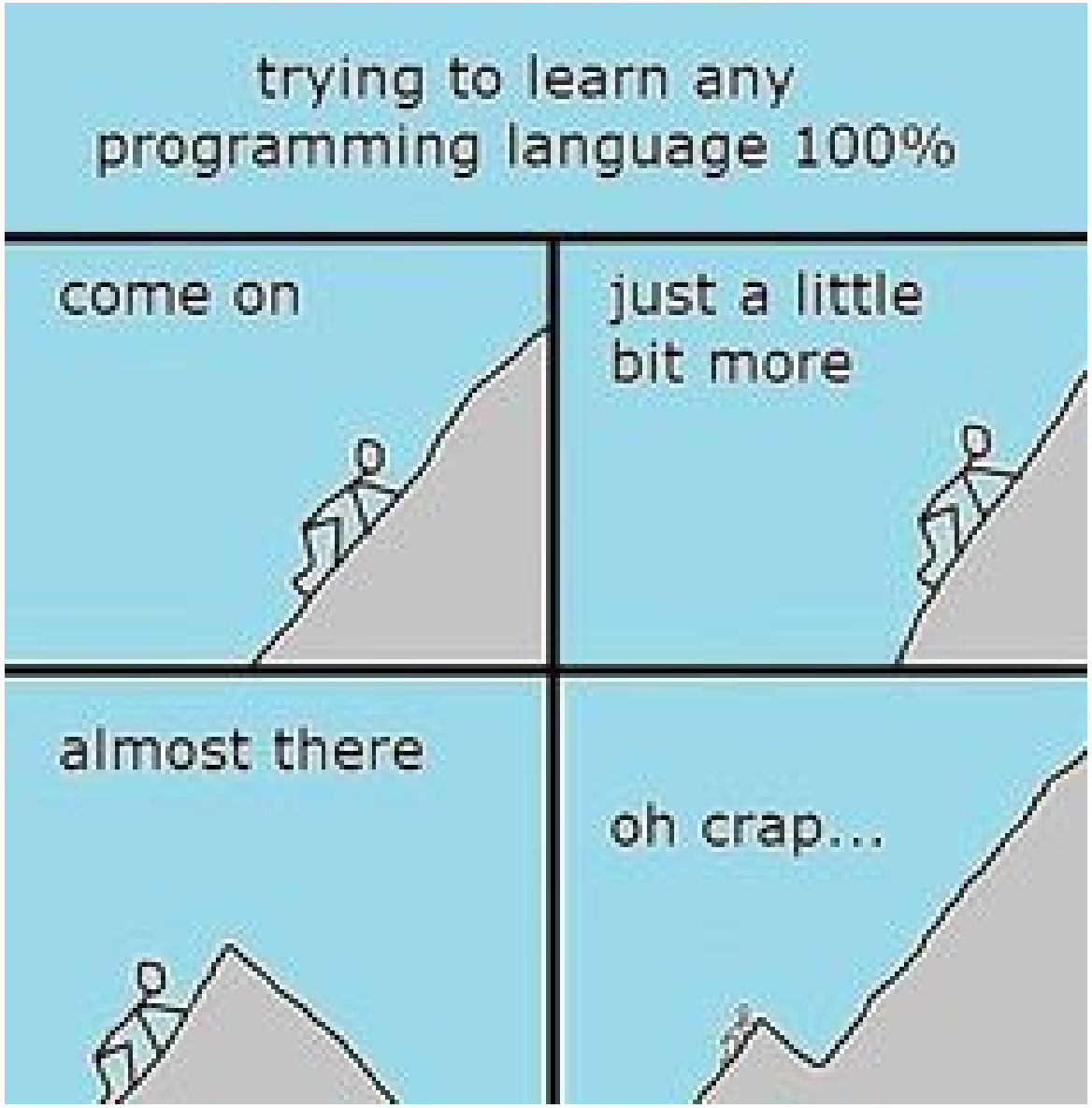
Overview of MATLAB


trying to learn any programming language 100%
come on
just a little bit more
almost there
oh crap...

