



# Learn To Program

by Jason Chalom and  
Wits SoftDev Student Interest Group 2014  
Version 1.0



A decorative graphic on the left side of the slide, consisting of a network of white lines and circles on a dark blue background, resembling a circuit board or a neural network.

# License

## The MIT License (MIT)

Copyright (c) 2014 Jason Chalom, and The Mathematical Sciences  
Department of The University of the Witwatersrand (Wits Software Development Student Group).

Permission is hereby granted, free of charge, to any person obtaining a copy of  
this documentation and associated software, to deal in  
this documentation and associated software without restriction, including without limitation the rights to  
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of  
the Software, and to permit persons to whom the Software is furnished to do so,  
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all  
copies or substantial portions of this documentation and associated software.

THE DOCUMENTATION AND ASSOCIATED SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS  
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR  
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER  
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN  
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a neural network.

# Objectives of this Tutorial

- Learn basic coding principles for imperative languages
- Learn what an algorithm is
- Think algorithmically
- Do examples in MatLab
- Take a problem and be able to map it to a programmed solution
- Be able to use and understand documentation for further learning

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a neural network.

# Why would you want to learn to code?

- Programming is used in many industries and jobs, for instance in banking all the financial models are programmed. SAS is an example of a statistical language geared towards that kind of computation.
- Programming is more than just some mathematical technique. It is a way of algorithmically thinking about problems and their solutions. One important change in thought is breaking down a problem into its base components which you then solve to solve the whole problem. (Abstraction)
- Being able to transmute a problem from a definition, equation or implicit overview to a solid algorithm which is able to solve that problem gives you a unique and in-depth understanding of that problem.

# Before we begin

- This tutorial will go over programming basics using MatLab. MatLab is an imperative programming language which is a way of expressing a program via a set of statements which describe how the problem should be solved.
- MatLab is also an interpretive language which means it executes statements directly rather than first converting code into some more advanced code such as assembly or binary.
- Keep in mind that there are other types of programming languages which work very differently. The kind of language used depends of the problem and situation. In business the best, fastest and most beautiful solution would mostly be too time consuming and costly to implement, debug and use.
- MatLab is a programming language designed to be used to solve mathematical problems. It will not be the kind of language used to make the next text editor or some such program. MatLab is geared towards using matrices. That is its strong suit. Mathematica is more powerful in other areas as a mathematical programming language.

# Before we begin

- MatLab is quite cost-prohibitive. As a student you will most likely have access to discounts, free versions or local copies at the University.
- There is a free open-source alternative which is very similar to MatLab called Octave. It uses the same syntax and is very similar. However there are differences which mainly have to do with more advanced features of MatLab not covered in this tutorial.
- Something to lookup would be source control. If you are doing a group project or have many versions of source code for the same project source control can help keep things tidy and simple. GitHub is my personal favourite source control and variant of the git system. (See: <https://github.com/edu>, <http://nvie.com/posts/a-successful-git-branching-model/>)
- Also Remember **DON'T PANIC!**



