Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference

Computational Notebooks for AI Education

Keith J. O'Hara
Computer Science Program
Bard College
Annandale-on-Hudson, NY
kohara@bard.edu

Douglas Blank
Computer Science Department
Bryn Mawr College
Bryn Mawr, PA
dblank@brynmawr.edu

James Marshall
Computer Science Department
Sarah Lawrence College
Bronxville, NY
jmarshall@slc.edu

Abstract

Computational notebooks are documents that serve dual purposes: they serve as an archive format containing code, text, images and equations; but they can also be run like computer programs. This paper explores the use of these new computational notebooks to teach AI and introduces tools that we have developed — ICalico and Calysto — to facilitate that use. Not only do these new tools broaden the languages and contexts available to students exploring notebook-based AI computing, but they offer a new mode of teaching and learning for the AI classroom.

A computational notebook is a document that can be read like a journal paper and run like a computer program. Although the idea is not new, the computational notebook approach has just recently begun to be widely adopted by computer science educators. For example, Peter Norvig recently shared notebooks on the web presenting topics in Artificial Intelligence such as the Traveling Salesperson Problem (TSP, see Figure 1). A new computational notebook system, called Jupyter (Pérez and Granger, 2007), is being developed and appears to be an excellent medium for teaching many topics, especially AI. This paper explores the use of these new computational notebooks to teach AI and introduces tools that we have developed — ICalico and Calysto — to facilitate that use. Not only do these new tools broaden the languages and contexts available to students exploring notebook-based AI computing, but they offer a new mode of teaching and learning for the AI classroom.

Jupyter Computational Notebooks

This paper focuses on the use of the Jupyter computational notebook project. The Jupyter system itself evolved from the IPython project (Pérez and Granger, 2007). Originally, IPython was just a better console-based read-eval-print loop for Python—it had many conveniences for programming including command-line history, command completion, and a set of built-in macros called "magics." However, in the last few years, IPython has evolved into a vast client-server architecture for running Python programs. Recently IPython was expanded to allow any language to be used, and the Jupyter project was born. Jupyter has three language-agnostic clients, or frontends, in which users can enter and edit code and text. There is a plain-text frontend, called "console" and a graphical frontend, called "qtconsole". However, the frontend of interest to us is the web-based frontend, called "notebook". The notebook client allows users to combine code, text, images, videos and mathematical equations (via LaTeX), in a web browser. In addition, graphs and plots produced by code appear directly in the browser. Finally, not only do Jupyter notebooks provide the live, interactive interface for a running programming language, but can also serve as a static, archived view of the document. In that manner, a notebook can be rendered in a manner that allows it to be viewed by anyone (see Figure 1). This makes notebooks serve a dual-purpose: they are the direct interface to the computational engine, and also serve as a well-formatted archive of that computation.

Although Jupyter has not been released as of this writing (January 2015) it is slated to be released in the next few weeks. We have been developing, exploring, and teaching with a pre-released version of Jupyter, and find it to be a useful tool in the classroom. To facilitate its use in courses such as AI, we have developed additional tools. First, we have expanded the list of languages that can be used to include Prolog, Java, and Scheme (among others). Secondly, we have added additional features to these languages to make their use in the AI classroom especially effective. For example, we have added non-deterministic backtracking to Scheme. Finally, we have developed libraries for integration into projects such as ROS, the Robot Operating System. These extensions will be discussed below.

