Coding Guidelines

For MATLAB®, Processing®, z-Tree and many other programming languages.

Adrian Etter

University of Zurich
Department of Economics
Winterthurerstrasse 30
CH-8006 Zurich

E-Mail: adrian.etter@econ.uzh.ch
URL: http://www.econ.uzh.ch/faculty/etter.html

13 February 2013

University of Zurich[UZH]

Abstract

This document is intended to provide basic tips and tricks in style, alignment and programming technics to be used at the laboratories of the Department of Economics at the University of Zurich. This style guide is not mandatory, but adhering to a style guide makes the collaboration between researchers a lot easier. Furthermore it improves the reusability of codes and scripts and if codes / scripts have to be reconstructed / reconducted after a while it's a lot easier if that code was written according to a style guide. Well-formed code is a lot easier to maintain and debug. Other researchers benefit from well written code and it helps to provide a consistent code repository.
All the examples are done with MATLAB, but can easily be adapted to other programming languages!

Table of contents:

Prerecord

English There's only one language to program. It's English! This applies also to comments.

DRY Don't repeat yourself!

KISS Keep it simple, stupid!

Data Storage

Generic path On all the lab computers you have a separate drive with a user folder. In there you find the group folders. For the structure inside your group, it's up to your group to keep it tidy.

Be aware: Network drives are not as reliable as local drives. Never load a file from a network drive during your experiment nor store data on the network path while your experiment is running. It could interrupt or crash.

Generally used software – like Cogent, Psychtoolbox and other SNS-Lab toolboxes – should be saved in the system path.
What's only for your experiment/script has nothing to do in the global path and should not be saved there.

Example

```
[directory, filename, extension] = fileparts(which(mfilename));
directoryPath = [directory, filesep, 'subfolder'];
addpath(directoryPath);
:; % Here comes the code that uses functions from a subfolder
rmpath(directoryPath);
```

- which: returns the path to the called function.
- mfilename: returns the name of M-file itself – the currently executed M-file[1].
- filesep: returns the directory separator on the actual operating system – '\' on windows, '/' on mac/Unix.
- addpath(tempPath): adds the specified path **temporarily** to the MATLAB path.
- rmpath(tempPath): removes the path. Try to always remove your added paths. If someone after you uses MATLAB without restarting MATLAB, it can disturb his script.

In general leave computers and programs as they were found upon arrival. If you always have to add things that you think they are for common use and you can't find them, ask the technical staff to make it available for everyone.

1

Header

Help

If you type 'help function' to the MATLAB command line, it will return the first continuous block of comment lines of an M-file below the function name. If you create a function, always make a helpful description in the header, see the example below.

Lookfor

The 'lookfor' command is a mighty function to quickly search for a function. Every first comment line below the function name is included in the lookfor command. Make sure to add useful keywords in the first comment line.

Author
Change History

To follow up changes and authorship it's always a good habit to add an author and version line. For smaller projects it's sufficient to add a version and last revision line. For major projects you should consider a revision control tool as e.g. Git or SVN.

Example

```
function example_header( ~ )
%EXAMPLE_HEADER Shows an example of a M-file header.
%   This function is part of the SNS-Guideline examples. It is an example
%   for how a header in an M-file should look like.
%   The first line of text after the function is for the lookfor command.
%   The first continuous block of command lines will be displayed if
%   'help example_header' is called! Try it out!

%
% Syntax:  example_header()
%
% Inputs:
%    no input
%
% Outputs:
%    no output
%
% Example:
%    example_header();

%
% Other M-files required: none
% Subfunctions: none
% MAT-files required: none

%
% See also: OTHER_FUNCTION_NAME1,  OTHER_FUNCTION_NAME2

% Author: Adrian Etter
% email: adrian.etter@econ.uzh.ch
% © SNS-Lab, University of Zurich
% Version 1.0 2011/Apr/04
% Last revision: 2011/Apr/04 by John Do
% -Added version history

%% HERE STARTS THE CODE
clc; % Clear the screen
fprintf('\n\nThe lookfor command: lookfor header;\n\n');
lookfor header;
fprintf('\n\nThe help command: help example_header;\n\n');
help example_header;
```

```
%% HERE ENDS THE CODE

% I always add an end at the end of a function. It's not necessary but it
% indicates that the function ends. It makes your code more readable
% especially if you use subfunctions. In other programming languages it's
% most common to have an ending tag.
end
```

Naming convention

Name the issue
No cryptologic competition!

