

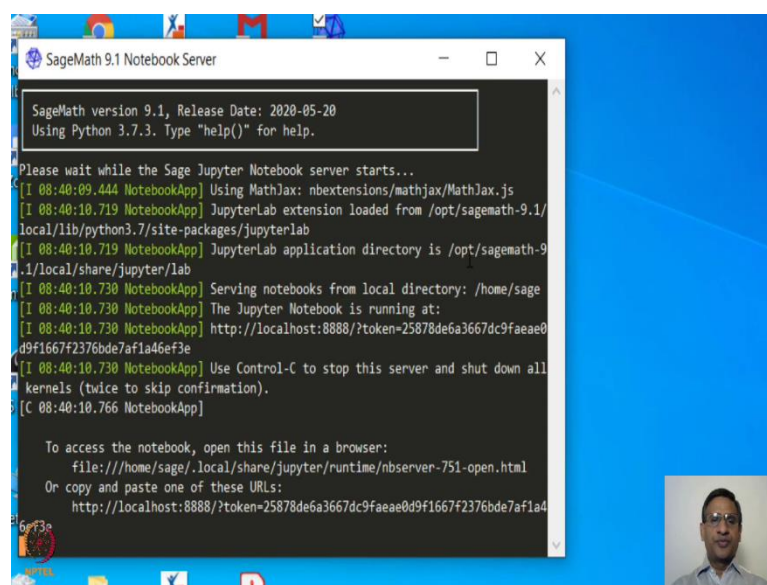
Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture – 16
Exploring integers in SageMath

Welcome to the 16th lecture on Computational Mathematics with SageMath. Now, onwards we will start using SageMath and we will explore mathematical concepts using SageMath. So, I hope all of you have managed to install Sage in your system or at least you have figured it out how to use it online. In case you have problem in installing in your system you can use it online that I explained in the last video.

So, let us get started. I have already installed in my system and in my computer system it has created three icons on my desktop. One is SageMath Notebook another one is SageMath 9.1 another one is SageMath Cell. So, we will be using SageMath Notebook.

(Refer Slide Time: 01:07)



```
SageMath 9.1 Notebook Server

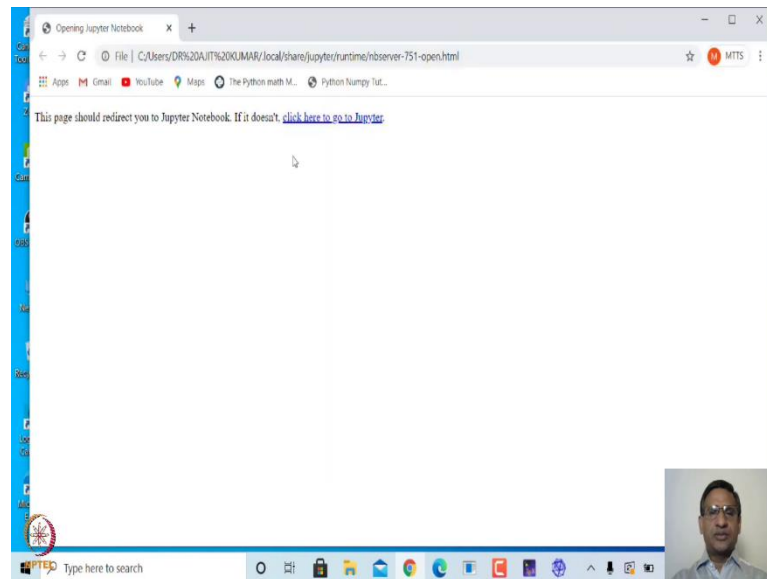
SageMath version 9.1, Release Date: 2020-05-20
Using Python 3.7.3. Type "help()" for help.

Please wait while the Sage Jupyter Notebook server starts...
[I 08:40:09.444 NotebookApp] Using MathJax: nbextensions/mathjax/MathJax.js
[I 08:40:10.719 NotebookApp] JupyterLab extension loaded from /opt/sagemath-9.1/
local/lib/python3.7/site-packages/jupyterlab
[I 08:40:10.719 NotebookApp] JupyterLab application directory is /opt/sagemath-9
.1/local/share/jupyter/lab
[I 08:40:10.730 NotebookApp] Serving notebooks from local directory: /home/sage
[I 08:40:10.730 NotebookApp] The Jupyter Notebook is running at:
[I 08:40:10.730 NotebookApp] http://localhost:8888/?token=25878de6a3667dc9faae0
d9f1667f2376bde7af1a46ef3e
[I 08:40:10.730 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[C 08:40:10.766 NotebookApp]

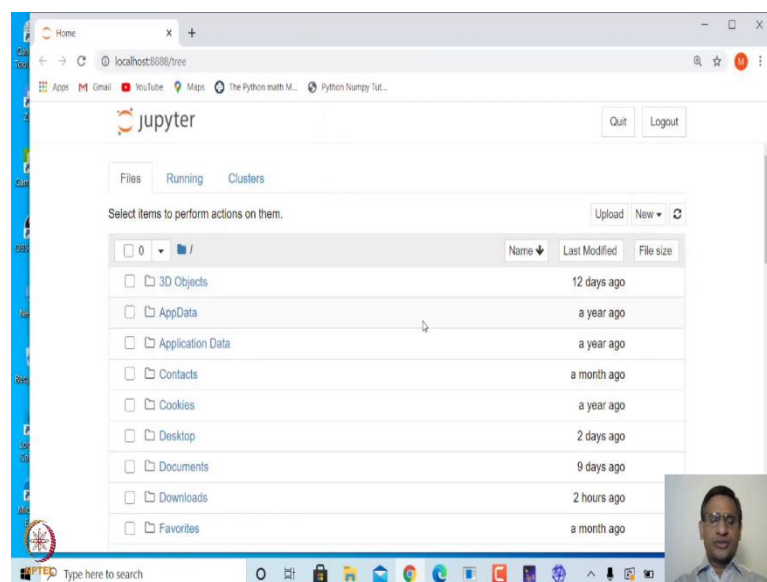
To access the notebook, open this file in a browser:
file:///home/sage/.local/share/jupyter/runtime/nbsrvr-751-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=25878de6a3667dc9faae0d9f1667f2376bde7af1a4
6ef3e
```