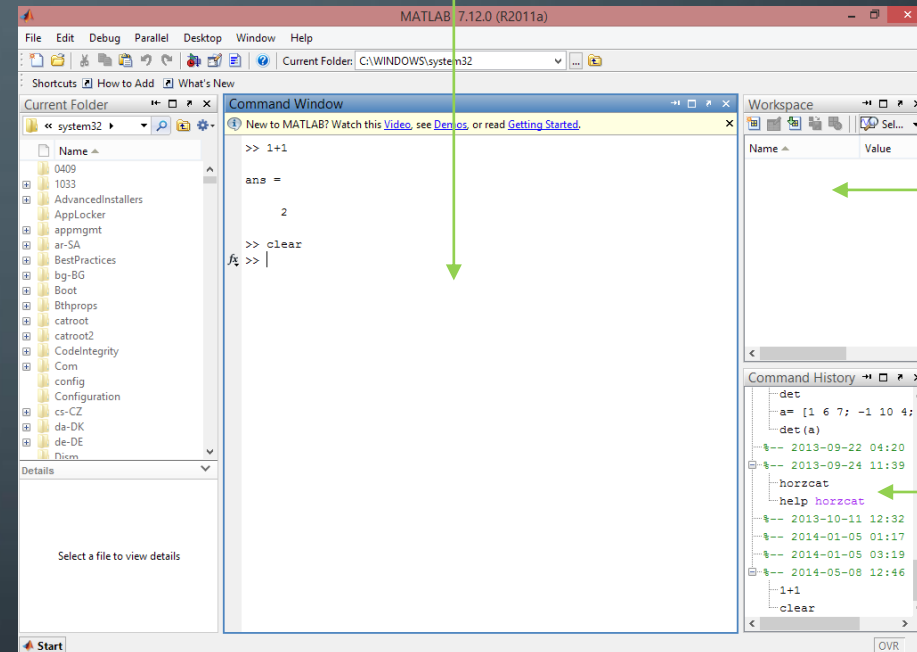
## Some conventions and standards

- I prefer to use self-documenting code which means that rather than writing pseudo-code or comments explaining what my code does, the code itself is written in an easily readable way. I do sometimes write a block of comments above my code for documentation, especially when I am doing something very mathematical and I'm reasoning my way through it.
- I prefer to use spaces in places where they don't affect the code but rather make the code look nicer and easier to read. I do the same for new lines (Enter).
- There are many conventions and ways of writing code which comes with practice. When you are not sure, first look online (sites like [www.stackoverflow.com](http://www.stackoverflow.com) and official documentation: <http://www.mathworks.com/help>) help to clarify problems and standards, if you still don't know then ask someone for help. With programming always try to do research first.

# Part I: Basic Interpretive Programming

- Ok lets begin:
- With an interpretive language like MatLab, you have two ways to execute commands.
  - By typing in a command and pressing enter at the command line where it will be executed right away
  - Or by writing a script file and then executing the code in a block.
- First we familiarise ourselves with the command line.
- Open MatLab

This is the command line or command window



This is the workspace box which shows you variables in memory and what variables are assigned to them

This is the history box which shows you past code which has been executed.



# Part I: Basic Interpretive Programming

- There are many operators in MatLab, Here is a short list of ones you would come across.
- The basic ones are (+ - / \*). Plus, minus, divide and times
- ^ denotes help
- = is an assignment operator used to assign one entity to another (This is not equals which is a Boolean operator)
- Boolean operators return true/false or 1/0
  - == is equals
  - ~= is not equals
- Logical Operators:
  - | is OR (UNION)
  - & is AND (INTERSECTION)
  - || conditional OR (for use with Booleans)
  - && conditional AND (for use with Booleans)
- Special characters:
  - : (colon) creates an auto generated list.
  - There are two or three required values
    - (starting number):(end number)
    - (starting number):(stepping size):(end number)
    - This can be used with matrices and also generates vectors
  - Dot (pointwise operators)
    - The .^ is a pointwise power (i.e. for each element). This is used in a matrix.
    - The .\* is a pointwise product
    - The ./ is a pointwise divide
    - The .+ is a pointwise addition

# Part I: Basic Interpretive Programming

- Try type in and then press enter

`1+1`

`2-1`

`5*5`

`1/2`

`'hello'`

- The last one is known as a string. The `'` tells MatLab that it is not a function or command but rather text to be outputted.
- Now type `clc` to clear the command window. `clear all` will clear the 'memory space' storing all variables and answers. The workspace window will be cleared. Using `clear 'name of variable'` (without quotes) will clear only that variable.



```
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

>> 1+1
ans =
    2

>> 2-1
ans =
    1

>> 5*5
ans =
   25

>> 1/2
ans =
  0.5000

fx >>
```

# Part I: Basic Interpretive Programming

- A variable is an entity which can be associated with a specific value. This can even be a matrix

- Try:

`A=33`

`B=10`

`A*B`

In MatLab calling a variable is quite easy. Just using alpha-numeric characters and an assignment will create a stored variable. You can also create specific variables i.e. an Integer but that is a bit harder and usually not needed.

- Now:

`clear all`

`clc`

`A=[1 2 3 4 5; 6 7 8 9 10]`

`B=2`

`B*A`

The `;` here denotes the end of one line and start of The next.

