How to get started with developing Sage

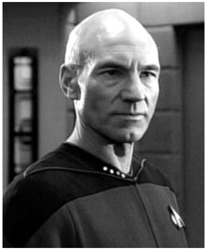
Sage Days 16, Barcelona, June 23, 2009

How to start developing Sage

1. get tenure
2. ???
3. profit

Outline

Outline



Enhancing Sage for your own research

- There is some bug which just plainly annoys you . . .
- There is some function which just isn't documented properly and you keep getting it wrong . . .
- There is this functionality (which would be easy to add) but no developer bothered . . .

"It's easy, implement it and send me a patch."

– William Stein

Enhancing Sage for your own research

- There is some bug which just plainly annoys you . . .
- There is some function which just isn't documented properly and you keep getting it wrong . . .
- There is this functionality (which would be easy to add) but no developer bothered . . .

"It's easy, implement it and send me a patch."

– William Stein

Enhancing Sage for your own research

- There is some bug which just plainly annoys you . . .
- There is some function which just isn't documented properly and you keep getting it wrong . . .
- There is this functionality (which would be easy to add) but no developer bothered . . .

"It's easy, implement it and send me a patch."

– William Stein

Enhancing Sage for your own research

- There is some bug which just plainly annoys you . . .
- There is some function which just isn't documented properly and you keep getting it wrong . . .
- There is this functionality (which would be easy to add) but no developer bothered . . .

"It's easy, implement it and send me a patch."

– William Stein

Enhancing Sage for your own research

- There is some bug which just plainly annoys you . . .
- There is some function which just isn't documented properly and you keep getting it wrong . . .
- There is this functionality (which would be easy to add) but no developer bothered . . .

"It's easy, implement it and send me a patch."

– William Stein

Writing your own non-trivial Sage programs

- there isn't that much of a difference between developing for the Sage core library and writing your own program.
- Most of the comments below apply to both.
- Please consider submitting your code to Sage if it implements functionality not in Sage yet (or better than Sage)

Using Sage as a frontend for your own library

- you wrote this amazing C/C++/whatever library and need to test it out
- you can either write tedious testcode or
- you can write a slim Sage interface which allows to test your library much more rigorously.
- you can use this interface to convert between your native format and many other systems (Pari, Magma, Mathematica, ...)

Let's face it

You are stuck at this workshop anyway so you might as well write some code while you are here.

Outline

Python

python & Cython

- Sage fundamentally depends on Python.
- Speaking Python is a requirement for properly using and developing Sage.
- Python is a very easy to learn language.
- Learning Python has benefits far beyond Sage, it is **widely** used.

`http://www.diveintopython.org/` and
Python in a Nutshell by Alex Martelli

Cython

Sage depends on Cython to provide

- a compiled fast language for low-level arithmetic and
- a language to easily interface with C/C++ code and libraries.

If you want to work on this level, it makes sense to learn some Cython:

`http://docs.cython.org/`

The Preparser

Sage commands get **preparsed**.

- `1/2` is `0` in Python, but `1/2` in Sage
- `P.<x,y> = GF(2)[]` is not valid Python, but valid in Sage

```
sage: preparse("1/2")
'Integer(1)/Integer(2)'
sage: preparse("P.<x> = ZZ[]")
"P = ZZ['x']; (x,) = P._first_ngens(1)"
sage: preparse("0.5")
"RealNumber('0.5')"
```

When writing code for the Sage Library you must write valid Python and you have to expect Python behaviour (e.g. `1/2 == 0`).

Components

- Sage comes in various SPKGs: Sage Packages.
- Sage 4.0.1 contains 99 such SPKGs like bzip2, MPIR, Pari, NTL, FLINT, Maxima, Singular etc.
- This way Sage is self contained and behaves reasonably similar across platforms.
- The code which ties all these packages together is the "**Sage Library**". Naturally, it comes in an SPKG.
- Each copy of Sage allows you to hack the Sage Library straight away – batteries included.

I will focus on modifying the Sage Library in this talk.

