Meowlang Programming Language Reference Manual

Language Guru: Carolyn Chen (cec2192)

Manager: Megan Frenkel (mmf2171)

System Architects: William Penney (wjp2114) & Lauren Pham (lyp2106)

Tester: Michelle Lin (ml4080)

Programming Languages and Translators
Spring 2021

Contents

1 Introduction and Overview

Meowlang is an esoteric programming language inspired by LOLCODE, a language created by Adam Lindsay based on internet lolspeak. Meowlang pushes the boundaries of language design in a creative, humorous way while still providing functionality and usability. Overall, Meowlang abbreviates and/or adjusts original LOLCODE keywords and syntax where needed to improve readability. But more specifically, Meowlang improves on the safety and functionality of LOLCODE in two ways. Firstly in safety, unlike LOLCODE's dynamic typing, Meowlang enforces strong static typing to reinforce the integrity of users' code. Secondly in functionality, Meowlang provides an implementation of arrays and limited support for object-oriented programming, both of which LOLCODE currently lacks. Arrays are an important and versatile construct in programming, while the introduction of object oriented programming allows the creativity of the esolang to shine and provides helpful constructs for organizing data and functionality to Meowlang programmers.

2 Language Paradigm

Meowlang is heavily inspired by LOLCODE, sharing similar syntax and keywords. The original LOLCODE can be found here: `https://github.com/justinmeza/lolcode-spec/blob/master/v1.2/lolcode-spec-v1.2.md`.

However, Meowlang has some key differences in its language paradigm. Meowlang shares similarities with both C and Java, with its lack of garbage collection and object-oriented programming features, respectively. Meowlang has object-oriented features like classes, class variables, class methods/functions, instance variables and instance methods/functions. Memory is manually managed through constructors and destructors.

3 Programming in Meowlang

3.1 Conventions

3.1.1 Whitespace

Spaces are used to demarcate tokens in Meowlang, although some keyword constructs may include spaces (see **3.1.3**). However, Meowlang is not whitespace sensitive; multiple spaces and tabs are treated as single spaces and are otherwise irrelevant. Indentation is also irrelevant. This means that statements may span multiple lines, as long as the end of the statement is properly marked using the "." character, as explained in section **3.1.4**. Nevertheless, it is recommended to make thoughtful use of tabs, new lines and spaces and use the Meowlang conventions illustrated in the following code samples to maximize readability.

3.1.2 Identifiers

Identifiers in Meowlang cannot be strictly numeric (i.e., '123'). Convention dictates that identifiers do not begin with numeric characters, but they can be a combination of alphanumeric characters. Meowlang uses the following capitalization conventions for different types of identifiers:

- Classes: All capitalized characters: `[A-Z][A-Z0-9]*`
- Functions: First character is capitalized: `[A-Z][a-z0-9]*`
- Variables: Lower case characters: `[a-z][a-z0-9]*`

Code examples provided in this reference manual illustrate these identifier rules.

3.1.3 Keywords

Meowlang utilizes a set of "reserved" words or keywords that cannot be used as identifiers. Keywords have all capital letters and are case sensitive. The following is a list of keywords Meowlang relies on, by category:

Listing 1: `keywords.meow`

```
1 PSST (1) Structure and Control of Flow
2 IZ GIMME HAI ITZ ME KBYE GIVE WIT R AN NEW MAEK BLEEP PURR  O
     RLY? YA RLY NO WAI IM IN YR LOOP UPPIN NERFIN
3
```

```
4 PSST (2) Types
5 CLASS FUNC YARN BOO AYE NAY NUMBR NUMBAR BUCKET
6
7 PSST (3) Operators
8 CAT SUM DIFF PRODUKT QUOSHUNT MOD BIGGR SMALLR BOTH EITHER NOT
    SAEM DIFFRINT THAN OF
```

3.1.4 Blocks and Scope

As a whitespace-insensitive language, Meowlang requires programmers to make use of two different constructs to denote (a) the end of statements and (b) the beginning and end of function and class definitions.

