Sage Tutorial

Justin C. Walker, Alex Ghitza, Mike Hansen, William Stein

March 10, 2009

The following notes provide some basic hints for using Sage from the command-line. You begin by typing 'sage':

```
$ sage[CR]
# starts the program
sage: ^D
$
# quits the program.  Alternatives are
sage: quit
$
# and
sage: exit
$
```

Sage understands the **EMACS** key bindings (because Sage uses IPython, and IPython comes equipped with GNU ReadLine support). In particular, the following are useful keystrokes. The ↑ means CTRL key/modifier (e.g., ↑A is CTRL-A). META is the "meta" or "alt" key (OPTION, on Macs).

- ↑A, ↑E: Beginning and End of the current line
- ↑P, ↑N: previous, next line in history
- ↑K: kill to end of line (from cursor)
- ↑U: kill to beginning of line (from cursor)
- ↑Y: "yank" text from kill buffer
- META-F, META-B: move to next, previous "word" in current line
- META-D: kill from cursor position to end of word
- META-B: kill from cursor position to beginning of word

Any keystroke that "kills" text puts the text into the kill buffer (so ↑Y retrieves it).

Sage has a lot of content. There is documentation available

- on line (sagemath.org: PDF, HTML)
- in the package you downloaded and installed (HTML)
- in the program itself
  TAB, .TAB, ?, ??
- in the mailing lists (search on Google Groups pages)

Given a Sage can load it for you in several ways:

```
sage: load /path/to/foo.sage
```

In this case, Sage reads the file in, parses it, and adds the content to its internal list of "known things". Anything defined in the file is then available for use. If the file

`/tmp/foo.sage`

contains

```
x = 47
y = sin(2.0)
```

then if the variable "y" is not defined, it will be after the load.

If you attach the file:

```
sage: attach /path/to/foo.sage
```

then Sage will monitor the file, reloading and parsing it whenever a change is detected. This can be disconcerting if you make a change, save an interim version with syntax errors, and hit **CR**: Sage detects the error on reload and reports it.

In general, if there is a syntactic error in your code, the parser will find it and complain, giving you a line number where it thinks the problem is. This is typically the last line of what can be a long "traceback", which tells you where in the computation you were.

If there's an undefined symbol, you probably won't find out until you run the code containing that symbol.

If there's a logic error, well, that is called debugging.

You can also load the file so that Sage tracks the file and reloads it whenever you make changes:

```
sage: attach /path/to/file
```

Once the file changes, merely typing **CR** will cause the system to reload (and reparse) the file. This can create a situation where just typing **CR** will cause a long error message to get blatted to the screen.

Note that, when you load/attach a file containing a sequence of commands, no output will show up unless those commands explicitly print something. If the file "/tmp/x.sage" contains this:

```
Zx.<x> = PolynomialRing(Integers())
f = x^2-15
K.<a> = NumberField(f); K
OK = K.maximal_order(); OK
A = 5*OK; A
FF = A.prime_factors(); FF
P = FF[0]
P.is_prime()
P.ramification_index()
P.residue_class_degree()
```

then you see this with a load or attach:

```
sage: load /tmp/x.sage
sage
```

The 'iload' command interactively loads and executes one line at a time from the file. The line is displayed and executed when you type **CR**.