So, let us double click on this and once you double click it will start SageMath server in a Jupyter Notebook.

(Refer Slide Time: 01:15)

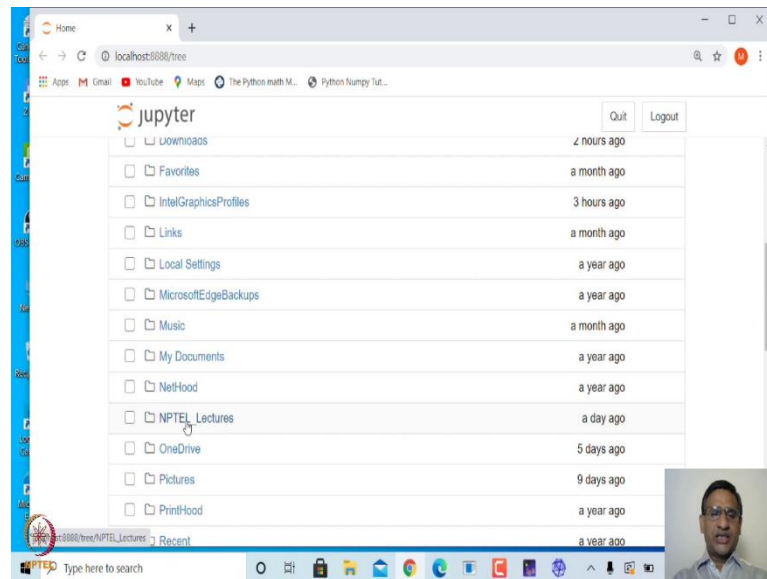


(Refer Slide Time: 01:17)



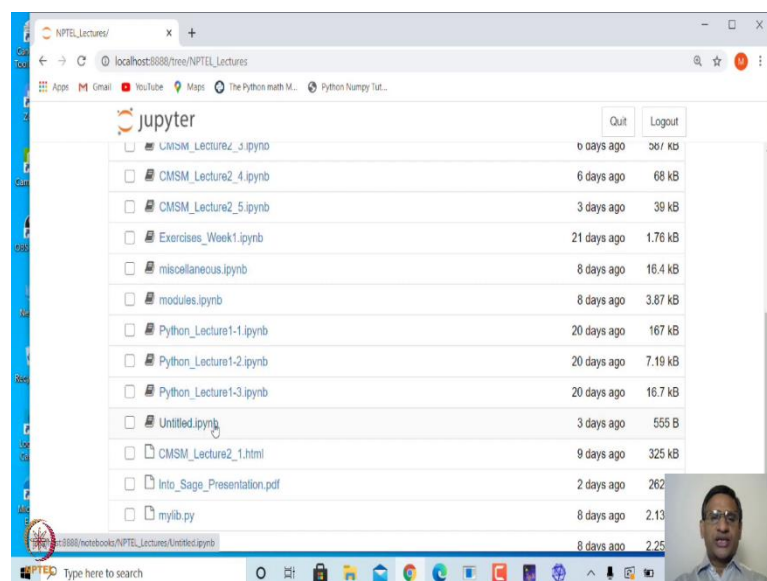
Initially, it may take little time, but subsequently it will be faster.