Directory structure

`$SAGE_ROOT`

- `local` where the SPKGs are installed to
  - `bin` executables go here (e.g. `Singular`)
  - `lib` e.g. shared libraries go here (e.g. `libsingular.so`)
- `devel/sage` the Sage Library
  - `doc` the reference manual, tutorial etc. sources
  - `sage` the code that makes things happen
  - `c_lib` some low-level code, can be ignored by most
- `spkgs` this is where the SPKGs are stored

Directory structure of the Sage library

Excerpt:

algebras	free, group, quaternion, steenrod . . .
combinat	very comprehensive combinatorics
graphs	all things graph theory
groups	group theory
interfaces	interfaces to other system (e.g. Magma)
libs	raw interfaces to C/C++ libraries
matrix	matrices over all kinds of fields
misc	a lot of useful utility functions!
modular	modular forms and symbols
plot	2d and 3d plotting
quadratic_forms	guess what.
rings	integers, rationals, finite fields, polynomials, *p*-adics
schemes	curves (e.g. elliptic, hyperelliptic)
server	the notebook server
structure	coercion and Sage parent–element infrastructure
symbolic	all things symbolic manipulation

Finding that function I

```
sage: search_src("Integer","create")
...
misc/preparser.py:We create a raw integer.
misc/preparser.py:first one computes a list of SAGE integers ...
monoids/free_monoid.py:        One can create a free monoid
...
combinat/sf/sfa.py:           integer n as its input and  ...
server/notebook/js.py:                                 string  ...
matrix/matrix2.pyx:        We create the zero matrix over...
misc/parser.pyx:variables) and how integer and floating ...
rings/integer.pyx:        You can create an integer from ...
rings/integer.pyx: A global  pool for performance when  ...
rings/integer.pyx:   if available, otherwise a new Integer ...
rings/integer_ring.pyx:To create an ``Integer``, coerce  ...
rings/integer_ring.pyx:    We can create integers from ...
...
```

Finding that function II

```
sage: search_def("rank")
algebras/free_algebra_quotient.py:    def rank(self):
combinat/choose_nk.py:    def rank(self, x):
...
combinat/subset.py:    def rank(self, sub):
misc/functional.py:def rank(x):
modules/free_module.py:    def rank(self):
modules/matrix_morphism.py:    def rank(self):
combinat/posets/hasse_diagram.py:    def rank(self,element=None):
...
combinat/words/alphabet.py:    def rank(self, letter):
libs/mwrank/interface.py:    def rank(self):
modular/abvar/abvar.py:    def rank(self):
...
modular/modsym/ambient.py:    def rank(self):
quadratic_forms/genera/genus.py:    def rank(self):
...
matrix/matrix0.pyx:    def rank(self):
matrix/matrix_integer_dense.pyx:    def rank(self):
matrix/matrix_mod2_dense.pyx:    def rank(self):
matrix/matrix_modn_dense.pyx:    def rank(self):
matrix/matrix_modn_sparse.pyx:    def rank(self, gauss=False):
matrix/matrix_rational_dense.pyx:    def rank(self):
...
```

Finding that function III

edit() tries hard to an editor at the right line in the right file $^{\text{TM}}$.

```
sage: edit(ZZ)
cdef class IntegerRing_class(PrincipalIdealDomain):
    r"""
    The ring of integers.

    In order to introduce the ring `\ZZ` of integers, we
    illustrate creation, calling a few functions, and
    working with its elements.
    ...
```

Finding that function IV

```
sage: ZZ??
Type:       IntegerRing_class
Base Class: <type 'sage.rings.integer_ring.IntegerRing_class'>
String Form:Integer Ring
Namespace:  Interactive
File: .../local/lib/python2.5/site-packages/sage/rings/integer_ring.so
Docstring:

        The ring of integers.

        In order to introduce the ring `ZZ` of integers, we
        illustrate creation, calling a few functions, and working with its
        elements.
```

- sage.rings.integer_ring.IntegerRing_class
- .../site-packages/sage/rings/integer_ring.so

$\longrightarrow$ $SAGE_ROOT/devel/sage/**sage/rings/integer_ring.pyx**

site-packages vs. devel

- on the last slide the source file for `ZZ` was given as `local/lib/python2.5/site-packages/sage/rings/integer_ring.so`
- this is where the code that is actually run is stored
- the sources are in `devel/sage/sage/rings/integer_ring.pyx`
- the command **sage -b** compiles and copies the sources for you

Always edit the files in the `devel` subdirectory and call **sage -b** to (compile and) copy them over.