BASICS

```
% SCALARS ... (RED is MATLAB input, BLUE is MATLAB output)

>> 410

ans = 410

% Assign a value to a variable --- use =

>> a = 410

a = 410

>> a

a = 410

% pi is predefined but NOT protected

>> pi

ans = 3.1416

% Change output format to show full precision (about 16 digits)

>> format long
```

```
>> pi

ans = 3.14159265358979

>> sin_b = sin(pi/3)

sin_b = 0.866025403784439

>> cos_b = cos(pi/3)

cos_b =  0.500000000000000

>> sum_b = sin_b^2 + cos_b^2

sum_b = 1

>> exp(1)

ans =  2.71828182845905

>> log(exp(1))

ans =  1

>> log10(10^2)

ans =  2
```

```
% ROW VECTORS ...

% Use [ ] to create vectors in-line, separating elements with white
space or commas

>> vec1 = [2, 3, 5, 7]

vec1 =

   2   3   5   7

>> vec2 = [11 13 17 19]

vec2 =

   11   13   17   19

% Index using regular parenthesis (indexing is from 1, not 0)

>> vec1(3)

ans =  5
```

```
% COLON OPERATOR ...

>> vec1 = 1 : 10

vec1 =

    1    2    3    4    5    6    7    8    9   10

>> vec1(8)

ans =  8

>> vec1a = 1 : 1 : 10

vec1a =

    1    2    3    4    5    6    7    8    9   10

>> vec1 - vec1a

ans =

   0   0   0   0   0   0   0   0   0   0
```

```
>> vec2 = 10:-1:1

vec2 =

   10    9    8    7    6    5    4    3    2    1

>> vec1 + vec2

ans =

   11   11   11   11   11   11   11   11   11   11

% Colon operates with floats as well

>> format short

>> vec3 = 0.0:0.25:1.5

vec3 =

    0.00000    0.25000    0.50000    0.75000    1.00000    1.25000
    1.50000
```

```
% MATRICES ...

% Define row by row, separating rows with semi-colons

>> mat1 = [ [1, 0, 0]; [0, 1, 0]; [0, 0, 1] ]

mat1 =

    1    0    0
    0    1    0
    0    0    1

% Index with ( ), supplying 2 indices

>> mat1(1,1)

ans =  1

>> mat1(2,1) = -1

mat1 =

    1    0    0
   -1    1    0
    0    0    1
```

```
% Rows defined using colon operator

>> mat2 = [ [1:4]; [5:8]; [9:12]; [13:16] ]

mat2 =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

% Transpose operator, '

>> mat2'

ans =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

```
% 4 x 4 matrix of random numbers between 0 and 1

>> mat3 = rand(4)

mat3 =

    0.800587   0.739155   0.707768   0.129989
    0.980077   0.594170   0.925259   0.365658
    0.895373   0.525012   0.079183   0.997303
    0.244606   0.644317   0.318224   0.066753


% Length 4 row vector of random numbers

>> rr4 = rand(1, 4)

rr4 =

    0.8147    0.9058    0.1270    0.9134
```

```
% Matrix inverse computed in two ways

>> inv(mat3)

ans =

    4.77502   -2.44260    0.51729   -3.64688
    0.43686   -0.96806    0.17759    1.79887
   -3.66848    3.32607   -0.86490    1.84601
   -4.22570    2.43849    0.51347    2.18060

>> mat3^(-1)

ans =

    4.77502   -2.44260    0.51729   -3.64688
    0.43686   -0.96806    0.17759    1.79887
   -3.66848    3.32607   -0.86490    1.84601
   -4.22570    2.43849    0.51347    2.18060
```

```
% When operating on matrices, * is matrix multiplication

>> mat3 * mat3

ans =

   2.03088   1.48628   1.34795   1.08888
   2.28486   1.79884   1.43305   1.29183
   1.54622   1.65792   1.44312   0.45391
   1.12857   0.77372   0.81572   0.58922

>> mat3 * inv(mat3)