Because Jupyter's notebook interfaces with a programming language through a web browser, one can run programs over the web. The Jupyterhub[1] project does exactly that: it allows users to login, create notebooks, and execute code on a remote server. With some caveats (discussed below), this is a very useful manner of managing classroom materials. For example, it allows students to begin working immediately without having to install anything. In addition, getting all of the packages installed for running AI

[1] https://github.com/jupyter/jupyterhub

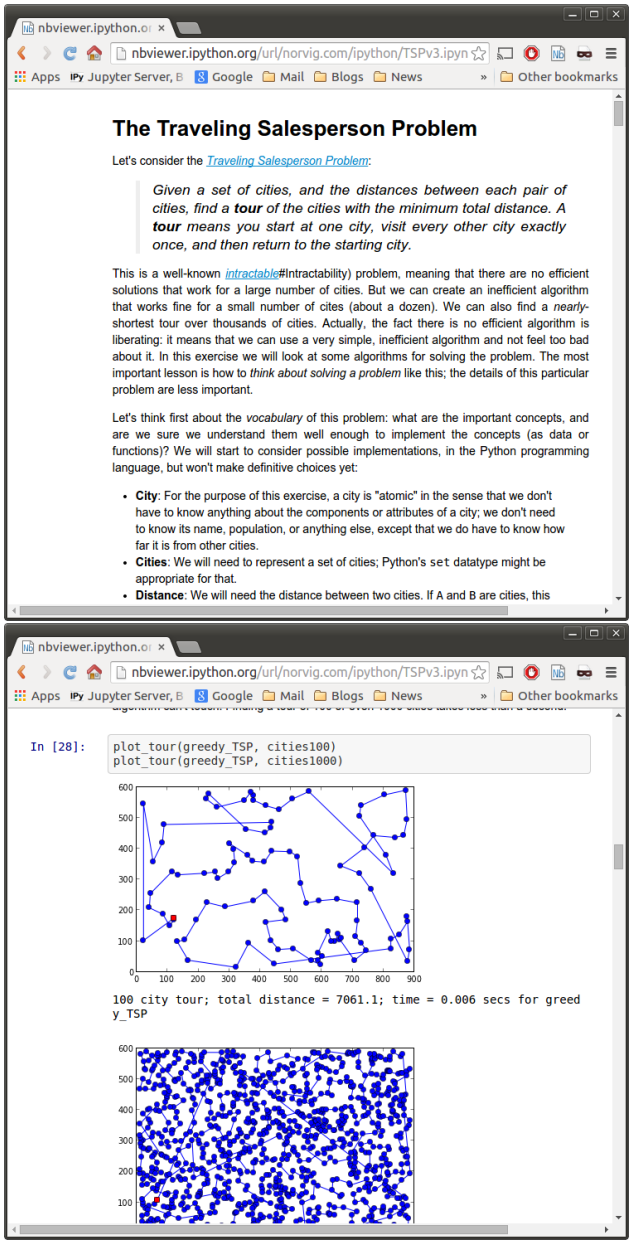


Figure 1: Peter Norvig's TSP exploration using a computational notebook. The document can be downloaded and run by anyone using the Jupyter system. http://nbviewer.ipython.org/url/norvig.com/ipython/TSPv3.ipynb

experiments can be tricky. Having a server-based solution means that the system can be set up once for all students to use. Although it does require a server, the requirements and set up are fairly minimal. However, for classes with a large number of students, computational resources may be a concern. If the computational requirements are minimal, one could expect a standard desktop computer could support many dozens of students. One could also consider having students use a system like SageMathCloud (Stein, 2014); however, you can't control a student's environment (they would need to install and configure all required libraries), and there could be extensive lag in processing.

We have explored four uses for these computational notebooks in the AI classroom: 1) Lectures/Discussions; 2) Readings/Flipped classroom; 3) Homework; and 4) Exams. Using notebooks (instead of presentation software) with a digital projector can be quite effective for lectures and discussions. There is a fairly low bar for this use: one simply needs to create the notebook as one works, and use it as the basis for a lecture/discussion. There are three options for lectures: use a static version of the notebook (like those shown in Figure 1); use a notebook connected to a live language backend with outputs precomputed; or use a live notebook with no output data shown. One can also export the notebook as an HTML slideshow. Exporting to HTML has two options: shown as a static, pre-rendered HTML page; or a slideshow connected to a live computational engine (called a "kernel"). These slideshows have all of the glitter of modern presentation software, such as animated transitions. However, one can also run code in them as well. Ensuring that lectures are exactly reproducible reduces any discrepancies between old-style lectures and code, making an easy path for the student to duplicate the demonstration.