(Refer Slide Time: 01:27)

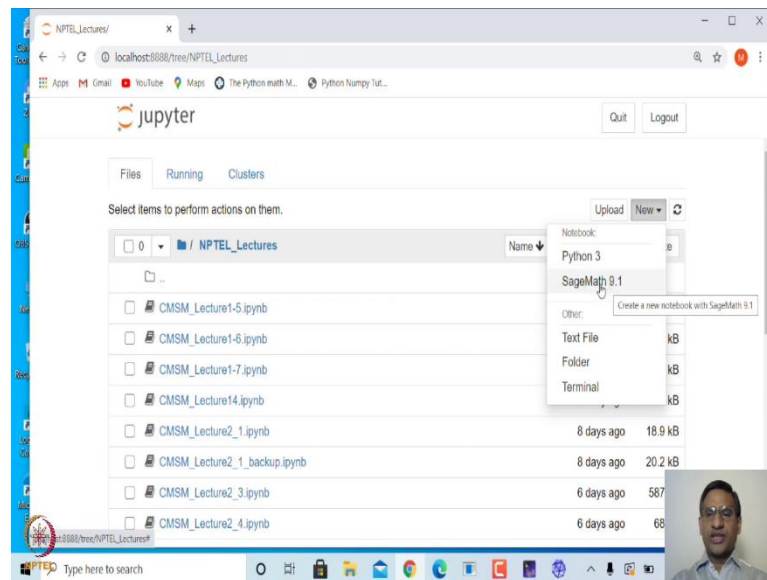


So, this is the local host for SageMath server. You can see your files directories and all other things etc. So, let me go to one directory which I have created as NPTEL Lectures.

(Refer Slide Time: 01:40)

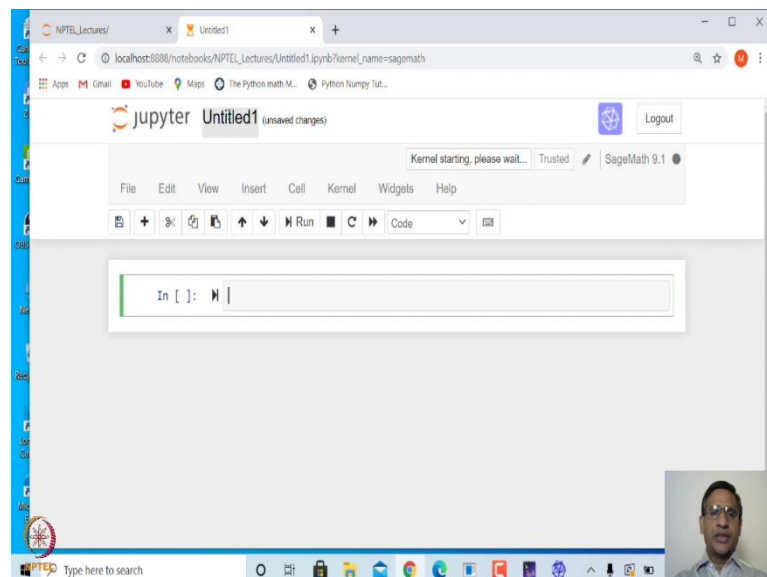


(Refer Slide Time: 01:43)



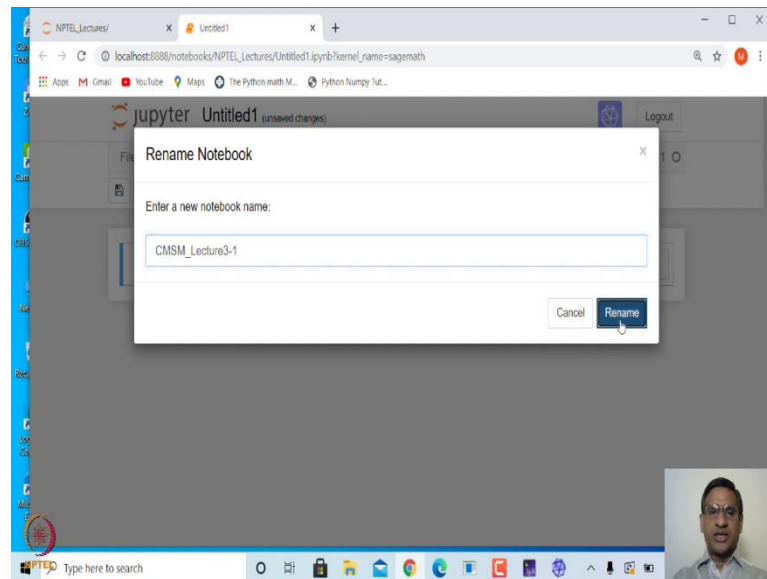
So, if I click on this there are files which we created last week. So, now, let us go to new and click on new. In case you want to just do Python programming that also option is available. So, you can click on Python 3. We will be using SageMath 9.1. So, let us click on SageMath 9.1.

