Lytescript (Revision 1) Specification

# Abstract

Lytescript (also referred to as Lyte) is a stack-based, multi-paradigm programming language that strives to have as little syntax as possible while still maintaining flexibility. With its “batteries-included” philosophy towards its standard library, this means that the language is easy to learn but powerful nevertheless. Lyte is primarily a mixture of the functional, loosely-typed and object-oriented programming language paradigms each of which affect the language differently. For example, it considers everything to be an object and reduces all arithmetic operations to function-calls.

# Syntax Overview

## In-line Functions

{ /\* Function Body \*/ }

@( /\* Arguments \*/ ) { /\* Function Body \*/ }

Lytescript provides two ways to create in-line functions, also called blocks, which simply create a new block and push it onto the stack. The first of these is simply the function’s body surrounded by curly-brackets whereas the second method denotes a set of named arguments. The latter is especially useful whenever arguments that would normally be lost after being popped off of the stack need to be used multiple times.

## Function Invocation

F(a, b) // Traditional  
b a F // Stack-based  
b `F` a // Infix notation

Lytescript provides multiple ways to invoke a function for the sake of code readability. The first of which is the “traditional” method which “invisibly” wraps each of its arguments in blocks and pushes them onto the stack in reverse-order. The stack-based method is the simplest way to invoke a function, as it simply assumes that the programmer has prepared the arguments on the stack and invokes the function. Lastly, using infix notation swaps the order of execution of the statement surrounded by back ticks and the following statement. Because of these properties, all three of the above examples are equivalent. Note, the infix method does not imply anything about the order of operations besides the aforementioned swapping; the limitation of this can be seen in the example below.

4 `+` 2 `\*` 6 /\* This results in 36 rather than the result accounting for  
 the order of operations, 16 \*/

## Primitives

### Numbers & Strings

4 4.0 0b100 0x4 04 4E0  
"An example string"

Lytescript provides several ways to represent a number, from right to left (in the above example) integer, floating point, binary, hexadecimal, octal and scientific. Each of these will push a representative Number object onto the stack. Strings are single-line chunks of text surrounded by double-quotes that allow for Javascript-style escapes.

### Objects

%{ Key: Value, Key: Value, ... }

Objects are sets of key-value pairs (aka maps) that allow for keys that are valid identifiers, numbers or strings while values can be of any type. Note, the underlying type for all keys is string therefore one should exercise caution when using numeric strings with trailing zeros as a key (such as in the case of A[“1.0”]) since they might be interpreted incorrectly.

### Lists

%[ Value, Value, ... ]

Lists are integer-indexed collections of values of any type.

### Ranges

%[ Start:Finish ]  
%[ Start:Step:Finish ]

Ranges are lists of numbers over that step by 1, -1 or a given amount over a given range (inclusive).

## Assignment Operations