ans =

   1.0000e+00   3.3307e-16   2.6368e-16  -1.6653e-16
  -1.5543e-15   1.0000e+00   1.1102e-16   7.7716e-16
  -1.7764e-15   8.8818e-16   1.0000e+00   8.8818e-16
  -2.2204e-16   1.6653e-16   6.2450e-17   1.0000e+00
```

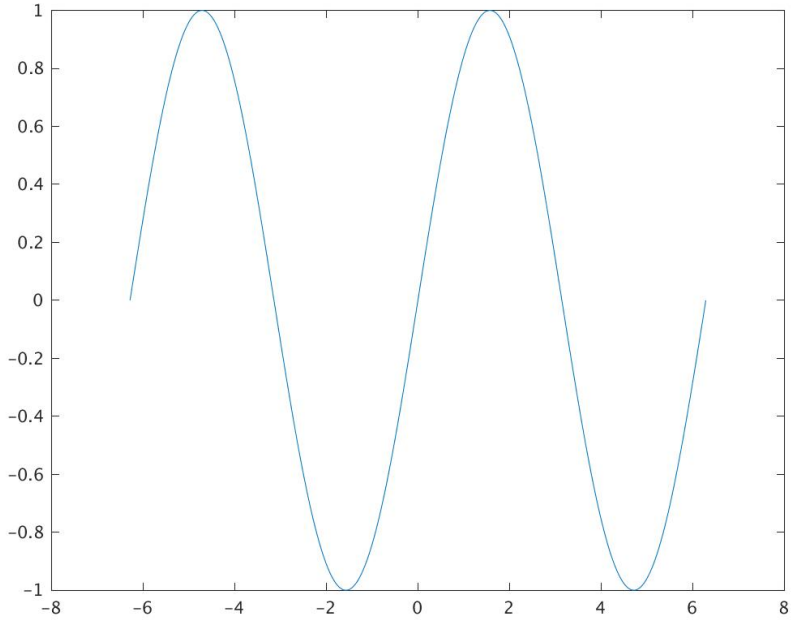
```
% eye: built in command for generating the identity matrix

>> eye(4)

ans =

Diagonal Matrix

   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1

>> mat3 * inv(mat3) - eye(4)

ans =

  -8.8818e-16   3.3307e-16   2.6368e-16  -1.6653e-16
  -1.5543e-15   8.8818e-16   1.1102e-16   7.7716e-16
  -1.7764e-15   8.8818e-16   0.0000e+00   8.8818e-16
  -2.2204e-16   1.6653e-16   6.2450e-17   0.0000e+00
```

```
%-------------------------------------------------------
% MATRIX:              m rows by n columns (m x n)
% ROW VECTOR:          1 row  by n columns (1 x n)
% COLUMN VECTOR:       m rows by 1 column  (m x 1)
%-------------------------------------------------------

% COLUMN VECTORS ...

% Define like a Matrix, each row is one value so don't need
% inner [ ] .

>> vc1 = [2; 3; 5; 7]

vc1 =

    2
    3
    5
    7
```

```
% Can also transpose a row vector

>> vc2 = [2, 3, 5, 7]'

vc2 =

    2
    3
    5
    7
```

```
% linspace command: another way to generate a row vector with uniformly
% spaced elements
%
% Syntax: linspace( <first element>, <last element>, <# of elements> )

>> vec4 = linspace(0.0, 1.0, 21)

vec4 =

 Columns 1 through 8:

   0.00000   0.05000   0.10000   0.15000   0.20000   0.25000   0.30000
   0.35000

 Columns 9 through 16:

   0.40000   0.45000   0.50000   0.55000   0.60000   0.65000   0.70000
   0.75000

 Columns 17 through 21:

   0.80000   0.85000   0.90000   0.95000   1.00000
```

```
% PLOTTING ...

% Plot a vector of sin(x)'s vs x with 1001 uniformly spaced values
% of x from -2*Pi to 2*Pi

% ; suppresses output

>> vx = linspace(-2*pi, 2*pi, 1001);

>> vsinx = sin(vx);

>> plot(vx, vsinx)

% Save the plot as a JPEG image

>> print('sin.jpg', '-djpeg')
```




Get credit for a taste of the GRAD SCHOOL research experience with open-ended exploration of beautiful and exotic natural phenomena in PHYS 409
*Prerequisite: One of PHYS 309, PHYS 319
REGISTRATION STILL OPEN
PHYS 409A = Term 1
PHYS 409B = Term 2
PHYS 409C = Terms 1&2
UBC DEPARTMENT OF
PHYSICS ASTRONOMY

PROGRAMMING BASICS

Relational and logical operators

MATLAB follows the C-language approach of

1. Returning the integers

   0 for false

   1 for true

   when evaluating relational or logical expressions

2. Treating the value 0 (integer OR floating point) as false, and any non-zero value as true in contexts where relational/logical expressions are expected (e.g. if statements)

   That is, although 0 is the unique “false value”, and although comparisons and logical operations will always return 0 or 1, there is no unique “true value”

Recent versions of MATLAB also incorporate a true logical type, including logical constants true and false

RELATIONAL OPERATORS

Operator	Definition
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
==	Equal
~=	Not equal

LOGICAL OPERATORS

&	Logical AND
|	Logical OR
~	Logical NOT
&&	Logical AND (short circuit: evaluation of logical expressions stops as soon as overall truth value is known)
||	Logical OR (short circuit)

Control Structures

SELECTION/CONDITIONAL: The if-elseif-else-end statement

DEFINITIONS