Because the notebook is also executable, using well-written notebooks as out-of-classroom readings fits in very well with the notion of the "flipped classroom" (Bergmann and Sams, 2012). The flipped classroom is a teaching philosophy that replaces lectures with classroom discussions, moving the lecture material to be consumed outside of the classroom. Typically, outside classroom activities include prepared videos, or reading chapters from a textbook. In-class time often revolves around discussions or hands-on activities. We have found that computational notebooks can also be used as material for the outside classroom activity.

Requiring students to create notebooks for homework is a useful method to instill *literate computing*. Literate computing is a term coined by Fernando Pérez (2013). We take literate computing to be substantially different from its uncle, "literate programming" (Knuth, 1992). The focus of literate programming is to document a program. In this manner, it is an inward-facing document, designed to explain itself. On the other hand, literate computing is meant to focus on the computational goals, rather than on the specific details of the program. The goal of literate computing is not to explain the workings of a program to programmers, but to explain a computational problem to a wide audience. Literate computing does not subsume literate programming — it is something altogether different. Thus the goals of literate computing align very well with the goals of student writing in the college environment.

Using notebooks as the method for examinations has had mixed results, and warrants further study. Like any on-line computational exam, students can get bogged down on the details of any single problem. However, if one is in an environment that supports "take home" exams, that can alleviate in-class limited time pressures. Even allowing for that, students have reported that they often work in a non-linear fashion, and it can be hard to keep track of which problems on a test have been completed, and which ones are left to

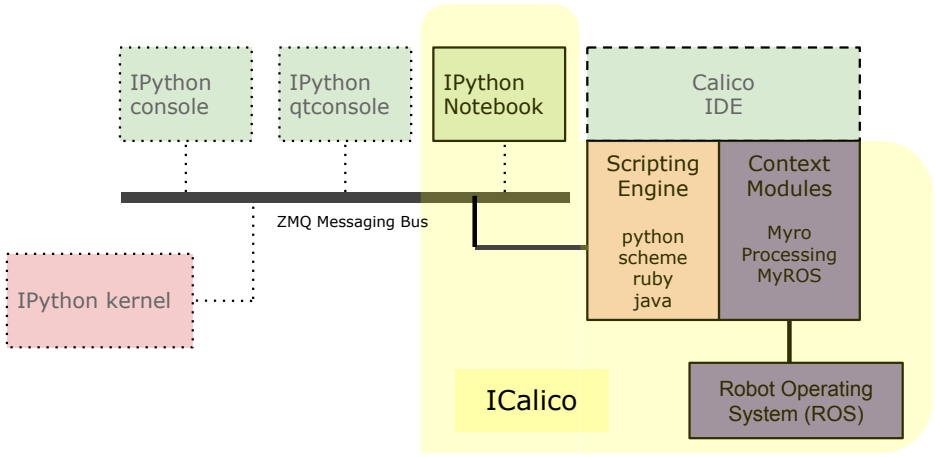


Figure 2: The Architecture of ICalico: integrating Calico, Robot Operating System (ROS), and IPython/Jupyter.

complete. Automatic grading of code is a project under development[2], and could be used at least for homework assignments.

ICalico and Calysto

Calico is a .NET/Mono-based software framework that includes a stand-alone Integrated Development Environment (IDE) for writing programs in a number of programming languages using a variety of libraries (Blank et al. 2012). It is designed to support the beginning student, provide appropriate scaffolding as students develop computational skills, continue to support them as they gain more experience, and provide assistance to teachers to help make them more effective. Calico is designed to support a smooth continuum from beginner to expert. ICalico makes the Calico languages and libraries available to Jupyter. The ICalico architecture is visualized in Figure 2.

