Email Lang Language Tutorial

Elliot Hagyard, M. Fay Garcia, Yosef Jacobson

November 29 2023

Warning

The purpose of Email Lang™ is to be obtuse and incredibly verbose.
On behalf of Aperture Science, we apologize in advance for any negative experiences caused by interaction with this language.

1 Introduction

1.1 Description

With Email Lang™, we sought to leverage existing synergies to bridge the gap between programmers and the corporations they represent and, above all, to boost productivity and profitability.

In the workforce, programmers rarely complain about programming. Instead, they bemoan the constant interruptions of meetings and emails. To reduce interdepartmental friction, we seek to join the two activities together into one cohesive whole.

The concept is an email/corporate lingo programming language in the spirit of languages like the Rockstar programming language or the Tabloid Programming Language. So, the idea is to define the language in such a way that a program could look like an actual, perhaps nonsensical, email.

1.2 Language Descriptions

Email Lang™ is an esolang (Esoteric Programming Language). It is a compiled, strongly & statically typed language. Type declarations are given by the first capitalized letter in a variable name and variables are mutable. It is a C-like procedural, imperative language. Every sentence ends with a period. Operators are left associative, since we read from left to right. There are no comments because the language speaks for itself.

2 Tutorial

The syntax of Email Lang™ is reminiscent of the email that took you three and a half hours of googling and pandering to write, but to which your corporate overlord shot back “yep, works for me” in less than thirty seconds. It has several useful keywords and operators that really highlight the brevity of corporate jargon.

Like a typical email, your Email Lang™ program must open with the word “Dear” followed by your command-line arguments, which must be the names of variables. We will explain variables more down below. If there are no command-line arguments to be had, your program must open with “To whom it may concern,”.

Similar to Java or C, an entire Email Lang™ program my technically be written on one line, as only spaces are required to delimit different expressions within the language. But would this be an appropriate way to write an email, or, for that matter, a program? Not quite. Thus Email Lang™ also inherits a C-like style.

So, we’ve established that your program must have good grammar. Following with this, each sentence or expression that you write must be followed by a period.

Additionally, as an email begins, so it must end. Your Email Lang™ program must close with ”Best, ...”

To this end (among others, perhaps), the following sections cover the basic operators for types as well as some information and instructions on how to use variables, loops, and conditional statements. By the end of this document, we at Aperture Science hope that you might be able to write your very own Email Lang™ program! Go forth and make sure to give it 110%!

2.1 Types

Email Lang™’s types are mutable and support comparison by default.

2.1.1 Basic Types

The types of Email Lang™ are similar to those of other languages such as Java. They include Boolean values, integers, and Strings.

The keywords for Booleans are a little different than your typical Java or C program. In Email Lang™, we seek to optimize productivity and every opportunity! Thus, we treat Booleans as follows:

```
    true = yep
    false = nope
```

Integers and Strings are as you know them.

2.2 Declarations and Definitions

2.2.1 Declaration of Variables

Variables are declared using the keyword *said*. The type of a variable is specified by the first letter of the variable's name. Variables must begin with either "B", "I", or "S". Best practice is to make variable names human names, but this is not strictly necessary. Variable names beginning in a "B" are of type Boolean. Those beginning with "I" are of type integer, and those beginning with "S" are of type String.

For example, the variable *Bob* would be a Boolean, *Ian* would be an integer, and *Sarah* would be a string. Below are some examples of variable declaration:

```
Bob said yep.
Ian said 5.
Sarah said "sounds good!".
```

Once a variable is declared, you can use it in other expressions (where appropriate, of course). And, since variables are mutable, their values can be reassigned as necessary.

2.2.2 Functions

Email Lang™ supports function declarations using the following syntax:

```
Subject: <function_name>.
    Dear <input_variable1>, <input_variable2>, ...,
    <statement1>,
    <statement2>,
    ...,
    RE: <return_value>.
Best, <return_type>.
```

<return_type> is a name that defines the return type of the function.
It can be any name that starts with "B", "I", or "S", even if that name is used elsewhere.

Functions with no input parameters may begin with "To whom it may concern," as a greeting instead of "Dear,".

Functions are called using the following syntax:

```
SEE: <function_name> with: <input_variable1>, <input_variable2>, ...,
```

Each Email Lang™ program requires a special function named "Main" which will execute when the program is run. Input variables defined in "Main" represent command line arguments for the program.

2.2.3 Loops

Email Lang™ supports basic while loops. The syntax is

```
Keep <bool_expr> in the loop regarding:
    <code to execute>
Thanks.
```

Where <code to execute> is executed continuously until <bool_expr> evaluates to yep.

2.2.4 Conditionals

Conditional statements have the following syntax:

```
Suppose <bool_expr>: then <then code>; otherwise, <otherwise code>
Thanks.
```

If <bool_expr> evaluates to yep, the code in <then code> is executed.
If it evaluates to nope, the code in <otherwise code> is executed.

2.2.5 Nesting

As Email Lang™ is designed to capture the workflow of a typical workplace, traditional nesting is not permitted.

Instead, nesting must be achieved through functions in order to simulate the constant sending and receiving of lengthy emails with little useful information.

An example of an if statement nested inside of a loop is given on the next page. The same basic structure can be used to nest multiple loops or if statements, and deeper nesting can be achieved with additional function declarations.

```
Subject: If.
    Dear Ian,
    Suppose Ian is greater than 1: then
        Highlight Ian;
        otherwise, highlight "less than 1"
    Thanks.
    RE: Ian.
Best, Ian.

Subject: Main.
    To whom it may concern,
    Ivan said 0.
    Keep Ivan is less than 10 in the loop regarding:
        SEE: If with: Ivan,
        Highlight "\n",
        Ivan said Ivan piggybacking off of 1,
    Thanks.
Best, Ivan.
```

2.3 Operators

To print to the console, use the unary operator “Highlight”:

```
Highlight "Hello".
Highlight Sarah.
Highlight Igor.
```

2.3.1 Arithmetic Operators

Email Lang™ supports the basic arithmetic operators:

```
x + y = x piggybacks off of y
x - y = x drills down on y
x * y = x joins forces with y
x div y = x leverages y
x mod y = x remains to be seen of y
```

2.3.2 Boolean Operators

Email Lang™ employs the typical boolean operators by using the literal words. Thus “or”, “and”, and “not” are the pertinent keywords here.

2.3.3 Comparison Operators

Email Lang™ has several robust comparison operators that return boolean values.

1. To compare the equality of two variables of the same type, use the key-phrase “on the same page as”. Strings, integers, or booleans can be compared for equality. For example:

   ```
   Sarah is on the same page as Sebastian.
   ```

2. To see if one variable is less than another, use the key-phrase “less than”. This operator can only be used on integers.

   ```
   Ian is less than Isaac.
   Isaac is less 5.
   ```

3. Similarly, use the key-phrase “greater than” to see if one integer is greater than another.

   ```
   Iago is greater than 10.
   Iago is greater than Isaac.
   ```