Published on *Multilingual Online Translation* (http://www.molto-project.eu)

D6.2 Prototype of comanding CAS

Contract No.:	FP7-ICT-247914
Project full title:	MOLTO [1] - Multilingual Online Translation
Deliverable:	D6.2. Prototype of comanding CAS
Security (distribution level):	Public
Contractual date of delivery:	M23
Actual date of delivery:	February 2012
Type:	Prototype
Status & version:	Final (evolving document)
Author(s):	Jordi Saludes [2], Ares Ribó
Task responsible:	UPC [3]
Other contributors:	

Abstract

The present paper is the cover of deliverable D6.2 as of WP6 [4]. It gives description and installation instructions for the executables included in this deliverable.

Dependencies

The following table describes whats is needed in order to use the executables. In all case you'll need GF [5] and Sage [6].

`gfsage` is the simple dialog executable [7], `shell` denotes the component that allow using natural language inside Sage [8] and `shell-complete` is the same with auto-completion of commands.

Component	O. S.	Extra requirement	Spoken output	autocompletion
gfsage	Mac OS X, Linux Ubuntu	ghc, curl	OSX[1], Linux	yes
shell	all[2]	—		no
shell-complete	Linux	gf python bindings		yes

1. 10.7 ↩
2. Not tested on Windows, but in this case Sage runs inside of a Linux virtual box. ↩

Installation

Depending on your permission settings you might have to run some of these command as `sudo`. For all of these first you have to checkout the *Mathematics Grammar Library* from:

```
svn co svn://molto-project.eu/mgl
```

Be warned that develoopment will continue for some time in this HEAD branch. For a frozen version of it, checkout from:

```
svn co svn://molto-project.eu/tags/D6.2
```

You'll find detailed instructions for installing each executable in the following pages. For the moment, note that it is necessary to modify some files in your Sage [6] files, for these executables to run. Usually, we have to make these changes just once: The first time, the installation procedure will warn you about it:

```
Please add 'sage.nlgf' to /usr/local/sage-4.7.2/devel/sage/setup.py
```

Since ours is not a regular Sage [6] package, we must add a package reference manually by tweaking `setup.py` given above (Notice that yours may have a different path). This is a python file that Sage [6] reads to configure the system using the command `setup`. Please find it in the file, mine is at line 882 and looks like this:

```
code = setup(name = 'sage',
```

The *setup* command lists several items; Please locate *packages* (which is a python list) and add `'sage.nlgf'` (quotes included) among the other packages listed there. Python is picky about indentation and doesn't like to have spaces and tabs mixed. Please check that you're using the same spacing as the rest of the file.

The installation has been tested on Sage 4.7.1, 4.7.2 and 4.8

gfsage: a natural language interface for Sage

The goal of this work is to develop a command-line tool able to take commands in natural language and have them executed by Sage, a collection of Computer Algebra packages presented in a uniform way. We present here instructions on how to build the interface and examples of its intended use.

Building the executable

You'll need:

- ghc [9] with `cabal`, as in **Haskell platform**
- curl
- a way to call Sage [6] on a terminal (usually `sage` command. It assumes it's in your PATH)
- A POSIX system
- The source version of GF [5].

You can get this source version by:

```
cabal install gf
```

We can install the other dependencies too by:

```
cabal install json curl
```

Checkout the mathematics grammar library from:

```
svn co svn://molto-project.eu/mgl
```

This is the active branch. For the fixed one use:

```
svn co svn://molto-project.eu/tags/D6.2
```

Go into the **mgl/sage** directory (**D6.2/sage** if you're using the fixed branch) and make it:

```
cd mgl/sage
make
```

The first time you **make** it will fail, asking you to make modifications in the Sage [6] installation. Please refer to the installation page [10].

Now try to build `gfsage` again. All these build operations will ask Sage [6] to "rebuild" itself. Be warned that the first rebuild takes some time:

```
make
```

The system as been tested in Mac (OS X 10.7) and Linux (Ubuntu).

