1. Sameer desu. Indojin desu. Yoroshiku onegaishimasu. Before I start with my presentation let me tell you a quick story. In Japan a wife respectfully calls her husband otosama or otosan. But In india women have a very special name for their husbands. Last year we had some japanese guests at our house. They would always laugh when my mother called my father otosan in our language. My mother could not understand WHY they would laugh, so on the third day she finally asked them, why do you always laugh when I call my husband? SO they said, ok we'll telll you but can you call your husband again?' and so she ok ok I'll call my husband, just a moment... and then she goes 'ahooo'. Unfortunately they couldn't stop lauging so still doesn't know what it means.

2. I come from India.

3. The land of kare (kha-reh – curry in JP). Which is also home...

4. The mighthy Ganga, also known as the mother river.

5. To the Snowy Himalayas.

6. And the great Taj Mahal.

7. My country is now on the cusp of an industrial and scientific revolution, and given our culture and heritage dating back to thousands of years, just like other Asian countries like Japan, we need to master Western science and yet preserve our culture and heritage.

8. Japan is the only Asian country that has managed to realize this dream of preserving ones roots and gaining a mastery of modern science and technology. Therefore, I am extremely excited to be in the midst of the very civilization that is a role model for all Asian countries.

9. I am from the city of Pune, which is a city with about 6 million people. It is also known as the Oxford of the East, because of the 2 million strong student population studying in almost 700 colleges across the city. I am myself an undergraduate student of Computer Engineering at the University of Pune.

10. My name is Sameer Deshmukh, also known as v0dro on github and twitter.
11. Most of family has consisted of doctors.

12. In fact, my great grandfather <click> was a doctor, my grand father <click> is a Doctor, both my parents are doctors <click>, yet for some reason I turned out to be an engineer and looked like this <click> not too long ago.

13. In my spare time I also play the bass guitar for my band Cat Kamikazee. You can check out our music on sound cloud, it's some really cool experimental and instrumental stuff. I was here at the conference yesterday and I have been pretty....

14. terrified by the massive size of this venue and the kind of audience that it attracts. Ever since the speaker's schedule was released I couldn't help thinking that a 23 year old like me with like zero experience is speaking after Tanaka-san, the creator of Narray, and before Murata-san, the maintainer of BigDecimal. And I think, that the cat in

15. This picture over here perfectly describes what I'm feeling like right now. I feel very honoured in sharing the stage with such senpai and am very grateful to the organizers of Ruby Kaigi for having me over.

16. My city Pune is also home to the Pune Ruby Users Group. We have montly meetups and I give talks over there almost every month. It is one of the most active Ruby user groups in India and annually conducts the Deccan Ruby Conference in Pune each year on a not for profit basis.

17. I am a core team member of the Ruby Science Foundation. The Ruby Science Foundation, or SciRuby in short, is an open source software organization dedicated to making Ruby a viable language for scientific computing and data science by developing high quality open source tools for this purpose. Make sure you

follow us on twitter to stay up to date with latest news. SciRuby has also sponsored my travel and stay for this conference and I am extremely grateful to them for helping me make the most of this opportunity. This trip would not have been possible without their contribution.

18. I have been associated with SciRuby for almost a year and a half. My first contact with them was during Google Summer of Code 2015, when I was selected as a student intern to work on some SciRuby projects. Later that year I also received the Ruby Association Grant for further development of Ruby's numerical computing infrastructure. This year, for Google Summer of Code 2016, I was the GSOC admin for SciRuby and also co-mentored a student.

19. My talk for today is titled 'Data Analysis in Ruby with daru'. In this talk, I will try to shed some light upon daru...

20. ... which is an acronym for Data Analysis in Ruby, and is a library that we've been working on primarily for simplifying data analysis tasks in Ruby. Interestingly In India's national language Hindi

21. , daru means alcohol. And the only thing that alcohol reminds me of when in Japan is the amazing sake that I've been consuming ever since I arrived here 4 days ago. I hope that daru the library becomes as important to Ruby as sake is important to Japan.

22. To summarize daru in a sentence, it is a library for analysis, cleaning, manipulation and visualization of data.

23. Daru gives you some powerful features like advanced data indexing capabilities, reading and writing data from various sources like CSV files, SQL databases, Excel files and even ActiveRecord. It also has a whole lot of statistics methods and works very well with 'wild' data. By 'wild' data I mean data that's missing or is not in a form that can be analyzed by conventional means. Daru also has many statistics, querying and sorting functions, all of which work very well with missing data or 'wild' data.

24. Daru primarily performs its functions through two major data structures, first of which is called the Daru::Vector. The Vector is a one dimensional heterogenous Array-like data structure. Each element in a vector can be independently labelled via daru's indexing functionality. We will go through the indexes in a short while.