All statements in Meowlang must end with the period (`"."`) character to indicate completion. Its use is analogous to the use of the semi-colon in the C programming language and should be used in the same way.

The keywords `HAI` and `KBYE` denote the beginning and end of both class and function declarations, respectively. These keywords enclose the statements relevant to the function/class, similar to the way curly brackets are used in C, except that the opening bracket in Meowlang *precedes* the function/class definition. Variables declared within a `HAI` and `KBYE` block are local variables available only within the scope of these "brackets." Taking into consideration the verbosity of Meowlang, it is recommended that additional `HAI` and `KBYE` brackets are used outside of declarations to add structure and readability to code.

3.1.5 Comments

The `PSST` keyword precedes every comment and continues until the end of the line, for single line comments and comments inline with code. There are no multi-line comments.

Listing 2: `comment.meow`

```
1 PSST This is a valid single-line comment
2 ITZ ME YARN kitty IZ "Furry".  PSST Comments inline w/ code
```

3.2 Built-In Data Types

Four data types are supported by Meowlang out of the box: strings (`YARN`), integers (`NUMBR`), floats (`NUMBAR`) and booleans (`BOO`).

3.2.1 Strings

What other languages refer to as a "string" Meowlang refers to as an instance of the `YARN` data type. `YARN` literals must be demarcated by double quotation marks, and the content of a string can be any sequence of characters. Under the hood, strings are arrays of characters, but there is no separate type for characters. A character is a string of length 1. The `YARN` type is immutable, so manipulations of a `YARN` variable always produce a new `YARN`.

Listing 3: `valid_and_invalid_strings.meow`

```
1 PSST These are valid strings
2 ""
3 "hi, I'm a string"
4
5 PSST These are not valid strings
6 "Something seems incomplete here...
7 'I am not a string'
```

3.2.2 Integers and Floats

Numeric data types referred to as integers and floats in other languages correspond to the `NUMBR` and `NUMBAR` data types in Meowlang, respectively.

Integer literals are a sequence of digits and can be expressed as: `['0'-'9']+`. Note that if your program provides an integer literal with leading zeros, such as `002`, this is read as the integer literal `2`.

Because Meowlang relies on the period character `"."` to terminate statements, float and integer literals have one key difference from other language implementations: whereas other languages would consider `2.` a float declaration, Meowlang considers it an integer. Writing `2.` would otherwise be ambiguous and could mean either a float declaration without a termination or an integer with a termination character. If you want to declare a float, you

must include a value after the decimal point (i.e., `2.0`). Note that if you choose to utilize expressions using `e` or `E`, such as `1e-10`, the data type will be interpreted as a float.

To be explicit, the following regular expression is specified for floats, where `digits` is equivalent to `[‘0’-‘9’]+`:

```
digits '.'  ((digit+ | ( ['e' 'E'] ['+' '-']? digits))
    | (digit* ( ['e' 'E'] ['+' '-']? digits )))
```

3.2.3 Boolean

Meowlang supports boolean values with the `BOO` data type. Items of type `BOO` can be have either the value `AYE` (true) or `NAY` (false), which underneath the hood correspond to values of 1 and 0, respectively. Note that boolean values can neither be used in substitution of 1 and 0 and cannot be cast into integer values 1 and 0.

3.3 Variables

Variables are declared with the keyword phrase `ITZ ME` followed by the variable type and selected identifier. To create new identifiers, please refer to the identifier rules in section **3.1.2**. Below is a template variable declaration:

```
ITZ ME <type> <identifier>.
```

Note that is is possible to define and declare a variable at the same time. Defining a variable requires use of the keyword `IZ`, which acts like the assignment operator “=” in other languages. Variable identifiers must be unique within a scope.

```
ITZ ME <type> <identifier> IZ <value>.
```

Below, find examples of variable declarations and definitions for various built-in data types.