Usage

Run the tool as:

```
./gfsage english
```

giving the input language as argument. It will take some seconds to start the server. After that it will reply with some server information and will show the prompt:

```
sage>
```

You can then enter your query:

```
sage> compute the product of the octal number 12 and the binary number 100.
(3) 40
answer: it is 40 .
```

To show that a CAS is actually behind the scene, let's try something symbolic:

```
sage> compute the greatest common divisor of x and the product of x and y.
(4) x
answer: it is x .
```

and compare it with:

```
sage> compute the greatest common divisor of x and the sum of x and y.
(5) 1
answer: it is 1 .
```

Sage does the right thing in both cases, x and y being unbound numeric variables.

```
sage> compute the second iterated derivative of the cosine at pi.
(6) 1
answer: it is 1 .
```

Exiting

Exit the session by issuing CRTL+D: This way the server exits cleanly.

Just another example in a different language:

```
./gfsage spanish
Login into localhost at port 9000
Session ID is c1ef10dfd49e4fdb3214fa6d3a3b9c92
waiting... EmptyBlock 2
finished handshake. Session is c1ef10dfd49e4fdb3214fa6d3a3b9c92
sage> calcula la parte imaginaria  de la derivada de la exponencial en pi.
(4) 0
answer: es 0 .
```

More recent examples involving integer literals and integration:

```
sage> compute the sum of 1, 2, 3, 4 and 5.
(3) 15
answer: it is 15 .

sage> compute the summation of x when x ranges from 1 to 100.
(4) 5050
answer: it is 5050 .

sage> compute the integral of the cosine from 0 to the quotient of pi and 2.
waiting... (5) 1
answer: it is 1 .

sage> compute the integral of the function mapping x to the square root of x from 1 to 2.
(6) 4/3*sqrt(2) - 2/3
answer: it is 4 over 3 times the square root of 2 minus the quotient of 2 and 3 .
```

Other invocation options

Use english:

```
gfsage
```

Use *LANGUAGE*:

```
gfsage LANGUAGE
```

General invocation:

```
gfsage [OPTIONS]
```

where *OPTIONS* are:

short form	long form	description
`-h`	`--help`	Print usage page
`-i LANGUAGE`	`--input-lang=LANGUAGE`	Make queries in *LANGUAGE*
`-o LANGUAGE`	`--output-lang=LANGUAGE`	Give answers in *LANGUAGE*
`-V LEVEL`	`--verbose=LEVEL`	Set the verbosity *LEVEL*
`-t FILE`	`--test=FILE`	Test samples [11] in *FILE*
`-v[VOICE]`	`--voice[=VOICE]`	Use voice output [12]. To list voices use `?` as *VOICE*.
`-F`	`--with-feedback`	Restate the query when answering.

Limitations

- On Darwin (OS X 10.6 and 10.7) a bug in the Sage part makes the system unresponsive after some computations (between 7 and 10)
- On some machines, it takes time for the Sage server to respond.

This condition is signaled by the message:

```
gfsage: Connecting CurlCouldntConnect
```

I used a Linux virtual machine to reproduce this condition and find that, sometimes, it takes about 10 retries for the server to catch, but then it stays running ok for hours. My guess is that is related to some *timeout* limit in the server. Killing the orphaned python processes from the previous retries might help too (`killall python`).

Realsets

`realsets.py` is a Sage [6] module to support subsets of the real field consisting of intervals and isolated points and was developed to demonstrate set operations of the MGL `Set1` module.

It is based of previous work from Interval1Sage [13] adding integration on real sets and real intervals.

An object in this module consists of a list of disjoint open intervals plus a list of isolated points (not belonging to these intervals). Notice that `Infinite` is acceptable as interval bound. Therefore, one can define:

- All sort of real intervals: open, close and half-open
- Finite sets
- Unbounded intervals
- And combinations of these by union, intersection and taking complements.

Represent a set that can be the union of some intervals and isolated points. It consists of:

- A list of disjoint open non-empty intervals.
- A list of points. Each of these points belongs at most to one interval.

