Octave CheatSheet

Basics

- not equal ~=
- logical AND &&
- logical OR ||
- logical XOR `xor(1,0)`

A semicolon `;` at the end of a line will suppress the output
`v = 1:5` creates a 1x5 Matrix (linear vector) with the numbers one to five. `v = 1:0.1:2` also creates a vector but with the numbers from one to two with a step size of 0.1.

```
>> v = 1:5
v =
   1   2   3   4   5
>> v = 1:0.1:2
v =
1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000
1.7000    1.8000    1.9000    2.0000
```

To display the 5th number of this vectors use `v(5)`. *Beware that ocatve is 1 based.* Displaying a range of numbers use `v(3:5)` to display the 3rd, 4th, and 5th number.

Useful functions

`help _function name_` will give you the man page of this function.
`ones(x [, y])` creates a matrix with ones. If only x is provided it is a squared matrix of size x. If y is provides it has the size x cross y.
`zeros(x [, y])` behaves the same as `ones()` but will give a zero-matrix.
`rand(x [, y])` Return a matrix with random elements uniformly distributed on the interval (0, 1).
`randn(x [, y])` Return a matrix with normally distributed random elements having zero mean and variance one. The arguments are handled the same as the arguments for `rand`.
`hist(x [, y])` Produce histogram counts of plots. `y` is the number of buckets.
`eye(x [, y]])` Produces an identity matrix.
`size(A [, DIM])` Return the number of rows and colums of A.
`DIM = 1` number rows
`DIM = 2` number columns
`length(A)` Return the *length* of the object A. For Matrix objects, the length is the number of rows or columns, whichever is greater (this odd definition is used for compatibility with MATHLAB).

```
log(a)      % natural logarithm
abs(a)      % absolute value
sign(a)     % signum function
exp(a)      % compute e^a
```

`who` Lists currently defined varibales matchin the given pattern. `whos` Provide detailed information on currently defined variables.
`sum(a)`, `sum(A,1)` sums up all columns. `sum(A,2)` sums up all rows `prod(a)` takes the product of each column
`floor(a)` rounds down `ceil(a)` rounds up
`clear` will clear all variables or only the ones who are named.

Save and Load Data

You can save and load data in octave easily with the two commands `save` and `load`.
To save a specific variable v to the file `filename.dat`

```
save filename.dat v; % save data of v in binary format
save filename.txt v --ascii % save data of v in readable version
```

Loading data is as simple as saving.

```
% both are equivalent
load filename.dat
load('filename.dat')
```

The file will be saved in the current directory and will be loaded from the current dir.

Matrixes

Let A be the matrix: `A = [1 2; 3 4; 5 6]`

```
A =
    1   2
    3   4
    5   6
```

The semicolon indicates a new row and a space or comma is a new column. Instead of typing a semicolon it is also possible to hit `enter`.
Output last element in Matrix

```
>> A(end, end)
ans = 6
```

Display Data

```
% show number in first row and second column
>> A(1,2)
ans = 2

% show second row: colon means 'all'
>> A(2,:)
ans =
    3   4

% show second column
>> A(:,2)
ans =
    2
    4
    6
```

Assign new Data

```
% replace second column by 10, 11, and 12
>> A(:,2) = [10,11,12]
% in this case it doesn't matter if comma or semicolon is used
>> A(:,2) = [10;11;12]
```

Concatinating Matrixes

```
>> B = [20 21; 22 23; 24 25]
>> C = [A B] % A B
ans =
    1   2
    3   4
    5   6
    20  21
    22  23
    24  25

>> D = [A; B] % A append B
ans =
    1   2   20  21
    3   4   22  23
    5   6   24  25
```