Listing 4: `definitions.meow`

```
1 ITZ ME NUMBR num IZ 2.
2 ITZ ME YARN random_string IZ "This is a string".
3 ITZ ME NUMBAR value IZ 2.0.
4 ITZ ME BOO fact IZ AYE.
5 ITZ ME BOO fiction IZ NAY.
```

3.3.1 Type Casting

Among the different variable types, only integers (`NUMBR`) and floats (`NUMBAR`) may be casted. Casting a `NUMBR` to a `NUMBAR` concatenates a '.0' to the integer value. Casting a `NUMBAR` to a `NUMBR` truncates the float to the decimal point. Casting is demonstrated below using the keyword `IZ`.

To convert a float to an integer:

```
NUMBR <int variable> IZ NUMBAR <float variable>.
```

To convert an integer to a float:

```
NUMBAR <float variable> IZ NUMBR <int variable>.
```

3.4 Arrays

An array is called `BUCKET` in Meowlang. Meowlang supports the creation of fixed-length arrays with size known at compile time or dynamic arrays created at runtime. Each `BUCKET` can only hold elements of one type; valid types include primitives as well as user-defined classes. To create a new `BUCKET`, you have several options:

1. Declare a new `BUCKET`, specifying the size of the `BUCKET` and the type of each element in the `BUCKET`. You cannot provide more elements than the number specified by the given size:

   ```
   MAEK <identifier> NEW <bucket_type> BUCKET OF <bucket_size>,
       WIT element1_value AN element2_value AN ... AN elementX_value.
   ```

2. Declare a new `BUCKET` with a specified size, but no elements initialized. Be careful in using these arrays as the memory will be allocated, but the contents will be "garbage" until the program sets the values explicitly:

   ```
   MAEK <identifier> NEW <bucket_type> BUCKET OF <bucket_size>.
   ```

3. Declare a new `BUCKET` with both size and contents unspecified:

   ```
   MAEK <identifier> NEW <bucket_type> BUCKET.
   ```

Meowlang requires that users provide either a integer variable (with a value `>0`) or a integer literal (`>0`) for the array size (i.e., what is referred to as `<bucket_size>` in the templates above). In other words, the compiler will not accept any arbitrary expression. This decision was made for practicality purposes; Meowlang is a verbose language and this rule is in place to ensure readability. Instead, users should create an integer variable, define it using whatever complex expression they require, and use that variable in the array declaration, as in the code example in **Listing 5** below.

Individual `BUCKET` elements may be accessed using standard bracket notation to access element in index `i`: `<bucket_identifier>[i]`. Array indexing starts at index `0` and ends at `array_length-1`. Below, we create an array of pet names and access the first element, which is the string "Lucky." You cannot access elements beyond the defined size of the array.

Listing 5: `array_declaration.meow`

```
PSST Creating a new array of three strings called my_pets
ITZ ME YARN BUCKET OF 3 my_pets
    WIT "Lucky"
    AN "Elliot"
    AN "Wellington".

PSST Accessing the first element of my_pets, "Lucky"
ITZ ME YARN first_pet IZ my_pets[0].

PSST Bucket size is left unspecified
MAEK unknown_array NEW YARN BUCKET.

PSST Bucket size specified, no contents
MAEK empty_array NEW BOO BUCKET OF 10.

PSST Array size calculation prior to creation of array
ITZ ME NUMR complex_expr IZ SUM OF 2 AN 4.
MAEK complex_array NEW YARN BUCKET OF complex_expr.
```

3.5 Basic Operations

Meowlang has support for a number of built-in operators that allow programmers to perform basic math, boolean expressions and comparisons out of the box. These operators rely on prefix notation, taking the form:

Unary operators:

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

Binary operators:

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

More specifically, Meowlang supports the basic operations outlined below.

