Cheatography

MATLAB Repo Cheat Sheet

by [deleted] via cheatography.com/52740/cs/14288/

Naming conventions

Variable names in camel case	`moveToPosition`
Functions & methods in camel case	`sendFilesOverNetwork`
Acronyms follow camel case	`htmlPage, isJpeg`
Use meaningful names	`wage = hourlyRate * nHours`
n prefix for amounts	`nFiles`
No suffix for specific numbers	`measurementNo`
Cnt suffix for iterator variables	`sampleCnt = 1:nSamples`
Constant uppercase + underscore	`SPEED_OF_LIGHT`
Classes & stuctures capitalized	`RadarSensor`
Non-default units in variable name	`BOTTLE_CAPACITY_CL = 50`

Layout

Split lines if needed	`longMethodName(... long, list, of, params)`
Align for better readability	`value = (10 * nDimes) + ... (5 * nNickels) + ... (1 * nPennies);`
Commas are followed by spaces	`add(a, b, c)`
Brackets are NOT surrounded by spaces	`mean(profitPerMonth)`
Separate logical groups of code by a blank line	
Separate functional groups of code by using `%%`	

Cheatography

MATLAB Repo Cheat Sheet

by [deleted] via cheatography.com/52740/cs/14288/

Best practices

Write code as functions when possible

> The main role of scripts is in development because they provide direct visibility of variables. Functions modularize computation by using internal variables, and tend to be cleaner and more flexible. It is also much easier to manage changes if code appears in only one file.

Don't (always) use `clear all`

> Whenever Matlab meets new code it pre-compiles it, making re-runs faster. Make sure you don't use `clear all` when you meant to use `clear` - the former not only clears variables but also the pre-compiled code.

Helper functions

> A single .m file can contain multiple functions. Although only the first function is exposed, this allows for better structuring and reuse of the code.

Subfunctions

> It's possible to nest functions, allowing for better structuring while strongly associating the subfunction with it parent.

Comments

Write comments only when needed

For comments after code, prefix '␣␣%␣'

```
SAMPLE_FREQ = 3e9   % Hz
```

Align for better readability

```
var = 1;        % Comment
otherVar = 2;   % Other
comment
```

Documentation

Class header gives a description of the class and a summary of all its functions

Function headers give information about the inputs & outputs

Include a demo file for non-trivial classes and functions

Speed-ups

Preallocation

> Allocate empty matrices to store incremental results generated by loops. If the number of results is unknown, you can allocate the for maximum amount and prune afterward (if possible).

Speed-ups (cont)

Vectorization

> Placing a period (.) before the operators *, /, and ^, transforms them into array operators.
> Some functions also accept vectors/matrices as input and perform their computations on element/row-wise.
>
> *Example*
>
> ```
> t = 0:.01:10;
> y = sin(t);
> ```

Indexing

> Use vector or boolean indexing instead of loops where possible
>
> *Example*
>
> ```
> someNumbers(someNumbers < 0) = 0;
> ```

Profiler

> To determine which parts of your code are the most time consuming use the `tic toc`, `timeit`, or `profile` commands.

Workflow

Create a new branch for your feature

```
git checkout -b featurename
```

Use git as you normally would

```
git add file1.m file2.m
git commit -m "message"
git pull origin featurename
git push origin featurename
```

Regularly merge the master branch into the feature branch

```
(git checkout featurename)
git fetch origin
git merge origin/master
```

One the feature is completed, file a merge request on GitLab.

> If your request if rejected.

Sources

Guidelines for writing clean and fast code in MATLAB - Nico Schlömer

MATLAB Style Guidelines 2.0 - Richard Johnson

The Elements of MATLAB Style - Richard Johnson