Built-in documentation I

```
sage: ZZ?
...
        The ring of integers.

        In order to introduce the ring `ZZ` of integers, we
        illustrate creation, calling a few functions, and working with its
        elements.

        ::

            sage: Z = IntegerRing(); Z
            Integer Ring
            sage: Z.characteristic()
            0
            sage: Z.is_field()
            False

        We next illustrate basic arithmetic in `ZZ`::

            sage: a = Z(1234); b = Z(5678); print a, b
            1234 5678
...
```

Built-in documentation II

```
sage: help(ZZ)
 Help on IntegerRing_class object:

class IntegerRing_class(sage.rings.ring.PrincipalIdealDomain)
 |   File: sage/rings/integer_ring.pyx (starting at line 104)
 |
 |   The ring of integers.
 |
 |   In order to introduce the ring ‘\ZZ‘ of integers, we
 |   illustrate creation, calling a few functions, and working with its
 |   elements.
 |
 |   ::
 |
 |       sage: Z = IntegerRing(); Z
 |       Integer Ring
 |       sage: Z.characteristic()
 |       0
 |       sage: Z.is_field()
 |       False
...
```

Source code

```
sage: ZZ??
...
   def __init__(self):
        ParentWithGens.__init__(self, self, ('x',), normalize=False)
        self._populate_coercion_lists_(element_constructor=integer.Integer,
                                       init_no_parent=True,
                                       convert_method_name='_integer_')

    def __cinit__(self):
        # This is here because very old pickled integers ...
        global number_of_integer_rings
        if type(self) is IntegerRing_class:
            if number_of_integer_rings > 0:
              self._populate_coercion_lists_(\
                     element_constructor=integer.Integer, \
                     init_no_parent=True, \
                     convert_method_name='_integer_')
            number_of_integer_rings += 1

    def __reduce__(self):
        """
        TESTS::

            sage: loads(dumps(ZZ)) is ZZ
            True
        """
        return IntegerRing, ()

...
```

Writing documentation I

- Every function or class added to Sage **must** have documentation of its functionality and inputs.
- Sage docstrings are formated using the ReStructuredText.

Build the reference manual by typing `sage -b` first and then `sage -docbuild reference html` and check

- that it produces no errors and
- that the HTML looks okay.

Writing documentation II

Examples

- `*foo*`: *foo*
- `**foo**`: **foo**
- ``` ``x^i`` ```: `verbatim environment`
- `` `x^i` ``: LaTeX math mode x^i
- Any line ending with `::` means that the following indented things are a literal block (usually `sage:` commands)

See

`http://www.sagemath.org/doc/developer/sage_manuals.html`

for more details.

Running tests I

- Every new function included with Sage **must** have doctests.
- doctests are examples on how to use a function and are run on a regular basis to test for regressions.

```
sage: e.log?
EXAMPLES::

    sage: Integer(124).log(5)
    sage: Integer(125).log(5)
    3
    sage: Integer(125).log(5,prec=53)
    3.00000000000000
    sage: log(Integer(125))
    log(125)

For extremely large numbers, this works::

    sage: x = 3^100000
```

For tests which are not end-user friendly use `TESTS::`.

Running tests II

- You can run doctests on a single file as `sage -t filename_or_directory`.
- You can run doctests in parallel as `sage -tp num_threads filename_or_directory`
- You can and should also add and run doctests on your private code to test for regressions in Sage and/or your code.
- Before submitting a patch to Trac you should run doctests on the complete Sage tree (`make test`).
- If your computer is too slow for this, ask William about an account on `http://sage.math.washington.edu`.

Outline

Mercurial

Mercurial is the source control system that is used with Sage.

See

`http://www.selenic.com/mercurial/`

for full documentation on Mercurial.

Batteries included

All of the Mercurial repositories related to Sage are included with Sage. Thus the complete change history and setup for doing development is available in your copy of Sage.

Before using Mercurial, make sure to define your username so the patches you make are identified as yours. Make a file `~/.hgrc` in your home directory like this one:

```
[ui]
username = John Doe <doe@example.com>

[extensions]
hgext.mq =
```

Running Hg

There are several ways to run Mercurial:

- from the command line, run `sage -hg` (`Hg` is the chemical symbol for mercury),
- or from within Sage, run `hg_sage`. Most of the examples below use the second method.

Before you modify Sage library files, you might want to create a copy of the Sage library in which to work.

Do this by typing `sage -clone myver`, for example; then Sage will use Mercurial to clone the current repository and call the result `myver`.

You can switch between copies by `sage -b main` and `sage -b myver`.

Building changed source files

Once you have copied the library to a new branch `myver` and edited some files there, you should build the Sage library to incorporate those changes: type `sage -b myver`, or just `sage -b` if the branch `myver` is already the current branch: that is, if `SAGE_ROOT/devel/sage` links to `SAGE_ROOT/devel/sage-myver`.

Note that `devel/sage` is a symlink to whatever branch/clone is currently "active."

You can also type `sage -br myver` to build the library and then to immediately run Sage.

How to revert changes

- If you did `sage -clone myver` you can simply `sage -b main` to return to the upstream version of Sage
- You can also enter `hg_sage.revert()` to undo uncommitted changes.
- You can use `hg_sage.update()` to return to a previous revision to "undo" committed changes.

Preparing patches I

If you want to submit your changes to the Sage development team for refereeing (and inclusion into Sage if the referee's report is positive), you should produce patch files.
To do this:

- Type `hg_sage.status()` and `hg_sage.diff()` to see exactly what you've done (you can pass options to `diff` to see information about certain files).
- If you've added new files, not just edited existing ones, type `hg_sage.add([filenames])` to add those new files to your repository.

Preparing patches II

- Commit your changes by typing `hg_sage.commit([optional filenames])` to commit the changes in files to the repository – if no filenames are given, all files are committed. First the output of `hg diff` is displayed: look at it or just enter `q`. Then you are dumped into an editor to type a brief comment on the changes. The default editor is vi, so type `i`, write some meaningful one line description, hit `Escape` and type `:wq`.
  (In bash, to make emacs the default editor, type `export EDITOR=emacs`.)

Preparing patches III

- Now create a patch file using `hg_sage.export(...)`. This command needs a revision number (or list of revision numbers) as an argument; use `hg_sage.log()` to see these numbers. An optional second argument to `hg_sage.export(...)` is a filename for the patch; the default is `(changeset_revision_number).patch`.
- Then post your patch on the Sage Trac server (see below).

Other stuff

Finally, if you want to apply a patch file (perhaps you've downloaded a patch from the Trac server for review), use the command `hg_sage.patch('filename')`.

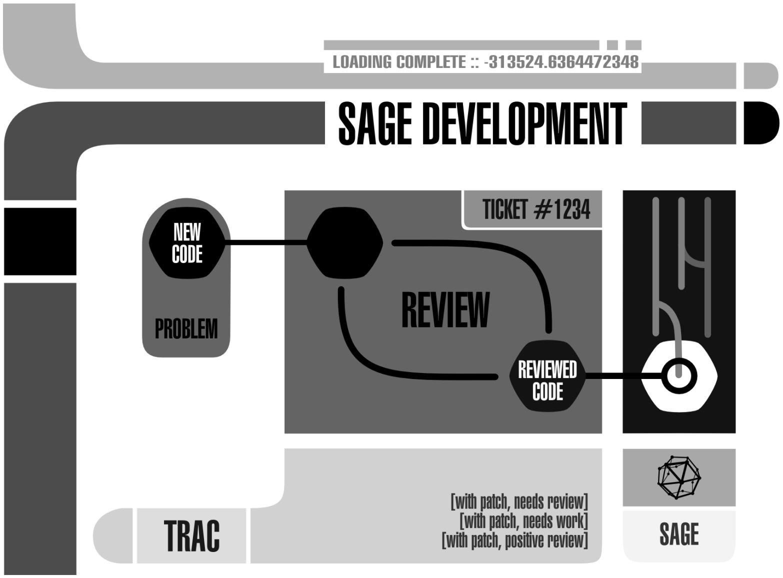
Most Sage developers seem to use a Mercurial extension called Mercurial Queues these days to manage their patches. See