3.5.1 Math

Basic math operators are binary prefix operators. These operations are available for use on items of type `NUMBR` and `NUMBAR` only.

```
SUM OF X AN Y       PSST +
DIFF OF X AN Y      PSST -
PRODUKT OF X AN Y   PSST *
QUOSHUNT OF X AN Y  PSST /
MOD OF X AN Y       PSST modulo
BIGGR OF X AN Y     PSST max
SMALLR OF X AN Y    PSST min
```

Note that each `X` and `Y` below could be another math expression, such that the operators can be nested. As expected, multiplication and division have higher precedence than subtraction and addition. The use of a prefix operator also clearly denotes how an expression should be interpreted, without further parentheses, and operators are right-associative. This means that if you wanted to write an expression for `2 * 4 + 5`, you could write either:

```
SUM OF 5 AN PRODUKT OF 2 AN 4.
SUM OF PRODUKT OF 2 AN 4 AN 5.
```

3.5.2 Boolean

Boolean operators are limited to the following set of operators. Note here that `X` and `Y` must themselves be boolean (`AYE`/`NAY`) conditions:

```
BOTH OF X AN Y      PSST X and Y
EITHER OF X AN Y    PSST X or Y
NOT X               PSST not X
```

3.5.3 Comparison

Comparisons of two items can be performed using the following operators. Note that comparisons of two `NUMBR`s use integer math, and floating point math if one or both is a `NUMBAR`. You cannot compare a string and an integer, or a primitive with any custom object. Only comparisons between primitives are supported.

```
SAEM X AN Y         PSST AYE (true) if x == y, else NAY (false)
DIFFRINT X AN Y     PSST AYE (true) if x != y, else NAY (false)
SMALLR X THAN Y     PSST AYE (true) if x < y, else NAY (false)
BIGGR X THAN Y      PSST AYE (true) if x > y, else NAY (false)
```

3.5.4 Concatenation

Two variables of type `YARN` can be concatenated into a single `YARN` using the `CAT` operator. Note that this creates a new `YARN`.

```
CAT X AN Y              PSST x + y = xy
```

Listing 6: `basic_operators_demo.meow`

```
1 PSST ~~ Math Examples ~~
2 SUM OF PRODUKT 3 AN 4 AN 5          PSST (3 * 4) + 5
3 BIGGR OF 15 AN QUOSHUNT 100 AN 10   PSST 15 > (100 / 10)
4
5 PSST ~~ Boolean Examples ~~
6 PSST (2 < 4) && (10 > 12) = false
7 BOTH OF SMALLR 2 THAN 4 AN BIGGR 10 THAN 12
8
9 PSST ~~ Concatenation Example ~~
```

```
10 PSST create string "one fish two fish red fish blue fish"
11 CAT CAT CAT "One fish " AN "two fish " AN "red fish " AN "blue
       fish"
```

3.6 Control Flow

3.6.1 If-Then-Else

Branching in Meowlang is accomplished by if-then-else statements, as in other languages. The template for a set of if-else conditions is as follows:

```
<expression>          PSST Conditional expression
O RLY?                PSST If keyword
YA RLY HAI            PSST Then keyword and opening bracket to begin code block
     <code block>     PSST Jump here if expression is true
KBYE                  PSST Closing bracket on code block
NO WAI HAI            PSST Else keyword and opening bracket to begin code block
     <code block>     PSST Jump here if expression is not true
KBYE                  PSST Closing bracket on code block
```

Note that it is possible to omit the `HAI` and `KBYE` keywords if the code block includes exactly a single statement, as in:

```
<expression>
O RLY?
YA RLY
    <statement>
NO WAI
    <statement>
```

It is also possible to omit the `ELSE` if you only need an `IF` condition:

```
<expression>
O RLY?
YA RLY
    <statement>
```

The following code snippet illustrates usage of the if-then-else construct in Meowlang. Note carefully that there is no period (".") after the comparison expression and that although the construct spans multiple lines, the whitespace is not required.