(Refer Slide Time: 02:05)



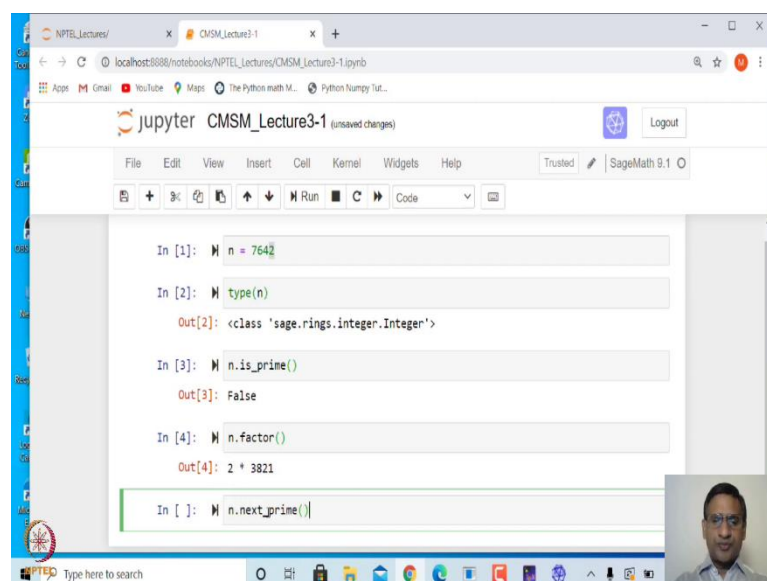
Once you click SageMath 9.1 it will create an untitled Jupyter Notebook.

(Refer Slide Time: 02:14)



So, we can rename this. So, let us rename this. I will call this as computational mathematics with SageMath and let me say this is Lecture 3-1. So, that is the name I am giving to this worksheet.

(Refer Slide Time: 02:33)



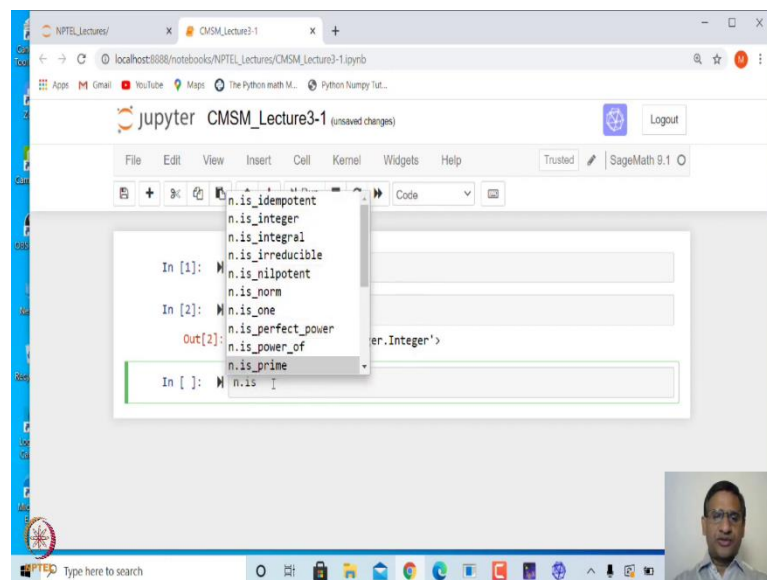
So, by default it will create an extension i.e. .ipynb that is the Jupyter Notebook default extension. Now, this is where you can type all the commands and you can run this exactly in a same way as we did when we wrote Python programs using Jupyter Notebook.

So, this SageMath as I said in the introductory video is very useful and you can explore most of the mathematical concepts using SageMath. So, let us get started. Suppose we want to let us say explore integers. So, how do we do that?

We will define let us say n is equal to some integer. Let us say 7642. So, let us store this. Now, when we have created $n = 7642$, actually n is an object of integer class. So, if I ask for what is type of n , it will tell you that it is coming from `sage.rings.integer.Integer`.

Now, all the methods that you want to apply on n can be obtained using dot tab operator because this as I said this SageMath is Python based. So, all the functionality of Python will also be available here.

(Refer Slide Time: 04:17)



So, if I say n dot and then press tab you will see all the methods that can be applied to this object n . So, for example, you can go through this list and explore maybe few of them. So, let us see suppose we want to look at this integer and I can ask whether this integer is prime or not a prime number.

