# Intro Tutorial Chennai

# Introductory Sage Tutorial - Welcome!

This Sage worksheet is for the course in Sage and programming at the Institute of Mathematical Sciences in Chennai, circling around Sage Days 60. It is based on one in a series of tutorials developed for the MAA PREP Workshop "Sage: Using Open-Source Mathematics Software with Undergraduates" (funding provided by NSF DUE 0817071). It is licensed under the Creative Commons Attribution-ShareAlike 3.0 license (CC BY-SA).

## What is Sage, and where do I use it?

Sage is comprehensive open-source mathematics software. With it, you can compute, manipulate, model, and visualize in your desired domain of mathematics.

Because it is based on a widespread, easy-to-learn computer language, it is very powerful and yet usable for a novice. We'll start using it in just a second!

### Ways to use Sage

We will be using this interface, the so-called "Sage notebook", throughout the course. It is fairly intuitive and similar to the graphical interface for other programs. However, there are several other options you may find useful.

- The Sage cell server at http://sagecell.sagemath.org is a great way to do one-time computations from anywhere.
  - We can compute the number of partitions of the number 9.
    - As you can see, it is divisible by 5 - appropriate to confirm Ramanujan's result here in Chennai!
    - We can list the partitions of the number 9 to confirm this.
  - This link plots several familiar functions together nicely.
  - The Android and iOS apps are based on using this functionality.
- As another cloud-based option, Sage can be used complete with a whole filesystem via the SageMathCloud, though it requires slightly more bandwidth.
  - It is a little more work to set up an account, but you gain a lot of power, with a lot of collaborative options.
  - You can basically do anything you can do on a Linux computer, which is a lot. There are a few tutorials, like this one.
  - (Demo)
- For those comfortable with command-line interfaces to computers, you can download Sage to your

desktop.
  - This works natively on Mac and most Linux distros.
  - On Windows it requires a virtualization process.
  - An advantage is that your data is on you whether you are online or not.
  - (Demo)
- Finally, we have the notebook.
  - This does require a login and some brief steps to open a worksheet like this one.
  - How to get started is well-documented in this tutorial.

From now on, we'll stick with the notebook, but nearly everything is independent of how you interact with Sage.

## Evaluating Sage Commands

### (i.e., How do I get Sage to do some math?)

Below, and throughout this *worksheet*, are little boxes called *input cells* or *code cells*. They should be about the width of your browser.

Evaluating the content of an input cell is very easy.

- First, click inside the cell so that the cell is active (i.e., has a bright blue border).
- Then, just below the cell on the left, an "evaluate" link appears; clicking this link evaluates the cell.

Try evaluating the following cell.

```
2+2
```

```
4
```

Sage prints out its response just below the cell (that's the "4" above, so Sage confirms that $2+2=4$). Note also that Sage has automatically made the next cell active after you evaluated your first cell.

You can also evaluate a cell using a keyboard shortcut.

- If the following cell isn't active, click in it.
- Then hold down the Shift key while you press the Enter key.

We call this "Shift-Enter". Try doing Shift-Enter with this cell.

```
factor(2014)
```

```
2011
```

An input cell isn't much use if it can only do one thing, so you can edit a cell and evaluate it again. Just click inside, and then make any changes you wish by typing as usual.

Try changing the number "2014" above to "2011" and evaluate the cell to find its factorization (surprised?); then try your own favorite number.

To do more math, we'll need to be able to create new input cells. This is also easy.

- Move your cursor over the space above or below another cell.
- A blue horizontal line as wide as the browser should appear.
- Click on the line to insert a new cell.

If for some reason you need to remove or delete an input cell, just delete all the text inside of it, and then press backspace in the now-empty cell.

Try creating a few new input cells below, doing some arithmetic in those cells, and then deleting one of the input cells.

```

```

```

```

There are a lot more special keyboard strokes to do fancier things with the cells, especially once you are a more adept programmer and will see their utility.  See the "Help" link and search for the "key and mouse bindings" if you are curious.

## Functions in Sage

To start out, let's explore how to define and use normal symbolic functions (or, "callable symbolic expressions") in Sage.  It's useful to start with really elementary material.

