* High level language
* Programming language and development environment
* Built-in development tools
* Numerical manipulation
* Plotting of functions and data
* Implement algorithms
* Create models and applications
* Many built in functions
* Interface with other languages
* Create graphical interfaces
The MATLAB Environment
Common Arithmetic Operators

+   Addition
-   Subtraction
*   Multiplication
/   Division
^   Exponential
()  Order operations
Semicolons in MATLAB

Suppress the output from a MATLAB expression
Comments %

Suppress the output from a MATLAB expression

```matlab
>> cos(2*pi) %cosine operator

ans =

    1

>> 3^2; %you can write anything here
```
Variables

Variable name = variable value

Be careful of i,j
MATLAB is CASE SENSITIVE

```matlab
>> x
x = 9

>> X
Undefined function or variable 'X'.

Did you mean:
>> x
x = 9
```
Initially Workspace is empty
Load your workspace
Variables to result of expression

If an expression is not stored as a variable it will be stores as ‘ans’
Multiple assignments and ‘who’

Command Window

```
>> a=4;b=2;c=a*b

c =

>> who

Your variables are:

a  ans  b  c  x  y  z
```
whos

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>ans</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

>> clear a
>> whos

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ans</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>1x1</td>
<td>8</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

>> clear
>> whos
format

```matlab
>> x=pi
x =
  3.1416

>> format long
>> x=pi
x =
  3.141592653589793

>> format short
>> x
x =
  3.1416

>> format rat
>> pi
ans =
  355/113

>> format bank
>> pi
ans =
  3.14
```

‘format short’
to get back to normal
Numbers are actually 1x1 Matrices

```matlab
>> format short
>> x
x =
    3.1416
>> size(x)
ans =
    1   1

>> rowVector = [1,2,3,4,5]
rowVector =
    1   2   3   4   5
>> rowVector = [1:5]
rowVector =
    1   2   3   4   5
>> size(rowVector)
ans =
    1   1
>> size(rowVector)
ans =
    1   5
```
Operations apply to matrices

```plaintext
>> rowVector
rowVector =
    1  2  3  4  5

>> secondVector = [4,3,2,1,0]
secondVector =
    4  3  2  1  0

>> rowVector + secondVector
ans =
    5  5  5  5  5  5
```
Operations apply to matrices

`>> rowVector + secondVector`

`ans =`

```
  5   5   5   5   5   5
```

`>> columnVector = [1;2;3;4;5]`

`columnVector =`

```
1
2
3
4
5
```

`>> columnVector + rowVector`

`Error using +`

`Matrix dimensions must agree.`
Put . before an operator to make it element-wise.

```plaintext
Command Window
>> myMatrix = [1,2,3;4,5,6]
myMatrix =
    1     2     3
    4     5     6

>> mySecondMatrix=[6,5,4;3,2,1]
mySecondMatrix =
    6     5     4
    3     2     1

>> myMatrix + mySecondMatrix
ans =
    7     7     7
    7     7     7

>> myMatrix * mySecondMatrix
Error using *
Inner matrix dimensions must agree.

>> myMatrix .* mySecondMatrix
ans =
    6    10    12
   12    10     6
```

disp(variable) displays the contents of a variable.
fprintf – print something to command window

```matlab
>> numberOfDays = 3;
>> nameOfInstructor = 'Simon';
>> fprintf('We are learning MATLAB with %s over %d days',nameOfInstructor, numberOfDays)
We are learning MATLAB with Simon over 3 days
>> fprintf('We are learning MATLAB with %s over %d days\n',nameOfInstructor, numberOfDays)
We are learning MATLAB with Simon over 3 days
```

%s  Format as a string.
%d  Format as an integer.
%f  Format as a floating point value.
%e  Format as a floating point value in scientific notation.
%g  Format in the most compact form: %f or %e.
\n  Insert a new line in the output string.
\t  Insert a tab in the output string.
Concatenating matrices

>> myMatrix = [0,1,2,3];
>> myOtherMatrix=[4,5,6,7];
>> cat(1,myMatrix,myOtherMatrix)

ans =

    0     1     2     3
    4     5     6     7

>> cat(2,myMatrix,myOtherMatrix)

ans =

    0     1     2     3     4     5     6     7

>> [myMatrix,myOtherMatrix]

ans =

    0     1     2     3     4     5     6     7

>> [myMatrix;myOtherMatrix]

ans =

    0     1     2     3
    4     5     6     7
Concatenating matrices

```matlab
>> [myMatrix; myOtherMatrix]
ans =
     0     1     2     3
     4     5     6     7

>> length(ans)
ans =
     4

>> size(ans)
ans =
    1     1

>> size([[myMatrix; myOtherMatrix]])
ans =
    2     4
```

