MATLAB Style Guidelines Cheat Sheet

Compiled by Jason Nicholson, jashale@yahoo.com Edited by Richard Johnson, Version 1.3, 2014-January-29

This is a quick reference. For more information, see *The Elements of MATLAB Style*, ISBN 0521732581, by Richard Johnson linked in the references at the bottom.

Naming Conventions

Variables

Variable names should be mixed case starting with lower case.
```
    velocity, angularAcceleration.
```
Variables with a large scope should have meaningful names.
Variables with a small scope can have short names.
```
    Small scope: x, y, z
    Large scope: velocity, acceleration
```
The prefix n should be used for variables representing the number of objects.
```
    nFiles, nCars, nLines
```
Use a convention on pluralization consistently.
```
    point, pointArray
```
Variables representing a single entity number can be suffixed by *No*.
```
    tableNo, employeeNo
```
Iterator variables should be named or prefixed with *i, j, k,* etc.
```
    iFiles, jColumns
```
For nested loops, the iterator should be alphabetical order and helpful names.
```
    for iFiles = 1:nFiles
        for jPositions = 1:nPositions
        …
        end
    end
```
Avoid negated boolean variable names.

    ~~isNotFound~~ instead use `isFound`

Acronyms, even if normally uppercase, should be mixed or lower case.
```
    Use html, isUsaSpecific
```
Avoid using a keyword or special value name.
```
    Just don't do it.
```

Constants

Named constants should be all uppercase using underscore to separate words.
```
    MAX_ITERATIONS, COLOR_RED
```
Constants can be prefixed by a common type name.
```
    COLOR_RED, COLOR_GREEN, COLOR_BLUE
```

Structures

Structure names should be mixed case and begin with a capital letter.
```
    Car, DumpTruck
```
Do not include the name of the structure in the field name.
```
    Use Segment.length
    Avoid Segment.segmentLength
```

Functions

The names of functions should document there use.

Names of functions should be written in lower or mixed case.
```
    width(), computeTotalWidth()
```
Functions should have meaningful names.
```
    Use computeTotalWidth
    Avoid compwid
```
Functions with single output can be named for the output
```
    shearStress(), standardError()
```
Functions with no output argument or which only return a handle should be named after what they do.
```
    plotfft()
```
Reserve the prefix *get/set* for accessing an object or property.
```
    getobj(), setappdata()
```
Reserve the prefix *compute* for methods where something is computed.
```
    computeSumOfResiduals(),
    computeSpread()
```
Reserve the prefix *find* for methods where something is looked up.
```
    findOldestRecord()
```
Reserve the prefix *initialize* for instantiating an object or concept.
```
    initializeProblemState()
```
Reserve the prefix *is* for boolean functions.
```
    isCrazy, isNuts, isOffHisRocker
```
Use complement names for complement operations.
```
    get/set, add/remove, create/destroy,
    start/stop, insert/delete,
    increment/decrement, old/new, begin/end,
    first/last, up/down, min/max,
    next/previous, open/close, show/hide,
    suspend/resume, etc.
```
Avoid unintentional shadowing of function names. Use which -all or exist to check for shadowing.

General

Abbreviations in names should be avoided.
```
    Use computeArrivalTime
    Avoid comparr
```
Consider making names pronounceable.

All names should be written in English.

Files and Organization

M-Files

Modularize code. Use small well designed pieces to make the whole.

Write functions that are easy to test.

Make interaction clear. Use inputs and outputs rather than global variables.

Replace long lists of arguments with structures.

Partitioning. All sub-functions and most functions should do one thing very well.

Use existing functions rather than custom coded functions when possible.

Move blocks of code used in multiple m-files to functions.

Use sub-functions when a function is only called by one other function.

Write test scripts for every function.

Input/Output

Make input and output modules for large functions.

Format output for easy use. For humans, make it human readable. For machines, make it parsable.

Statements

Variables and constants

Variables should not be reused unless required by memory limitations.

Related variables of the same type can be declared in a common statement. Unrelated variables should not be declared in the same statement.
```
    persistent x, y, z
```
Document important variables in comments near the start of the file.

Document constants with end of line comments.
```
    THRESHOLD = 10; % Max noise level
```

Global Variables

Minimize use of global variables and constants.

Consider using a function instead of a global constant.

Loops

Variables used in loops should be initialized immediately before the loop.
```
    result = zeros(nDays,1);
    for iDay = 1:nDays
        result(iDay) = foo(iDay);
    end
```
Minimize the use of *break* and *continue* in loops.

The end lines in nested loops can have comments to clarify the code block.
```
    for index=1:2
        if index==1
                dosomething(index);
                …
        end % End if
    end % End for
```

Conditionals

Avoid complicated conditional expressions. Use temporary logical variables instead.
```
    isValid = (v >= lowerLimit) &
              (v <= upperLimit);
    isNew = ismember(v, valueArray);
```
Avoid the conditional expression *if 0*.

An *if-else* sequence should include the else condition.

The usual case should be put in the *if*-part and the exception in the *else*-part of an *if-else* statement.

A *switch* statement should include the *otherwise* condition.

Use a *switch* sequence if the variable is a string.

Use a *switch* statements in place of many *if-elseif-else* statements when possible.

General

Avoid cryptic code. You should be able to look at it a month from now and know what it does.

Use parentheses for clarity even if not need because of operator precedence.

Minimize the use of numbers in expressions. Use a named constant instead.

Always use a zero before the decimal point.
```
    THRESHOLD = 0.5
```
Make floating point comparisons with caution.

Layout, Comments, and Documentation

Layout

Contents should be kept within the first 80 columns.

Lines should be split after commas, spaces, and operators.

Align a continued line with the beginning of the expression on the previous line.
```
    totalSum = a + b + c …
               d + e;
```
Basic indentation should be 4 spaces.

In general, a line of code should contain only one executable statement.

Short single statement *if*, *for*, or *while* statements can be written on one line.
```
    if (condition), statement; end
```

White Space

Surround =, &, and | by spaces.

Follow commas by a space.

Keywords should be followed by a space.

Blocks of code should be separated by three blank lines or a section break.

Use code alignment wherever it enhances readability.

Comments

Comments cannot justify poorly written code.

Comments should agree with the code but not restate the code.

Comments should have the same indentation as the statement(s) referenced.

Traditional function header comments should support *help* and *lookfor*. *help* prints the first continuous block of comments. *lookfor* searches the 1st comment line of all m-files on the path.

Function headers should discuss any special requirements for the input/output argument and describe any side effects of the function.

Write the function name using correct case in the function header comments.
```
    function runEverything
    % runEverything runs all mfiles
      in its folder
```
Put any copyright lines and change history after the function header with a blank line in between.

All comments should be in English.

Documentation

Write header comments with text markup to provide user documentation. Include sections that correspond to a help page: syntax, description, example, and see also.

Consider writing the documentation first to better define inputs, outputs and functionality.

Consider using a source control tool such as SVN or GIT. If you do not use a source control tool, document changes by adding change history comments after the function header or near the top of the script.

References

Johnson, Richard. "MATLAB Programming Style Guidelines." Version 1.5. Datatool. 2002-Oct. Accessed 2014-Jan-15. http://goo.gl/uyGVM5
Johnson, Richard. "*The Elements of MATLAB Style.* Cambridge University Press, 2010-Dec-31. http://goo.gl/cUA7Gr
Johnson, Richard. "Updates to The Elements of MATLAB Style." 2012-May-07. Accessed 2014-Jan-29. http://goo.gl/GwHqNp