit.

Example:

```
   LOL IDX R 0
   LOL LIMIT R 10
   IM IN YR LOOP UPPIN YR IDX TIL BOTH SAEM IDX AN 7
      VISIBLE IDX
      VISIBLE! "<--"
      LOL IIDX R 0
      IM IN YR ILOOP UPPIN YR IIDX TIL BOTH SAEM IIDX AN
LIMIT
         VISIBLE  IIDX
         VISIBLE " "
      IM OUTTA YR ILOOP
```

VISIBLE! ""
IM OUTTA YR LOOP

17 Command Reference

$CAT

String concatenate, use it to concatenate two string variables or to add a string to a string variable.

$CAT is used with the LOL command, see below.

Example:

```
I HAS A VAR$
LOL VAR R „ONE „
LOL VAR R $CAT  „TWO“
VISIBLE! VAR
LOL VAR R $CAT „ „
LOL VAR R $CAT VAR
VISIBLE! VAR
```

%2$

Integer to String, converts an integer into a string variable.

Example:

```
I HAS A FIELD$
LOL FIELD R %2$ 47
VISIBLE "FIELD=>"
VISIBLE FIELD
VISIBLE! "<"
```

$2%

String to Integer, converts a string or string variable into an integer variable.

Example:

```
I HAS A NUMFIELD%
LOL NUMFIELD R $2% "12345"
VISIBLE "NUMFIELD=>"
VISIBLE NUMFIELD
VISIBLE! "<"
```

@

The GoSub command, call a subroutine.
@SUBROUTINE [PARAMETER1]....[PARAMETER-N]

Example:

```
@DISPLAY_PAGE „The header“ „The text“ „The footer“
```

*****

An asterik * in colum one starts a comment, see BTW command.

Example:

```
* Your comment here
```

BBHF

BBHF leaves a subroutine.

Example:

```
LULZ testsubroutine
   your lolcode here
BBHF
```

BTW

BTW starts a comment, see * (asterik) command.

Example:

```
BTW your comment here.
```

CAN HAS

CAN HAS is used to include libraries (unused right now)

Example:

```
CAN HAS STDIO?
```

CAN HAZ

CAN HAZ is used to include copybooks into programs.

Example:

```
CAN HAZ COPYBOOK?
```

Includes the copybook COPYBOOK.cpy into your program.

DBDUMP

DBDUMP shows the content of the internal data base in physical, unsorted order.

Example:

```
DBDUMP
```

DOWN

decrements an integer variable.

Example:

```
DOWN VAR2
```

EAT MA

EAT MA is used to reset system variables.
For examples the ERRORCODE, when you have handled it successfully.
Example:

```
EAT MA ERRORCODE
```

resets the ERRORCODE to zero.
For example, the IMP command can set an error code, FILE NOT FOUND.
To check the error code after an IMP command, write

```
I HAS A RC%
IMP „file.txt“ AS „myfile“
LOL RC R ERRORCODE
IZ RC BIGGER THAN 0
YARLY
   VISIBLE! „Error after IMP command!“
NOWAI
   VISIBLE! „No Error after IMP command.“
KTHX
```

EEKS!

EEKS! terminates the program and produces a core dump, if option -d was specified

Example:

```
EEKS!
```

ENUF

ENUF leaves a loop, see IM IN YR command.

GIEF

GIEF [DATA] WITH [KEY] FROM [CATEGORY]
Retrieves data from the internal data base, identified by a category and a key.

Example:

```
I HAS A CATEGORY$
I HAS A DATA$
LOL CATEGORY R „testcat"
GIEF DATA WITH „key1" FROM CATEGORY
VISIBLE „Retrieved data >"
VISIBLE DATA
VISIBLE! „<"
```

This data can be stored with the **ST0R** command, see there.

KEY and CATEGORY can be string variables or string constants.

DATA must be a string variable.

GIMMEH

GIMMEH [STRING VARIABLE]

GIMMEH reads a single line from a given file descriptor, default is STDIO.

