1 Python 2.7 vs. 3

Python 2.7 is the final version of the Python 2.x line – all future development work will focus on Python 3. It may seem strange to learn an "old" language. The reasons for using 2.7 are:

- There are more modules available for Python 2.7. While all of the core python modules are available for both Python 2.7 and 3, some of the more esoteric modules are either only available for 2.7 or have not been extensively tested in Python 3. Over time, many of these modules will be available for Python 3, but they aren't ready yet.
- The language changes relevant for numerical computing are very small – and these notes explicitly minimize these so that there should few changes needed to run against Python 3+ in the future (ideally none).
- Configuring and installing 2.7 is easier.
- Anaconda defaults to 2.7 and the selection of packages available for Python 3 is limited.

Learning Python 3 has some advantages:

- No need to update in the future.
- Some improved out-of-box behavior for numerical applications.

1.1 2.A Relevant Differences between Python 2.7 and 3

Most differences between Python 2.7 and 3 are not important for using Python for analysis.

print

print is a function used to display test in the console when running programs. In Python 2.7, `print` is a keyword which behaves differently from other functions. In Python 3, `print` behaves like most functions. The standard use in Python 2.7 is

```
print 'String to Print'
```

while in Python 3 the standard use is

```
print('String to Print')
```

which resembles calling a standard function. Python 2.7 contains a version of the Python 3 `print`, which can be used in any program by including

```
from __future__ import print_function
```

at the top of the file.

Division

- Python 3 changes the way integers are divided. In Python 2.7, the ratio of two integers was always an integer, and so results are truncated towards 0 if the result was fractional. For example, in Python 2.7, 9/5 is 1.
- Python 3 gracefully converts the result to a floating point number, and so in Python 3, 9/5 is 1.8. When working with numerical data, automatically converting ratios avoids some rare errors.
- Python 2.7 can use the Python 3 behavior by including

```
from __future__ import division
```

at the top of the program. We will assume that all relevant programs will include this statement.

`range` and `xrange`

It is often useful to generate a sequence of number for use when iterating over the some data. In Python 2.7, the best practice is to use the keyword `xrange` to do this, while in Python 3, this keyword has been renamed `range`. I will always use `xrange` and so it is necessary to replace `xrange` with `range` if using Python 3.