There are a variety of languages supported in ICalico, including Python, Java, Scheme, Logo, Basic, Ruby, F#, and Boo (similar to Python, but with types and macros). All of these languages are treated as dynamic languages, allowing students to interactively enter code snippets and incrementally develop programs. In addition to a particular language's standard libraries, ICalico also contains a number of additional language-agnostic libraries. For example, ICalico comes with a rich library for exploring introductory robots, called Myro. The Myro library allows students to control a real or simulated robot, take pictures, do image processing, make the robot speak, go through a maze, draw a picture, etc. (However, to use a real robot requires a student to run the notebook locally, rather than on a server.) ICalico also contains libraries for making artwork, creating graphics games and physics simulations, exploring GIS, developing distributed systems, connecting to the Arduino (Banzi, 2008), accessing the Kinect, and many other libraries.

[2] https://github.com/jupyter/nbgrader

Building upon the Myro module, ICalico includes a prototype module for the Robot Operating System[3] (ROS) called MyROS (described below). MyROS provides two main functions: it makes ROS available to all of the ICalico languages, and it handles much of the complexity for starting ROS-based servers. The effect is that users gain access to sophisticated robotics (real and simulated) and much prepackaged, sophisticated functionality (such as face recognition) without having to wrestle with irrelevant system details.

The Calysto project takes all the Python-based components from Calico and makes them available without the .NET/Mono framework. This allows Calysto-based languages and libraries to use standard CPython-based libraries, such as numpy and matplotlib. To make our Calysto-based languages have the same kind of utility that Python has in IPython, we have worked with other open source developers to create *metakernel*[4]. Metakernel adds magics, command completion, command-history, shell access, parallel processing support, and more to Calysto-based languages. Thus, students can use parallel processing with Calysto Scheme and Calysto Prolog. The following sections describe how one can explore topics in AI through Jupyter notebooks.

Non-Determinism and Search

As part of the Calico project, we developed a version of the Scheme programming language. Our Scheme is written in Scheme, and can be converted to run on either .NET/Mono or Python. The .NET/Mono version (Calico Scheme) can use native Common Language Runtime (i.e., Windows) libraries, whereas the Python version (Calysto Scheme) can use native CPython libraries (e.g., numpy, matplotlib). One interesting aspect of our Scheme is that it has a nondeterministic choose function built into the language (see

[3] http://www.ros.org/

[4] https://github.com/blink1073/metakernel

"Baker, Cooper, Fletcher, Miller, and Smith live on different floors of an apartment house that contains only five floors. Baker does not live on the top floor. Cooper does not live on the bottom floor. Fletcher does not live on either the top or the bottom floor. Miller lives on a higher floor than does Cooper. Smith does not live on a floor adjacent to Fletcher's. Fletcher does not live on a floor adjacent to Cooper's. Where does everyone live?"

```
(define floors
  (lambda ()
    (let ((baker (choose 1 2 3 4 5))
          (cooper (choose 1 2 3 4 5))
          (fletcher (choose 1 2 3 4 5))
          (miller (choose 1 2 3 4 5))
          (smith (choose 1 2 3 4 5)))
      (require (not (= baker 5)))
      (require (not (= cooper 1)))
      (require (not (= fletcher 5)))
      (require (not (= fletcher 1)))
      (require (> miller cooper))
      (require (not (= (abs (- smith fletcher)) 1)))
      (require (not (= (abs (- fletcher cooper)) 1)))
      `((baker ,baker) (cooper ,cooper) (fletcher ,fletcher)
        (miller ,miller) (smith ,smith)))))
```

Figure 3: To the right is the Scheme code necessary to solve the stated constraint satisfaction problem (Abelson et al. 1996).

Figure 3). In addition, we have incorporated a version of Prolog (written in Python) into both ICalico and Calysto. Both Scheme's choose function and Prolog allow students to easily explore logic-based, declarative programming.

Machine Learning

Because Machine Learning (ML) is heavily based on statistics and many statisticians have already adopted notebook-based computing, many ML examples are widely available (see Figure 5). As can be seen, the visualizations depicting ML analysis makes the notebook a nice vehicle for relaying ML concepts and processes to the student.

