The MT Programming Language

Ramsay Carslaw

February 4, 2021

Contents

Chapter 1

Introduction

1.1 What is MT?

The MT programming language is an open source, dynamically typed programming language intented to bridge the gap for new programmers between python and C. MT was implemented in C using a source to byte code compiler into a bytecode interpreter. This means MT is a resonably fast language.

1.2 Who is MT for?

MT is not intended for any serious develpoment projects, more a tool for learning. MT is a good halfway point between Python and your next language for new programmers. If you are looking to write an operating system MT is probably not the language for you, however if you are looking for a language to practice your data structures then MT will be a safe choice.

1.3 Who is this manual for?

This manual is for people with some programming experience that would like to learn MT. It's not too high level so even basic programming experince should be enough to learn MT. More advanced users can use this book as a refernce for syntax where as beginners can learn by completing the excercises at the end of each section.

1.4 Design Ideas

Although, MT supports object oriented programming this is by no means forced. It is perfectly fine to just use functions or code imperativley. Objects are in MT for the cases where it will make your life easier, they are not forced or nesseccerry.

1.5 Installing MT

Depending on your platform getting MT is slightly differnt. Once you have followed the relevant instructions you can run 'mt' in your terminal to test if everything has installed correctly.

1.5.1 macOS

MT was developed on a mac so this one is pretty easy

```
git clone git clone https://github.com/ramsaycarslaw/mt.git
cd mt
make
make install
```

This does require that the XCode command line tools are intsalled, to do this just download XCode from the Mac App store.

1.5.2 GNU/Linux

Depending on your GNU/Linux Distrobution you may need to configure the makefile. By default the makefile uses clang which is not always installed on linux systems. To install clang use:

```
sudo pacman -S clang
```

Alternativley, change the CC line in the Makefile to whatever C compiler you use. (Note: This may effect the performace of MT). After that follow the following steps.

```
git clone git clone https://github.com/ramsaycarslaw/mt.git
cd mt
make
make install
```

1.5.3 Windows

I don't have a windows machine to test any of this on but you will have to compile from source.

Chapter 2

The Basics

2.1 Saying Hello

2.1.1 Hello World

The very first program needed by any language is the classic hello world. Hello World in MT is very simple.

```
print ``hello, world!'';
```

There are a few interesting features of MT's hello world. First print is a keyword. Usually, print is a function and it's contents will be contained in brackets. In MT however print is a keyword which makes it more beginner friendly as it works the same as all the other keywords in MT and they can program without understanding the concept of a function. Aside from print being a keyword everything else is pretty standard. The double quotes denote a string and the semi-colon the end of a statement.

2.1.2 Hello MT

To run the program above save it as hello.mt in your favorite text editor. You can then navigate to the directory where you saved that file and run mt hello.mt to run the code. You should see 'hello, world!'. If MT warns you about the file not existing check you are in the right directory and that you saved the file. If you get an error like 'no such command mt' check the installation process as something has gone wrong.

2.1.3 Hello User

The next program we will cover is getting input from the users of your program. This is essential for any fully featured program so we will cover that next. This will also introduce a few other concepts like string concatenation.

```
var name = input(``What is your name? '');
print ``Hello, '' + name;
```

In this case input is actually a built in function as it needs the brackets. The message (or string) inside the brackets is prineted as a prompt for the user to type in. The plus sign in this case is used to join the two strings so if the user types in bob for the input then the result will be "Hello, bob". Notice the var keyword as well, this is used to tell MT you have a new variable.

2.2 Numbers Game

That covers the basic of getting text in and out of MT. Now it's time we actually compute something. You don't even need to create a file to do some of this. Open your terminal and type in 'mt'. You should see something that looks a bit like:

```
mt>
```

This is called the REPL, you can use it to evaluate MT statements without a file. For exanple you can type in:

```
mt> 10 + 12;
```

You will notice that nothing has happend. This is becuase we haven't told MT what to do with the result. To see the answer simply add a print beforehand.

```
mt> print 10 + 12;
22
```

This time it showed us the answer. This also works with the other basic operators

```
mt> print 10 * 12;
120
mt> print 10 - 12;
-2
mt> print 10 / 12;
0.833333
mt> print 10 ^ 12;
1e+12
```

There is no need to specify integers or floats in mt. This is becuase it is dynamically typed and can change during run time. You can also evaluate more complex statements invoving parenthesis.

```
mt> print (10 + 12 ^ (99 / 6 - (10 + 3.1415)));
4221.44
```

2.3 The Truth, The Whole Truth and Nothing but the Truth

Booleans are also, fully supported in mt, the syntax for which is familiar to anyone who has used a C-style langauge before.

```
mt> print 10 == 12;
false
mt> print 10 != 12;
true
mt> print !((10 != 12) && (10 == 12));
true
```

The basic operators are, equity `==` which checks that two values are the same. Non-equatity `!=` which verifys that two values are not the same. And (`&&`), Or (`||`) and Not (`!`).

2.4 My Type

There are several types in mt although many are abstracted for the sake of ease of use. For example there is no concept of integers and floating point numbers there is just the general `number` type. mt is dynamically typed which means the type of variables can change during runtime (think python). This removes type errors completly from the equation although you cannot disregard it enteirly if you are writing a library. All variables are created with the `var` keywword. Some examples are shown below.

```
var name = "The mt Programming Language"; // string
var age = 0.8; // number
var active = true; // boolean
var keywords = ["if", "else", "while", "for"]; // string array
```

After you have declared a variable you can update its value without the `var` keyword and change it to any other type.

Chapter 3

Advice on writing librarys

3.1 The use keyword

If you have chosen to write a library, first of all: thank you! Second there are some things you should consider. First of all the `use` keyword. The `use` keyword is for importing code and splitting projects across multiple files. It must be followed by a string with the *absoloute path* to the file you wish to include (note: this may change in the future as we introduce the concept of a standard library).

Currently, `use` should be considered unsafe, it is like C's `include` but without header guards. For this reason make sure you have no import loops in your code or your program will break.

3.1.1 Syntax

Below is an example of how to create a simple math library.

File: math.mt

```
// math.mt

fn square(x)
{
  return x*x;
}
```

File: main.mt

```
// main.mt

// main.mt is in the same folder as a directory called math
// math.mt is inside that folder
use "./math/math.mt";
```

```
print square(10);
```

3.2 Type Checking

If you are writing a library you may notice that you need to validate the types of some of your functions. Better type checking functions are being worked on but in the meantie the best advice is to document well and use the casting functions wherever possible.