Why is this 1,1?
Many Functions work on columns

```matlab
myBigMatrix =
    0  1  2  3
    4  5  6  7
>> max(myBigMatrix)
ans =
    7  7  7  7
>> min(myBigMatrix)
ans =
    0  0  0  0
>> sum(myBigMatrix)
ans =
    4  10  16  22
```
3 Ways of running code
- Command Window
- Scripts
- Functions

m-files
New Script

>> edit
>> edit myNewFile.m

Current Folder

<table>
<thead>
<tr>
<th>Name</th>
<th>Date Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>myNewFile.m</td>
<td>23/04/2016 18:42</td>
</tr>
<tr>
<td>myFile.mat</td>
<td>23/04/2016 16:57</td>
</tr>
</tbody>
</table>
Our First Script

```matlab
genesExp1 = 260;
genesExp2 = 58;
genesExp3 = 79;
totalGenes = genesExp1+genesExp2+genesExp3;
avgGenes=totalGenes/3;
disp(avgGenes);
```

Command Window

```matlab
>> edit
>> edit myNewFile.m
>> myNewFile
132.3333
```
Scripts can create variables in the workspace.
Data Types

* Type declarations are not necessary in MATLAB

* MATLAB automatically decides data type
Data Types

**single** - single precision numerical data

**double** - double precision numerical data

**logical** - logical values of 1 or 0, represent true and false respectively

**char** - character data (strings are stored as vector of characters)

**cell array** - array of indexed cells, each capable of storing an array of a different dimension and data type

**structure** - named fields capable of storing an array of a different dimension and data type

**function handle** - pointer to a function

**user classes** - objects constructed from a user-defined class

Int8 uint8 int16 uint16 int32 uint32 int64 uint64 – don’t worry about these
Data Types

>> edit dataType.m

```
1 - geneName = 'nfbka';
2 - disp(geneName);
3 - geneExp = 367.54323;
4 -
5 - doubleVal = double(geneExp);
6 - intVal = uint32(geneExp);
7 -
8 - disp(doubleVal);
9 - disp(intVal);
10 -
11 - disp(isinteger(doubleVal));
12 - disp(isinteger(intVal));
13 -

Command Window

>> dataType
nfbka
    367.5432

    368

    0

    1

fx >>
```
Testing Data Types

```matlab
x = [1 2 3]
ispinteger(x)
isfloat(x)
isvector(x)
isscalar(x)
```

```
x =

1 2 3

ans =

0

ans =

1

ans =

1

ans =

0
```
Relational Operators

< Less than
\geq \text{ Less than or equal to }
> \text{ Greater than }
\geq \text{ Greater than or equal to }
== \text{ Equal to }
\neq \text{ Not equal to }

Useful for if statements!
If Statement

- If condition is true, execute conditional code.
- If condition is false, do nothing.

```plaintext
exp1 = 400;
exp2 = 500;
if (exp1 >= exp2)
    max = exp1;
end
```

Never run
If else Statement

```plaintext
1 - exp1 = 400;
2 - exp2 = 500;
3 - if (exp1 >= exp2)
   max = exp1
4 - else
5 - max = exp2
6 - end
```

Command Window

```
>> edit ifStatement.m
>> ifStatement

max =

500
```

fx >>
If elseif Statement

```plaintext
if (exp1 > exp2)
    max = exp1
elseif (exp1==exp2)
    max = 'Both are equal'
else
    max = exp2
end
```

Command Window
```
max =
Both are equal
```
Nested if Statement

define \( \text{exp1} = 440 \);
define \( \text{exp2} = 400 \);
if \( \text{exp1} > \text{exp2} \)
    if \( (\text{exp1}-\text{exp2}) > 50 \)
        fprintf('\text{exp1 is much bigger}')
    else
        fprintf('\text{exp1 is slightly bigger}')
    end
end
max = 'exp1'
elseif \( \text{exp1} == \text{exp2} \)
    max = 'Both are equal'
else
    max = \( \text{exp2} \)
end

Command Window

>> ifStatement
exp1 is slightly bigger
max =

exp1
For loops

```matlab
for a = 10:20
    fprintf('value of a: %d\n', a);
end
```

Command Window:

```matlab
>> edit forLoops.m
>> edit forLoopExample.m
>> forLoopExample
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```
Write a **script** that takes a list of gene names and gene expression values and outputs only those gene names over a threshold.

**PSUEDO CODE:**
For each gene in a list:
  
  if its expression value is over the threshold
  
  print the gene name and the expression value
Combining What We’ve Learned

PSUEDO CODE:
For each gene in a list:
    if its expression value is over the threshold
    print the gene name and the expression value
Combining What We’ve Learned

```
exp1Genes = {'IL1A', 'NFKBIA', 'BCL2', 'STAT5A', 'CCL5', 'MYC', 'IRF1'};
exp1Results = [4, 56, 21, 12, 39, 21, 3];
threshold = 20;

for i=1:length(exp1Results)
    if (exp1Results(i) > threshold)
        fprintf('Gene %s expression %d
', exp1Genes{i}, exp1Results(i));
    end
end
```