Robotics and Computer Vision

Although Myro can control the Scribbler, Finch, Hummingbird, and Arduino robots, Calico includes a prototype interface to the Robot Operating System (ROS). The Robot Operating System (Quigley et al., 2009) is an open source environment for building end-to-end robot applications. ROS is rapidly becoming the standard in the robotics research and development community. ROS abstracts away the details of interfacing with hardware and provides many standard algorithms. Our ROS module allows students to easily access and configure ROS nodes from Windows, Mac, and Linux. Traditionally, development of ROS programs meant students must work inside Linux using C++, Common Lisp or Python. In addition to a wide variety of robotics hardware platforms, ROS provides higher-level software capabilities, like computer vision routines, robot localization and navigation, and a 3D point-cloud library. ROS can be used at three different levels within ICalico depending on the background of the user and the intended application.

ROS robots can be made accessible via the Myro interface. ROS is completely hidden at this level. Students in a CS1 class that would like to explore computing via simple ground robots and commercial quadcopters like the ARDrone (see Figure 6). ROS robots can be accessed via ROSs provided Python wrappers. However, this option only works on Linux. Although this would not use ICalico, some users might want to go the route of using only Jupyter and ROS. The computational notebooks would still be useful in this case even if the Calico back-end is not used. Or somewhere in the middle, ROS robots be accessed via MyROS, a full-fledged visible ROS module. Although more complex than Myro, MyROS still provides the user with a wide variety of programming languages and utilities for configuring and maintaining ROS nodes. The various possibilities are visualized in Figure 4.

Calico was used in a coures called (De-)Coding the Drone. In this class, students interfaced three different types of robots using the same Python API. The students began with Arduino, then moved on to the Scribbler, and finally to the Parrot ARDrone. The students used the Arduino to build a simple streetlight and telegraph. Using the Scribbler and the ARDrone the students explored autonomous and teleoperated operation. Other non-robotics assignments included an Eliza-based chatbot and a graphical avatar. Students explored all these varied contexts using the same Python-based environment.

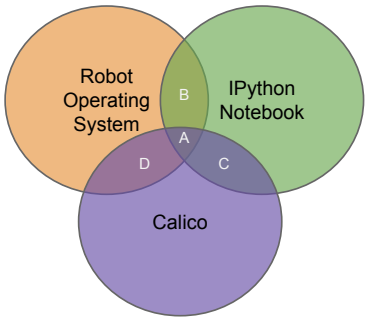
Notebooks are also well suited for computer vision applications, particularly as an interface to OpenCV. OpenCV, a C++ library for state-of-the-art computer vision, provides Java and Python APIs. Students in a 300-level class called The Computational Image used the notebooks to explore camera calibration[5] and augmented reality[6]. The online tutorials mix prose, mathematical foundations, images, and code and are fertile ground for students relying on the notebook-based approach. The students are able to recreate the tutorials step-by-step by writing a notebook using images and text to document milestones as they proceed.

Computational notebooks complement and supplement the static, read-only tutorials[7] already made available by the OpenCV and ROS communities. These tutorials already mix prose with images and code, and are generally very good. However, there are a number of issues with such tutorials: a) the tutorials are not executable they require the student

[5] http://docs.opencv.org/trunk/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration

[6] http://docs.opencv.org/trunk/doc/py_tutorials/py_calib3d/py_pose/py_pose.html#pose-estimation

[7] http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics



A The intersection of the three core technologies. Example: using computational notebooks to describe graph search as applied to motion planning in robotics. Various algorithms are described, compared, contrasted, and then implemented in Java, Python, or Scheme.

B IPython with ROS. Example: Students use CPython, taking advantage of scipy and BLAS. This configuration limits the student to working in Python on Linux, but allows the students to use many popular CPython libraries. Requires maintenance of many subsystems, which could be beyond many instructors' abilities.