Example:

```
I HAS A VAR1$
VISIBLE "Please type some text, then hit ENTER :>"
```

```
GIMMEH VAR1
VISIBLE "var1=>"
VISIBLE VAR1
VISIBLE! "<"
```

HAI

HAI is the main function of every LoLCode 1337 program and shuld be, like in C programs, at the end of the source code.

No variables can be specified in the HAI section.

IMP

IMP [FILENAME] AS [CATEGORY]

IMP imports a text file into the internal data base.
The category will be the given [CATEGORY] from the IMP command,
the key will be the line number, starting at zero.

Example:
```
IMP „customers.txt“ as „customers“
```

I HAS A

Is used for defining variables.

Example:
```
I HAS A CNT%
```

which defines the variable CNT, which is an integer variable, indicated by the %.

There are two types of variables in LoLCode 1337:

Integer, defined with the % at the end of the I HAS statement and

Strings, defined with the $ at the end of the I HAS statement.

When you use the variables in your source code, there is no need to type the % or $, because the compiler knows the variable type already from the I HAS definition.

You can't define arrays in LoLCode 1337, for that use the (coming soon) internal database.

IM IN YR

IM IN YR [LOOP NAME]

is the begin of an endless loop.
To leave this loop, use the ENUF command.

Example:
```
IM IN YR COUNTLOOP
   your lolcode here
IM OUTTA YR COUNTLOOP
```

IM IN YR [LOOP NAME] UPPIN YR [VAR] TIL BOTH SAEM [VAR] AN [NUM]

is the begin of a loop incrementing [VAR] until [VAR] is equal to [LIMIT]

[VAR] must be a numeric variable
[LIMIT] can either be a number or a numeric variable

Example:
```
I HAS A IDX%
LOL IDX R 0
IM IN YR LOOP UPPIN YR IDX TIL BOTH SAEM IDX AN 10
    VISIBLE! IDX
IM OUTTA YR LOOP
```

IM OUTTA YR

is the end of an endless loop.
To leave this loop, use the ENUF command.

Example:

```
IM IN YR COUNTLOOP
   your lolcode here
IM OUTTA YR COUNTLOOP
```

IZ

IZ is the if-statement of LoLCode 1337.

It has two branches, YARLY and NOWAI to handle the positive and negative result of the comparison.

The statement ends with KTHX.

Example:

```
IZ VAR2 BIGGER THAN 10?
   YARLY
     ENUF
   NOWAI
     VISIBLE "*"
KTHX
```

Valid comparisons are BIGGER, SMALLER, EQUAL.

KTHX

KTHX ends an IZ statement.

Example:

```
IZ VAR2 BIGGER THAN 10?
   YARLY
     ENUF
   NOWAI
     VISIBLE "*"
KTHX
```

KTHXBB / KTHXBYE

Both commands end the HAI section, there is no difference between both commands, use the one you like more :)

LOL

LOL assigns a value to a variable, which can include some simple mathematics or, for more complex calculations, PRON code.

Simple values:

The value can be a string or number or another variable. However, the variable types must match, you can't assign a string to an Integer variable and so on.

Example:
```
LOL VAR3 R "var3 = TestString"
```

PRON calculations

You can use LOL to assign the result of a PRON calculation, a FORTH like way to write formulas.

Example:
```
LOL VAR2 R PRON 2 dup + 4 -
```

Simple mathematics

Like A = B + 3, which would look in LoLCode like this:

Example:
```
LOL A R B UP 3
```

where
UP is +
NERF is -
TIEMZ is *
OVAR is /

Special assignments

Assign a random number to an integer variable.

Example:
```
LOL VAR2 R RND 64
```

Assign the system error code to an integer variable.

Example:
```
LOL RCODE R ERRORCODE
```

LULZ

LULZ [SUBROUTINE NAME] [PARAMETERS]

LULZ defines a subroutine and it's parameters.

Example:
```
LULZ testsub2 param$
```

testsub2 is the name of the subroutine, which has one parameter, param$, which is a String.

To call a subroutine use the @ command.

Example:
```
@testsub2 „this is the string parameter“
```

NOWAI

NOWAI belongs to the IZ command, see there.