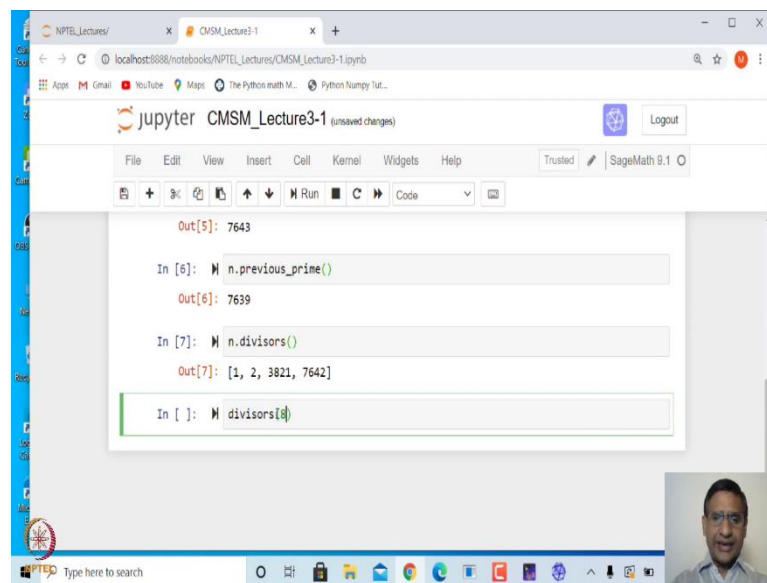
So, of course, we know that this is even integer. So, it cannot be prime. So, but we can still check whether it is a prime or not. How do we check that? So, if you again press the tab you will see an option.

There is an option called `is_prime()` and then you give this empty parenthesis because this is a method you want to access this method known as underscore prime from this

integer class and you want to apply on an object n . So, this is what you have to do. So, this says that this is not a prime number.

So, if it is not a prime number you can ask various questions. So, for example, I can say $n.factor()$. So, what are the factors of this? So, it says that it has only two factors, namely, 2 and 3821. You can ask for $n.next_prime()$. So, that will give you a number which is prime and it is immediately after this.

(Refer Slide Time: 05:58)



```
Out[5]: 7643

In [6]: n.previous_prime()
Out[6]: 7639

In [7]: n.divisors()
Out[7]: [1, 2, 3821, 7642]

In [ ]: divisors[n]
```

So, let us run this it gives you 7643. So, it was 7642 and the next number next integer is a prime number. So, that is very nice. So, you can see here for example, $next_prime()$ or $is_prime()$. First of all $is_prime()$ checks primality of this number n .

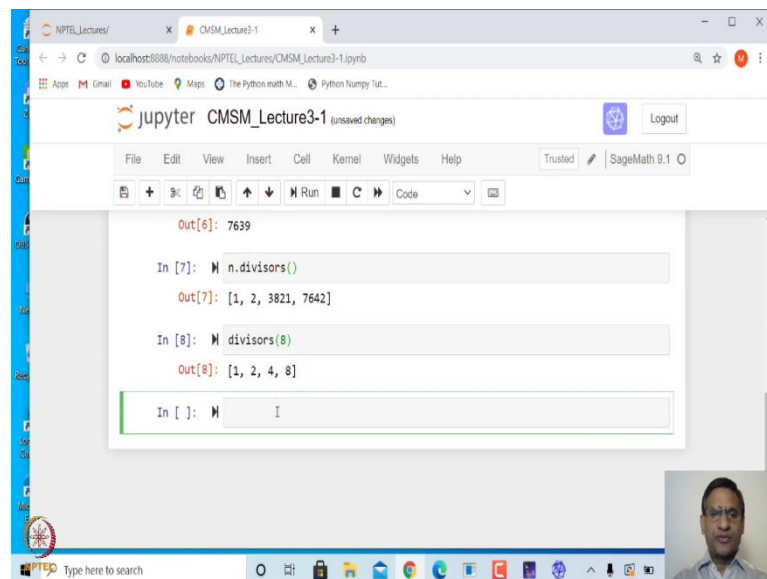
So, whenever you want to check something let us say you want to check whether some integer is even or odd or check prime, not prime, check matrix which is singular, non-singular etc, generally the name of the method will start with 'is'.

And since it is combination of is and prime a general philosophy in SageMath is to combine these two using underscore (`_`) and in some other software it may be some other philosophy. But, in SageMath two keywords are combined to make a single method or function using underscore.

So, this is the next prime. If I say `n` dot if I want to find previous prime I can say `pre` and you can after typing let us say `pre` or just `pr` you can press tab, then it will show you all the options that can exist with two characters `pr`. So, there is an option called previous prime.