- If you put a `;` at the end of the statement it will suppress the message output.
- Press `ctrl-c` to terminate any executing code if it is stuck or infinitely executing.
- A function I will mention now is the help function.
  - `help 'any command, operator or topic'`
    - This will display help topics and documentation

```
>> A=33
```

```
A =
```

```
33
```

```
>> B=10
```

```
B =
```

```
10
```

```
>> A*B
```

```
ans =
```

```
330
```

```
>>
```

```
>> A=[1 2 3 4 5; 6 7 8 9 10]
```

```
A =
```

```
1 2 3 4 5
6 7 8 9 10
```

```
>> B = 2
```

```
B =
```

```
2
```

```
>> A*B
```

```
ans =
```

```
2 4 6 8 10
12 14 16 18 20
```


```
>> B*A
```

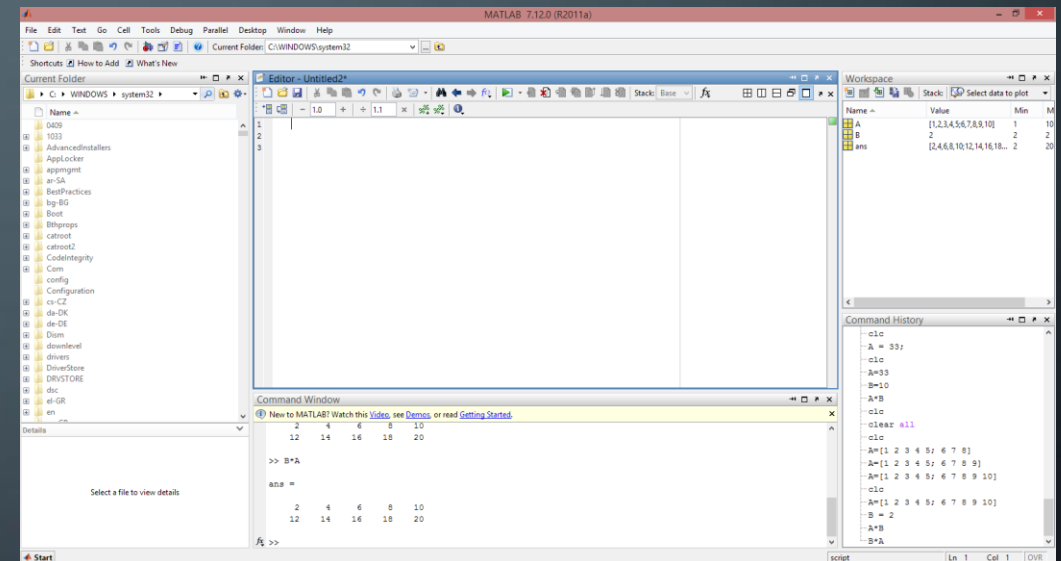
```
ans =
```

```
2 4 6 8 10
12 14 16 18 20
```

```
>> |
```

# Part I: Basic Interpretive Programming

- Now we will start with scripting
  - File -> New -> Script (Or the new icon)
  - Later on we will go over functions.
  - As you see a new window is opened in MatLab
  - A script is a file which stores a whole collection of related commands which are then executed together. (For this tutorial the code will execute sequentially MatLab does support more advanced code execution like multi-threading)
- You can save and open scripts
- To run a script click on 
  - Or press F5
- Comments are created by putting % in front of any text or numbers. They turn green.



# Part I: Basic Interpretive Programming

- Before an example with scripts can be done some more basics need to be given.
- The condition statement or if-statement is a statement which will execute one command if some condition is met or will execute something else if that condition is not met.

```
if (expression)
    statements
elseif (expression)
    statements
else
    statements
end
```

- An example would be:

```
A=2;
if (A == 1)
    fprintf('hello');
elseif (A == 10)
    fprintf('goodbye');
else
    A = 1;
end
```

- The end statement is to tell MatLab to end a block of code. We will use this later for functions.
- This example shows the difference between = and ==
- fprintf is a function to print text onto the screen (command window), this will be covered later.

# Part I: Basic Interpretive Programming

- A SWITCH-CASE statement is similar to the if-statement.
- Conditions here are used to execute one set of statements from several different cases pertaining to one specific variable.
- Computationally the switch-case statement is slower than the if-statement due to how the computer handles the behind-the-scenes code. It is more useful to use to create more readable and user-friendly code

```
switch (variable to check)
    case (a condition which the variable equals)
        The code to execute
    Case (another condition which the variable equals)
        More code
    otherwise
        Code for the default case
end
```

- An example would be:

```
variable = 'hello';
switch (variable)
    case 'goodbye'
        fprintf('cheers\n');
    case 'hello'
        fprintf('hello world\n');
    otherwise
        fprintf('why you say no hello?\n');
end
```

- The end statement is to tell MatLab to end a block of code. We will use this later for functions.
- Here the condition is implied and can only be a equal truth condition.
- The otherwise code-block is a default code-block you can choose to add in so that if none of the statements are true this one will execute by default.



