Pair Programming Online Practicals for IAML

Maria Wolters

September 18, 2015

iPython notebooks look like the most sensible and straightforward tool to me if it is important to you that students learn to document what they do, show how they got their results, and reflect on the results of their analyses. What follows assumes that this is indeed one of the key learning outcomes.

Throughout, we will assume a simple introductory lab where students calculate key descriptive statistics.

1 General Ideas

Rewrite Lab Sheets to Support Different Roles. Pair programming involves a coder and a note taker. In one lab, it makes sense to switch roles once or twice. To reinforce that the notetaker needs to do something, too, you could have the note taker write documentation, the pair could write two functions that build on each other, the note taker could test the function the coder has written, . . .

Should students need to work in groups of three, they will need adapted lab sheets.

Train Tools and Procedures First. It takes a while to get used to the division of labour, and to establish ways of working together. The coder needs to learn to think aloud while coding, so that the note taker knows what's going on, and the note taker needs to learn how to make sure they can follow what the coder is doing. A tightly scripted first lab sheet (Lab 0) would help, where students work in tandem with the tutor.

The first time students work together in pairs, the lab itself should be on the easy side, and students should be encouraged to chat and get to know each other a little first before they work on the content.

Decide How to Make Tutors Available. Tutors can either be online when pairs are working, and join their Skype/Google Hangout in case of problems, or they can review the iPython notebooks asynchronously. If a tutor works at a time when several pairs of students are working, they can maintain an Instant Messenger console where students can alert the tutor to problems, and the tutor can then join existing calls.

Example: Eve is tutoring a lab with two pairs, Alice and Bob and Carol and Dave. Alice and Dave are the coders, Bob and Carol are the note takers. The task is to calculate the mean and standard deviation for an array of variables stored in a csv file. Alice messages the tutor "Hi, I just had a look at the data sets and some idiot put NA into the second data file???". Dave messages the tutor "Hi, where's numpy on my Mac?" The tutor responds to Bob: "NA is a standard code that's used when a number missing. How would you propose to cope with it in your code?". To Dave: "Could you invite me to your call?"

2 Infrastructure

In order to match students, use employment status/schedule, time zone, operating system / infrastructure, and experience. Many of the students we take in initially will be working, and so will be fairly inflexible when it comes to scheduling their joint lab sessions. Tutor support should also be structured around the times when students are available, which may be outside normal working hours.

Matching operating systems matters. Some quirks and bugs, in particular when it comes to installation, will be OS specific, and students with the same OS can support each other better.

Since iPython works on a local host, students would either need to run iPython from a directory on a mounted University file system (afs style) or they would have to share code through svn.

The advantage of using a common file system is that there is no confusion about what the current version of the file is, whereas version control allows one person to edit the file while the other person has the previously committed version in their iPython Notebook browser. Then, after the edits are done, there can be a hand off process (commit, check out) which can be scripted into the lab sheet.

Example: Alice writes a function that takes a data set and computes the mean, standard deviation, minimum, and maximum of all integer and float columns. She tests it on a very simple test data set. Alice then commits the notebook with her function, and Bob loads it into his system. He uses the function on a larger data set that has been designed to test for common programming mistakes. Bob finds a problem—the data set contains NA in some cells, R speak for missing value. Alice and Bob discuss how to deal with missing values.

At the beginning, all students need is the video sharing app of their choice (Skype or Google Hangout). The way you organise these depends on whether you want to keep track of whether students show up to the labs, and whether you want to have a tutor available when students meet virtually to complete their tasks.

As IAML grows and brings in more funds, Informatics should consider moving to a good videoconferencing system such as WebEx (`http://webex.co.uk`) that is appropriately supported on Windows, Mac, and Linux. Blackboard Collaborate only supports Ubuntu from 9.10 upwards. There are issues with GoToMeeting,

which is another widely used tool.[1]. Blackboard Collaborate Ultra, which does not require Java but is completely HTML5-based, will be available soon through the University. I have no experience with it (just with Collaborate), and given that students will be working in very small groups, there are no reasons why we should favour this system over solutions that students are familiar with.

3 Excerpts From A Sample Lab

In this course, we will use the practice of pair programming. This means that one person codes, while the other watches to make sure that the coder doesn't make any mistakes. You no longer need separate code reviews; instead, the code is reviewed as it is written. This is a very powerful tool and efficient tool for working together.

Starting Off with iPython

[Explanation of the iPython interface here] **Coder:** Add a new cell to the notebook. Type `print(''Hello World'')`. Click on the Cell Menu and then on Run. Add a cell below the current cell. Change its type to Markdown. Type `This is my first iPython program.` Again, click on Run. What happens?

You can go back and edit cells at any time. Go back to the print statement, change what it says, and run it. What happens?

What happens at later stages in the notebook affects the context of all cells. Add a new cell, and type `a = np.array([[1,2],[3,4]])`. Try to execute it.

Now add another cell below the last cell, and type `import numpy as np`. Run it. Then run the previous cell again.

Save and check in the complete Notebook.

Notetaker: Take notes of what happens at any step. There is a step where we are asking the coder to make a mistake. Point it out ot the coder at the right step.

Switch over now

Coder: You used to be the note taker. Load the notebook into your iPython. Rearrange the cells to move the numpy import to the top, import it, run the cell that defines the array, and then compute the mean of the whole array, the first axis, and the second axis using numpy's `mean` function. Hint: Use the `axis` parameter to specify the axis. Add a cell where you describe which dimension corresponds to which axis parameter.

Now add a new cell below the last cell, and define the array `a` again. This time, define a 3 x 3 array. Run the cell where you compute the means again.

[1]The Linux requirements for Collaborate are here (one of the best examples) `http://ischool.sjsu.edu/current-students/technology-support/blackboard-collaborate/conferencing/faqs#q`, for WebEx: `https://www.webex.co.uk/support/support-system-requirements.html` and a good summary of the only known workaround for GoToMeeting: `http://www.itsprite.com/linux-how-to-use-gotomeeting-in-linux/`

Notetaker: You used to be the coder. Look up the numpy function required while the coder sets everything up, be prepared to help the coder. Watch out for any problems.