Sage FAQ

Release 9.2

The Sage Development Team

Oct 25, 2020

CONTENTS

CHAPTER

ONE

FAQ: GENERALITÀ

1.1 Perchè esiste questo progetto?

La missione fissata per Sage è di essere un'alternativa open-source a Magma, Maple, Mathematica e Matlab. I predecessori di Sage, noti come HECKE e Manin, furono creati perché William Stein ebbe bisogno di scriverli come parte della sua ricerca sulla Teoria dei Numeri. Iniziato da William nel 2005 quando era all'università di Harvard, Sage combina alcuni fra i miglori software open-source per la matematica in un'unica intefaccia comune. Da allora Sage viene utilizzato non solo da ricercatori nel campo della Teoria dei Numeri, ma da ricercatori in tutte le scienze matematiche.

Sage si avvale ed estende le funzionalità di molti dei pacchetti inglobati. Anche dal principio, quando Sage veiniva usato principalmente per la Teoria dei Numeri, includeva: Givaro, MPIR, NTL, Pari/GP, e molti altri troppo numerosi per essere elencati qui. Studenti, insegnanti, professori universitari, ricercatori di tutto il mondo usano Sage perché vogliono un pacchetto open-source comprensivo per la matematica che offra calcolo sia simbolico che numerico. Perlopiù le persone sono contente di quanto offre Sage.

Com'é comune nell'ambito del software open-source (FOSS), spesso ci sono persone che individuano casi in cui Sage non dispone delle funzionalità richiesta da loro. Quindi si immergono nel codice sorgente di Sage per estenderlo per il loro scopo, o ancora per esporre delle funzionalità dei pacchetti inglobati in Sage in modo da poter usarne le funzioni che loro preferiscono dall'interfaccia di Sage. La squadra Sage-Combinat e' costituita da ricercatori in Algebra Combinatoria. La missione che si è data tale squadra è quella di migliorare Sage come uno strumento estendibile per la sperimentazione a computer nell'ambito dell'Algebra Combinatoria, e favorire lo scambio di codice sorgente fra ricercatori di questa materia. Per informazione dettagliate sul perché esiste Sage, vedere il seguente link: biografia matematica personale di William, settore software.

1.2 Cosa vuol dire Sage e come si deve pronunciarlo?

Nei primi anni di esistenza di Sage, il progetto era chiamato "SAGE", acronimo inglese di "software per la sperimentazione in Algebra e Geometria". A cominciare dal 2007 ed inizio 2008, fu largamente adottato il nome "Sage". Considera "Sage" come il nome di un progetto software FOSS per la matematica, esattamente come "Python" è il nome di un linguaggio FOSS di programmazione di uso generale. Ovunque possibile, per cortesia usa "Sage" non "SAGE", che invece è un progetto per un computer (SAGE), così da evitare confusioni. "Sage" si pronuncia nello stesso modo della parola inglese "sage" che significa uomo saggio, o anche indica la pianta della salvia. Alcune persone lo pronunciano in modo simile a "sarge", un po' come si pronuncia Debian Sarge.

Comunque lo pronunci, per cortesia non confonderlo con il software di contabilità americano che ha lo stesso nome.

1.3 Chi c'è dietro al progetto?

Sage è un progetto basato sull'opera di volontari. Il suo successo è dovuto all'opera gratuita di una grande squadra internazionale di studenti, insegnanti, professori universitari, ricercatori, ingegneri informatici e persone che lavorano in vari ambiti della matematica, delle scienze, dell'ingegneria, dello sviluppo software e a tutti i livelli della scuola. Lo sviluppo di Sage ha potuto usufruire di fondi asegnati da numerose istituzioni e ha potuto includere sia componenti preesistenti che in corso di sviluppo da parte di numerosi autori.

Una lista di coloro che hanno dato un contributo diretto è reperibile al link "mappa di sviluppo di Sage" (Sage Development Map) e la storia delle modifiche può essere reperita al link "changelog di alto livello" (changelog). Fai riferimento alla Pagina dei riconoscimenti del sito web di Sage per una lista aggiornata di coloro che ci sostengono finanziariamente o a livello di infrastruttura, a livello di siti mirror ed altri contributi indiretti.

1.4 Perché Sage è un software libero ed open-source?

Una regola universale nella comunità matematica è che tutto dev'essere pubblico e aperto ad analisi. Il progetto Sage ritiene che non seguire lo stesso principio nel software per la matematica è quantomeno scortese e maleducato, o peggio una violazione delle pratiche comuni nella scienza. Un principio filosofico sottostante Sage è di applicare la regola di scambio libero e confronto fra pari, che caratterizza la comunicazione scientifica, anche allo sviluppo di software per la matematica. Né il progetto Sage né la sua squadra di sviluppo hanno la pretesa di essere gli originali proponenti di questo principio.

Il modello di sviluppo di Sage è largamente ispirato del movimento del software libero di cui + stata pioniere la Free Software Foundation ed il movimento open-source. Una fonte di ispirazione all'interno della comunità matematica è Joachim Neubüser, come espresso nell'articolo

- J. Neubüser. An invitation to computational group theory. In C. M. Campbell, T. C. Hurley, E. F. Robertson, S. J. Tobin, and J. J. Ward, editors, *Groups '93 Galway/St. Andrews, Volume 2*, volume 212 of London Mathematical Society Lecture Note Series, pages 457–475. Cambridge University Press, 1995.

ed in particolare nella seguente citazione dal suo articolo:

```
Puoi leggere il teorema di Sylow e la sua dimostrazione nel libro di Huppert
in biblioteca senza nemmeno comprare il libro e poi usare questo teorema per
il resto della tua vita senza dover pagare una tariffa, invece...per molti
sistemi di algebra computazionale devi pagare regolarmente delle licenze
per tutto il tempo in cui li utilizzi. Per proteggere ciò per cui devi pagare,
non ti viene dato il codice sorgente ma soltanto l'eseguibile del programma,
cioè un oggetto sigillato sul quale premi bottoni ed ottieni risposte così come
ottieni belle immagini dal tuo televisore: in entrambe le situazioni non puoi
controllare il modo in cui sono create.

In questa situazione sono violate due delle più basilari regole di condotta in
matematica: in essa l'informazione è passata gratuitamente e tutto può essere
consultato e verificato. Non applicare queste regole ai sistemi di algebra
computazionale usati per la ricerca in matematica...significa andare in una
direzione assolutamente non desiderabile. Ancora più importante:
possiamo aspettarci che qualcuno creda al risultato di un programma il cui
funzionamento non gli è permesso di vedere? Inoltre: vogliamo veramente far
pagare a colleghi in Moldavia molti anni di stipendio per un
sistema di algebra computazionale?
```

Simili idee sono state anche espresse da Andrei Okounkov, come si può leggere in

- V. Muñoz and U. Persson. Interviews with three Fields medalists. *Notices of the American Mathematical Society*, 54(3):405–410, 2007.