Examples

A closed interval:

```
? RealSet.cc_interval(1,4);
[ 1 :: 4 ]
```

A single point:

```
? RealSet.singleton(1)
{1}
```

Union

Union is supported with intervals and can be nested :

```
? I = RealSet.co_interval(1, 4)
? J = RealSet.co_interval(4, 5)
? M = RealSet.oc_interval(7, 8)
? I.union(J).union(M)
[ 1 :: 5 [ ∪ ] 7 :: 8 ]
```

Intersection

```
? I.intersection(J)
()
? I.intersection(RealSet.cc_interval(2,5))
[ 2 :: 4 [
```

Queries

Is a point in the set?

```
? I = RealSet.oo_interval(1, 3)
? 2 in I
True
? 3 in I
False
```

Is a set discrete (i.e: does not contain intervals)?

```
? RealSet.oo_interval(0,1).discrete
False
? RealSet(points=(1,2,3)).discrete
True
```

Size of a discrete is the number of points:

```
? RealSet(points=range(5)).size
5
? RealSet.oo_interval(0,3).size
+Infinity
```

A is subset of B

```
? A = RealSet.oo_interval(0,1)
? B = RealSet.cc_interval(0,1)
? RealSet().subset(A)
True
? B.subset(A)
False
? A.subset(B)
True
? A.subset(A)
True
? A.subset(A, proper=True)
False
```

Return the infimum (greatest lower bound)

```
? RealSet(points=range(3)).infimum()
0
? RealSet.oo_interval(1,3).infimum()
1
```

The opposite of a set: –A = {-x | x ∈ A}

```
? -RealSet.oo_interval(1,2)
] -2 :: -1 [
```

Return the supremum (least upper bound)

```
? RealSet(points=range(3)).supremum()
2
? RealSet.oo_interval(1,3).supremum()
3
```

The complementary of a set:

```
? RealSet.oo_interval(2,3).complement()
] -Infinity :: 2 ] ∪ [ 3 :: +Infinity [
? RealSet(points=range(3)).complement()
] 0 :: 1 [ ∪ ] 1 :: 2 [ ∪ ] 2 :: +Infinity [ ∪ ] -Infinity :: 0 [
```

The set difference of `A` and `B`: `\{x \in A, x\notin B\}`

```
? I = RealSet.oo_interval(2,+Infinity)
? J = RealSet.oo_interval(-Infinity, 5)
? I.setdiff(J)
[ 5 :: +Infinity [
? J.setdiff(I)
] -Infinity :: 2 ]
```

gfsage internal workings

`gfsage` is a prototype to demonstrate two-way natural language communication between a user and a Sage [14] system.

When you invoke the `gfsage` command interactively:

- A Sage process is started in the background, listening for incoming http requests;
- A GF [5] pgf module is read and set to mediate between the user and the Sage process;

The details of these components are given below.

The GF [5] side

A GF [5] module acts as a post office translating messages between the different parties (*nodes*) composing a *dialog*. This section is more a description of a proposed design strategy for a generic *postoffice* interface based on GF [5]. The actual code implements ideas of this design, but, for instance, it contains no *edges* or *nodes* as explicit entities.

NODES AND EDGES

`gfsage` deals with just 2 agents:

1. The user
2. The Sage system

in the case whether the *input language* is different of the *output language*, we may consider a third node (the output user).

There is a unique `pgf` module containing all GF [5] information for the dialog system to work: `Commands.pgf`. Each node has a **language** (a GF [5] *concrete module*) assigned: the user uses a natural language (i.e., `ComandsEng` for English).

A node reacts to received messages by sending a reply. The chain of messages between two nodes is called a *dialog*. An active node as the user can start a dialog by sending a message. A passive node, like the Sage system here, just replies to the received messages.

A node can *receive*:

- A **regular message** from another node: This is a GF [5] *linearization* in the receptor *language*.
- A `no_parse` **message** from the postoffice telling that a previous outgoing message cannot be parsed.
- An `is_ambiguous` **message** from the postoffice related to a previous message sent by the node, specifying that it was ambiguous and carrying additional info for the node to decide among the possible meanings. To respond to this, the node must send a `disambiguate` **message** to the postoffice (see below).