```
http://www.selenic.com/mercurial/wiki/MqExtension
     http://wiki.sagemath.org/MercurialQueues
```

for details.

Outline

How Alice gets code into Sage I



How Alice gets code into Sage II

1. Alice writes some awesome code, opens a trac ticket at `http://trac.sagemath.org` and attaches her patch. The subject line of the ticket is now: `[with patch, needs review] ...`
2. Bob reviews Alice's patch and finds an issue. The subject line of the ticket is now: `[with patch, needs work] ...`
3. Alice fixes the issues, Bob checks the changes and decides it is okay now: `[with patch, positive review]`.
4. Eve is acting release manager for Sage X.Y.Z and applies Alice's patch to her tree. If everything is fine she closes the ticket, otherwise the ticket gets `[with patch, needs work]`.

Requesting a Trac Account

- To prevent Spam, we have had to disable anonymous creation and editing of tickets.
- Please write an email to wstein@gmail.com and provide an account name and password.
- The account name should be non-silly, i.e. no first names, no leet-handles.

Opening Tickets

Tickets can either be opened to submit a patch or to request a bugfix or enhancement.

- Before opening a ticket, make sure that nobody else has opened a ticket about the same or closely related issue.
- It is better to open several specific tickets than one that is very broad.
- Be precise: If foo doesn't work on OSX, but is fine on Linux, mention that in the title. Also use the keyword option to make searches pick up the issue.

Finding someone to review your patch

- Components on Trac have defaults assigned, so hopefully that will take care of it already.
- If you know someone who can review your patches, it makes sense to ask directly.
- If you are fixing a bug in/adding documentation to `sage -hg annotate -u filename` is a good tool to identify who wrote that function.
- You can ask the release manager or on [sage-devel] to find someone to review your patch.

Reviewing Patches

- The code makes sense and reads okay.
- 100% doctests: All new code must be 100% doctested. **There is no way around this.**
- Bug fixes must be doctested: The patch that fixes an issue must also contain a doctest specifically to test the problem. This is not always possible, so this is not enforced in certain situations.
- Test the reference manual: `sage -docbuild reference html` must produce no errors
- Test the Sage Library: `make test` or `make ptest` (edit number of threads in makefile before using ptest!)

Outline

Asking Questions

Definitely do ask a lot of questions!

sage-devel	the main development list: 900 members, 1000 messages per month
sage-support	end-user support list: 1220 members, 800 messages per month
#sage-devel	on freenode: main development IRC channel, 30 members, medium activity, usually at least two ‘core’ developers around
real people	this might be a good moment for the Sage developers in the room to stand up and introduce themselves.

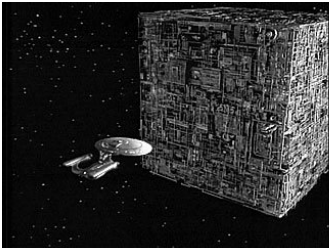
Netiquette

- Many Sage developers are volunteers and have day jobs or other obligations.
- Usually, questions are answered quickly if someone knows the answer straight away.
- If your question goes unanswered, it is probably just because people didn't get around to it yet. Don't hesitate to ask again!
- Keep it friendly though, we take pride in the fact that our mailing lists hardly see any insults and flame wars.

Other Resources

- The wiki (`http://wiki.sagemath.org`) a wonderful collection of useful outdated information.
- The developer guide (`http://www.sagemath.org/doc/developer/`) should contain everything to get started, if not **write it and send us a patch!** or let us know at least.
- The archive of [sage-devel] (`http://groups.google.com/group/sage-devel`) contains tons of useful information.

Thank You!



Building the Cube instead of reinventing the Warp drive!