Transpose a matrix or a vector

```
>> A'
ans =
    1   3   5
    2   4   6
```

max values and find()

```
>> max(magic(4)) % returns a 1x4 vector with max-values per column
```

```
ans =
    16  14  15  13

>> [val, ind] = max(magic(4)) % retuns 2 vectors: 1. values, 2. indexes
val =
    16  14  15  13
ind =
    1   4   4   1
```
To find the largest value in a matrix you can either chain to `max()` functions or take the complete matrix.
```
>> max(max(A)); % Both are equivalent
>> max(A(:))
ans =
    6
```
You can search thru matrixes and vectors with a condition
```
>> find(A < 3) % returns a index-vector where number < 3
ans =
    1
    4
```

Flipping a matrix
Sometimes it is necessary to flip a matrix upsidedown.
```
>> flipud(A)
ans =
    5   6
    3   4
    1   2
```
Hint: useful in combination with the identity matrix `flipud(eye(4))`

Sum of diagonals in a square matrix
Example: A magic matrix is a square matrix where the sum of rows, columns and diagonals are the same result. Let's check the sum of diagonals
```
>> M = magic(4);
>> sum(sum(M.*eye(4))) % sum of diagonal top left to bottom right
ans = 34

>> sum(sum(M.* flipud(eye(4) ))) % sum of diagonal bottom left to top
right
ans = 34
```

Inverse matrix
There exists serveral ways to create a inverse matrix in octave
```
>> inv(M)           % warning: yes
>> pinv(M)          % warning: no
>> M^-1             % warning: no
>> M\eye(size(M))   % warning: yes
```

Ploting
`plot(x, y)` plot a graph with `x` as x-axis and `y` as y-axis. `x` and `y` has to match in length.
`hold on` will plot updated in the current plot instead of replace the current plot.
`plot(x, y2, 'r')` will plot a graph in red.
`axis([a b c d])` X-axis will start at `a` and goes to `b`, and Y-axis will start from `c` and goes to `d`. Example `axis([0.5 1 -1 1])`
`clf;` clear figure
`imagesc(A)` grid of colors for matrix `colorbar` display colorbar, kind of legend `colormap gray` display only gray-scale
Example: `imagesc(A), colorbar, colormap gray;`

Labels
`xlabel('Zoidberg')`, `ylabel('Bender')`, `legend('fry', 'lila')`, `title('Futurama')`
`print -dpng 'myPlot.png'` export plot as PNG file. For other format see `help plot`
`close` close the plot, as simple as that.
`figure(1); plot(x, y)` plot graph as *figure 1* `figure(2); plot(x, y2)` plot graph as *figure 2*
`subplot(x,y,POS);` plot multiple graphs in one windows as a X-by-Y grid at position `POS`
```
>> subplot(1,2,1);
>> plot(x, y);
>> subplot(1,2,2);
>> plot(x, y2);
```

Functions & control statements
Functions are saved in files with the file-ending `.m` for MATHLAB. The syntax is quite simple:
```
function y = function_name(x)
    y = x^2;

% y is the return value
% x is a parameter
% is also possible to return multiple values

function [y1, y2] = function_name(x)
    y1 = x^2
    y2 = x^3
```
Call your function in octave like `function_name(5)`. You have to be in the same dir as the file or add it to your search-path (`addpath()`).
`for`- and `while`-loops are easy and useful in octave.
```
>> for i=1:10
>>      disp(i)
>> end;

>> i = 1;
>> while (i ~= 10)
>>      disp(i);
>>      i = i+1;
>> endwhile;
```
It is also possible to use `if`-conditions
```
% i = 10
>> if (i == 10)
>>      sprintf('yes')
>> else
>>      sprintf('no')
>> endif
ans = yes
```

Various
The following commands can be used within octave:
- `cd`
- `pwd`
- `ls [-la]`
- `dir`

It is possible to change octave's prompt. Use `PS1('>> ');` to change the promt to `>>`
`disp(a);` will display the variable `a`. Useful in loops or conditions.
`magic(x)` Create an N-by-N magic square.
For more controll in displaying stuff use `sprintf('%i', a);`. It follows the C-standard.
`format long` or `format short` adjusts the length of output in octave.
`addpath('_path_')` will add a path to the search-path for octave. I.e. Octave will also look in this path for functions and files.