Listing 7: `if_else.meow`

```
1  HAI ITZ ME FUNC conditions_example,
2          PSST Test if-then-else
3          ITZ ME YARN condition.
4
5          PSST IF THEN
6          SMALLR 4 THAN 10
7          O RLY?
8          YA RLY
9              condition IZ "math is the only Truth".
10
11         PSST IF THEN ELSE
12         SAEM 4 AN 4
13         O RLY?
14         YA RLY
15             condition IZ "same same".
16         NO WAI
17             condition IZ "not same same".
18
19         PSST IF THEN ELSE with code blocks
20         SAEM "kittens" AN "puppies"
21         O RLY?
22         YA RLY HAI
23             condition IZ "this is not the correct answer. ".
24             condition IZ CAT condition AN "pick a team!".
25         KBYE
26         NO WAI HAI
27             condition IZ "clearly kittens are cuter".
28             condition IZ CAT condition AN "or are they?".
29         KBYE
30
```

```
31          GIVE condition.
32  KBYE
```

3.6.2 For Loops

The only count-controlled loop in Meowlang is the for-loop. The loop may either increment an index variable value (using the `UPPIN` keyword) or decrement an index variable value (using the `NERFIN` keyword) with each iteration. A general template for creating a loop is provided below for the increment case (replace `UPPIN` with `NERFIN` for the decrement case):

```
IM IN YR LOOP <index var identifier> UPPIN <index var assignment (optional)>
    AN <termination condition>
HAI
    <code block>
KBYE
```

Note that you may initialize the value of the index variable in the loop, as is customary in many languages. For example, a common programming pattern is to set an index variable to `0` at the beginning of a loop. However, the index variable assignment can be omitted when you have previously declared and instantiated the index variable you want to use, as in the first two loops in the **Listing 8** example below.

Listing 8: `for_loops.meow`

```
1  HAI ITZ ME FUNC loops_test,
2
3          ITZ ME NUMBR count.
4          ITZ ME YARN condition.
5          count IZ 0.
6
7          PSST Incrementing with already instantiated index
8          IM IN YR LOOP count UPPIN AN SMALLR count THAN 10
9              condition IZ "count is still not 10".
10
11         PSST Decrementing with already instantiated index
12         count IZ 20.
13         IM IN YR LOOP count NERFIN AN BIGGR count THAN 10 HAI
```

```
14              condition IZ "count is still more than 10".
15              condition IZ CAT condition AN "count is".
16              condition IZ CAT condition AN count.
17          KBYE
18
19          PSST Instantiating index and incrementing
20          IM IN YR LOOP index NERFIN index IZ 15 AN BIGGR index
        THAN 10
21              condition IZ "index is still more than 10".
22  KBYE
```

3.7 Functions

Functions in Meowlang have a fixed number of zero or more arguments and can return at most one value. The types of arguments and return types must be specified in the function definition. A new function can be defined using the following syntax pattern, which makes use of the `HAI` and `KBYE` scoping keywords. The list of parameters uses the `WIT`.. `AN` .. construct, as previously seen in array declarations. Note that the number of arguments is may extend indefinitely, though it is recommended to keep the number of arguments to no more than four for readability:

```
HAI ITZ ME <return_type> FUNC <function_identifier>
    WIT <arg1_type> <arg1> AN <arg2_type> <arg2> ... ,
    ... all statements go here...
KBYE
```

To return a value from a function, use the keyword `GIVE` and any expression:

```
GIVE <expression>.
```

Below, we define a new function called ‘Chase‘ that takes two `YARN` arguments and returns another `YARN` representing the concatenation of the two provided strings with “chases”.

Listing 9: `chase.meow`

```
1  # PSST return values and arguments are optional
2  HAI ITZ ME FUNC Do_Nothing,
3      PSST This function does nothing
```

```
4 KBYE
5
6 HAI ITZ ME YARN FUNC Chase WIT YARN bad_cat AN YARN poor_mouse,
7     GIVE CAT CAT bad_cat AN " chases " AN poor_mouse.
8 KBYE
```

Once a function is defined, it can be called using the keyword `PURR` in the following way, which again makes use of the `WIT`.. `AN` .. pattern:

```
PURR <function_identifier> WIT <arg1> AN <arg2>.
```

The code sample below calls the `Chase` function defined in **Listing 9**. Note that we capture the returned string in a new variable of type `YARN`:

Listing 10: `chase_call.meow`

```
1 PSST This function call will return "Silvester chases Gary"
2 ITZ ME YARN message IZ
3     PURR Chase WIT "Silvester" AN "Gary".
```

3.8 Modules/Libraries

Meowlang source files end in `.meow`. By convention, source filenames should be camel-case and consist of lowercase letters only. Standard libraries and source code in other files can be imported into another source file with the keyword `GIMME` and will end with the character "?" as in the following pattern:

```
GIMME <module_name>?
```

Note that the `module_name` that should be used for the import is actually the name of the `.meow` source file, but transformed into uppercase and without the `.meow` suffix. This means that if you defined a source file `hello_world.meow` it becomes importable into another '`.meow`' file as `HELLO_WORLD`. The Meowlang compiler will always look for the `hello_world.meow` source file first in current directory, then the system `PATH`. Note that importing functions/variables of the same name causes a compiler error; you cannot duplicate identifiers.

3.8.1 Standard Library

A basic standard library, entitled `STDIO`, is implemented to provide basic support for printing (`MEOW`) and retrieving input from users (`SCAN`). This module is imported and used in the code sample below. Note that `SCAN` always read input in as a `YARN`, so the variable accepting the input must also be a `YARN`.

Listing 11: `hello_world.meow`

```
1 GIMME STDIO?
2
3 HAI ITZ ME FUNC MAIN,
4     MEOW "Hello World".    PSST Say hello to user
5
6     ITZ ME YARN message.
7     SCAN message.          PSST retrieve user's hello message
8 KBYE
```

3.9 Object Oriented Programming

Unlike the original LOLCODE, which does not have support for custom objects, Meowlang allows programmers to define their own classes with instance variables and methods. Objects are referred to by reference. Classes are composed of class variables and class functions. Inheritance and interfaces are not supported in this version of Meowlang.

You can define a new class using syntax similar to that used when defining a new function with the `HAI ITZ ME` set of keywords, specifying that you are creating a new class with the `CLASS` keyword. As previously mentioned, the Meowlang convention is for class name to be given in all capital letters (see section **3.1.2**); when an instance of a class is created, only the first character of its identifier is capitalized. Below is the template for class creation; the ellipses (`...`) specifies that there can be an indefinite number of instance variables and methods for a given class.

```
HAI ITZ ME CLASS class_name,
    ITZ ME type instance_variable1.
        ...
    HAI ITZ ME return_type FUNC method_name1 WIT type1 arg1 AN type2 arg2,
```

```
        <statements>
    KBYE
    ...
KBYE
```

Below, we create a new custom `MOUSE` class, which has a single instance variable `cookies` of type `NUMBR`. The class also has three methods: `squeek`, which simply prints out the sound that a mouse makes, `count_cookies` which returns the number of cookies that the instance has stored in the `cookies` instance variable, and `give_cookie`, which takes a `NUMBR` representing the number of cookies to increment `give_cookie` by.

Listing 12: `mouse_class.meow`

```
1  PSST Create a custom MOUSE class, class convention is in all-
       caps
2  HAI ITZ ME CLASS MOUSE,
3
4      PSST This is an instance variable
5      ITZ ME NUMBR cookies IZ 0.
6
7      PSST This is a class method
8      HAI ITZ ME FUNC squeek,
9          MEOW "Eeeeeeek!".
10     KBYE
11
12     HAI ITZ ME NUMBR FUNC count_cookies,
13         GIVE cookies.
14     KBYE
15
16     HAI ITZ ME FUNC give_cookie WIT NUMBR count_cookies,
17         PSST This uses the SUM prefix operator
18         cookies IZ SUM OF cookies AN count_cookies.
19     KBYE
20 KBYE
```

In order to create a new instance of a class, you must use the `MAEK` and `NEW` keywords, which allow memory to be allocated on the heap for the new object. Note that because Meowlang

does not support automatic garbage collection, the memory associated with the object must be freed using the `BLEEP` keyword once there is no use for it anymore. Here is the general pattern:

```
MAEK object_identifier NEW class_name.
BLEEP object_identifier.
```

Here we create a new instance of the `MOUSE` class and subsequently delete it. By convention, the first letter of an instance identifier is capitalized.