C Calico backend with an IPython frontend. Example: a class taught using Java with no robots or simple robots, as Calico supports a variety of robots without ROS. Lacks access for exploring advanced robotics.

D Calico with ROS. Example: the Calico IDE provides a more traditional IDE, stepper, and other debugging support. Students are unable to explore the integrated literate computing approach for developing computational narratives and explanations.

Figure 4: Synergies between different components of ICalico.

to cut-and-paste to replicate; b) they take a long time to construct and format to look good and include media and output; c) the example code can become out-of-sync with the ROS codebase. The approach this paper puts forth is a way to more tightly couple code, documentation, and explanation in a way that improves understanding of concepts through code, and understanding of code through concepts. Moreover, rather than only ROS experts writing these tutorials, students can be tasked with writing as well as reading tutorials. By combining ROS with Calico we have the recipe for accessible robots: many hardware platforms, rich robotics system software, and support for many languages all with front-ends on the major three operating systems.

Conclusion

We believe that using Jupyter, ICalico, and Calysto in the AI classroom could have the following benefits:

- Combined with Jupyterhub, students can start quickly as there is nothing for them to install.
- Materials previously given by lectures can be made available by notebooks, ala the flipped classroom.
- Encourages the creation of visualizations to help explain complex topics.
- Emphasizes reproducible research and literate computing.
- Increases the availability of topics in AI through easy-to-use interfaces.

We believe that writing with combined text, code, and equations in the form of an executable, computational notebook is fundamentally different from current paradigms and worth exploring as a means of teaching advanced artificial intelligence topics. In this paper, we identified and explored the possible uses of notebooks in AI education. In future research we plan to explore their effectiveness.

References

M. Banzi. 2008. Getting Started with Arduino. O'Reilly Media/Make.

J. Bergmann and A. Sams. Flip Your Classroom: Reach Every Student in Every Class Every Day. 2012. International Society for Technology in Education.

D. Blank, J.S. Kay, J. Marshall, K.J. O'Hara, and M. Russo. 2012. "Calico: A Multi-Programming-Language. Multi-Context Framework Designed for Computer Science Education." ACM Technical Symposium on Computer Science Education (SIGCSE).

D. Knuth. *Literate Programming*. 1992. Stanford, California: Center for the Study of Language and Information.

L. Lamport. *LaTeX: A Document Preparation System*, 2/E. 1994. Addison-Wesley Professional.

F. Pérez, B.E. Granger. 2007. IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, http://ipython.org

F. Pérez. 2013. URL: http://blog.fperez.org/2013/04/literate-computing-and-computational.html

M. Quigley, K. Conley, B.P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. 2009. ROS: an open-source Robot Operating System, in ICRA Workshop on Open Source Software.

R Development Core Team. 2008. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