```
<Bexpr> = Boolean expression (should be a scalar), also known as a
          conditional expression
<ss>    = Statement sequence
```

GENERAL FORM

```
if <Bexpr1>
    <ss>
elseif <Bexpr2>
    <ss>
elseif <Bexpr3>
    <ss>
     .
     .
     .
else
    <ss>
end
```

EXAMPLES

```
>> a = 2; b = 3;

>> if a == b
      a + b
   else
      a - b
   end

ans = -1


>> aa = 2;

>> if aa == 1
      10
   elseif aa == 2
      20
   else
      30
   end

ans = 20
```

ITERATION (REPETITION, LOOPS)

THE for STATEMENT

DEFINITIONS

<lvar> = loop variable

<vector-expression> = expression that defines ROW vector

<ss> = statement sequence

GENERAL FORM

```
for <lvar> = <vector-expression>
  <ss>
end
```

For loop: TYPE 1

<vector-expression> generated using colon notation:

<first> = first value of <lvar>

<step> = lvar increment (step)

<last> = last value of <lvar>

```
for <lvar> = <first> : <step> : <last>
    <ss>
end
```

or, if the loop variable increment is 1

```
for <lvar> = <first> : <last>
    <ss>
end
```

As the loop executes, <lvar> takes on the values

<first>, <first> + <step>, <first> + 2 * <step>, ...
and where the last value of <lvar> is always <= <last>

<first>, <step>, <last> don't have to be integers, but usually will want them to be to avoid possible problems with roundoff errors

EXAMPLES

```
>> for k = 1 : 3
      k
   end

   k =  1
   k =  2
   k =  3

>> for k = 4 : -1 : 2
      k
   end

   k =  4
   k =  3
   k =  2

>> for k = 1 : 4 : 14
      k
   end

   k =   1
   k =   5
   k =   9
   k =  13
```

For loop: TYPE 2

<vector-expression> created using any other command/expression that defines/returns a row vector

```
for <lvar> = <vector-expression>
    <ss>
end
```

EXAMPLES

```
>> for k = [1 7 13 sqrt(2)]
      k
   end

   k =  1
   k =  7
   k =  13
   k =  1.4142
```

For loop: TYPE 2

<vector-expression> created using any other command/expression that defines/returns a row vector

```
for <lvar> = <vector-expression>
    <ss>
end
```

EXAMPLES (continued)

```
>> for a = linspace(2.0, 3.0, 6)
      a
   end

   a = 2
   a = 2.2000
   a = 2.4000
   a = 2.6000
   a = 2.8000
   a = 3
```

THE while STATEMENT

DEFINITIONS

```
<Bexpr>    = Boolean / conditional expression
<ss>       = statement sequence
```

GENERAL FORM

```
while <Bexpr>
    <ss>
end
```

NOTE

The while loop executes until <Bexpr> evaluates to 0 (false). If <Bexpr> is 0 upon initial entry to the loop, the body of the loop does NOT execute.

In other words, while <Bexpr> is true the looping continues.

It is up to the programmer (i.e. you) to do something within the loop so that, eventually, <Bexpr> evaluates to 0 (false), or your program will be stuck in the proverbial "infinite loop"

EXAMPLE

```
>> q = 1
   while q <= 16
      q = 2 * q
   end

   q =  2
   q =  4
   q =  8
   q =  16
   q =  32
```

THE break STATEMENT

break causes an immediate exit from the (innermost) loop where it is executed

EXAMPLE (contrived!)