A node can *send*:

- A **regular message** to another node: This is a parseable string for the emitter *language*.
- A `disambiguate` **message** sent in response to an **ambiguous** message. In this message the node chooses one of the options or aborts the transaction.

A **regular message** between two given nodes corresponds to a fixed GF [5] category. In the case of `gfsage` it is `Command` for messages traveling from User to Sage and `Answer` for messages going the other way.

UP AND DOWN PIPELINE

A regular message from node N1 to node N2 goes through the following steps:

1. Input string is **lexed**, that is: separated into parse-able units (*tokens*);
2. It is then **parsed** using the node N1 *language* and *edge category* (i.e. node N1 to node N2) into a set of GF [5] *abstract trees*;

3. This set is, hopefully, reduced by paraphrasing the trees and removing duplicates (it is the **compute** step);
4. Now, If the resulting set is empty, a `no_parse` message is sent back to the sending node. If it contains more than one entry, an `is_ambiguous` message is sent. In the previous cases, the process stops here; Only when the computed set contains just an entry, is this pushed downstream to the node N2.
5. The abstract tree is **linearized** using the node N2 *language*;
6. The result is **unlexed**, that is: assembled into a string that is delivered to the receiving node.

The Sage side

For Sage to work alongside GF [5], we need a http sever listening to Sage commands and some scripts to set up the environment and respond to the type of queries that can be expressed in the *Mathematics Grammar Library*, MGL.

THE SAGE SERVER

A Sage process is started in the background by the `start-nb.py` script in `-python` mode. This script starts a Sage **notebook**, as described in Simple server API [15], listening on port 9000 and up to requests in http format. It also installs a handler for cleanly disposing of the notebook object whenever the parent process terminates.

The parent process sends then an initial request to load some functions and variables that we'll need in the dialog system defined in `prelude.sage` and goes into the main evaluation loop.

SAGE SCRIPTS

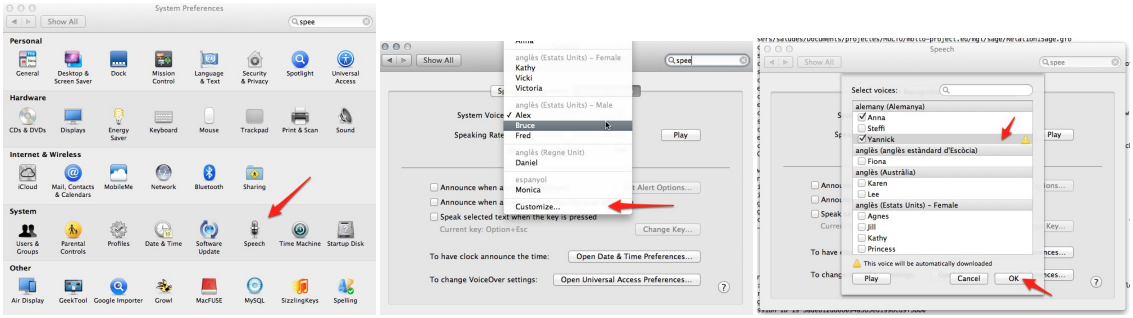
`realsets.py`
: is a Sage module developed to support set operations as described in the `Set1` module of the MGL [16]. (See the page about it [17])

`prelude.sage`
: defines Sage functions to implement derivation on the style of the MGL and state storing for numbers, sets, functions and sets to support anaphora in the dialog.

Adding voice output to gfsage

Description

OS X has voice output buit-in, usable from the shell by way of the `say` command. You can use several voices in English or download more for other languages.