Listing 13: `ephemeral_mouse.meow`

```
1 MAEK Gus NEW MOUSE.   PSST Create a object using 'NEW' keyword
2 BLEEP Gus.            PSST Release memory associated with gus
```

Note that classes have implicit constructors that allow you to set the values of instance variables within a class instance. For example, our previously defined `MOUSE` class contains an instance variable `cookies`. An instance of a class is initialized with either specified default values or garbage values. To instantiate an instance with specific instance variable values, use the following pattern:

```
MAEK object_identifier NEW class_name,
    WIT instance_variable1 IZ value1
    AN instance_variable2 IZ value2
    ...
    AN instance_variableX IZ valueX.
```

3.9.1 Accessing Class Variables

Class variables may be accessed with the `IN` keyword. Values can be assigned with the `IZ` keyword.

```
    cookies IN Gus.
    cookies IN Gus IZ 100.
```

3.10 Writing a Complete Program

The Meowlang programming language expects the programmer to define a single "main" function that is the first function that gets executed when a program starts running. This

`Main` function is declared like any other function.

Currently, Meowlang requires that source files have a three part structure: first module imports, then functions, starting with `Main`, then custom classes. It is not necessary for a single source file to contain all three, they simply must be ordered in this fashion if they all are present. This structure is subject to change.

Below is a example Meowlang program made up of a single `Main` function. Although it is rather useless and unextraordinary, simply creating a variable and ending, this is sufficient for a full Meowlang program!

Listing 14: `example_main.meow`

```
1 HAI ITZ ME NUMBR FUNC Main,
2     ITZ ME YARN cat_name IZ "Silvester".
3     GIVE 0.
4 KBYE
```

4 Code Examples

The following code sample uses our examples above to create a single Meowlang program.

Listing 15: `mouse.meow`

```
GIMME STDIO?  PSST Import the Standard Library

HAI ITZ ME FUNC Main,

    PSST Declaring a bunch of variables...
    ITZ ME YARN cat_name IZ "Silvester". PSST string
    ITZ ME YARN mouse_name IZ "Gary".
    ITZ ME NUMBR num IZ 2.               PSST int
    ITZ ME NUMBAR value IZ 2.0.          PSST float
    ITZ ME BOO mouse_is_small IZ AYE.    PSST boolean
    ITZ ME BOO cat_is_friendly IZ NAY.

    PSST Create an array of 2 names
    MAEK names NEW YARN BUCKET OF 2 names,
        WIT cat_name
        AN mouse_name.

    PSST Prints "Silvester chases Gary"
    MEOW PURR Chase WIT names[0] AN names[1].

    PSST Create new MOUSE instance
    MAEK Gus NEW MOUSE,
        WIT cookies IZ 3.

    MAEK Jerry NEW MOUSE.    PSST no specification of cookies

    PSST Release memory associated with gus and jerry
    BLEEP Gus.
    BLEEP Jerry.
KBYE
```

```
31
32 PSST custom MOUSE class, class convention is in all-caps
33 HAI ITZ ME CLASS MOUSE,
34
35     ITZ ME NUMBR cookies IZ 0.    PSST instance variable
36
37     HAI ITZ ME FUNC squeek,       PSST class method
38         MEOW "Eeeeeeek!".
39     KBYE
40
41     HAI ITZ ME NUMBR FUNC count_cookies,
42         GIVE cookies.
43     KBYE
44
45     HAI ITZ ME FUNC give_cookie WIT NUMBR count_cookies,
46         PSST This uses the SUM prefix operator
47         cookies IZ SUM OF cookies AN count_cookies.
48     KBYE
49 KBYE
50
51 HAI ITZ ME YARN FUNC Chase WIT YARN bad_cat AN YARN poor_mouse,
52     GIVE bad_cat CAT " chases " CAT poor_mouse.
53 KBYE
```