

# Going from Matlab to Python

## A generic workflow

October 6<sup>th</sup>, 2022 | Sandra Diaz Pier

# Where to start

- Identify the type of computation you are performing
- Identify computational steps in your code
- Do basic profiling on your original Matlab code
- Identify input / output
- If you need parallel, think parallel from the beginning

# What to do next

- Take one step at the time and translate the functionality
- Build a unit test which allows you to compare to known results => your Matlab results
- Document your code as you write it
- Commit frequently
- Check code performance

Try to adhere to programming standards (PEP8):  
[Python Enhancement Proposal](#). A proper IDE or plugins (autopep) will help you to do so, too.

# How to implement your steps

- Each computation step has an input and an output
- Identify the interfaces among steps

# How to implement your steps

- Identify the correct data structures in your computation
- Python provides a larger diversity of data structures (lists, dictionaries, arrays, matrices, etc...)

# How to implement your steps

- Remember Python has a large community of users. Look for modules which can make your computation easier.
- If you use Matlab commands from a specific toolbox, look for equivalent Python modules.
- Do not reinvent the wheel.
- Use sites such as stackoverflow etc. (know your sources or how to search)
- Do not forget to acknowledge the source

# How to implement your steps

- Break down computation into functions, classes or even modules.
- (Always) think about clean / reusable / maintainable code

# Debugging

- PDB
    - Python's interactive source code debugger
    - Available as a module; *import pdb*
  - Print and test
    - Better to use with a filename and line number.
- ```
if x > 23:  
    print("Debugging: my_file.py, line 11")  
    print("!!! x is {} !!!".format(x))
```



# Debugging

- Set breakpoints in Matlab and inspect variables in workspace
- compare with debug mode in python
- interactively compare functions such as **norm** (Matlab) and **linalg.norm** (Python)
  - $\text{norm}(m) \neq \text{linalg.norm}(m) \mid m \text{ some Matrix}$
- *Interactively*: Create a minimal example inside Matlab workspace and IPython shell

# Testing & Comparing

- if randomness is involved set the seed
- save random created variables
- load them into the other program
- read the documentation
- Useful sites
  - <https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html>
  - <http://mathesaurus.sourceforge.net/matlab-numpy.html>

# Pointers

- Do not forget the indexing, Matlab starts with 1 and Python 0
- Numpy matrices vs. arrays, go with arrays, unless you need something specific (e.g. sparse matrices)
- Column view vs. row view
- Dot and element-wise multiplication
- In Matlab every object/variable can be in workspace even over several scripts. Not directly possible in Python.