25. The next data structure is called Daru::DataFrame.This is a 2D spreadsheet like data structure that can be indexed on both rows and columns. It also leverages the indexing functions that are used by Daru::Vector for the labelling.

26. Now, generating informative visualization is one of the most important tasks for any data analyst for generating quick insights into data. Therefore, daru creates some very intuitive abstractions over libraries like nyaplot, gnuplotrb and gruff for plotting. I will now make a small demonstration of daru, but before I do so, let me introduce you to a Ruby library that is particularly useful for all sorts of interactive computation and data analysis.

27. The library is called the iruby notebook.

28. Iruby is basically a browser-based Ruby REPL shell for interactive computing. It functions by running an instance of pry in the backend, and providing a good looking development front end to the user right in their browser instead of having to look at a rather bland and sometimes difficult to read command line.

29. This is what an iruby notebook looks like. Notice that it's running in my Google Chrome browser. It has two major components, the first one is the input cell that provides a syntax highlighted text box for writing your Ruby script, and the second is an output cell that generates good looking output from the executed Ruby code, like the Matrix that you can see over here. Let me now shift over to an iruby notebook to show you a demo of the basic capabilities of daru.

1. This is an iruby notebook. As I said earlier, it runs in the browser and supports input and output cells for code. For documenting notebooks you can also specify markdown in the cells.
2. In this first cell, I am requiring daru.

30. One of the most important tasks faced by any data analyst is that of cleaning data so that useful information can be derived from this data by using conventional means of statistical inference and visualization.Various sources say that almost 60% of a data analyst's time is spent in cleaning data. Daru tries to make life easier for data analysts by providing powerful functionality for data cleaning. All operations defined in daru support missing data, including sorting, joining and querying of data frames.
31. Now, daru as a library by itself provides little more than a container for holding and manipulating data. What makes it truly shine is that it can be coupled with a variety of other Ruby scientific libraries like statsample, statsample-glm and statsample-timeseries for statistical analysis, mixed_models for computing mixed models and daru-td for importing and manipulating data from a treasure data server. As an example, let me show a demonstration of daru being used to perform some statistical analysis.
32. For this purpose let's take a use case where we're coupling the statsample-glm gem with daru to perform some basic machine learning tasks by using logistic regression.
33. To briefly talk about the statsample-glm gem, it is a gem for computing a wide variety of regression models in Ruby.
34. It also supports a formula language exactly like R or patsy for specifying the response and predictor variables of the regression. So basically in this example over here, y is the response variable, and a,b,c and d are the predictor variables. This formula will be internally expanded to the below equation and this equation will be used to read the respective columns from a Daru::DataFrame container, perform operations on them using the operations that are specified in the expanded equation, and ultimately perform the regression to predict the beta constants.
35. Let me demonstrate the usage of daru with statsample-glm with a use case derived from a Kaggle data set about the outcomes of animals in some animal shelters. Let's shift to an iruby notebook and have a look at how this can be done.
36. All of you seem weary from having all that daru. So now, let me shift my focus to something that does not have so much daru in it..
37. I will now take some time to talk about some new ideas that SciRuby is experimenting with, which we feel will make Ruby a much more useful tool than it is now.
38. Arthur Clarke once said that any suffiencietly advanced technology is indistinguishable from magic, and so we want it to be with these new ideas about the future of Ruby that we are working towards making a reality.
39. I think this quote from Steve Jobs holds true to this conference and what the people who come here stand for.
40. As I'm sure most of you are aware, since you might have attended Sutuo-san's (Soo-toh) talk in the morning on creating C bindings for Ruby, there are majorly 3 ways in which C library functions can be called from Ruby, those being FFI, Rice and having to manually write C code using the CRuby extension API. Now, the problem with the first two is that they don't really allow much flexibilty when there is a lot of glue code involved in making the C extension work with the Ruby interpreter. The third way is most flexible, but honestly, I do not enjoy writing a low level language like C after writing Ruby, and evertime I have to write a C extension, I feel there is some evil presence..
41. Like Freiza Standing behind my back and forcing me to write that C code. I simply don't enjoy writing C extensions for Ruby, and something tells me that most of you don't either, except maybe the Ruby commiters.

42. So let me explain to you why I do not like to write Ruby extensions with a small example. Consider this very simple recursive Ruby method called 'factorial' that is being used for calculating the factorial of a number. Now for reasons of your own you decide to port this code for calculating a factorial to a C extension.

43. You start by writing a simple C function called calc_factorial, and under that you write a function called cfactorial that calls the calc_factorial method with all these calls to macros from the CRuby API. Now as if this wasn't hard and confusing enough...

44. You also need to write an intiliazation function that will call certain functions from the Cruby API that will initialize a class called Fact and an instance method inside that class called factorial, which will in reality call the cfactorial method defined in the earlier slide.

45. Finally, you can call these functions from Ruby by intializing a class called 'Fact' and calling the factorial method that is defined inside it.