# Part I: Basic Interpretive Programming

- Loops:

- There are two basic kinds of loops, the for-loop and the while-loop
- The while-loop is a loop which has a terminating condition. When this condition is met it will terminate.
- Usually the terminating condition uses a variable which has been defined before the loop code-block. In the example it's the count variable.
- The loop will continue to execute until the condition stops being true
- In Boolean logic a condition where the statement hold is seen as true.
- When using the not operator ~ this makes any statement the opposite so here the loop will end when the statement is the opposite i.e. true.

```
while (condition)  
    Code to repeat  
end
```

- An Example would be:

```
count = 0;  
while (count ~= 5)  
    count = count + 1  
end
```

# Part I: Basic Interpretive Programming

- The for-loop is a loop which goes from a defined start point to a defined end point. There is also a counter which can be set to increase or decrease.
- The interesting thing about it is that the end condition does not have to be connected to the counter meaning it can be used similar to a while loop although this is not advised.

```
for index variable = start index : end  
index
```

Code to repeat

```
end
```

- An example would be:

```
A = zeros(5,100);
```

```
for m = 1:5
```

```
    for n = 1:100
```

```
        A(m, n) = 1/(m + n - 1);
```

```
    end
```

```
end
```

A decorative graphic on the left side of the slide, consisting of a network of white lines and small circles on a dark blue background, resembling a circuit board or a neural network.

# Part I: Basic Interpretive Programming

- Included with this tutorial are some example scripts. The one with examples from before is called Script1.m It can be opened and executed in MatLab.

# Part I: Basic Interpretive Programming

- Functions are an important part of any imperative programming language.
- Just like in mathematics you can get a function with one or even many variables which are 'sent into' the function and a result of some kind is returned the same is true for programming.

$$f(x) = x^2$$

- Programming is a mathematical discipline and follows similar rules to some respect.
- A function in MatLab is a type of script. There are others but this tutorial will not go over them.

- Functions are used for abstraction purposes.
- A function reduces the amount of code needed by taking repeated code and putting it in one place.
- It also allows a programmer to break up a difficult problem into smaller bits which makes that problem easier to understand.
- There are many functions in MatLab which can help make things easier.
- There are two ways of declaring functions in MatLab.
  - Where there is one output
  - Where there is multiple outputs

# Part I: Basic Interpretive Programming

- Declaring Functions with one output:

```
function output variable = function name(input variable/s)
```

```
Any kind of code
```

```
output variable = some value or computation;
```

```
Any kind of code (Although not outputted from FN)
```

```
end
```

- In MatLab the function name has to be the same name of the m file it is stored in.
- Usually the function is stored in its own file but you can have a script and a function together. (see later)
- A script or another function in a different file or even the command window can access and use a function within those computations.
- It just needs to be open in MatLab, or be in a recognised path. (Dialogue when you try to run it with F5 or the run button)

- An Example:

```
function y = average(x, z)
```

```
if isvector(x)
```

```
error('Input must not be a vector')
```

```
end
```

```
y = (x+z)/2;
```

```
end
```

- The `isvector()` is a built in MatLab function which will return a 0 (true) if the variable sent into it is a vector or 1 (false) if it is not.

# Part I: Basic Interpretive Programming

- Declaring Functions with multiple outputs:

```
function [output variable 1, output variable 2] = Name of FN(x)
```

```
    Any kind of code
```

```
    output variable1 = some value or computation;
```

```
    output variable2 = some value or computation;
```

```
    Any kind of code (Although not outputted from FN)
```

```
end
```

- Any number of output or input variables can be used.

