CS 132 – Geometric Algorithms
Getting Started with Python

Mark Crovella

In this course you will use `python` to complete assignments that require coding. `Python` is an exceptionally versatile language and it is a natural one for discussion of algorithms related to linear algebra.

This document contains notes that relate to the use of `python` in this course. You should read it and understand it fully before starting the homework that requires `python` coding.

Learning Python. If you have used `python` before, you can skip this section. You will need to know the basics of `python` to complete the homeworks in this course. Luckily, it does not take very long to master the basics. There are many ways to get up to speed, but some that I recommend are:

- Run through the first part of the Python tutorial at `codecademy.com`. This is at `https://www.codecademy.com/learn/learn-python-3`. You only need to complete up to and including the "loops" lesson. This should only take a few hours.
- Run through the first part of the Python tutorial at `http://www.learnpython.org/`. You only need to complete up to the "Functions" lesson.
- Check Youtube if that's what works best for you. There are lots of tutorials there too.

Installing Python. We will use python version 3. This is the same version used in CS 111. Python is available on all the lab machines. If you want to install it on your own machine, you need to make sure to install the `numpy` and `matplotlib` packages (these are very standard.) A good system available for performing the install, and managing the packages, is Anaconda: `https://anaconda.com`. Be sure to install the Python 3 version.

Using Python. Rather than interacting with the `python` interpreter directly, I recommend that you use *ipython,* which is a greatly enhanced interpreter that makes development very easy. `Ipython` is installed by Anaconda and is available on all the lab machines. To get started, run `ipython` and then type a question mark at the prompt for an introduction.

For example, once you start ipython you can use the %run command to execute a file, and then you can look at the output and related variables afterwards in the interpreter, which can help for debugging. You can execute code snippets in the interpreter to see if they are doing the right thing. This is *much* easier than using python in batch mode, e.g., "python myprogram.py".

Another option is "Jupyter notebook" which is becoming the standard for laboratory (reproducible) research work in python.

There is also an IDE called "Spyder" that is installed by Anaconda. You may find this the most convenient programming environment, although my personal preference is for a text editor + ipython.

Floating Point. Python has both integer and floating point types. In python 3, dividing two integers will yield a floating point value (which is what we will always want).

You will need to understand that floating-point calculations are not exact, because real numbers are not stored exactly when converted to floating-point representation. It's not a bad idea to read `http://en.wikipedia.org/wiki/IEEE_floating_point` if you haven't had CS210 yet.

To give an illustration of how this can affect your computations, consider this example. For two floating point numbers `a` and `b`, performing the computation `a/b` may not yield exactly the true value, but rather a value that is only *very close* to the true value. So, mathematically we would expect that `b * (a/b)` should yield exactly `a`, but computationally, `a/b` is not stored exactly, so this may not happen. So in particular, `a - (b * (a/b)))` may not be exactly zero (though it will be a very small number). This will be covered in lecture; you will need to keep this in mind throughout the course.