For a typical mathematical function, it's pretty straightforward to define it.  Below, we define the function

$$f(x) = x^2 \ .$$

```
f(x) = x^2
```

Since all we wanted was to create the function $f(x)$, Sage just does this and doesn't print anything out back to us.  (Notice we needed to use the "caret" to get exponentiation.)

We can check the definition by asking Sage what $f(x)$ is:

```
f(x)
```

```
x^2
```

If we just ask Sage what $f$ is (as opposed to $f(x)$), Sage prints out the standard mathematical notation for a function that maps a variable $x$ to the value $x^2$ (with the "maps to" arrow $\mapsto$ as "|-->").

```
f
```

```
x |--> x^2
```

We can evaluate $f$ at various values.

```
f(3)
```

```
9
```

```
f(3.1)
```

```
9.61000000000000
```

```
f(31/10)
```

```
961/100
```

Notice that the output type changes depending on whether the input had a decimal; that is a key feature.

Naturally, we are not restricted to $x$ as a variable. In the next cell, we define the function $g(y) = 2y - 1$.

```
g(y) = 2*y - 1
```

(If you've never used a computer algebra system before, you may not want to type $2 \times y$ instead of just $2y$, but it's important to use the "times" symbol.)

In the next cell, we see what happens if we try to use a variable all by itself.

```
z^2
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'z' is not defined
```

This is explained in some detail elsewhere. At this point, it suffices to know using the function notation (like "g(y)") tells Sage you are serious about "y" being a variable.

One can also do this with the "var('z')" notation below.

```
var('z')
z^2
```

```
z^2
```

This also demonstrates that we can put several commands in one cell, each on a separate line. The output of the last command (if any) is printed as the output of the cell.

Sage knows various common mathematical constants, like $\pi$ ("pi") and $e$.

```
f(pi)
```

```
pi^2
```

```
f(e^-1)
```

```
e^(-2)
```

In order to see a numeric approximation for an expression, just type the expression inside the parentheses of "N( )".

```
N(f(pi))
```

```
9.86960440108936
```

Another option, often more useful in practice, is having the expression immediately followed by ".n( )" (note the dot).

```
f(pi).n()
```

```
9.86960440108936
```

For now, we won't go in great depth explaining the reasons behind this syntax, which may be new to you. As you will learn later, Sage often uses this type of syntax (known as "object-oriented") because...

- Sage uses the Python programming language, which uses this syntax, 'under the hood', and
- Because it makes it easier to distinguish among
  - The mathematical object,
  - The thing you are doing to it, and
  - Any ancillary arguments.

For example, the following numerically evaluates ('n') the constant $\pi$ ('pi') to twenty digits ('digits=20').

```
pi.n(digits=20)
```

```
3.1415926535897932385
```

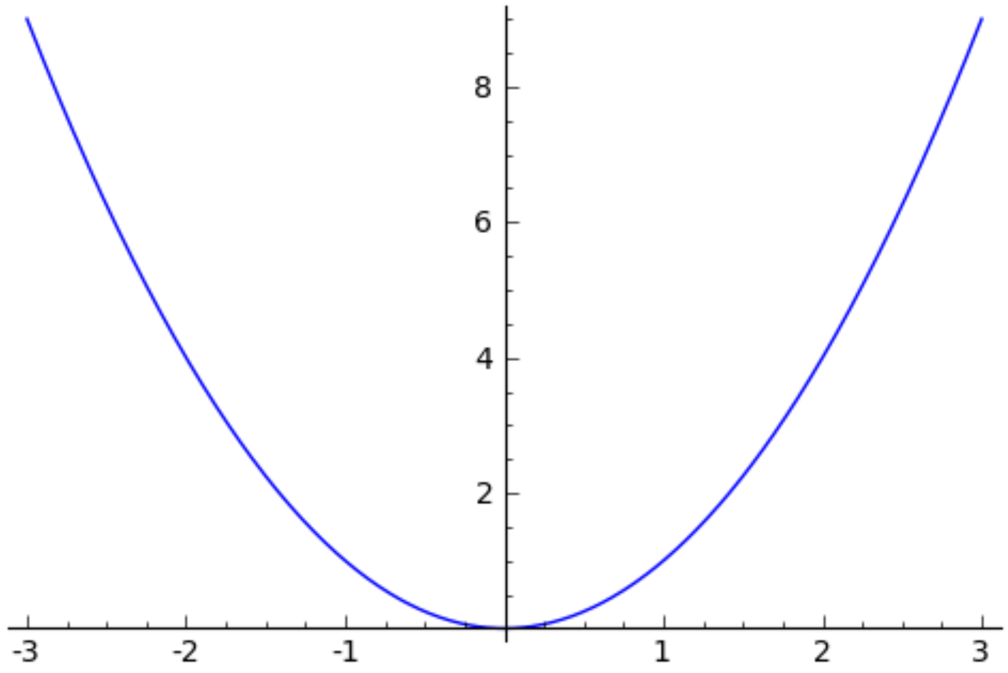
If we want things to look nicer, we can use the "show" command.

```
show(sqrt(2))
```