ed in particolare nella seguente citazione:

```
I computer non sono una minaccia per i matematici più di quanto i robot da
cucina lo siano per i cuochi. Poiché la matematica diviene sempre più complessa
mentre il ritmo delle nostre vite accellera, dobbiamo delegare il più possibile
alle macchine. Ed intendo sia il lavoro in campo numerico che in quello
simbolico. Alcune persone possono andare avanti senza lavastoviglie, ma penso
che le dimostrazioni vengano fuori molto più pulite quando il lavoro di
routine è automatizzato.

Questo porta con sè parecchie questioni. Non sono un esperto ma penso che
abbiamo bisogno di uno standard a livello di calcolo simbolico per rendere le
manipolazioni al computer più facili da documentare e verificare. Con tutto il
rispetto per il libero mercato, forse in questo non dobbiam essere dipendenti
da un software commerciale. Un progetto open-source potrebbe, forse, trovare
risposte migliori a problemi ovvi come la disponibilità, i bug, la
compatibilità all'indietro, l'indipendenza dalla piattaforma, le librerie
standard, ecc. Si può imparare dal successo di TeX e da software più
specializzati come Macaulay2. Spero veramente che le agenzie per finanziamenti
governativi stiano considerando questo.
```

1.5 Perché avete scritto Sage da zero, invece di usare software e librerie preesistenti?

Sage non è stato scritto da zero. La maggior parte delle sue funzionalità sono realizzate attraverso progetti FOSS come

- ATLAS — libreria software per Algebra Lineare ottimizzata automaticamente.
- BLAS — sottoprogrammi per Algebra Lineare di base.
- FLINT — libreria C per Teoria dei Numeri.
- GAP — sistema di calcolo per algebra discreta, con particolare enfasi sulla teoria dei gruppi.
- Maxima — sistema di calcolo simbolico e numerico.
- mpmath — libreria in puro Python per aritmetica floating-point di precisione.
- NumPy — algebra lineare numerica ed altre funzioni di calcolo numerico per Python.
- Pari/GP — software matematico per calcolo veloce in Teoria dei Numeri.
- Pynac — versione modificata di GiNaC che rimpiazza la dipendenza da CLN con Python.
- R — linguaggio ed ambiente operativo per calcolo statistico e grafici relativi.
- E molti altri troppo numerosi per essere elencati qui.

Una lista aggiornata può essere reperita alla seguente link: repository dei pacchetti standard.

I principali linguaggi di programmazione di Sage sono Python e Cython. Python è il principale linguaggio di programmazione e di interfacciamento, mentre Cython è il principale linguaggio per ottimizzare funzionalità critiche e per interfacciarsi con le librerie C e le estensioni C per Python. Sage integra oltre 90 pacchetti FOSS in un'interfaccia comune. Sopra questi pacchetti sta la libreria Sage, che consiste in oltre 700.000 righe di codice Python e Cython scritto ex-novo. Vedi openhub.net per l'analisi del codice sorgente dell'ultima release stabile di Sage.

1.6 Come posso ricevere aiuto?

Per ricevere aiuto sull'uso di Sage, ci sono due opzioni

- Il sito web di domande e risposte `ask.sagemath.org` : https://ask.sagemath.org/questions/
- La lista email `sage-support`: http://groups.google.com/group/sage-support

Per aiuto sullo sviluppo di Sage, c'è una list email `sage-devel` : https://groups.google.com/forum/#!forum/sage-devel

Consulta http://www.sagemath.org/help.html per una lista con altre risorse.

1.7 Non sarebbe meglio se Sage non fosse distribuito come un gigantesco aggregato di pacchetti?

La distribuzione SageMath continua a construire le proprie versione dei sofware richiesti ("SPKGs") che funzionano bene assieme.

Tuttavia, per ridurre i tempi di compilazione e le dimensioni dell'installazione di Sage, fin dalle versioni 8.x c'è stato uno sforzo a livello di sviluppo che ha reso possibile utilizzare molti pacchetti software già presenti nel sistema operativo (or da distribuzioni di Homebrew o conda-forge) invece di costruire delle copie solo per SageMath.

Il meccanismo nominato "spkg-configure" è utilizzato all'inizio del processo di installazione dal codice sorgente durante la fase `./configure`.

Per assicurasi chec SageMath si installi e funzioni correttamente su un'ampia gamma di sistemi, noi utilizziamo dei test automatici. Vai al capitolo Test di portabilità nella Guida Per Sviluppatori per maggiori informazioni.

1.8 Perché ci sono così tanti bug in Sage, con centinaia di modifiche in corso, perché non producete una versione stabilizzata?

Ogni software contiene bug. In qualcosa di così complesso come Sage nessuno, né la squadra di sviluppo di Sage né la sua comunità, ha alcuna pretesa che esso sia libero da bug. Farlo sarebbe un atto di disonestà.

Un ciclo di rilascio di Sage di solito dura alcuni mesi, con molte versioni beta che si susseguono a intervalli di 2-3 settimane. Ogni ciclo di rilascio è presieduto da un singolo gestore che si occupa dell'albero di integrazione pacchetti per tutta la durata del ciclo. In questa fase tale gestore deve spesso dedicare del tempo equivalente ad un lavoro a tempo pieno alla gestione della qualità e deve interagire attivamente con la comunità internazionale degli utenti, degli sviluppatore e dei potenziali contributori a Sage.

Ci sono stati alcuni casi in cui due contributori a Sage sono stati affiancati come gestori di rilascio per un ciclo di rilascio di Sage. Comunque in genere poche persone hanno l'equivalente di 3 settimane di tempo libero per dedicarsi alla gestione del rilascio. Se vuoi aiutare nella gestione del rilascio iscriviti alla mailing list sage-release.

Fin dall'inizio del progetto Sage, i contributori hanno cercato di ascoltare e di riflettere su cosa potesse aumentare la possibilità che altri potenziali validi contributori dessero effettivamente un aiuto. Cosa incoraggia un contributore può scoraggiare un altro, quindi bisogna trovare degli equilibri. Decidere che un rilascio stabilizzato dovrebbe includere le patch di correzione dei bug, e solo quelle, probabilmente scoraggerebbe qualcuno dal contribuire, nel momento in cui gli fosse detto in anticipo che la sua aggiunta, anche se giudicata positivamente, non verrebbe integrata nel rilascio.

La comunità Sage crede nel principio "pubblica subito, pubblica spesso". Il modo in cui il progetto Sage è organizzato e gestito differisce parecchio da quello di una azienda di software commerciale. I contributori sono tutti volontari e questo cambia totalmente la dinamica del progetto da quella che sarebbe se Sage fosse un'iniziativa software commerciale con sviluppatori stipendiati a tempo pieno.

