Final Project Report

Math 480A

Topic: Adding Line Numbers to the Sage Notebook

Authors: Alex Juarez
Sean Johnson
Priya Rao Chagaleti

Introduction

For software developers, one of the main concerns is to catch bugs in the code they have written. The industry average for the total number of bugs per thousand lines of code is about 15-50, depending on the application being developed. To catch these bugs, the most common approach is to either raise an exception (in order to check the pre-conditions) or handle the exception in an elegant way. However, most of the code written fails to handle exceptions in an elegant way, thus leaving it to the user defined code to handle the exceptions thrown. Given that the code could potentially be several thousands of lines, the best way to start debugging is to follow the stack upto the line which caused the error by referring to the line number of the last executed line of code. The Sage Notebook prints this line number, which is very convenient, but it fails to provide optional line numbers, thus making it very difficult to spot the bug. For example, if the code is a thousand lines long, and the bug is on line number 736, the Sage Notebook doesn't leave the coder any option but to run through the lines from the top, manually counting until line 736 is reached. This might be a highly time-consuming process, and counting may not be entirely error-free. This was the major motivation for us to implement optional line numbers, in order to make the process of debugging quicker and more error-free.

Another essential feature in most programming language editors is the ability to recognize keywords. Through the use of color coding, the coder can avoid bugs such as using keywords for variables or not maintaining the format of a function/method. The lack of this feature in the Sage Notebook makes writing code more troublesome than if it were written in another editor such as Notepad++. This motivated us to impliment CodeMirror into our redesigned coding cell, which highlights and color codes the keywords as well as function names. This color coding helps the user to check for the wrong usage of any keywords in the code.

To implement these features into the Sage notebook, we primarily used two resources, TinyMCE and CodeMirror. TinyMCE is a platform independent web-based Javascript/HTML editor released as

open source. TinyMCE has the ability to convert HTML TextArea fields or other HTML elements to editor instances. TinyMCE is very easy to integrate into other Content Management Systems.

CodeMirror is a JavaScript library that can be used to create a relatively pleasant editor interface for code-like content, such as computer programs, HTML markup, etc. If a mode has been written for the language you are editing, the code will be coloured, and the editor can optionally help you with indentation (although we were unable to implement this). Below is a description of our coding process.

Setting up the workspace (for Windows)

- Download the following tools:

- Latest Sage source code for Linux (http://sagemath.org/download-linux.html). Choose the closest source. If you have 2GB of RAM or more, get 64-bit (otherwise, get the 32-bit).
- VMware (www.vmware.com/products/player)
- Ubuntu (http://www.ubuntu.com/download/ubuntu/download)

- Install VMware, create a Virtual Machine for Ubuntu.

- Install Ubuntu. To get the apps to run Sage, run the following commands:

    sudo apt-get install built-essential

    sudo apt-get install gfortran

    sudo apt-get install m4

    sudo apt-get install mercurial

    sudo apt-get install open-vm-tools

- Drag and drop the Sage tarball into your Linux Virtual Machine. Right-click the ball to extract to Ubuntu.

- Navigate to the Sage folder in your terminal. Run the following command:

    Sudo ./sage

The process

- Examined the TinyMCE and CodeMirror source code, located in the sagenb data repository.

- Took an intallation of TinyMCE and created a mock-up of the line numbers application, including CodeMirror (for highlighting).

- Emplemented the mock-up to the existing cell code in the sagenb data repository.

- Created a Mercurial patch

Problems encountered

- No well-documentated way to develop for Sage in Windows.

- Getting Windows and Linux to communicate (to transfer files between them)

- There wasn't the right tools in Linux to run Sage

- No user name for Mercurial in Linux installation