W. A. Stein et al. 2014. Sage Mathematics Software. URL: http://www.sagemath.org.

H. Abelson, G.J. Sussman, and J. Sussman. 1996. *Structure and Interpretation of Computer Programs*, 2nd ed.

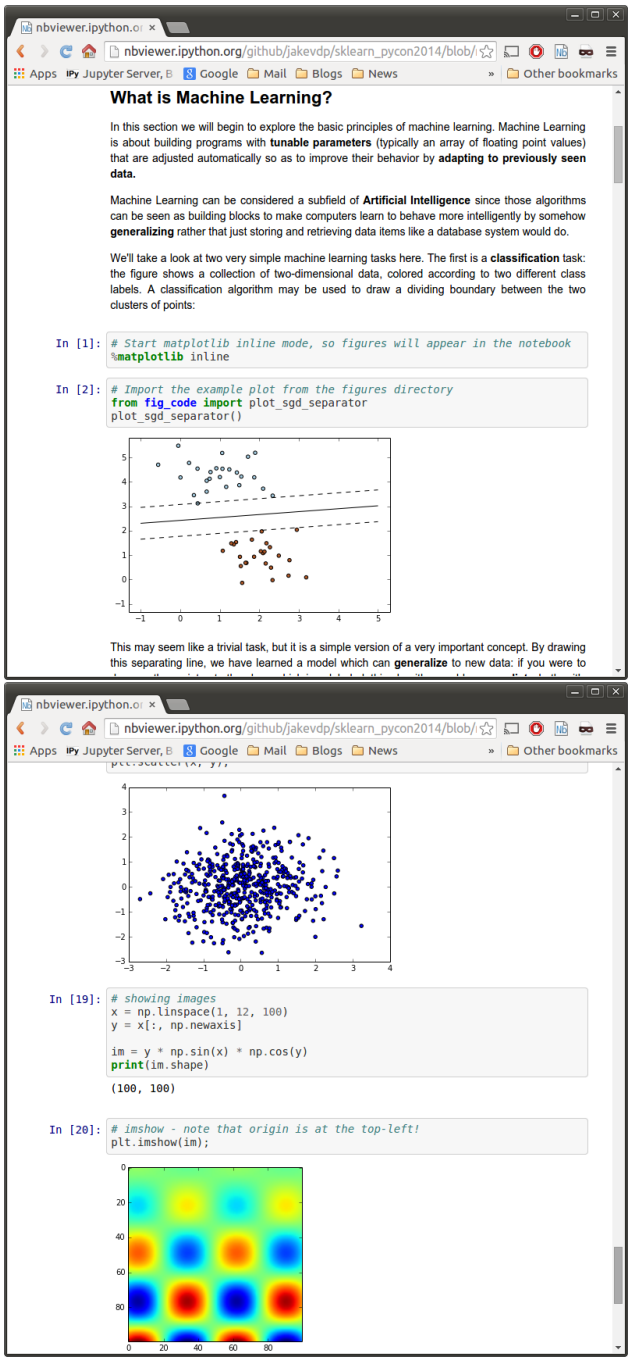


Figure 5: Examples of a Jupyter notebook on Machine Learning by Jake Vanderplas. Students can read the produced, static notebook. More importantly they can also download the notebook and re-run the experiments exactly. http://nbviewer.ipython.org/github/jakevdp/sklearn_pycon2014/blob/master/notebooks/01_basics.ipynb

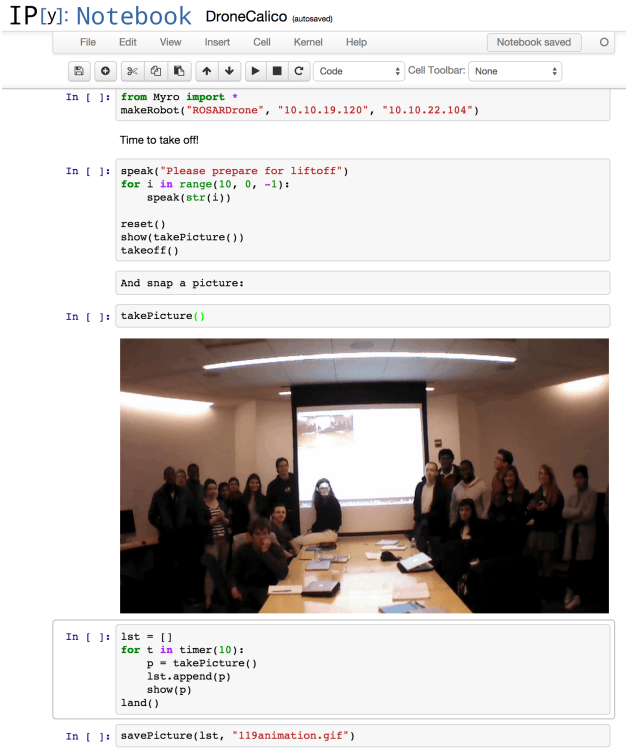


Figure 6: Flying a quadcopter drone using the ICalico prototype, ROS, and Calico. Creating still photos and short animated GIFs, both displayed in the browser inline with the text and code.