Work in progress notes of interest from *MATLAB programming with Applications for Engineers* by Stephen J. Chapman 2013.

Chapter 6 & 7

MATLAB passes by value, i.e. copies are made of input arguments to functions. Apparently, there is some mechanism in place that detects if an input variable is changed; and if not, then pass by reference is used. The exact operation this process was not exactly explained in detail and feels pretty sketchy to rely on an automatic process that may largely affect a programs operation.

Function Notes

Functions have specific commands that may be useful:

- nargin
- nargout
- inputname
- nargchk

MATLAB allows for `persistent` variables to be declared in a function. These variables are not cleared at the completion of a funciton and instead remain in the the functions local workspace.

Function Functions

MATLABs 'function functions' are functions that accept other functions as input. Some useful function functions are:

- eval('some string') % evaluates input string as if it were typed into the command window.
- feval('someFunc', vals) % evaluates given named function with input values.

Function Search Order

1. Sub functions (functions defined inside a named function)
2. Private functions (functions defined inside a folder named `private` in the current directory)
3. Current directory functions (functions defined in current folder)
4. MATLAB path

Function Handles

Function handles can be used to 'rename' a function.

```
fHandle = @funcName; % creates handle fHandle that would act the same as funcName
fHandle = str2func('funcName'); % same operation as above.
```

Anonymous Functions

One line functions without a name. Returns a function handle that accepts defined input arguments.

```
>> aFunc = @(x) x^2;
>> aFunc(4)
ans =
    16
```

Chapter 9

While MATLAB is focused on using of matrices, or arrays, more advanced structures exist that aid in the manipulation of data.

Cell Arrays

Cell Arrays contain data structures. They are often used for collection of strings as each entry can be a different length. Parenthesis are used to return data structure information in a cell. Indexing using braces {} returns data. Further indexing of data can then be performed by parenthesis ().

```
% Cell Creation
a{1,1} = rand(2,3)
a{2,1} = magic(4)
a{2,2} = 'This is a cell'

% Cell indexing examples
a(1,1) % Display data strucuture information from ROW 1, COL 1
a{1,1} % Display all data in cell ROW 1, COL 1
a{2,1}(2,2) % Display data entered in ROW 2, COL 2 from item in cell ROW 2, COL 1

% Cell specific functions
celldisp(a) % Display cell data strucuture information
cellplot(a) % plot visual representation of cell
```

Structured Array

A structured array is based on the idea of objects and fields. All objects in a structured array contain identical field names, but unique data. As a result, each field may contain different types of data. Some useful structured array specific functions include:

- getfield(struct, fieldName)
- setfield(struct, fieldName, data)
- fieldnames(struct)

While the get and set field functions are easy to use, MATLAB will often recommend using *Dynamic Field Names*. The below code provides examples of these functions and slight explanations.

```
clear; clc; close all
%% Cell Creation
a{1,1} = rand(2,3)
a{2,1} = magic(4)
a{2,2} = 'This is a cell'

%% Structure Creation
s.a = a
s.t = 2
s.Name = 'First'

%% Addition to Structure
s(2).Name = 'Second'
% not each field is required to have the same type of data
s(4).a = 'This is a string'
% structures do not have to be created sequentially
s(5).Name = 'Fifth'
% definition of field via setfield function
setfield(s(5), 't', 56)

%% Simple Dynamic Field Name Example
s(2).Name % display current data
s(2).('Name') % same as above, but using dynamic field format
f = 'Name'; % Using dynamic field names with variables is more useful
s(2).(f) %

%% Some structure manipulations
fNames = fieldnames(s); % returns field names as cell
% for each structured array
for sNdx = 1:size(s, 2)
    % Introduce each strucutred array
    fprintf('*** s.(%d) has Fields:\n', sNdx)
    % display all fields
    for fNdx = 1:size(fNames)
        fprintf('%s: \n', fNames{fNdx} ) % display field name
        if isempty( getfield(s(sNdx), fNames{fNdx}) )
            % clarify existance of empty fields
            disp('   Empty')
        else
            % display data of each field using dynamic field names
            disp( s(sNdx).(fNames{fNdx}) )
            % same as above but using getfield function
            disp( getfield(s(sNdx), fNames{fNdx}) )
        end
    end
end
```