1.9 Come posso scaricare la documentazione di Sage così da poterla leggere offline?

Per scaricare la documentazione standard di Sage in formato HTML o PDF, visita Help and Support sul sito web di Sage. Ogni release di Sage dispone della documentazione completa che costituisce la documentazione standard di Sage. Se hai scaricato un rilascio di Sage in formato binario, la versione HTML della sua documentazione si trova già disponibile nella cartella `SAGE_ROOT/src/doc/output/html/`. Nel corso della compilazione da sorgente viene preparata anche la documentazione HTML. Per construire la versione HTML della documentazione, lancia il seguente comando dopo essersi posizionati in `SAGE_ROOT`:

```
$ ./sage -docbuild --no-pdf-links all html
```

La preparazione della documentazione in formato PDF richiede che sul tuo sistema sia installata una versione funzionante di LaTeX. Per preparare la documentazione in formato PDF puoi lanciare il comando, dopo esserti posizionato in `SAGE_ROOT`:

```
$ ./sage -docbuild all pdf
```

Per altre maggiori opzioni disponibili da riga di comando fai riferimento alle istruzioni stampate dai seguenti comandi:

```
$ ./sage -help
$ ./sage -advanced
```

CHAPTER

TWO

FAQ: USO DI SAGE

2.1 Come posso partire?

Puoi provare Sage senza dover scaricare nulla:

- **CoCalc™:** Vai su https://cocalc.com e crea un account gratis.

  Se accedi al sito, avrai accesso all'ultima versione di Sage e molti altri programmi

  Ricorda che questo sito è un servizio commerciale indipendente da Sage.

- **Sage cell:** Una singola versione di Sage, disponibile per fare una computazione alla volta. https://sagecell.sagemath.org/

Per scaricare una distribuzione binaria **precompilata** di Sage vai al link http://www.sagemath.org/download.html e clicca sul link relativo al file binario per il tuo sistema operativo.

Il **codice sorgente** di Sage è anche disponibile al download: vai al link http://www.sagemath.org/download-source.html per scaricare l'archivio TAR di qualunque rilascio di Sage.

Le sessioni di lavoro Notebook di Sage sono eseguite all'interno di un browser web. Puoi lancianer il Notebook di sage attraverso il seguente comando purché `sage` sia nella variabile `PATH`

```
$ sage -notebook
```

2.2 Quali sono i prerequisiti di Sage?

La maggior parte delle dipendenze sono incluse all'interno di Sage. Nella maggior parte dei casi puoi scaricare il binario precompilato ed usarlo senza dover installare alcun pacchetto dipendente. Se usi Windows avrai bisogno di intallare VirtualBox, che puoi scaricare dal link http://www.virtualbox.org/wiki/Downloads. Dopo aver installato VirtualBox devi scaricare una distribuzione di Sage per VirtualBox al link http://www.sagemath.org/download-windows.html. Segui bene le istruzioni che trovi a quella pagina. Poi puoi lanciare la macchina virtuale Sage usando il software VirtualBox.

Puoi scaricare il codice sorgente completo di Sage per compilarlo sul tuo sistema Linux o Mac OS X. Sage si trova in una cartella isolata e non va ad interferire col sistema in cui si trova. Viene dato con tutto il necessario per lo sviluppo, il codice sorgente, tutte le dipendenze ed il changelog (cioè la lista delle modifiche operate) completo. Sui sistemi Linux come Debian/Ubuntu puoi dover installare il pacchetto `build essential` ed il processore di macro `m4`. Il tuo sistema deve disporre di un compilatore C funzionante se vuoi compilare Sage da codice sorgente. Su Debian/Ubuntu puoi installare questi prerequisiti come segue:

```
sudo apt-get install build-essential m4
```

Se hai un sistema multiprocessore puoi scegliere una copilazione parallela di Sage. Il comando

```
export MAKE='make -j8'
```

abiliterà 8 threads per quelle parti della compilazione che supportano il parallelismo. Al posto del numero 8 metti il numero di processori/core del tuo sistema.

2.3 Come posso far riconoscere la mia attuale installazione di Tcl/Tk all'interprete Python di Sage?

Potresti avere la libreria Tcl/Tk installata e l'interprete Python del tuo sistema la riconosce ma l'interprete Python di Sage no. Per risolvere questo ti basta installare la libreria di sviluppo Tcl/Tk. Su Ubuntu lancia, da riga di comando:

```
sudo apt-get install tk8.5-dev
```

o qualcosa di simile. Poi reinstalla l'iterprete Python di Sage con:

```
sage -f python
```

Questo aggancerà automaticamente la libreria Tcl/Tk. Dopo aver reinstallato correttamente l'interprete Python di Sage, lancia i seguenti comandi dall'interfaccia a riga di comando di Sage:

```
import _tkinter
import Tkinter
```

Se non ti viene segnalato alcun errore di `ImportError` allora il problema è risolto.

2.4 Come faccio ad importare Sage in uno script Python?

Puoi importare Sage in uno script Python come faresti con una libreria. La cosa a cui fare attenzione è che devi lanciare quello script Python usando l'interprete Python interno a Sage (versione 3.7.x per Sage 9.2). Per importare Sage metti la seguente istruzione in cima al tuo script Python:

```
from sage.all import *
```

Quando poi esegui il tuo script devi lanciare Sage con l'opzione `-python` che farà sì che venga eseguito dalla versione dell'interprete interna a Sage. Ad esempio, se Sage è nella tua variabile d'ambiente `PATH`, puoi scrivere:

```
sage -python /path/to/my/script.py
```

Un altro modo è scrivere uno script Sage e lanciarlo usando Sage stesso. Uno script Sage ha estensione `.sage` ed è simile ad uno script Python ma utilizza funzioni e comandi specifici di Sage. Puoi poi lanciare tale script Sage in questo modo:

```
sage /path/to/my/script.sage
```

Questo si occuperà di caricare le variabili d'ambiente necesssarie ed eseguire gli import di default al posto tuo.

2.5 Come posso ricaricare uno script Python in una sessione di Sage?

Puoi caricare uno script Python in una sessione Sage usando il comando **load**. Ad esempio possiamo usare Sage per importare un file di nome "simple.py" con:

```
load("simple.py")
```

e ripetere questo comando ogni volta che cambiamo il file. Invece digitando:

```
attach("simple.py")
```

ogni cambiamento al file verrà automaticamente aggiornato anche in Sage.

2.6 Posso usare Sage con la versione 3.x di Python?

Dalla versione 9.0 del Gennaio 2020, SageMath utilizza Python 3.

2.7 Vedo un errore di "Permission denied" (accesso negato) su un file di nome "sage-flags.txt.

Quando Sage viene compilato dal sorgente, tiene traccia di quali istruzioni speciali supporta la tua CPU (quali ad esempio SSE2) e le memorizza. Così se provi ad eseguire il codice su un'altra macchina, che non supporta queste istruzioni speciali extra, ti vengono segnalati degli errori in maniera intelleggibile anzichè dei generici "segfault" (segmento di memoria errato) o "illegal istruction" (istruzione non consentita). Poichè quest'informazione dev'essere memorizzata in Sage stesso anziché nella cartella `.sage`, dev'essere creata da qualcuno con le necessarie autorizzazioni sul sistema. Quindi se vedi qualcosa del genere

```
Traceback (most recent call last):
  File "/usr/local/sage-4.0.2/local/bin/sage-location", line 174, in <module>
    t, R = install_moved()
  File "/usr/local/sage-4.0.2/local/bin/sage-location", line 18, in install_moved
    write_flags_file()
  File "/usr/local/sage-4.0.2/local/bin/sage-location", line 82, in write_flags_file
    open(flags_file,'w').write(get_flags_info())
IOError: [Errno 13] Permission denied:
  '/usr/local/sage-4.0.2/local/lib/sage-flags.txt'
```

probabilmente significa che hai compilato/installato Sage usando un determinato account (nome utente), ma poi non l'hai eseguito così da permettergli di generare il file `sage-flags.txt`. Ti basta eseguire Sage una volta con lo stesso account con cui è stato installato per risolvere questo problema. Questo si dovrebbe risolvere facilmente anche lanciando Sage una volta nel corso del processo d'installazione (cfr. trac ticket #6375).