$\sqrt{2}$

We can also plot functions easily.

```
plot(f, (x,-3,3))
```



The preferred syntax has the variable and endpoints for the plotting domain in parentheses, separated by commas. We will not go further into plotting here, but there is *lots* of documentation! If you are feeling bold, plot the "sqrt" function in the next cell between 0 and 100.

But we are here to use Sage with more than just $x^2$. How would we find out how to use it and what Sage can do?

# Help and Information inside Sage

There are various ways to get help for doing and finding things in Sage. It's important to be conversant with how to do it to get any serious work done, so on this first day we will spend time with common ways to get help as you are working in a Sage worksheet.

## Documentation

Sage includes extensive documentation covering thousands of functions, with many examples, tutorials, and other helps.

- One way to access these is to click the "Help" link at the top right of any worksheet, then click your preferred option at the top of the help page.
- They are also available any time online at the Sage website, which has many other links, like video introductions.
- The Quick Reference cards are another useful tool once you get more familiar with Sage.

For this course, you will want to start *now* to familiarize yourself with the comprehensive reference manual, especially the "combinatorics" section. This is accessible via "Help" or in the Sage online documentation.

As an example, here is the de Bruijn sequence documentation. If you don't know what that is, don't worry - I didn't either until writing this, but now I think they are pretty cool!

```
DeBruijnSequences(2,5).an_element()
```

```
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
1, 0, 1, 1, 0, 1, 1, 1, 1, 1]
```

If you go through this in a cycle, you can get every sequence of zeros and ones of length three. What are some other topics you will all be expecting to find? There is a search box which is helpful, and just doing a search for "my math topic sage" often works as well.

What about things we can do directly within this worksheet, though?

## Tab completion

The most useful help available in the notebook is "tab completion". The idea is that even if you aren't one hundred percent sure of the name of a command, the first few letters should still be enough to help find it. Here's an example.

- Suppose you want to do a specific type of plot - maybe a slope field plot - but aren't quite sure what will do it.
- Still, it seems reasonable that the command might start with "pl".
- Then one can type "pl" in an input cell, and then press the tab key to see all the commands that start with the letters "pl".

Try tabbing after the "pl" in the following cell to see all the commands that start with the letters "pl". You should see that "plot_slope_field" is one of them.

```
pl
```

To pick one, just click on it; to stop viewing them, press the Escape/esc key. This should work in all Sage interfaces.

What if I wanted algebraic structures related to matrices? This reveals a slight catch.

```
mat
```

We can plot a matrix and make one, but is that really all Sage has to offer? Before we give up, let's try one more thing.

```
Mat
```

Do you see what I did differently? Ah, there are the matrix algebras!

This is more of a parlor trick.  Far more useful is that you can use this to see what you can do to an expression or mathematical object.

- Assuming your expression has a name, type it;
- Then type a period after it,
- Then press tab.

You will see a list pop up of all the things you can do to the expression.