So, if I just use `previous_prime()` you get this previous prime which is 7639. So, 7639 and 7643 are two consecutive primes. Similarly, you can ask for let us say divisors. We already found out factors, but if I say `n` dot divisors, then it will give you all the divisors of this say including 1 and the number itself. So, all the divisors it will give you. It will give you divisors in case it is repeated. For example, if I say divisors of let us say 8.

(Refer Slide Time: 08:04)

A screenshot of a Jupyter Notebook interface. The browser address bar shows the URL 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-1.ipynb'. The notebook title is 'CMSM_Lecture3-1 (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running code, and other functions. The code area shows three input cells: the first contains 'n' and its output is '7639'; the second contains 'n.divisors()' and its output is '[1, 2, 3821, 7642]'; the third contains 'divisors(8)' and its output is '[1, 2, 4, 8]'. A fourth input cell is partially visible with 'In []: ' and a cursor. A small video feed of a man is visible in the bottom right corner of the notebook window.

```
Out[6]: 7639

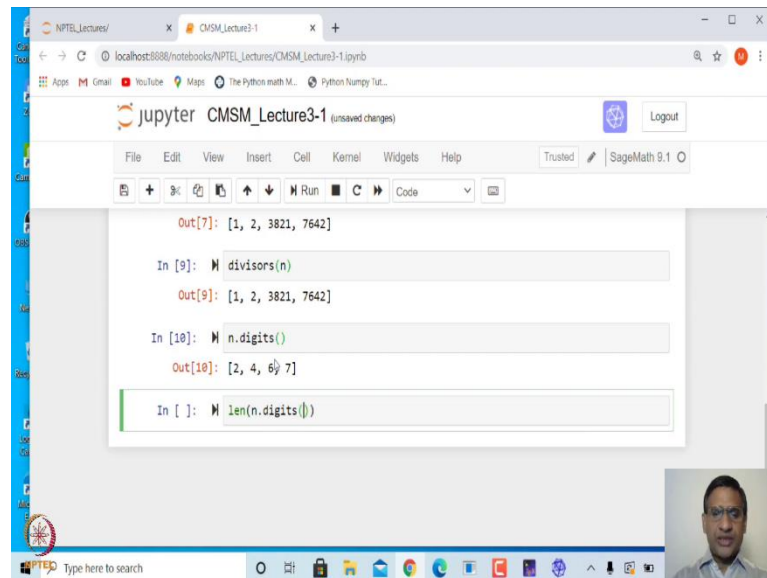
In [7]: n.divisors()
Out[7]: [1, 2, 3821, 7642]

In [8]: divisors(8)
Out[8]: [1, 2, 4, 8]

In [ ]: 
```

So, divisors of 8 are 1, 2, 4, 8. So, it will give you only the divisors as a list right. So, you can see here `n.divisors()` can be used or we can use `divisors` in the bracket `n` (`divisors(n)`) both are the same thing.

(Refer Slide Time: 08:22)



A screenshot of a Jupyter Notebook interface titled 'CMSM_Lecture3-1'. The notebook is running on a local host. The code in the cells is as follows:

```
Out[7]: [1, 2, 3821, 7642]

In [9]: M.divisors(n)
Out[9]: [1, 2, 3821, 7642]

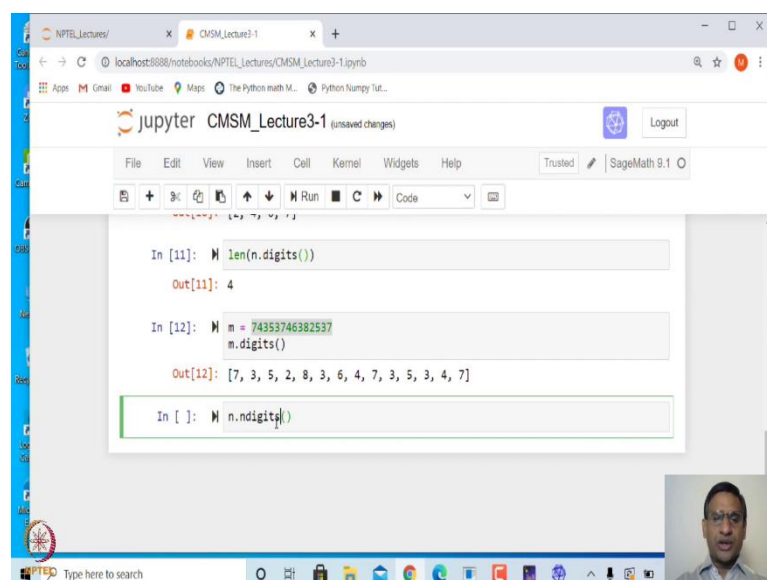
In [10]: M.n.digits()
Out[10]: [2, 4, 6, 7]

In [ ]: M.len(n.digits())
```

The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing 'SageMath 9.1'. A small video feed of a person is visible in the bottom right corner.