Usage

1. You must build the system on `mgl/sage` as described previously.
2. Check that you have at least one voice for your prefered languages: Go to **System Preferences** > **Speech** and click on **System Voice**
3. See that you have the right ones. If not, click **Customize** on the pop-up
4. Select the ones for you and click Ok. When downloading terminates, you may run the tool.
5. You can call gfsage in 3 different ways, but for voiced output you must use the one with *OPTIONS*:

```
    gfsage Use english
    gfsage LANGUAGE Use this language
    gfsage [OPTIONS] where OPTIONS are:
    -h --help print this page
    -i INPUT --input-lang=INPUT Make queries in LANGUAGE
    -o OUTPUT --output-lang=OUTPUT Give answers in LANGUAGE
    -v[VOICE] --voice[=VOICE] use voice output. To list voices use ? as VOICE.
    -F --with-feedback Restate the query when answering.
```

The options relevant here are `-v` and `-F`. Use the first to select voice output. With no argument it will pick the first available voice for the *OUTPUT* voice selected:

```
./gfsage -i english -v
Voiced by Agnes
```

... It will use *Agnes* as English voice. Notice that if you do not give a `-o` option, the *OUTPUT* language is assume to be the same as the *INPUT* language.

To list the available voices use:

```
./gfsage -i english -v?
Agnes, Albert, Alex, Bahh, Bells, Boing, Bruce, Bubbles, Cellos, Daniel, Deranged, Fred, Hysterical, Jun
```

It will list the English voices. To use a specific voice write:

```
./gfsage -i german -vYannick
Voiced by Yannick
```

The option `-F` is to make the system paraphrase your query on answering. First, get a simple answer:

```
./gfsage -i english
Login into localhost at port 9000
Session ID is df7ad7c769f2faac68b6bb9489bb97e2
waiting... EmptyBlock 3
sage&gt; compute the factorial of 5.
(4) 120
answer: it is 120 .
```

... and now the same with paraphrasing:

```
./gfsage -i english -F
Login into localhost at port 9000
Session ID is 88549994a28940fe0657eb9e506a5e84
waiting... EmptyBlock 3
sage&gt; compute the factorial of 5.
(4) 120
answer: the factorial of 5 is 120 .
```

So, to experience voice output in its full glory you have to use both `-v` and `-F`.

EXPERIENCES WITH GOOGLE VOICE

Following a suggestion from Aarne, I found some Google service for speech input, but the experiments are not encouraging:

1. I recorded `Compute this` into a mp4 file using QuickTime Player on the mac
2. Converted it to flac using:

   ```
   sox compute.m4a compute.flac rate 16k
   ```

3. And get into the service by:

   ```
   curl -H "Content-Type:audio/x-flac; rate=16000" "https://www.google.com/speech-api/v1
   /recognize?xjerr=1&amp;client=chromium&amp;lang=en-US" -F "myfile=@compute.flac
   ```

But got:

```
`{"status":0,"id":"56bdb158dd66b25fc2e221364004e620-1","hypotheses":[{"utterance":"coffee lol","conf
```

Other examples:

- `"I like pickles" ⇒ "I like turtles"`
- `"The determinant of x" ⇒ "new york" (with confidence 0.88!)`
- `"Compute this" ⇒ "coffee lol"`

Of course I'm not a native English speaker, but I expected a better performance.

Adding tests to gfsage

To help with regression testing I recently added a **test** option to gfsage for batch-testing the system by reading dialog samples from a file.

The samples must be in a text file and consist in a sequence of *dialogs* which are sequences of query/responses to the Sage system. Notice that a dialog might carry a state in the form of assumptions that are asserted or variables that are assigned. In the same way, each dialog is completely independent of the others.

Each dialog starts with a `BEGIN` or `BEGIN language` line. It specifies the beginning of dialog *triplets* and the natural language for these *triplets*. The dialog runs until an `END` line. The language specified becomes the *current language*. Dialogs with no given languages are assumed to be in the *current language*. At the start of a testing suite, the current language is English.

A *triplet* is a sequence of 3 lines:

- The query passed to Sage in the current language
- The Sage response in sage language
- This response translated to the current language.

EXAMPLE OF A TEST SUITE

```
BEGIN spanish
calcula el factorial del número octal 11.
```

```
362880
es 36280 .
END
BEGIN english
let x be 4 .
```

```
compute the sum of x and 5 .
9
it is 9 .
compute the sum of it and 5 .
14
it is 14 .
END
```

Notice that blank lines are relevant: they mark that Sage responded nothing to the query. Therefore, it is not allowed to insert blank lines neither between triplets nor dialogs.