46. Now, being a Rubyist, and having grown very comfortable with the 'Ruby way' of doing things, I find this approach extremely tiring and cumbersome for various reasons. The most important one being that we Rubyists are forced to move away from the very reason for which we embraced Ruby in the first place, that is not needing to care about small things.

47. The best solution to writing C extensions quickly, in my opinion, is a new language that I'm currently working on, called Rubex, that let's you write code that looks very much like Ruby, yet compiles down to C and thus gives you the goodness of both Ruby and C.

48. Rubex is basically a super set of Ruby that is inspired by Crystal. It let's you write C extensions that look exactly like Ruby, without having to go through all the pain that you typically need to go through when writing C extensions.

49. Here's a small example of what Rubex can do. The factorial example written in C that we saw earlier can simply be written this way with Rubex. As you can see, the only difference between the pure Ruby method and the Rubex method is that you can explicitly specify data types, which will be picked up by the Rubex compiler and then be compiled to C data types. This code will be read by the Rubex compiler and translated into C code that interfaces with the Ruby interpreter. The Rubex compiler will take care of the calls to the CRuby interpreter API, and thus, Rubysists will not need to worry about the intricacies of writing C code that can be called from Ruby.

50. Here's another example. In this example, I am using some Rubex specific keywords that will be picked up by the Rubex compiler for generating C code. So for example, the Static Array keyword lets you create a C array of a specific length and a specific data type. In this case I'm passing i32 in the first argument and 8 in the second argument. I32 is Rubex's way of specifying a 32 bit integer data type. Below that I'm initliazing a 32 bit integer called 'i' to 0, and under that, you can see this each loop with an argument. The argument in this case is the size of the array 'a', since C does not know the sizes of arrays by itself unlike Ruby. This each loop will taken apart by the Rubex compiler and be converted to a C for loop. Thus, as you can see, Rubex tries to be as similar to Ruby as possible, but with some add ons that allow it to be compiled directly to C code. Additionaly, it will also let you intermingle Ruby code with Rubex code, and will compile everything to the equivalent C API calls of the Cruby interpreter.

51. Rubex is still currently in alpha stage, but you can head over this repo and track the progress of the project.

52. The next project I'll be talking about pertains to scientific computation on JRuby. Now, currently C is the most common language used by scientists for conducting high speed computations. In fact, most of the common scientific libraries like ATLAS, LAPACK, FFTW, GSL were either first written in C or were ported to C from FORTRAN. C's importance is reflected by the fact that the two most common linear algebra

libraries for Ruby, that is

53. Nmatrix and narray, are both written for Cruby and internally consist of almost 90% C or C++ code. But it so happens that Jruby is also a very viable platform for conducting all sorts of computation.
54. So this year we came up with a Java backend for Nmatrix which runs on Jruby. It supports the exact same API as that of Nmatrix, the only difference being the Java backend.
55. It uses the Apache Commons math library for storing various types of data as Java arrays and uses jBLAS for performing computations.
56. If you want to have a look at the source code you can go to this link.
57. The next idea that I'll talk about is symbolic computation in Ruby. Symbolic computation is a scientific area that refers to the study and development of algorithms and software for manipulating mathematical expressions and other mathematical objects. A symbolic computation library emphasizes exact computation with expressions containing variables that have no given value and are manipulated as symbols.
58. For example, consider this expression.
59. It is now possible to represent an equation symbolically in Ruby using the symengine gem. In this code, I am assigning the variables x, y and z as symenging symbol objects. Any legitimate Ruby operator can then be applied on these objects while treating them exactly like the equation that we saw on the earlier slide. Calling the expand and then to_s methods on this equation will then expand it algebraically and produce output as you can see over here. You can also compare two equivalent equations with symengine.
60. You can have a look at this project on this link.
61. Finally we have another project going on that will make it possible for Ruby to be used in space research.
62. This project is about creating a Ruby wrapper over NASA's SPICE C library. SPICE was first conceptualized in by NASA to correctly analyze and interpret data consisting of readings taken by instruments on space missions. Today SPICE is routinely used in all phases of planetary missions, and sometimes in other space science missions as well.
63. To demonstrate the Ruby wrapper over SPICE, have a look at this code where we want to calculate the distance between the earth and moon at this point of time. <describe code>. This can be applied to any celestial body that is supported by the NASA database.
64. The library is available on this link.
65. As a parting note I would like to inform you that I am carrying some cool sciruby stickers with me that are up for grabs. Catch me anytime during or after the conference if you want one of these.
66. Also, many thanks to all these people who contributed their expertise to make this talk possible.
67. And a very big thank you to the Ruby Kaigi organizers for trusting me with delivering this talk and having me over for this conference.
68. END – Sameer desu. Indojin desu. Yoroshiko onigaishimasu.