Matlab Style Guide

> *"Guidelines are not commandments. Their goal is simply to help programmers write well."* – *Richard Johnson, MATLAB Programming Style Guidelines*
>
> *"Never offend people with style when you can offend them with substance."* – *Sam Brown*
>
> *"We cannot solve our problems with the same thinking we used when we created them."* – *Albert Einstein*

Maintainers of this style guide: Bolko Beutner (2757), Lars Fröhlich (3857)

Motivation & Generalities

- **Code is read more often than it is written.**
  Code should be easy to read. Not only for colleagues that have to debug or maintain it when you are not around, but also for yourself – ever try to remember how that tricky program works that you wrote three years ago? A clean, consistent style makes code more readable, maintainable, and debuggable. This style guide is a collection of a few core practices to help with this.
- **These are guidelines, not commandments.**
  You can deviate from the practices of this style guide whenever it seems like a good idea. But please, think about it carefully. It does not hurt to document your reasoning in the code as well.
- **Be consistent.**
  When editing someone else's code, you should follow the stylistic conventions used in that package, even if they do not agree with these guidelines. *Unless* you can agree with the original maintainer to retrofit the entire package, of course. You'll be our hero of the day.
- **Starting from scratch?**
  If you are writing a new piece of code, *please* follow these guidelines. Just consider what might happen if you don't: Your colleagues will sneer at you behind your back, there will be earthquakes, floods, and Berlin will be cooler than Hamburg. You don't want that.

Indentation

- **Use spaces instead of tabs.**
  Tabs may expand to different widths depending on editor settings, making source code hard to read.
- **Use 4 spaces to indent code.**
  4 spaces is Matlab's default setting and therefore the setting used in most of our code.

Names

- **Class names use CapitalCamelCase.**
  Class names always start with a capital letter and use camel case. No underscores are allowed:
  `MyCoolClass` (not my_cool_class or myCoolClass)
  Exception: In the hlc package, classes start with a lowercase "hlc" prefix:
  `hlcMyCoolClass`
- **Function and method names start with a verb and use_lowercase_and_underscores().**
  Function and method names should state clearly what the function does. They start with a verb, are all lowercase and use underscores to separate words:
  `prepare_emittance_measurement(), load_data_from_file(), plot_sine_wave()`
  Exception: In the hlc package, functions start with a lowercase "hlc_" prefix:
  `hlc_read_reprate()`
- **Variable names are lowercase and use underscores to separate words.**
  Variable names should state clearly what is stored in the variable, are all-lowercase and use underscores to separate words. Do not be afraid of long variable names:
  `normalized_emittance, full_filename, momentum_gain`
  Single- or two-letter variables are only acceptable where their purpose is obvious, like in loops:
  `n = 0; for i = 1:10; n = n + 1; end`

Interfaces

- **Document function parameters**.
  All parameters of a function should be documented if they are not very(!) obvious. Physical quantities should have units.