2.8 Ho scaricato il binario di Sage e va in crash quando lo lancio, con il messaggio "illegal instruction" (istruzione non permessa). Cosa posso fare?

Un modo di risolvere è compilare Sage interamente dal codice sorgente. Un'altra possibilità è correggere la tua installazione di Sage con la ricompilazione dei componenti MPIR e ATLAS (richiede da 15 a 20 minuti), da effettuarsi da riga di comando a partire dalla cartella `SAGE_ROOT` della tua installazione con le 2 istruzioni:

```
rm spkg/installed/mpir* spkg/installed/atlas*
make
```

È possibile che i binari siano stati compilati per un'architettura più recente di quella della tua macchina. Nessuno ha ancora trovato un modo di compilare Sage in maniera che MPIR ed ATLAS funzionino su qualunque hardware. Questo sarà prima o poi risolto. Qualunque aiuto in tal senso sarà apprezzato.

2.9 Ho usato Debian/Ubuntu per installare la versione 3.0.5 di Sage ed essa sta dando un sacco di errori. Cosa posso fare?

La versione di Sage distribuita con `apt-get` in Debian e Ubuntu (tipo la 3.0.5) è molto vecchia. Nessuno ha ancora avuto tempo di aggiornare la versione di Sage per Debian/Ubuntu. Qualunque aiuto in tal senso sarà molto apprezzato. Dovresti scaricare la versione più recente di Sage dal link di download del sito web di Sage. Se vuoi aiutarci ad aggiornare la versione di Sage per Debian/Ubuntu manda un'email alla mailing list sage-devel.

2.10 Faccio meglio ad usare la versione ufficiale o quella di sviluppo?

Ti consigliamo di usare la più recente versione ufficiale di Sage. Le versioni di sviluppo sono spesso annunciate sulle mailing list sage-devel e sage-release. Una maniera facile di aiutare con lo sviluppo di Sage è scaricare l'ultima versione di sviluppo, compilarla sul suo sistema, lanciare tutti i doctest e segnalare qualunque errore di compilazione o qualunque fallimento nei doctest.

2.11 È difficile imparare Sage?

Le funzionalità di base di Sage dovrebbero risultare facili da imparare quanto le basi di Python. Molti tutorial sono disponibili in rete per aiutarti ad imparare Sage. Per trarre il massimo da Sage ti consigliamo di impararare qualche elemento del linguaggio Python. Segue una lista, incompleta, di risorse su Python. Altre risorse possono essere trovate cercando sul web.

- Building Skills in Python di Steven F. Lott
- Dive into Python di Mark Pilgrim
- How to Think Like a Computer Scientist di Jeffrey Elkner, Allen B. Downey, and Chris Meyers
- Official Python Tutorial
- Python home page e Python standard documentation

2.12 Posso fare X in Sage?

Ti consigliamo di usare l'autocompletamento di Sage con il tasto TAB. Ti basta digitare qualche carattere, premere TAB e vedere se il comando che vuoi compare nella lista di autocompletamento. Se hai un comando che si chiama `mycmd`, allora digitalo e premi TAB per visualizzare la lista di funzionalità che sono supportate da quel comando. Per leggere la documentazione di `mycmd` scrivi `mycmd?` poi premi Invio e protrai leggerla. In modo similare, digitando `mycmd??` e poi Invio potrai visualizzare il codice sorgente di tale comando. Ti consigliamo anche di eseguire ricerche nel codice sorgente e nella documentazione di Sage. Per eseguire ricerche nel codice sorgente di Sage usa il comando `search_src("<search-keyword>")` mettendo al posto di `<search-keyword>` le parole chiave che vuoi cercare. Analogamente puoi effettuare ricerche nella documentazione di Sage usando il comando: `search_doc("<search-keyword>")`.

2.13 Cosa fa esattamente Sage quando digito "0.6**2"?

Quando scrivi "0.6**2" in Python, ti viene restituito qualcosa tipo 0.35999999999999999. Ma quando fai lo stesso in Sage ti viene restituito 0.360000000000000. Per capire perché Python si comporta in questo modo vedi il Python Tutorial, soprattutto il capitolo "Aritmetica floating-point: caratteristiche e limiti". Ciò che Sage fa è preprocessare l'input e trasformarlo come segue:

```
sage: preparse("0.6**2")
"RealNumber('0.6')**Integer(2)"
```

Così che ciò che viene *effettivamente* eseguito è:

```
RealNumber('0.6')**Integer(2)
```

Gli sviluppatori Sage (in pratica Carl Witty) decisero che i numeri floating-point di Sage dovessero, di default, stampare solo il numero di cifre decimali corrette, quando possibile, così da evitare il problema che ha Python. Questa decisione ha i suoi pro e contro. Nota che `RealNumber` e `Integer` sono specifici di Sage, quindi non puoi digitare quanto sopra nell'interprete Python ed aspettarti che funzioni, se prima non hai eseguito delle istruzioni di import quali:

```
from sage.all import RealNumber, Integer, preparse
```

2.14 Perché il comando "history" di Sage è diverso da quello di Magma?

Nell'uso di Sage non disponi di una funzionalità dell'interfaccia a riga di comando di Magma. In Magma, se immetti una linea recuperata dalla "history" (cioè dall'elenco dei comandi digitati precedentemente che viene automaticamente memorizzato) con il tasto "freccia in su'" e poi premi "freccia in giu'", viene recuperata anche la linea successiva nell'elenco. Questa funzionalità ti permette di recuperare dalla "history" tante righe consecutive quante vuoi. Ma Sage non ha una funzionalità simile: la riga di comando IPython utilizza la libreria "readline" (via pyreadline), che evidentemente non supporta questa funzionalit. Magma ha una sua propria libreria personalizzata simile alla "readline" che invece supporta questa funzionalità. (Dal momento che moltissime persone hanno richiesto questa funzionalità, se qualcuno trovasse un modo per implementarla sarebbe il benvenuto!)