Name it by its purpose! Any other name will be confusing. You don't name a hedgehog dog, just because dog is shorter ;)
It's always competitive to find appellative names even for experienced programmer. Try to keep the naming as simple as possible – name it by its purpose and keep it declarative!

Variables

Variables start with lower case

Variable can be in mixed case but should always start with a lower case, as it is common in most other programming languages.
Be careful with **underscores**. Underscores can make variables more readable but MATLABs TEX interpreter will read it as a switch for **subscript.**

```
E.g.: signalType, subjectAge, stimulationIntensity, stepWidth
```

Scope

Every variable should have a meaningful name. In particular you should take care that variables with a large scope have especially reasonable names. For variables with a small scope naming is less important. In general it's always good to give meaningful names.

Scratch variables

Variables whose value is only used within a few lines are known as scratch variables. They should always be initialized in sight of their usage.

Integer scratch variables

Common integer scratch variables are:

```
i, j, k, m, n
```

Double scratch variables

Common floating-point – double in MATLAB– scratch variables are:

```
x, y, z
```

Number of Objects

To name a number of objects variables should start with a prefixed n. A MATLAB specific addition is a prefixed m to indicate the number of rows – mRows. Keep attention on the ending **'s'**, pluralism always ends with an 's'!

```
E.g.: nTrials, nLines, nFiles, nSegments, nColumns, mRows
```

Pluralization

Pluralism always ends with a suffixed **'s'**. Sometimes there are two variables with the same name except for one ending with a suffixed 's'. Keep in mind that it could impair to read your code. Maybe you like to simply add a suffixed 'Array' instead of the 's'.

```
E.g.: point → points or pointArray

endowment → endowments or endowmentArray

distribution → distributions or distributionArray
```

Entity number

To numerate subjects or entities add a suffixing 'No'. A prefixed 'i' make the variables iterators.

E.g.: tableNo, subjectNo, studyNo, trialNo.

Iterators

Iterators should be named with a prefixed '**i, j, k etc.**'.
Note: If you use complex numbers, 'i' or 'j' should be reserved as the imaginary unit.

```
for iTrials = 1:nTrials
    ;
end
```

Units

If your variable is dimensioned to a specific unit, the unit should be suffixed.

`e.g. outputVoltage, velocitymeterpersecond`

Constants

UPPERCASE

Constants should be clearly stated as constant. In other programming languages it's very common to use UPPERCASE for constants. Why don't you do it likewise?

```
PARALLEL_PORT_ADDRESS = hex2dec('378');
COM_PORT = 'COM1';
MAX_OUTPUT = 123;
TRIAL_MODE = 'fmri';
```

Globals

Try to avoid them, as they are on one hand hard to identify, on the other hand you never know if a called function uses the same variable name and just overwrites its value. **Dangerous!**
If you cannot avoid a global variable, document it clearly and name it with a very significant name!

Structures

Structures start with a capital

Structures should always start with a capital letter to clearly separate them from ordinary variables.
Obviously there is no need to repeat the name of the structure in the fieldname.

Avoid:

```
Subject.subjectName
```

Use:

```
Subject.name
Study.id
Session.trialNo
```

Functions

Form follows function.
The name should self-explain its use!

Lower case

Functions should always be in lower case to avoid interoperability problems – the function name always is the filename. Windows is the only operating system with case insensitive path names. But as they move in mysterious ways there are paths on Windows 7 which are case sensitive.
MATLAB mentions that in future releases only a **matching case** on a function call will be executed!
For some unexplainable matter MATLAB write their functions in the help response in UPPERCASE…

Suitable prefixes

Use suitable prefixes according to its output for your functions!
e.g. "***compute****FunctionName*" if the response is a computation.
"***find****FunctionName*" if the output is the result of a search function
"***is****FunctionName*" if and **only if** the output is Boolean. Sometimes it fits better with a prefixed "*can, has, should"*!

```
e.g. isconnected(~), iscomplete(~), hasexperience(~)
```

Abbreviation

Avoid the use for abbreviation. Your code will become cryptic. There is no limitation in the length of a function name!
Common abbreviations like *max(~)* or *gcd(~)* should only be used, if their full qualified name is in the first comment line for clarity and the *lookfor* command.