So, its divisor is also a function at the same time it is also a method inside integer class. So, next let us say we want to find out how many digits are there. So, we can say `n.digits()` and if I say `n.digits()` then what will it create? It will create a list of digits; it will separate all the digits and create a list. So, and it will start with unit place. So, the 2, 4, 6, 7 our number was 7642. So, you can separate the digits starting from units place onwards and you can count how many digits are there.

(Refer Slide Time: 09:11)



A screenshot of a Jupyter Notebook interface titled 'CMSM_Lecture3-1'. The notebook is running on a local host. The code in the cells is as follows:

```
In [11]: M.len(n.digits())
Out[11]: 4

In [12]: m = 74353746382537
          m.digits()
Out[12]: [7, 3, 5, 2, 8, 3, 6, 4, 7, 3, 5, 3, 4, 7]

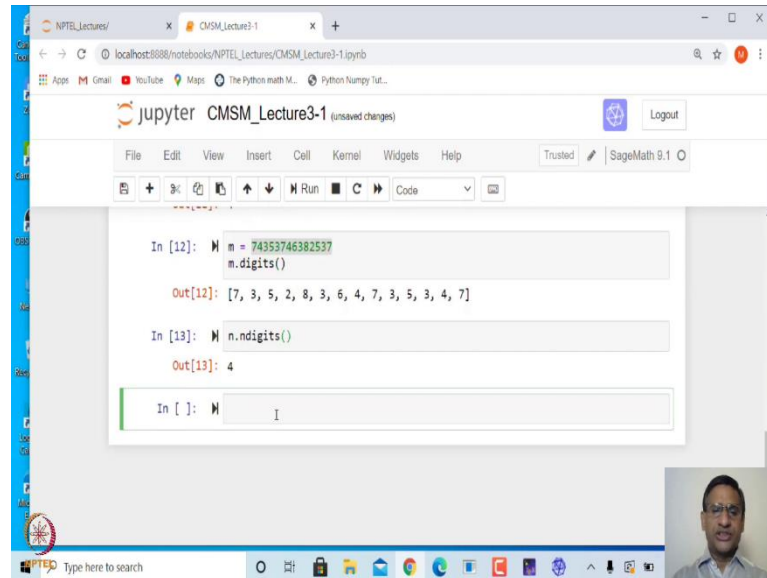
In [ ]: M.ndigitp()
```

The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing 'SageMath 9.1'. A small video feed of a person is visible in the bottom right corner.

I can say that length of `n.digits()` will give me number of digits. So, it has 4 digits and you can work with any big number. So, if I say let us say `m` is equal to some large number and then if I ask for what is `m.digits()`, then it will separate all the digit digits and you

can count the length. However, you can find the length of digits length of or number of digits in an integer using another option called *n.ndigits()*.

(Refer Slide Time: 09:49)



A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-1.ipynb'. The notebook title is 'CMSM_Lecture3-1 (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code area shows the following interactions:

```
In [12]: m = 74353746382537
         m.ndigits()

Out[12]: [7, 3, 5, 2, 8, 3, 6, 4, 7, 3, 5, 3, 4, 7]

In [13]: n.ndigits()

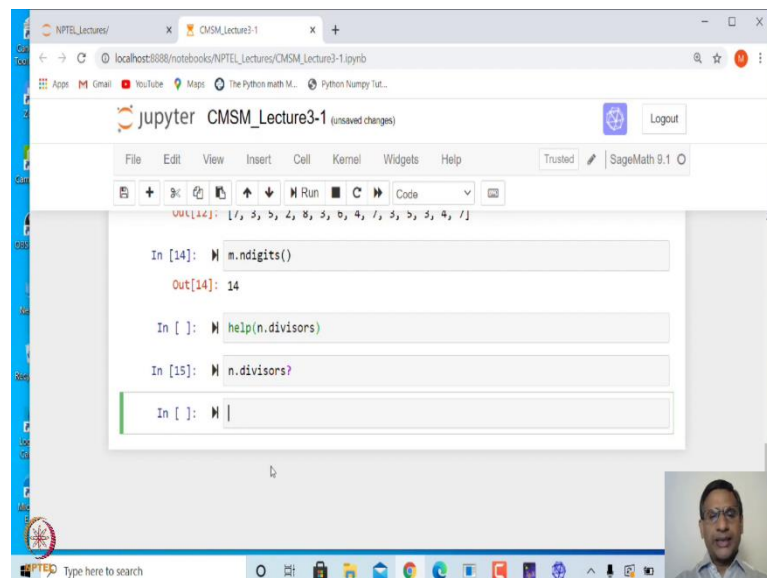
Out[13]: 4

In [ ]: |
```

The bottom of the screen shows a Windows taskbar with various application icons and a search bar. A small video feed of a man is visible in the bottom right corner.