To try this, evaluate the following cell to define a nice diagonal matrix.

```
M = diagonal_matrix([1,2,3])
```

Now put your cursor after the period and press your tab key.

```
M.
```

Again, Escape should remove the list.

One of the things in that (extremely long!) list above was "characteristic_polynomial".  Let's try it.

```
M.characteristic_polynomial()
```

```
    x^3 - 6*x^2 + 11*x - 6
```

Huh, that was interesting.  What else can I do?

```
M.cyclic_subspace(vector([1,1,0]))
```

```
    Vector space of degree 3 and dimension 2 over Rational Field
    Basis matrix:
    [1 0 0]
    [0 1 0]
```

## Finding documentation (question marks)

In the previous example, you might have wondered why I needed to put "M.cyclic_subspace(vector([1,1,0]))" rather than just "M.cyclic_subspace()", by analogy with "pi.n()", which we saw earlier.

To find out, there is another help tool one can use from right inside the notebook.  Almost all documentation in Sage has extensive examples that can illustrate how to use the function.

- As with tab completion, type the expression, period, and the name of the function.
- Then type a question mark.
- Press tab *or* evaluate to see the documentation.

To see how this help works, move your cursor after the question mark below and press tab.

```
M.cyclic_subspace?
```

To stop viewing the documentation after pressing tab, you can press the Escape key, just like with the completion of options.

The examples illustrate that the syntax requires "M.cyclic_subspace(v)" for some vector v. This makes sense, given that the subspace is defined based on a vector! But if one hadn't known this definition, this is valuable. The error if you didn't know is useful too.

```
M.cyclic_subspace()
```

```
    Traceback (click to the left of this block for traceback)
    ...
    TypeError: cyclic_subspace() takes at least 1 positional argument (0
    given)
```

If you would like the documentation to be visible longer-term, you can *evaluate* a command with the question mark (like below) to access the documentation, rather than just tabbing. Then it will stay there until you remove the input cell. However, it may cut off some of the content; you may be able to click along the left to get the rest.

```
M.cyclic_subspace?
```

**File:** /Users/karl.crisman/Downloads/Sage-6.2.app/Contents/Resources/sage/src/sage/matrix/matrix2.pyx

**Type:** <type ‘builtin_function_or_method’>

**Definition:** M.cyclic_subspace(v, var=None, basis=’echelon’)

**Docstring:**

Create a cyclic subspace for a vector, and optionally, a minimal polynomial for the iterated powers.

These subspaces are also known as Krylov subspaces. They are spanned by the vectors

$$\{v, Av, A^2v, A^3v, \dots\}$$

INPUT:

- `self` - a square matrix with entries from a field.
- `v` - a vector with a degree equal to the size of the matrix and entries compatible with the entries o matrix.
- `var` - default: `None` - if specified as a string or a generator of a polynomial ring, then this will be u construct a polynomial reflecting a relation of linear dependence on the powers $A^iv$ *and* this will polynomial to be returned along with the subspace. A generator must create polynomials with coe from the same field as the matrix entries.
- `basis` - default: `echelon` - the basis for the subspace is “echelonized” by default, but the keywo will return a subspace with a user basis equal to the largest linearly independent set $\{v, Av, A^2v, A^3v, \dots, A^{k-1}v\}$.

OUTPUT:

Suppose $k$ is the smallest power such that $\{v, Av, A^2v, A^3v, \dots, A^kv\}$ is linearly dependent. Then th subspace returned will have dimension $k$ and be spanned by the powers $0$ through $k-1$.

If a polynomial is requested through the use of the `var` keyword, then a pair is returned, with the polynoi and the subspace second. The polynomial is the unique monic polynomial whose coefficients provide a ı linear dependence on the first $k$ powers.

For less convenient, but more flexible output, see the helper method “_cyclic_subspace” in this module.