2.15 Ho problemi di tipo nell'utilizzo da Sage di SciPy, cvxopt e NumPy.

Stai usando da Sage le librerie SciPy, cvxopt e NumPy e hai degli errori tipo:

```
TypeError: function not supported for these types, and can't coerce safely to␣
↪supported types.
```

Quando digiti numeri in Sage, il preprocessore li converte in un anello base, come puoi vedere facendo:

```
sage: preparse("stats.uniform(0,15).ppf([0.5,0.7])")
"stats.uniform(Integer(0),Integer(15)).ppf([RealNumber('0.5'),RealNumber('0.7')])"
```

Sfortunamente il supporto che NumPy fornisce a questi tipi avanzati di Sage, quali `Integer` o `RealNumber` (numeri reali di precisione arbitraria), non è del 100%. Per risolvere ridefinisci `Integer` e/o `RealNumber` per cambiare il comportamento del preprocessore di Sage così che i decimali scritti vengano registrati come tipi float di Python anziché RealNumber di Sage e gli interi scritti siano registrati come tipi int di Python anziché Integer di Sage. Ad esempio:

```
sage: RealNumber = float; Integer = int
sage: from scipy import stats
sage: stats.ttest_ind(list([1,2,3,4,5]),list([2,3,4,5,.6]))
Ttest_indResult(statistic=0.0767529..., pvalue=0.940704...)
sage: stats.uniform(0,15).ppf([0.5,0.7])
array([  7.5,  10.5])
```

In alternativa sii esplicito circa il tipo di dato, ad esempio:

```
sage: from scipy import stats
sage: stats.uniform(int(0),int(15)).ppf([float(0.5),float(0.7)])
array([  7.5,  10.5])
```

Come terza alternativa puoi usare i suffissi semplici:

```
sage: from scipy import stats
sage: stats.uniform(0r,15r).ppf([0.5r,0.7r])
array([  7.5,  10.5])
```

Puoi anche disabilitare il preprocessore nel tuo codice tramite il comando `preparse(False)`. Puoi lanciare IPython da solo dalla riga di comando con `sage -ipython`, cosa che non precarica niente di specifico di Sage. O ancora puoi cambiare il linguaggio di sessione (Notebook language) in "Python".

2.16 Come faccio a salvare un oggetto così che non devo ridigitarlo ogni volta che apro un foglio di lavoro (worksheet)?

I comandi `save` e `load` rispettivamente registrano e caricano un oggetto. Nella sessione di lavoro Notebook la variabile `DATA` è la collocazione dello spazio di salvataggio del foglio di lavoro (worksheet). Per registrare l'oggetto `my_stuff` in un foglio di lavoro puoi digitare:

```
save(my_stuff, DATA + "my_stuff")
```

e, per ricaricarlo, ti basta digitare:

```
my_stuff = load(DATA + "my_stuff")
```

2.17 Sage contiene una funzione simile alla “ToCharacterCode[]” di Mathematica?

Potresti voler convertire caratteri ASCII come “Big Mac” nel corrispondente codice numerico per ulteriori elaborazioni. In Sage e Python puoi usare `ord`. Ad esempio:

```
sage: list(map(ord, "abcde"))
[97, 98, 99, 100, 101]
sage: list(map(ord, "Big Mac"))
[66, 105, 103, 32, 77, 97, 99]
```

2.18 Come posso scrivere le multiplicazioni in modo implicito come in Mathematica?

Sage ha una funzione che lo rende possibile:

```
sage: implicit_multiplication(True)
sage: x 2 x  # not tested
2*x^2
sage: implicit_multiplication(False)
```

Questo viene preprocessato da Sage in codice per Python. Potrebbe non funzionare in situazioni complicate. Per vedere cosa il preprocessore fa:

```
sage: implicit_multiplication(True)
sage: preparse("2 x")
'Integer(2)*x'
sage: implicit_multiplication(False)
sage: preparse("2 x")
'Integer(2) x'
```

Vai al link https://wiki.sagemath.org/sage_mathematica per maggiori informazioni su Mathematica vs. SageMath.

2.19 Posso far eseguire in automatico a Sage dei comandi all’accensione?

Sì, ti basta creare un file `$HOME/.sage/init.sage` ed esso sarà eseguito ogni volta che lanci Sage. Questo presuppone che la variabile ambiente di Sage `DOT_SAGE` punti alla cartella nascosta `$HOME/.sage`, cosa che avviene di default.

2.20 Quando compilo Sage il mio computer fa beep e si spegne o si blocca.

Compilare sage è piuttosto faticoso per il processore del computer. Il comportamento suddetto di solito indica che il computer si è surriscaldato. In molti casi questo si può risolvere pulendo il ventilatore del processore del computer ed assicurando adeguata areazione al computer. Puoi chiedere al tuo amministratore di sistema o ad un tecnico di provvedere, qualora tu non l'abbia mai fatto. Questa manutenzione del computer, se non fatta da persone preparate, potrebbe anche danneggiare il computer.

Per gli utenti Linux, se pensi che la compilazione fallisca per un problema di risorse di macchina, una soluzione potrebbe essere di modificare il file `/etc/inittab` per far partire Linux al runlevel 3. Tale file di solito contiene qualcosa del tipo:

```
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have
#   networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
```

Questo fa sì che la tua distribuzione Linux parta con la schermata di login grafico. Commenta la linea `id:5:initdefault:` e aggiungi la linea `id:3:initdefault:`, così da aver qualcosa come:

```
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have
#   networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
# id:5:initdefault:
id:3:initdefault:
```

Ora se riavvii il sistema ti troverai davanti all'interfaccia di login testuale. Questa ti permette di accedere al sistema con una sessione testuale all'interno di un terminale virtuale. Una tale sessione di solito non consuma molte risorse, come farebbe invece un'interfaccia grafica. Poi puoi compilare Sage da codice sorgente in tale sessione testuale. Dovresti assicurarti di essere in grado di riattivare successivamente l'interfaccia grafica, prima di tentare di accedere tramite un'interfaccia testuale.

2.21 Sage 2.9 o superiore non riesce a compilare ATLAS su Linux. Come posso risolvere?

La causa più probabile è l'abilitazione della gestione dell'alimentazione. Disabilitala per risolvere il problema. In base al tuo tipo di distribuzione ciò si può fare da interfaccia grafica oppure no. Digita da riga di comando, come utente root, quanto segue, per ogni CPU presente sul tuo sistema:

```
/usr/bin/cpufreq-selector -g performance -c #number CPU
```

Su Ubuntu, prova a disabilitare "Power Manager" (gestione alimentazione) via

```
System --> Preferences --> Sessions
```

nel menu "Startup Programs" (programmi di avvio) o utilizzando `cpufreq-set` da riga di comando.

2.22 Quando lancio Sage, SELinux segnala che "/path/to/libpari-gmp.so.2" richiede "text-relocation" (riallocazione del testo). Come posso risolvere?

Il problema può essere risolto eseguendo il seguente comando:

```
chcon -t textrel_shlib_t /path/to/libpari-gmp.so.2
```

2.23 L'aggiornamento di Sage è andato bene, ma adesso l'indicatore continua a mostrare la versione precedente. Come posso risolvere?