USAGE

```
gfsage --test
```

will test the dialogs in and tell about the differences. You got a summary of the results:

```
Dialog 'compute Gamma....' failed
18 out of 19 dialogs successful.
```

Using natural language inside Sage

By defining new Sage *interfaces* we can command the Sage shell and notebook server using natural language.

Installation

Move to the **sage** directory and build **sage-shell**:

```
cd mgl/sage
make sage-shell
```

The first time you build it, you may run into a warning as in the installation section of the front page [18], or:

```
Please add nlgf components to the interfaces list in /usr/local/sage-4.7.2/devel/sage/sage/in
```

We must inform Sage [6] that there are some new *interfaces* for it: We open `interfaces/all.py` (Notice that your actual path might be different), go to the end of the file and add something like this:

```
from nlgf import english, spanish
interfaces.extend(['english', 'spanish'])
```

The first line asks the system to load the interfaces for commanding Sage using English and Spanish. The next line add these to the list of available *interfaces*.

Now retry building:

```
make sage-shell
```

At the time of writing, the module `nlgf` provides `catalan`, `english`, `german`, and `spanish` interfaces.

SAGE SHELL WITH COMMAND AUTO-COMPLETION

In some systems you can have the commands Sage [6] shell auto-completed by pressing the tab key. This is experimental and you have to make the installation completely by hand.

First you have to build the Python bindings for GF [19] which, for the moment, only work in Linux. You'll find there a shared library called `gf.so`. Copy or move it into one of the directories that Python scans when resolving imports. Note that it may be the case that the Python instance run by Sage [6] be different of the one your machine runs by default; To be sure, do as follows:

```
sage -python -c 'import sys; print sys.path'
```

it will list all the directories that Sage/python scans.

You'll know it's all right when:

```
sage -python -c 'import gf'
```

exits with no complain: The next time you enter into the Sage [6] shell you'll have autocompletion for the GF [5] interfaces.

Usage

Shell interface

Start a Sage shell:

```
sage
```

and switch to one of the defined natural language interfaces:

```
sage: %english
```

will reply with:

```
--&gt; Switching to Gf &lt;--
```

If you didn't install autocompletion (which is the usual case, auto-completion being experimental), a warning will appear:

```
No autocompletion available
```

Now you're ready to issue sage commands in English:

```
english: compute the summation of x when x ranges from 1 to 100.
5050
english: add 3 to it.
5053
english: let x be the factorial of 6.
720
english: let y be the factorial of 5.
120
english: compute the greatest common divisor of x and y.
120
english: compute the least common multiple of x and y.
720
```

Go back to the standard interface by typing ctrl+D or typing `quit`.

Notebook interface

Sage [6] has a notebook interface that gives a more flexible way to interact with it. To use it, start the shell as above and then:

```
sage: notebook(secure=true, interface='')
The notebook files are stored in: sage_notebook.sagenb
**************************************************
*                                                *
* Open your web browser to https://localhost:8000 [20]  *
*                                                *
**************************************************
There is an admin account.  If you do not remember the password,
quit the notebook and type notebook(reset=True).
2012-02-13 12:48:19+0100 [-] Log opened.
...
```