EXAMPLES:

```
sage: A = matrix(QQ, [[5,4,2,1],[0,1,-1,-1],[-1,-1,3,0],[1,1,-1,2]])
sage: v = vector(QQ, [0,1,0,0])
sage: E = A.cyclic_subspace(v); E
Vector space of degree 4 and dimension 3 over Rational Field
Basis matrix:
[ 1  0  0  0]
[ 0  1  0  0]
[ 0  0  1 -1]
sage: F = A.cyclic_subspace(v, basis='iterates'); F
Vector space of degree 4 and dimension 3 over Rational Field
User basis matrix:
[ 0  1  0  0]
[ 4  1 -1  1]
[23  1 -8  8]
sage: E == F
True
sage: p, S = A.cyclic_subspace(v, var='T'); p
T^3 - 9*T^2 + 24*T - 16
sage: gen = polygen(QQ, 'z')
sage: p, S = A.cyclic_subspace(v, var=gen); p
z^3 - 9*z^2 + 24*z - 16
sage: p.degree() == E.dimension()
True
```

The polynomial has coefficients that yield a non-trivial relation of linear dependence on the iterates. Or, equivalently, evaluating the polynomial with the matrix will create a matrix that annihilates the vector.

```
sage: A = matrix(QQ, [[15, 37/3, -16, -104/3, -29, -7/3, 35, 2/3, -29/3, -1/
...                   [ 2, 9, -1, -6, -6, 0, 7, 0, -2, 0],
...                   [24, 74/3, -29, -208/3, -58, -14/3, 70, 4/3, -58/3, -2
...                   [-6, -19, 3, 21, 19, 0, -21, 0, 6, 0],
...                   [2, 6, -1, -6, -3, 0, 7, 0, -2, 0],
...                   [-96, -296/3, 128, 832/3, 232, 65/3, -279, -16/3, 232/
...                   [0, 0, 0, 0, 0, 0, 3, 0, 0, 0],
...                   [20, 26/3, -30, -199/3, -42, -14/3, 70, 13/3, -55/3, -
...                   [18, 57, -9, -54, -57, 0, 63, 0, -15, 0],
...                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 3]])
sage: u = zero_vector(QQ, 10); u[0] = 1
sage: p, S = A.cyclic_subspace(u, var='t', basis='iterates')
sage: S
Vector space of degree 10 and dimension 3 over Rational Field
User basis matrix:
[   1    0    0    0    0    0    0    0    0    0]
[  15    2   24   -6    2  -96    0   20   18    0]
[  79   12  140  -36   12 -560    0  116  108    0]
sage: p
t^3 - 9*t^2 + 27*t - 27
sage: k = p.degree()
sage: coeffs = p.list()
sage: iterates = S.basis() + [A^k*u]
sage: sum(coeffs[i]*iterates[i] for i in range(k+1))
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

```
sage: u in p(A).right_kernel()
True
```

TESTS:

A small case.

```
sage: A = matrix(QQ, 5, range(25))
sage: u = zero_vector(QQ, 5)
sage: A.cyclic_subspace(u)
Vector space of degree 5 and dimension 0 over Rational Field
Basis matrix:
[]
```

Various problem inputs. Notice the vector must have entries that coerce into the base ring of the matrix, polynomial ring generator must have a base ring that agrees with the base ring of the matrix.

```
sage: A = matrix(QQ, 4, range(16))
sage: v = vector(QQ, 4, range(4))

sage: A.cyclic_subspace('junk')
Traceback (click to the left of this block for traceback)
...
```

Try this with another function!

```
vector?
```

## Finding the source

There is one more source of help you may find useful in the long run, though perhaps not immediately.

- One can use *two* question marks after a function name to pull up the documentation *and* the source code for the function.
- Again, to see this help, you can either evaluate a cell like below, or just move your cursor after the question mark and press tab.

The ability to see the code (the underlying instructions to the computer) is one of Sage's great strengths. You can see *all* the code to *everything*.

This means:

- *You* can see what Sage is doing.
- Your collaborators can see what Sage is doing.

- And if you find a better way to do something, then you can see how to change it!