L'indicatore (banner in inglese) è memorizzato e non ricalcolato ad ogni esecuzione di Sage. Il fatto che non sia aggiornato non dovrebbe impedire a Sage di funzionare regolarmente. Digita `banner()` in una sessione di Sage per verificare la versione reale. Se vuoi l'indicatore corretto allora devi ricompilare Sage digitando `make build` in un terminale.

2.24 Come posso eseguire Sage come demone/servizio?

Ci sono parecchie possibilità. Puoi usare i programmi a riga di comando `screen`, `nohup` o `disown`.

2.25 Il comando show (mostra) per la visualizzazione di oggetti 3D non funziona.

La visualizzazione 3D in tempo reale per Sage dalla versione 6.4 in avanti usa il pacchetto Jmol/JSmol. Dalla linea di comando viene utilizzata l'applicazione Java Jmol, mentre per la visualizzazione dal browser viene usato puro javascript oppura una Java applet. In genere nei browser è usato javascript puro per evitare problemi con quei browser che non supportano i plugin per le applet Java (ad esempio Chrome). In ogni worksheet su browser c'è una casella da spuntare prima di generare una vista tridimensionale qualora l'utente voglia usare l'applet Java (essa è un po' più veloce con viste complicate).

La ragione più probabile di un malfunzionamento è che non hai installato l'ambiente runtime di Java (JRE) o che è più vecchio della versione 1.7. Se le cose funzionano dalla riga di comando, un'altra possibilità è che il tuo browser non abbia il plugin giusto per supportare le Java applet (al momento, nel 2014, tali plugin non lavorano con la maggior parte delle versioni di Chrome). Assicurati di aver installato il plugin IcedTea (su Linux vedi il tuo gestore dei pacchetti) o il plugin di Oracle Java (vedi: IcedTea e Java).

Se stai usando un server Sage sul web e anche la visualizzazione tramite javascript non funziona, potresti avere un problema con la funzionalità javascript del tuo browser, o potresti aver disabilitato javascript.

2.26 Posso usare gli strumenti di Sage in un ambiente commerciale?

Sì! Assolutamente! Fondamentalmente l'unico *limite* che hai è che se fai dei delle modifiche a Sage stesso e redistribuisci pubblicamente tale versione modificata, allora devi renderci disponibili tali modifiche cos' che le possiamo includere nella versione standard di Sage (se vogliamo). Altrimenti sei libero di usare quante copie di Sage vuoi per fare soldi, ecc. senza pagare alcuna licenza.

2.27 Voglio scrivere del codice Cython che usa l'aritmetica dei campi finiti, ma l'istruzione "cimport sage.rings.finite_field_givaro" non funziona. Cosa posso fare?

Devi segnalare a Sage di usare C++ (sia Givaro che NTL sono librerie C++) e che hai bisogno anche delle librerie GMP e STDC C++. Ecco un piccolo esempio:

```
# These comments are hints to Cython about the compiler and
# libraries needed for the Givaro library:
#
# distutils: language = c++
# distutils: libraries = givaro gmpxx gmp m
cimport sage.rings.finite_field_givaro
# Construct a finite field of order 11.
cdef sage.rings.finite_field_givaro.FiniteField_givaro K
K = sage.rings.finite_field_givaro.FiniteField_givaro(11)
print("K is a {}".format(type(K)))
print("K cardinality = {}".format(K.cardinality()))
# Construct two values in the field:
cdef sage.rings.finite_field_givaro.FiniteField_givaroElement x
cdef sage.rings.finite_field_givaro.FiniteField_givaroElement y
x = K(3)
y = K(6)
print("x is a {}".format(type(x)))
```

(continues on next page)

(continued from previous page)

```
print("x = {}".format(x))
print("y = {}".format(y))
print("x has multiplicative order = {}".format(x.multiplicative_order()))
print("y has multiplicative order = {}".format(y.multiplicative_order()))
print("x*y = {}".format(x * y))
# Show that x behaves like a finite field element:
for i in range(1, x.multiplicative_order() + 1):
    print("{} {}".format(i, x**i))
assert x*(1/x) == K.one()
```

Per saperne di più digita quanto segue al prompt di Sage

```
sage.rings.finite_field_givaro.FiniteField_givaro.
```

Poi premi TAB, ed usa `??` per avere più informationi su ogni funzione. Ad esempio:

```
sage.rings.finite_field_givaro.FiniteField_givaro.one??
```

fornisce informazioni sull'unità moltiplicativa nel campo finito.

2.28 La compilazione su Mac OS X fallisce in maniera incomprensibile. Come posso risolvere?

Cerca il file di log della compilazione (install.log) e controlla se c'è il seguente messaggio:

```
fork: Resource temporarily unavailable.
```

Se è così, prova a fare questo: crea (o modifica se c'è già) il file `/etc/launchd.conf` includendovi quanto segue:

```
limit maxproc 512 2048
```

Poi riavvia. Vedi il seguente link per maggiori dettagli.

2.29 Come disegno la radice cubica (o altre radici dispari) di numeri negativi?

Questa è una delle domande chieste più frequentemente. Ci sono molti metodi descritti nelle documentazione per la funzione plot, ma questo è il più semplice:

```
sage: plot(real_nth_root(x, 3), (x, -1, 1))
Graphics object consisting of 1 graphics primitive
```

Tuttavia, nota che il metodo più diretto:

```
sage: plot(x^(1/3), (x, -1, 1))  # not tested
```

produce il grafico corretto solo per valori di x positivi. La *ragione* per cui ciò avviene è che Sage restituisce dei numeri complessi per le radici dispari di numeri negativi, quando queste sono approssimate, il che è una convenzione standard:

```
sage: numerical_approx( (-1)^(1/3) )
0.500000000000000 + 0.866025403784439*I
```

2.30 Come va utilizzato in Sage l'operatore XOR bitwise?

L'OR esclusivo in Sage si fa con l'operatore ^^. C'è anche il corrispondente "operatore inplace" ^^=. Ad esempio:

```
sage: 3^^2
1
sage: a = 2
sage: a ^^= 8
sage: a
10
```

Se definisci 2 variabili e poi confronti:

```
sage: a = 5; b = 8
sage: a.__xor__(b), 13
(13, 13)
```

Puoi anche fare:

```
sage: (5).__xor__(8)
13
```

Le parentesi sono necessarie affinché Sage non supponga di avere a che fare con un numero reale. Ci sono molti modi di definire una funzione:

```
sage: xor = lambda x, y: x.__xor__(y)
sage: xor(3, 8)
11
```

Un'altra possibilità, che aggira il preparser di Sage, è

```
sage: def xor(a, b):
....:     return eval("%s^%s" % (a, b))
sage: xor(3, 8)
11
```

Puoi anche disattivare il preparser di Sage con il comando `preparser(False)`, a quel punto l'operatore ^ funzionerà esattamente come in Python. Puoi successivamente riattivare il preparser con il comando `preparser(True)`. Questo funziona solo dalla riga di comando di Sage. Se sei in una sessione Notebook, passa in "Python mode".

