KU LEUVEN

MATLAB

Reproducibility etc.

1

Research code

KU LEUVEN

2

Reproducible code

- Reproducibility: research should be able to be reproduced in order to validate its conclusions.
- Research code is it easy to reproduce?
  - Versions of software? Dependencies?
  - Toolboxes required?
  - Right data format for input?
  - Etc.

Reproducible code

Concerns:

- Coding/Programming style
- Project organisation
- Documentation
- Version control
- Testing

Reproducible code: Sources to check

- A concise guide to reproducible MATLAB projects (David Wilby)
  https://rse.shef.ac.uk/blog/2022-05-05-concise-guide-to-reproducible-matlab/
  - Matlab oriented
  - Structured overview
  - Useful links
- The Good Research Code Handbook (Patrick Mineault)
  https://goodresearch.dev/
  - Focus on Python projects

5

Programming style

6

Programming Style

- Purpose: write programs that are not only **correct** but also **understandable**.
- Will make programs easier to read.
- Allows to quickly understand what you did in your program when you look at it days, weeks, or years from now.
  Remember the style conventions are for the reader, but you will someday be that reader.
- Kerninghan: "*Well-written programs are better than badly-written ones – they have fewer errors and are easier to debug*"

Programming Style

- A set of conventions that the programmer follows to standardize its computer code to some degree and to make the overall program easier to read
- Consistent programming style gives your program a visual familiarity that helps the reader quickly understand the intention of the code
- A programming style consists of:
  - Visual appearance of the code
  - Conventions used for variable names
  - Documentation with comment statements
  - Etc.

Reference material

- Matlab Programming Style Guidelines
https://nl.mathworks.com/matlabcentral/fileexchange/46056-matlab-style-guidelines-2-0

- Guidelines for writing clean and fast code in MATLAB
https://nl.mathworks.com/matlabcentral/fileexchange/22943-guidelines-for-writing-clean-and-fast-code-in-matlab
- https://github.com/nschloe/matlab-guidelines

Programming Style

Advice:

- Follow the specifications.
- Comment the code.
- Provide user documentation.
- Explain major variables, methods used as well as functions.
- Favor clarity over brevity and ingenuity.
- Use indentation.
- Generalize when possible →Reuse code.
- Implement standard libraries if possible.

- Example:
  - *StyleGuide_1.m*
  - *StyleGuide_2.m*

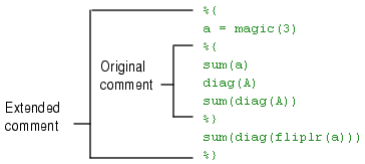
Matlab Comment

- % Matlab comment sign
- %{ %} block comment
- nesting comments is supported

Comment a block of code by adding %{ before the first line and %} after the last line.

```
a = magic(3)
%{
sum(a)
diag(A)
sum(diag(a))
%}
sum(diag(fliplr(a)))
```

Create a nested comment, that is, a block comment within a block comment.



When Writing

- **Indentation**
  - Follow a simple, key principle: **Indent Substructures.**
  - Comments for a group of statements should go above, with the corresponding statements indented below the comment.
  - Good programming involves breaking larger problems into smaller problems.
    The way the sub-problems fit into the larger problems should be immediately apparent from the code for maximum clarity.
  - In Matlab editor: **ctrl-i**

indendation

Which is easier to read?

```
n = length(x)
if mean(x) <0
   y(1)=0; y(n) =0;
   for k=2:n-1
      y(k) = x(k+1)-x(k-1);
   end
else
   y(1)=x(1); y(n) = x(n);
   for k=2:n-1
      y(k)=0.5*( x(k+1)+ x(k-1));
   end
end
```

How do you visualize your code?

```
n = length(x)
if mean(x) <0
y(1)=0; y(n) =0;
for k=2:n-1
y(k) = x(k+1)-x(k-1);
end
else
y(1)=x(1); y(n) = x(n);
for k=2:n-1
y(k)=0.5*( x(k+1)+ x(k-1));
end
end
```

Source: http://stommel.tamu.edu/~esandt/

KU LEUVEN

Variable Naming

- **Variable names should be short and meaningful**
- Short names are non-instructive, long(er) names give more information at a small cost
- camelCase
  - No spaces
  - firstLetterLowerCaseForVariablesAndFunctions
- use_underscores
- Use capital characters for constants
  - HOURS_DAY = 24;

KU LEUVEN

Variable Naming

- Advice:
  - Avoid names of less than 5 characters
  - Use as many variables as possible to make changes easier!
  - Use consistent notation

Variable	typical code	meaningful code
inside diameter	d=5;	ins_diam = 5;
thickness	t=0.02;	thickness = 0.02;
inside radius	r=d/2;	ins_radius = ins_diam / 2;
outside radius	r2=r+t;	out_radius = ins_radius + thickness;

Source: http://stommel.tamu.edu/~esandt/

Quiz

```
function v = func1(a,b)
    v = 1
    if a > 24 || a < 0
        v = 0
    end
    if b > 60 || b < 0
        v = 0
    end
```

```
function isValid =
validTime(hour, min)
    isValid = 1
    if hour > 24 || hour <
0
        isValid = 0
    end
    if min > 60 || min < 0
        isValid = 0
    end
```

Taken from Colorado Reed - mlg.eng.cam.ac.uk/creed/

Example function header

```
%
 % function sorted_array = sort( array )
 %
 % Author:  H. James de St. Germain (germain@eng.utah.edu)
 % Date:    Spring 2027
 % Partner: I worked alone
 % Course:  Computer Science 1000
 %
 % Function  : Sort
 %
 % Purpose    : The Sort Procedure sorts an array of
 %              integers using a standard bubble sort.
 %
 % Parameters : array             - an vector of integers
 %
 % Return Value : the sorted array
 %
 %
```

http://www.cs.utah.edu/classes/cs1001/Design_Patterns/style_guide.html

What should be in the header?

Comments section

- The name of the program and any key words in the first line.
- The date created, and the creators' names in the second line.
- The definitions of the variable names for every input and output variable.
  - definitions of variables used in the calculations
  - units of measurement for all input and all output variables!
- The (file)name of every user-defined function called by the program.
- Log of changes
- …

Summary

- **Design**
  - Do not rush into writing code.
  - First design your program by carefully writing out a flowchart for the algorithm.
  - Think carefully about all potential problems before rushing into implementation (writing the actual code).
  - Computers are extremely logical and so you need to be methodical to program them!
- **Structured code**
  - Structure the program into a number of smaller, self contained modular tasks.
  - The secret of getting the program to work quickly without bugs is to code each of these smaller tasks in separate self-contained subroutines which can be written and tested independently.
  - Subroutines should be the bricks you use to build your program.

Summary

- **Debugging**
  - If you do encounter an error, try to locate the error by using the debugger
  - Take particular care to match the parameters in function calls with those expected by the function header (check array sizes, …).
  - When you do locate a programming error, resist the temptation to correct it and run the program again immediately. Satisfy yourself that you can see why the error occurred, and that the correction will change things in the right way.
  - Understand exactly what you program is doing and why at each stage.
- **Document clearly**
  - Do not forget the programming style guide
  - Make your code as easily followed and neat as possible.
    If the code is easy to follow, you make fewer mistakes and it is much easier for some-one else to help you.
  - use indentation it helps enormously to identify the ordering of logical structures.
  - Add concise “meaningful” comments as you build up the code.
  - Use sensible and meaningful names for variables, subroutines, etc.
  - Use longer more meaningful names rather than single letter names