```
M.cyclic_subspace??
```

This is, to a large extent, the whole point of this course and the upcoming Sage Days! But we won't say much more about it now.

## Annotating with Sage

Whether one uses Sage in the classroom or in research, it is usually helpful to describe to the reader what is being done, such as in the description you are now reading.

Thanks to the mini-word processor TinyMCE and a TeX rendering engine called mathjax, you can type much more in Sage than just Sage commands. This math-aware setup makes Sage perfect for annotating computations.

To use the word processor, we create a *text cell* (as opposed to a *input cell* that contains Sage commands that Sage evaluates).

To create a text cell, do the following.

- First, move the cursor between two input cells, until the thin blue line appears.
- Then hold the Shift key and click on the thin blue line.
- (So to create an input cell, one merely clicks, but one "Shift-Click"s to create a text cell.)

Try inserting a text cell between the input cells below.

```

```

Someone I know loves $\int_{-\infty}^{\infty} e^{-x^2} dx$.

```

```

```

```

TinyMCE makes it easy for format text in many ways. Try experimenting with the usual **bold** button, underline button, different text fonts and colors, ordered and unordered lists, centering, and so on. Some of the shortcut keys you are familiar with from other word processors may also work, depending on your system.

There are two other things you can do which take advantage of the worksheet being on the web.

- It is easy to link to other helpful websites for additional information.
    - While in the editor, highlight a word or two, and then click on the little chain link toward the

bottom right of the buttons.
    - You can now type in a web address to link to.
    - Be sure to prepend http:// to the address. Normally, one should also select it to appear in a new window (so the Sage session isn't interrupted).
- You may have already noticed that some of the descriptions above had typeset mathematics in them. In fact we can add nearly arbitrary LaTeX to our text cells!
    - For instance, it isn't too hard to add things like

$$\zeta(s)=\sum_{n=1}^{\infty}\frac{1}{n^s}=\prod_p \left(\frac{1}{1-p^{-s}}\right).$$

    - One just types things like "$$\zeta(s)=\sum_{n=1}^{\infty}\frac{1}{n^s}=\prod_p \left(\frac{1}{1-p^{-s}}\right)$$" in the word processor.
    - Whether this shows up as nicely as possible depends on what fonts you have in your browser, but it should be legible.
    - More realistically, we might type "$f(x)=x^2$" so that we remember that $f(x)=x^2$ in this worksheet.

Here is a simpler example.

```
f(x)=x^2
f(9)
```

```
81
```

If $f(x)=x^2$, then $f(9)=81$.

Or, our matrix:

$$M=\begin{pmatrix}1 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 3\end{pmatrix}$$

It is simple to edit a text cell; simply double-click on the text. Try double-clicking on this text to edit this text cell (or any text cell) to see how we typed the mathematics!

```

```

Of course, one can do much more, since Sage can execute arbitrary commands in the Python programming language, as well as output nicely formatted HTML, and so on. If you have enough programming experience to do things like this, go for it!

```
html("Sage is <a style='text-decoration:line-through'>somewhat</a>
<b>really</b> cool! <p style='color:red'>(It even does HTML.)</p>")
```

`Sage is ~~somewhat~~ **really** cool!`

`(It even does HTML.)`

## Homework

You wouldn't want to leave without homework, right? You'll want to search the reference, use help, and perhaps learn a little programming for these.

Please come to the next class with *code*, not just answers. The point is to learn how to do this consistently; many you could do "by hand", after all.

1. Find three topics in the combinatorics directory you didn't know about, but should. Be ready to show someone else how to compute it.
2. Find a tutorial in the Sage documentation written by Anne Schilling, one of the presenters at Sage Days 60.
3. What was your favorite function in calculus? Can you calculate its integral?
4. Solve one of the harder Sudoku puzzles linked, using Sage.
5. How many graphs are there of order five? (Up to isomorphism.)
6. How many graphs are there of order five and size six?
7. Do Project Euler's problem seven.
8. Compute the sum of the primes below $n$, where $n = 30, 100, 1000$, and two million. For the last few, you may want to learn about a "list" from a good Python tutorial, such as this one, this one, or another one from this list.
9. *Use Sage* to calculate the determinant of the diagonal matrix of ranks one through ten with all ones on the diagonal. (You know the answer, but get Sage to do it for you!)
10. Look up the list of programs which Sage uses internally or has interfaces to. Have you used any of them?