2.31 Quando provo ad usare LaTeX in una sessione Notebook, dice che non trova "fullpage.sty".

La risposta più ampia, ma forse non la più utile, è che hai bisogno di installare `fullpage.sty` in una cartella acceduta da TeX. Su Ubuntu (e probabilmente anche su altre distribuzioni Linux) dovresti installare il pacchetto `texlive-latex-extra`. Se non è disponibile, prova ad installare il pacchetto `tetex-extra`. Se stai usando Mac OS X dovrai usare una qualunque distribuzione TeX che hai già per ottenere `fullpage.sty` (se usi MacTeX probabilmente ce l'hai già installato). Se stai usando l'immagine VirtualBox in Windows dovrai fare login in tale immagine ed da lì installare `texlive-latex-extra`.

2.32 Con degli oggetti “a” e “b” ed una funzione “f” ho digitato accidentalmente “f(a)=b” anziche’ “f(a)==b”. Questo mi ha dato un errore “TypeError” (come mi aspettavo) ma ha anche cancellato l’oggetto “a”. Perchè?

Questo è dovuto a come sono definite le funzioni in Sage con la notazione `f(x)=expr` usando il preparser. Nota anche che se fai quest’errore in un costrutto `if`, avrai un errore `SyntaxError` prima di qualunque altro comportamento errato, quindi, in questo caso, non hai il problema.

2.33 Come posso usare un browser internet diverso con il notebook di Sage?

Dovrai farlo dalla linea di comando. Semplicemente lancia un comando come questo.

- Linux (assumendo che hai Sage nella cartella `/usr/bin`):

```
$ env BROWSER=opera /usr/bin/sage --notebook
```

- Mac (assumendo che tu sia nella cartella dove hai scaricato Sage). Con il notebook Jupyter:

```
$ BROWSER='open -a Firefox %s' ./sage --notebook jupyter
$ BROWSER='open -a Google\ Chrome %s' ./sage --notebook jupyter
```

  Con il vecchio notebook SageNB:

```
$ BROWSER='open -a Firefox' ./sage --notebook
$ BROWSER='open -a Google\ Chrome' ./sage --notebook
```

2.34 Dov’è il codice sorgente di `<function>`?

Le funzioni e classi scritte in Python o Cython sono di solito accessibili tramite la linea di comando IPython con il comando `??`:

```
sage: plot??                                # not tested
Signature: plot(*args, **kwds)
Source:
...
```

Tuttabia gli oggetti che sono construiti in Python o IPython sono compilati e non verranno visualizzati. Ci sono molte funzioni in Sage construite come funzioni simboliche, i.e. possono essere usate come parte di espressioni simboliche senza dover essere calcolate. Il loro codice sorgente potrebbe non essere accessibile dalla linea di comando, sopratutto per le funzioni elementaru, poiché sono scritte in C++ (per ragioni di efficienza).

CHAPTER

THREE

FAQ: CONTRIBUIRE A SAGE.

3.1 Come posso iniziare a contribuire a Sage?

Il primo passo è usare Sage e incoraggiare i tuoi amici a usare Sage. Se trovi bug o della documentazione poco chiara, per favore riporta i problemi!

Due modi comuni per contribuire a Sage sono scrivere codice sorgente e creare documentazione o tutorial. Alcuni passi in entrambe le direzioni sono descritti di seguito.

3.2 Voglio contribuire a Sage. Da dove inizio?

Dai un'occhiata alla guida ufficiale per lo sviluppo di Sage. Come minimo, la lettura del primo capitolo è richiesta per ogni svilluppatore di Sage. Inoltre sii molto attento alle linee guida per trac. Puoi entrare nella lista email sage-devel o nel canale IRC `#sage-devel` su freenode. Mentre inizi a conoscere la comunità prendi una copia del codice sorgente di Sage e familiarizza con git, un software per il controllo versione.

Il migiol mode per diventare familiare con il processo di sviluppo di Sage è quello di scegliere un ticket dal server trac ed esaminare i cambiamenti proposti in quel ticket. Se vuoi costruire qualcosa, è buona pratica discutere le tue idee sulla lista email `sage-devel`, in modo che altri sviluppatori abbiano l'opportunità di esprimersi sulle tue idee/proposte. Sono anche aperti a nuove idee, come tutti i matematici dovrebbero essere.

La principale lingua di programmazione per Sage è Python. Alcune parte di Sage potrebbero essere scritte in altre lingue, specialmente le componenti che fanno l'elaborazione numerica più impegnativa, ma la maggior parte delle funzionalità sono realizzate in Python, incluso il codice di collegamento. Un aspetto valido di Python, che Sage eredita, è che è più importante scrivere codice funzionante piuttosto che codice veloce. Del codice veloce è prezioso, ma del codice chiaro e leggibile è importante. Nella comunità matematica i risultati inaccurati sono inaccettabili. La correttezza viene prima dell'ottimizzazione. Nel seguente articolo

- D. Knuth. Structured Programming with go to Statements. *ACM Journal Computing Surveys*, 6(4), 1974.

Don Knuth osserva che "dovremmo dimenticarci dell'efficienza di poco conto, diciamo per il 97% del tempo: l'ottimizzazione prematura sta alla radice di ogni male".

Se non conosci Python dovresti iniziare ad imparare il linguaggio. Un buon posto dove iniziare è il Tutorial Ufficiale per Python e altra documentazione si trova in Documentazione standard di Python. Un altro posto da guardare è al link Dive Into Python di Marc Pilgrim, che può essere veramente d'aiuto su temi specifici come lo sviluppo guidato dai test. Il libro Building Skills in Python di Steven F. Lott è adatto a chiunque sia già a suo agio nel programmare.

Se desideri, puoi iniziare a imparare Python usando Sage. Tuttavia, è utile sapere cosa è semplice Python e quando Sage sta usando la sua "magia". Ci sono molte cose che funzionano in Phyton, ma non in Sage e vice versa. Per di più anche quando la sintassi è identica, molti concetti di programmazione sono spiegati più dettagliatamente in risorse focalizzate su Python piuttosto che in risorse centrate su Sage; in quest'ultime, la prioirità viene data alla matematica.

3.3 Non sono un programmatore. C'è qualche altro modo in cui posso aiutare?

Certo. Come in ogni progetto FOSS ci sono molti modi in cui puoi dare il tuo aiuto nella comunità di Sage e programmare è solo uno di questi.

Molte persone amano scrivere tutorial tecnici. Una delle gioie di farlo è che impari qualcosa di nuovo nel farlo. Allo stesso tempo trasmetti delle conoscenze ai principianti, un'abilità che è utile anche in campi estranei alla stesura di documentazione tecnica. Un aspetto fondamentale della documentazione tecnica è che espone un dato tecnico a dei principianti, pertanto il gergo tecnico dev'essere ridotto al minimo. Darrell Anderson ha scritto alcuni suggerimenti sulla scrittura di documentazione tecnica, il quale consigliamo vivamente.

Per i designer grafici o gli artisti c'è la possibilità di aiutare migliorando l'aspetto del sito di Sage.

