As we start to get python code in Gromacs, it might be good to document style standards. I would propose a modified version of the Google python style guides (I may be biased because I used them at Google for a couple years and continue to use them in my own group, but I think they give good readable code). There's also a pylint template we can use.

https://google-styleguide.googlecode.com/svn/trunk/pyguide.html

Thoughts?

Erik suggested that we keep naming conventions consistent with the C/C++ code. I'm fine with that (we might want to revisit if we get a substantial amount of python code, but especially when it's relatively small in quantity it makes sense to prioritize codebase consistency over language idiom).

If this sounds reasonable, I'm happy to start working on a patch. If people want to discuss more here beforehand, that's good too.

Related issues:
- Related to GROMACS - Bug #2401: Python scripts use hardcoded #!/usr/bin/pytho... Closed
- Related to GROMACS - Feature #3032: Clean up dev-manual structure Feedback wanted

Associated revisions
- Revision fae19587 - 01/02/2020 02:53 PM - Eric Irrgang
  Add Flake8 linting to gmxapi tests.
  Add the flake8 package to requirements-test.txt and a run_flake8 docker entry point. This change does not require Python source checking, but facilitates it.
  Refs #1786
  Change-Id: l6a6c2d6012e4424d13d8ceeed0ea3696077042f6f

History

#1 - 07/28/2015 09:37 AM - Teemu Murtola
I won't mind if we have such standards, and the developer guide is the best place for this. Should this then apply also to all the small (and not-so-small) Python scripts we have (e.g., in releng)?

The most important thing to standardize, though, would be the expectations on what Python versions we want to support. If both 2 and 3, how do we deal with that, and how if someone wants to use features from newer Python for some limited-audience script?

#2 - 07/30/2015 01:14 AM - Peter Kasson
I think it should apply to all Python scripts going forward, and as with all legacy code we try to do clean-up as feasible.

2.7 versus 3.4 is an interesting question. I'd personally favor 2.7, but there's certainly an argument for 3.4 as the more forward-looking choice. There's also the reverse argument that well-written 2.7 code that uses modern Python idioms will have a trivial migration path to 3.4.

As with all our code, commits to master shouldn't introduce new external dependencies (such as newer features) unless specifically permitted in our dev standards, right?

#3 - 07/30/2015 07:11 AM - Teemu Murtola
Perhaps trivial, but not a no-op migration, since well-written code specifically targeting 2.7 likely uses methods such as itervalues() that no longer exist in 3.x. So unless explicitly managed, we are most likely stuck with just either 2.7 or 3.x until we completely drop the support for 2.7.

Also, there can be, but not necessarily should be, a difference between user-facing code (possibly even distinguishing an API and other scripts) and
All current scripts have been developed using 2.7, but a gmxtest.pl replacement that I have toyed around a bit with would benefit from subtest support in the Python unittest library, which is in 3.4 only. If we standardize on 2.7, is the only solution for such cases then to reimplement that standard library functionality?

#4 - 07/30/2015 05:45 PM - Peter Kasson

Hmm--so a few possibilities:

1. We choose python2.7 for everything and use a replacement for unittest subtests
2. We choose python3.4 for everything
3. All API's and user-facing code should be python 2.7-compatible (we could specify cross-compatible or just leave it as 2.7), and we allow engineering code to be 2.7 or 3.4.
4. All code could be either 2.7 or 3.4

If we go with #s 3 or 4, we should specify version. A lot of code starts to deprecate the shebang line, but we could A) require /usr/bin/python2 vs. /usr/bin/python3 in the shebang line or B) require use of the six library (which provides versioning support) either in all code or in a specified subset (code that uses python3-specific; code that uses python2-specific, ...)

#5 - 02/06/2018 05:23 PM - Mark Abraham

- Related to Bug #2401: Python scripts use hardcoded #!/usr/bin/python instead of #!/usr/bin/env python added

#6 - 12/26/2019 03:11 PM - Eric Irrgang

- Related to Feature #3032: Clean up dev-manual structure added

#7 - 12/26/2019 03:15 PM - Eric Irrgang

I would propose the following initial consolidated text, but I don’t see a particularly good place for it in the current dev manual structure. (See #3032)

Python code should be compatible with Python >=3.5 and should be :pep:`8` compliant.
Where additional guidance is necessary, refer to the Google

Refer to inline documentation
`examples <https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html>`__ from the Napoleon Sphinx extension that we use for Sphinx docs.

Inline Python documentation should comply with :pep:`257`
and follow Google docstring conventions, where possible.
Moderate use of Sphinx RST extended syntax is acceptable (see :pep:`287`) if it
improves readability of the automatically generated documentation without
hurting readability of the raw docstring.

Python code should pass a Flake8 linting test invoked with::

    python -m flake8 --ignore W503 --max-line-length 119

There should also be some sort of connection to the supported Python version for the repository infrastructure and for installed code with historic support constraints.