PRON

Polish Reverse Operation/Notation, the way how formulas are written in FORTH, or how ancient TI calculators work.

The following FORTH words are supported by the PRON command:
+, -, *, /, ROT, DROP, SWAP, OVER, DUP

Example:
```
LOL VAR2 R PRON 2 dup + 4 -
```

which will finally assign the value 0 to VAR2

RND

RND [LIMIT]

RND is used to assign a random number to an integer variable in a LOL statement. You can specify a limit, then the random number will be between 0 and smaller than the limit, or, without limit, the random number will be between 0 and the max value of an Integer.

Example:
`LOL VAR2 R RND 32`
will produce random numbers from 0 to 31

Example:
`LOL VAR2 R RND`
will produce unlimited random numbers

ST0R

ST0R [DATA] WITH [KEY] IN [CATEGORY]

written „**`ST zero R`**“, not „**`ST letter O R`**“!

Stores data in the internal data base, identified by a category and a key.

Example:

```
I HAS A CATEGORY$
LOL CATEGORY R „testcat“
ST0R „data1“ WITH „key1“ IN CATEGORY
```

This data can be retrieved with the **`GIEF`** command, see there.

DATA, KEY and CATEGORY can be string variables or string constants.

UP

Increments an integer variable.

Example:
```
UP VAR2
```

VISIBLE

VISIBLE [TEXT OR VARIABLE]

displays a text or the content of a variable. No carriage return/line feed is printed.

Example:
```
VISIBLE „HAI WORLD"
```

VISIBLE!

displays a text or the content of a variable. Crriage return/line feed is printed.

Example:
```
VISIBLE! „HAI WORLD"
```

YARLY

belongs to the IZ command, see there.

---

18 Code Sniplets

Here is a collection of code sniplets you can use to try out different commands.

Simple lolcode program – assign a value to a variable

```
* This is how to write a comment, it starts with a '*' in column 1
* Another way to write a comment is the BTW command, like this:
BTW comment
* but i find the '*' in column 1 more handy :)
*
*******************************************************
* ttt.lol                          2013-05-09         *
*                                                     *
* Author:                          the_piper          *
*                                                     *
* Description: sample program to show the structure   *
*              of a lolcode program                   *
*******************************************************
*************************************************************
* This is a subroutine named section1
*
LULZ section1
*
* First we define a variable named FIELD, type integer, thats
* what the % at the end of the variable name is for.
*
   I HAS A FIELD%
* assign the value 123 to the variable FIELD we dont need to write
* the variable type '%', lolcode knows it already from
* the declaration above
*
   LOL FIELD R 123
*
```

```
* Display some text..
*
     VISIBLE "FIELD = "
*
* and the content of FIELD with a carriage/return, thats what the
* '!' at the end of VISIBLE is for.
*
     VISIBLE! FIELD
*
* and bye bye, have fun, leave the subroutine :)
*
BBHF
************************************************************
* This is the main section, where the program flow starts, at the
* HAI command
*
HAI

* unused right now, just here for historical reasons
   CAN HAS STDIO?

* display some text
     VISIBLE! "in ttt.lol"
     VISIBLE! "============"

* call the subroutine named section1
     @section1

* display some more text
     VISIBLE! "End of ttt.lol"

* and thats it, okay, thanks, bye bye
KTHXBB
```

Or, shorter and more readable:

```
*******************************************************
* ttt.lol                          2013-05-09         *
*                                                     *
* Author:                          the_piper          *
*                                                     *
* Description: sample program to show the structure *
*              of a lolcode program                   *
*******************************************************
LULZ section1
   I HAS A FIELD%

   LOL FIELD R 123
   VISIBLE "FIELD = "
   VISIBLE! FIELD
BBHF
*********************************************************************
HAI
     CAN HAS STDIO?
     VISIBLE! "in ttt.lol"
     VISIBLE! "============"
     @section1
     VISIBLE! "End of ttt.lol"
KTHXBB
```

19 Contact

You can contact me via email the_piper@web.de, but i would prefer to use the forum, which can be found here:

http://lolcode1337.freeforums.org/index.php

Post there if you have any questions, problems, suggestions and so on.