Se sai parlare, leggere e scrivere in un'altra lingua, ci sono molti modi in cui il tuo contributo può essere molto prezioso all'intera comunità di Sage. Diciamo che conosci l'italiano. Allora puoi scrivere un tutorial per Sage in italiano, o aiutare a tradurre i tutorial ufficiali di Sage in italiano.

La suddetta è una lista molto breve. Ci sono molti, molti più modi in cui puoi dare il tuo aiuto. Sentiti libero di inviare una email alla mailing list sage-devel per chiedere in quali modi potresti essere d'aiuto, o per suggerire un'idea sul progetto.

3.4 Dove posso trovare risorse su Python e Cython?

Ecco una lista incompleta di risorse su Python e Cython. Ulteriori risorse possono essere trovate cercando sul web.

Risorse generali

- Cython
- pep8
- pydeps
- pycallgraph
- PyChecker
- PyFlakes
- Pylint
- Python home page e la Documentazione standard su Python
- Snakefood
- Sphinx
- XDot

Tutorial e libri

- Cython Tutorial di Stefan Behnel, Robert W. Bradshaw, e Dag Sverre Seljebotn
- Dive Into Python 3 di Mark Pilgrim
- Fast Numerical Computations with Cython di Dag Sverre Seljebotn
- Tutorial ufficiale di Python

Articoli e HOWTO

- decorator

- Functional Programming HOWTO di A. M. Kuchling
- Python Functional Programming for Mathematicians di Minh Van Nguyen
- Regular Expression HOWTO di A. M. Kuchling
- reStructuredText

3.5 Ci sono delle convenzioni di scrittura del codice sorgente che devo seguire?

Dovresti seguire le convenzioni standard di Python come documentato in **PEP 8** e **PEP 257**. Consulta anche la Guida dello Sviluppo Sage, specialmente il capitolo Convenzioni per scrivere codice sorgente in Sage.

3.6 Ho inviato al server trac una correzione molte settimane fa. Perchè la state ignorando?

Non stiamo cercando di ignorare la tua correzione. Le persone che lavorano su Sage lo fanno nel loro tempo libero. Con centinaia di ticket aperti aventi un impatto molto variabile sulla comunità di Sage, le persone che ci lavorano devono dedicare il loro tempo e sforzo principalmente a quei ticket che li interessano. A volte potresti essere la sola persona che comprende la tua correzione. In tal caso, ti invitiamo a fare uno sforzo supplementare per rendere l'esaminazione della tua patch il più semplice possibile. Ecco alcuni suggerimenti su come rendere la tua correzione facile da esaminare

- Hai descritto in modo chiaro il problema che la tua correzione vuole risolvere?
- Hai fornito ogni informazione di base rilevante al problema che la tua correzione vuole risolvere? Tali informazioni includono link a risorse online e ad articoli, libri, o altro materiale di riferimento.
- Hai descritto in modo chiaro come la tua correzione risolve il problema in oggetto?
- Hai descritto chiaramente nella tua correzione come effettuare i test dei cambiamenti?
- Hai elencato eventuali tickets da cui dipende la tua correzione?
- Se vi sono più correzioni, hai indicato chiaramente l'ordine in cui devono essere applicate ?
- La tua correzione segue le convenzioni importanti indicate nella "Guida dello sviluppatore"?

Se la tua correzione non ha la possibilità di essere aggiunta nell'albero dei sorgenti di Sage, non la ignoreremo ma semplicemente chiuderemo il ticket relativo con una spiegazione sul perché non possiamo includerla.

3.7 Come e quando posso ricordardare alla comunità di Sage una correzione a cui tengo?

Ti suggeriamo di fare uno sforzo ulteriore sul come ricordare alla comunità di Sage una correzione che vuoi venga inclusa nell'albero dei sorgenti di Sage. Potrebbe esserci un prossimo evento "bug squash sprint" o "Sage days" che è in relazione alla tua correzione. Tieni d'occhio le mailing list relative e rispondi educatamente ad ogni scambio di email relativo, spiegando chiaramente perché la tua correzione ha importanza. Tieni d'occhio il canale IRC `#sage-devel`, avendo cura di rammentare la questione al momento giusto.

3.8 Ho scritto del codice sorgente e voglio venga incluso in Sage. Però dopo aver rinominato il mio file `a.sage` in `a.py` ho degli errori di sintassi. Devo riscrivere tutto il mio codice in Python anziché in Sage?

La risposta sostanzialmente è sì, ma riscrivere è una parola grossa per ciò che bisogna realmente fare. C'è ben poco da fare dal momento che Sage per lo più segue la sintassi di Python. Le 2 maggiori differenze sono la gestione degli interi (vedi anche il link afterword per maggiori informazioni sul preparser di Sage) e la necessità di importare quello che ti serve.

- **Gestione degli interi:** dei fare i seguenti cambiamenti:
  - Notazione per l'elevamento a potenza: In Python `**` significa elevamento a potenza e `^` significa "xor".
  - Se devi restituire un intero all'utente, scrivi `return Integer(1)` invece di `return 1`. In Python, 1 è un intero Python (`int`), e `Integer(1)` è un intero Sage/Gmp. Inoltre gli `Integer` sono molto più potenti degli `int`; ad esempio hanno collegata ad essi l'informazione di primalità e la fattorizzazione.
  - Dovresti anche notare che `2 / 3` non significa più `Integer(2) / Integer(3)` che restituisce `2/3`, ma invece `int(2) / int(3)`, e pertanto restituisce `0` poichè la divisione è intera e trascura il resto. Se stai lavorando con i tipi `Integer` ma in realtà hai bisogno di eseguire una divisione intera puoi usare `Integer(2) // Integer(3)`.
- **Note sull'importazione:** la seconda cosa importante da tenere presente è la necessità di importare tutto ciò di cui hai bisogno. Nel dettaglio, ogni volta che usi una funzione Sage la devi prima importare all'inizio del file. Ad esempio, se hai bisogno di `PolynomialRing`, dovrai scrivere:

```
from sage.rings.polynomial.polynomial_ring_constructor import PolynomialRing
```

Puoi chiedere a Sage dove il comando per importare `PolynomialRing` usando:

```
sage: import_statements(PolynomialRing)
from sage.rings.polynomial.polynomial_ring_constructor import PolynomialRing
```

Se questo fallisce, puoi chiedere a Sage dove si trove `PolynomialRing` usando:

```
sage: PolynomialRing.__module__
'sage.rings.polynomial.polynomial_ring_constructor'
```

Questo corrisponde anche al percorso, che inizia dopo `site-packages`, restituito da Sage quando richiami l'help su `PolynomialRing`. A d esempio se scrivi `PolynomialRing?` otterrai:

```
Type:      function
[...]
File:      /path_to_sage_root/sage/local/lib/python3.7/site-packages/sage/rings/
↪polynomial/polynomial_ring_constructor.py
[...]
```

CHAPTER

FOUR

INDICE E TABELLE

- genindex
- modindex
- search

INDEX

P