Ambiguity

Avoid shadowing/ambiguity by checking if your function name is unique.

```
E.g.: which yourfunction -all, exist('yourfunction')
```

Layout

One per line

One line should only contain a single statement.

Indentation

The standard MATLAB indentation should be the default and you should style your code accordingly. In MATLAB the default tab spacing is 4 spaces.

Logical groups

Logical groups of statements should be separated by one blank line.

Cell mode

Cell mode helps block corresponding code

```
%% LOAD THE PATH TO YOUR TOOLBOX
[directory, filename, extension] = fileparts(which(mfilename));
directoryPath = [directory, filesep, 'toolbox'];
addpath(directoryPath);

%% PREPARE THE STIMULATION
% Loads Assembly, Prepares stimulation
initalizestimulation();

%% DEFINE THE CHANNELS YOU WANT TO USE
channels = [2,4,6:14];

%% TRIAL
runTrial(channels);

%% END OF STIMULATION
cleanupstimulation();
rmpath(directoryPath);
```

Line breaks
Alignment

Standard MATLAB editor recommends a line break after 80 columns. It's a common width for the most editors and terminals and you should stay with it.
Line breaks can be made:

- after a comma
- after a whitespace
- before an operator

Align the new line with the beginning of the expression.

```
[decisionMean, decisionVariance] = somefunction(ACONSTANT, ...
                                                firstDecision ...
                                                + secondDecision ...
                                                + thirdDecision ...
                                                * scale);
```

Single statement

A single statement can be used on one line.
E.g.:

```
if (isConnected()), disconnect(); end
while (condition), function(); end
for iTest = 1 : nTest, function(); end
```

White space Add whitespace as in prose. Surround '=', '&', '&&', '|', '||', '+', '−', '*', '/' , ':'

```
E.g.: result = (firstTerm + secondTerm) / thirdTerm

Somefunction(firstArgument, secondArgument, thirdArgument)
```

University of Zurich[UZH]

Documentation – Comments

Add information

Variables and constants should be named on its purpose and therefor self-explain their purpose. If not, document it. For statements don't explain what it does, that should be self-explanatory by the way it is coded. Explain why and how you implemented the algorithm.
Write like in a conversation, like to instruct someone.

Immediate Comment as you code

Document while you're coding and not after. Experience shows that the best practice is to comment your code while you are writing it.

Further implementation hints

Nested loops

In nested loops the iterators should be sorted in alphabetical order!

Negated Boolean

Should be avoided:

```
~isNotEmpty → isEmpty;
~isNotFound → isFound;
~isNotFinished → isFinished;
```

Acronyms

Acronyms in statements or functions should be lowercased
HTML → html, isHtml, isTiff

Ambiguity – MATLAB-Keyword

Check your functions if they do not already exist.

```
MATLAB: iskeyword('yourfunction');
```

Interoperability – Platform independency

Try always to code platform independent.

No hardcoded paths

Never use hardcoded paths!

[directory, filename, extension] = fileparts(which(mfilename));

```
directoryPath = [directory, filesep, 'subfolder'];
```

32 bit vs. 64 bit

Use a switch to call platform specific functions. Whenever possible try to handle all platforms:

```
switch computer
    case 'PCWIN
        % Windows 32 bit functions, e.g. mexw32 functions
    case 'PCWIN64'
        % Windows 32 bit functions, e.g. mexw32 functions
    case 'GLNX86'
        % Linux 32 bit functions, e.g. mexglx functions
    case 'GLNXA64'
        % Linux 64 bit functions, e.g. mexa64 functions
    case 'MACI64'
        % Mac functions, e.g. mexmaci64 functions
    otherwise
        error('This OS is not supported')
end
```

Acknowledgment

Many thanks to Stefan Schmid for revising, spotting typos and contribute with helpful hints

References

[1] Richard K. Johnson, "*MATLAB Elements of Style*", Cambridge, University Press, 2011