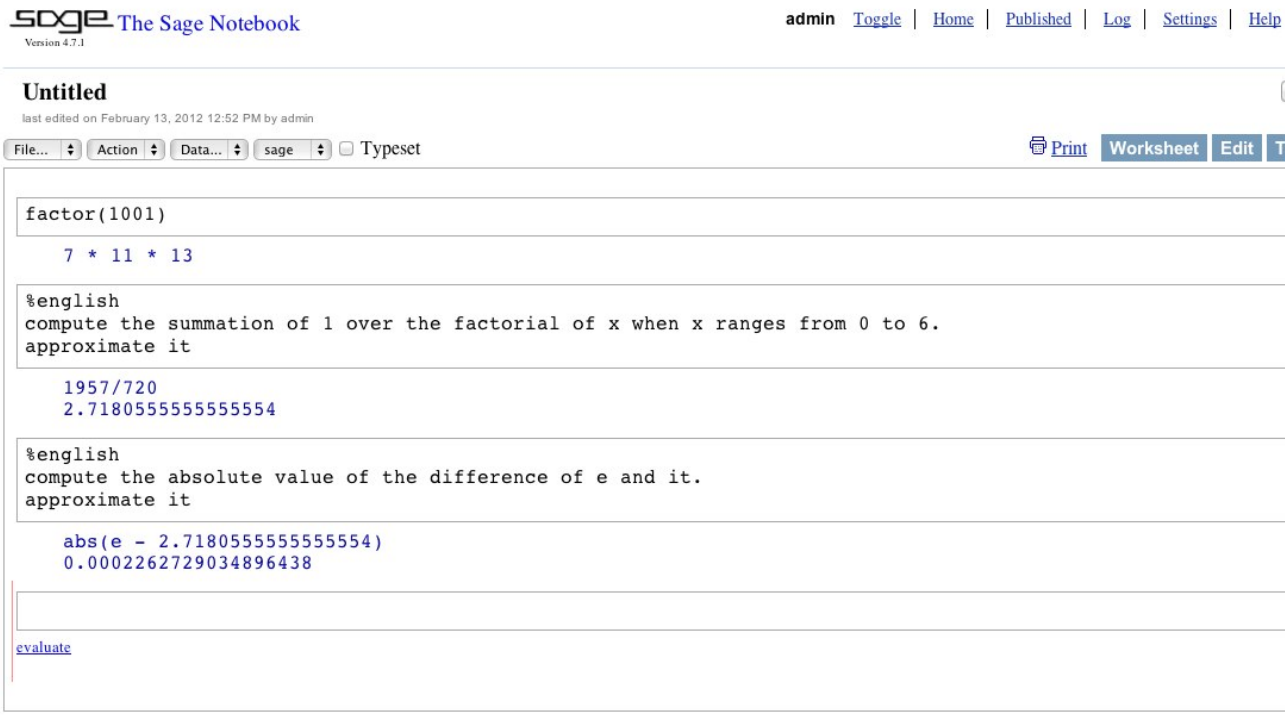
In some systems a browser will open simultaneously. Now you can use Sage from the browser.

Click on **New Worksheet**. You'll be asked to rename the worksheet (this is optional). A single **cell** will be ready for your input. Write your command and press **evaluate**. Notice that a cell can contain more than one command, separated by newlines.

Start a new cell by writing:

```
%english
```

and add one or more new lines with commands in English.



Attachment	Size
sage-notebook.jpg [21]	95.69 KB

Source URL: http://www.molto-project.eu/wiki/d62-prototype-comanding-cas

Links:
[1] http://www.molto-project.eu
[2] http://www.molto-project.eu/user/6
[3] http://www.molto-project.eu/Universitat Politècnica de Catalunya
[4] http://www.molto-project.eu/workplan/6
[5] http://www.grammaticalframework.org
[6] http://www.sagemath.org/
[7] http://www.molto-project.eu/story/simple-dialog-system-sage-natural-language
[8] http://www.molto-project.eu/wiki/using-natural-language-inside-sage
[9] http://www.haskell.org/ghc/
[10] http://www.molto-project.eu/node/1486
[11] http://www.molto-project.eu/node/1479
[12] http://www.molto-project.eu/node/1414
[13] http://www.mail-archive.com/sage-support@googlegroups.com/msg21326.html
[14] http://www.sagemath.org
[15] http://www.sagemath.org/doc/reference/sagenb/simple/twist.html
[16] http://www.molto-project.eu/node/1408
[17] http://www.molto-project.eu/wiki/realsets
[18] http://www.molto-project.eu/node/937
[19] http://www.molto-project.eu/wiki/gf-python-bindings
[20] https://localhost:8000
[21] http://www.molto-project.eu/sites/default/files/sage-notebook.jpg