```
>> q = 1
   while q <= 16
      q = 2 * q
      if q > 3
         break
      end
   end

   q =  2
   q =  4
```

If a break statement is encountered outside of any loop in a script or function, it terminates the execution of the file.

THE continue STATEMENT

continue is used within a loop to "short circuit" the execution of the loop body, and proceed to the next iteration

EXAMPLE

The rem command returns the remainder of division of the first operand by the second

```
>> for ii = 1:5
      jj = ii
      if rem(ii,2) == 0
         continue
      end
      jj = -ii
   end

   jj =  1
   jj = -1
   jj =  2
   jj =  3
   jj = -3
   jj =  4
   jj =  5
   jj = -5
```

THE return STATEMENT

The return statement causes an immediate return of a script or function to the invoking environment

EXAMPLE

```
function rval = errorreturn(a, b)
% errorreturn returns the sum of its two input arguments, providing
% that the first is strictly positive, otherwise it prints an error
% message and exits using the return statement.

  % Assign a "default" value to the output argument to ensure that it
  % IS defined before the function returns.
  rval = NaN;

  if a <= 0
     % Print the error message and return
     fprintf('errorreturn: First argument must be > 0');
     return;
  end

  % Return the normal value
  rval = a + b;
end
```

INVOCATIONS

With valid arguments ...

```
>> errorreturn(2.0, 3.0)

ans = 5
```

With invalid arguments ...

```
>> errorreturn(-2.0, 3.0)
errorreturn: First argument must be > 0

ans = NaN
```


Are you ill?
No, just feeling a bit off.
11111111
11111110
comic.browserling.com

Programming Units

Scripts and Functions

MATLAB code is prepared in source files with a .m extension as in

```
myscript.m
myfunction.m
```

MATLAB maintains a notion of the path for executable code, which is an ordered list of folders (directories) in which MATLAB will search for a source file with a name corresponding to a script or function which has been invoked.

The contents of the current path can be viewed using the 'path' command.

First note that a script is simply a source file containing an arbitrary sequence of MATLAB commands/statements. Assuming that a script file with the name 'myscript.m' has been created in some folder in the path, then the script can be executed as follows

```
>> myscript
```

(Note that there is no '.m' in the invocation.)

Similarly, assuming that myfunction is a function of two arguments, and has been coded in a file 'myfunction.m' located in a folder somewhere in the current path, then it may be invoked using

```
>> myfunction(1, 2)
```

DISPLAYING script/function DEFINITIONS: the type command

SYNTAX

```
type <script-or-function-name>
```

EXAMPLE

```
>> type sin

'sin' is a built-in function.
```

Function definition

FUNCTION DEFINITION: General forms

A MATLAB function can have an arbitrary number 0, 1, 2, 3, ... of input arguments, or formal parameters, each of which can be a scalar, vector, matrix, higher dimensional array, ...

A MATLAB function can have an arbitrary number 0, 1, 2 ... of output arguments, or return values, and each output can be a scalar, vector, scalar, vector, matrix, higher dimensional array, ...

DEFINITIONS

<ss> denotes arbitrary sequence of MATLAB statements/commands.

<inarg> = "input argument" (formal argument)

<outarg> = "output argument" (may sometimes call "return value")

Function definition: 0 output arguments

```
function <name>(<inarg1>, <inarg2>, ..., <inargm>)
    <ss>
end
```

Function definition: 1 output argument

```
function <outarg> = <name>(<inarg1>, <inarg2>, ..., <inargm>)
    <ss>
end
```

Function definition: 2 output arguments

```
function [ <outarg1>, <outarg2> ] = <name>(<inarg1>, <inarg2>, ...,
           <inargm>)
    <ss>
end
```

Function definition: n output arguments

```
function [ <outarg1>, <outarg2>, ... <outargn> ] =  <name>(<inarg1>,
           <inarg2>, ..., <inargm>)
    <ss>
end
```

DEFINING VALUES FOR OUTPUT ARGUMENTS

All outargs must be assigned a value before the function returns (i.e. reverts control to the invoking environment). This can happen in one of two ways

1. The end of the function is encountered (implicit return).
2. A return statement is executed (explicit return).

If there are multiple outargs, then to "capture" all of them upon return from the function, the function invocation must be on the RHS of an assignment statement with a row vector of names on the LHS, e.g.

```
>> [out1, out2] = fcn2(in1, in2)
```

otherwise only the first outarg is returned to the invoking environment.

FUNCTION DEFINITION: EXAMPLES

```
function res = myadd(sc1, sc2)
   res = sc1 + sc2;
end

>> myadd(2,3)

ans = 5


function [res1, res2] = myaddsub(sc1, sc2)
   res1 = sc1 + sc2;
   res2 = sc1 - sc2;
end

>> [val1, val2] = myaddsub(2,3)

val1 = 5

val2 = -1
```