- An Example:

```
function [a,m] = AddMultiply(x)
```

```
    %Show difference between
```

```
    %addition and multiplication
```

```
    a = x+x;
```

```
    m = x*x;
```

```
end
```



# Part I: Basic Interpretive Programming

- Declaring a function and a script in the same file.
- This is more to be informative than to be used.
- An Example:
- Functions can also have sub-functions
- A sub-function works under the main function and so does not follow the filename -> function name rule.
- Right now they are not really needed

**filename: myScript.m**

```
function [] = myScript()  
    y = 1:10;  
    out1 = myfunc(y);  
end
```

```
function [out1] = myfunc( x )  
    out1 = sqrt( 1 + (cos(x))^2 );  
end
```

# Part I: Basic Interpretive Programming

- List of useful built in MatLab functions:
- `fprintf()` – print the string which is sent into the function to the command window.
- `help “` – put anything in the quotes at the internal documentation will be called up.
- `sum(x)` – sum of the number of elements x. If two variables (x, dim) and x is a matrix then it will sum along the dimension dim. There are other configurations (see help)
- `isvector(x)` – checks if x is a vector and returns true/false or 0/1
- `sin(x), cos(x), tan(x)` – trigonometry functions
- `log(x), exp(x)` – logarithmic and exponential
- `sqrt(x)` – square root
- `abs(x)` – absolute of x

## Part II: A Problem Example, Mapping Mathematical Equations to Programs

- The main use of programming is to map a mathematical problem in equation form into a programmatically solved solution in numerical form.
- There are ways of getting analytical results using cellular automata but this tutorial does not cover that. Mathematica is better suited than MatLab for that kind of computation
- An equation can be quite easily mapped into a program
- The variables and inputs required need to be setup first.
- If Sigma is used then a loop is required.
- If there is a termination factor use a while loop
- If there are conditions use the if-statement
- Keep track of variables
- Output the results



## Part III: Iteration VS Recursion

- Iteration in computing is the repetition of a block of statements within a computer program
- Recursion in computer science is a method where the solution to a problem depends on solutions to smaller instances of the same problem as opposed to iteration.
- An important concept from computer science is that any algorithm which uses an iterative method could also use a recursive method to solve the same problem and visa versa.
- Recursion is usually slower than iteration.

# Part III: Iteration VS Recursion

- An exponential example of iteration:

*%x is the number, y is the exponent*

*%i is the counter*

```
x = 5;
y = 2;
if (y=1)
    ans = x
elseif (y ~= 0)
    for i = 1:y
        x*x;
    end
    ans = x
else
    ans = 1
end
```

- An exponential example of recursion:

*%x is the number, y is the exponent*

filename: CalcExp.m

```
function out = CalcExp(in, exp)
    if (y=1)
        out = in;
    elseif (y ~= 0)
        out = CalcExp(in*in, exp-1);
    else
        out = 1;
    end
end
```

filename: ExpRecursionScript.m

```
x = 5;
y = 2;
ans = CalcExp(x,y)
```

# Fibonacci examples using different methods

- Here we will look at three different ways to compute Fibonacci number sequences.
  - Iterative
  - Recursive
  - Matrix method
- The initial seed values are  $F_0 = 0$ , and  $F_1 = 1$

- Iterative:
- This is the method most people know.

- 1, 1, 2, 3, 5, ...
- $F_n = F_{(n-1)} + F_{(n-2)}$

Filename: **FibonacciIteration.m**

```
m = 0;
n = 1;
count = 5;
for i = 1:count
    ans = n+m;
    m = n;
    n = ans;
end
fprintf(ans);
```



# Fibonacci examples using different methods

- Recursive:

filename: *fib\_recur.m*

```
function y = fib_recur(n)
    if n == 0
        y = 0;
    else
        y = [fib_recur(n-1) fib(n)]
    end
```

filename: *FibonacciRecursive.m*

```
count = 5;
ans = fib_recur(count);
```

- Matrix:

- This method is a sort of hack. It uses a special Q-Matrix to help compute the sequence. (See references and solve)

- This Q matrix is defined as:

$$Q = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Q^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

$$G_m = \begin{pmatrix} m & 1 \\ 1 & 0 \end{pmatrix}$$

<http://www.cut-the-knot.org/arithmetic/algebra/FibonacciMatrix.shtml>

[http://www.emis.de/journals/AUA/pdf/16\\_220\\_paper-22-20-2009.pdf](http://www.emis.de/journals/AUA/pdf/16_220_paper-22-20-2009.pdf)