So, if I say n digits it will tell me number of digits.

(Refer Slide Time: 09:53)



A screenshot of a Jupyter Notebook interface, similar to the previous one. The code area shows the following interactions:

```
Out[14]: [1, 5, 3, 2, 8, 3, 6, 3, 4, 7, 3, 5, 3, 4, 7]

In [14]: m.ndigits()

Out[14]: 14

In [ ]: help(n.divisors)

In [15]: n.divisors?

In [ ]: |
```

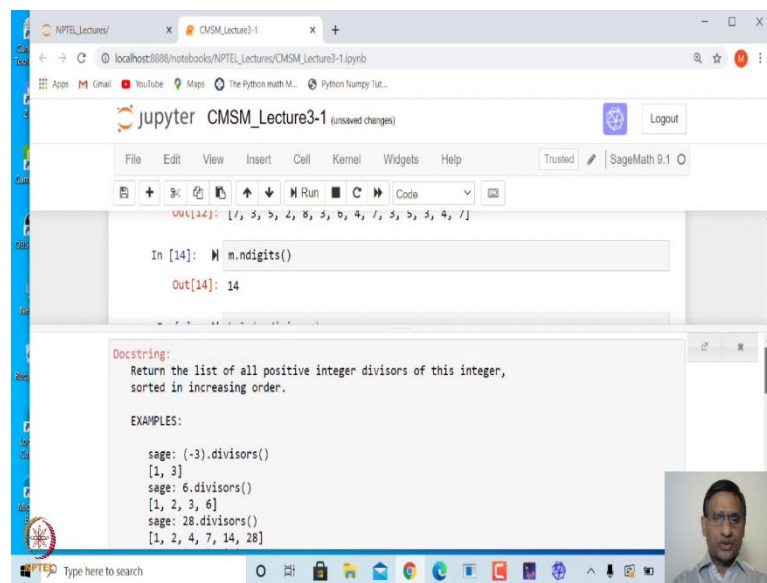
The bottom of the screen shows a Windows taskbar with various application icons and a search bar. A small video feed of a man is visible in the bottom right corner.

So, for example, if I say *m.ndigits()* it has 14 digits and that is what it gives me. So, once you have defined an integer any integer you can check various things about that integer

like whether it is prime, if it is not prime what are factors, next prime, previous prime all these things using dot tab operator.

So, suppose you want to for example, find out any about any function. So, for example, if I have said divisors, so, if I say `n.divisors()` and you can put either help i.e. you can write inside help you can write `n.divisors` 'help(`n.divisors`)' which is one way or you can even say `n.divisors` and put a question mark '`n.divisors?`'.

(Refer Slide Time: 10:59)



```
Out[14]: [1, 5, 7, 2, 10, 5, 10, 4, 1, 5, 7, 5, 4, 1]
```

```
In [14]: m.ndigits()
Out[14]: 14
```

Docstring:
Return the list of all positive integer divisors of this integer, sorted in increasing order.

EXAMPLES:

```
sage: (-3).divisors()
[1, 3]
sage: 6.divisors()
[1, 2, 3, 6]
sage: 28.divisors()
[1, 2, 4, 7, 14, 28]
```

Then it will open a help document which is inbuilt.

(Refer Slide Time: 11:02)

```

Out[14]: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 30, 36, 40, 45, 48, 60, 72, 80, 90, 96, 120, 144, 160, 180, 240, 360, 480, 720, 1440]

In [14]: m.ndigits()
Out[14]: 14

sage: (2^5).divisors()
[1, 2, 4, 8, 16, 32]
sage: 100.divisors()
[1, 2, 4, 5, 10, 20, 25, 50, 100]
sage: 1.divisors()
[1]
sage: 0.divisors()
Traceback (most recent call last):
...
ValueError: n must be nonzero
sage: (2^3 * 3^2 * 17).divisors()
[1, 2, 3, 4, 6, 8, 9, 12, 17, 18, 24, 34, 36, 51, 68, 72, 102, 136, 153, 204, 306, 408, 612, 1224]

```

Just to tell you in case you are opening this first time it may take little time, but of subsequently it will be faster. So, you can see here it tells you that returns the list of all positive integers divisors of this integer and sorted in increasing order. So, after finding the divisors it will sort in increasing order that is what it means.

(Refer Slide Time: 11:25)

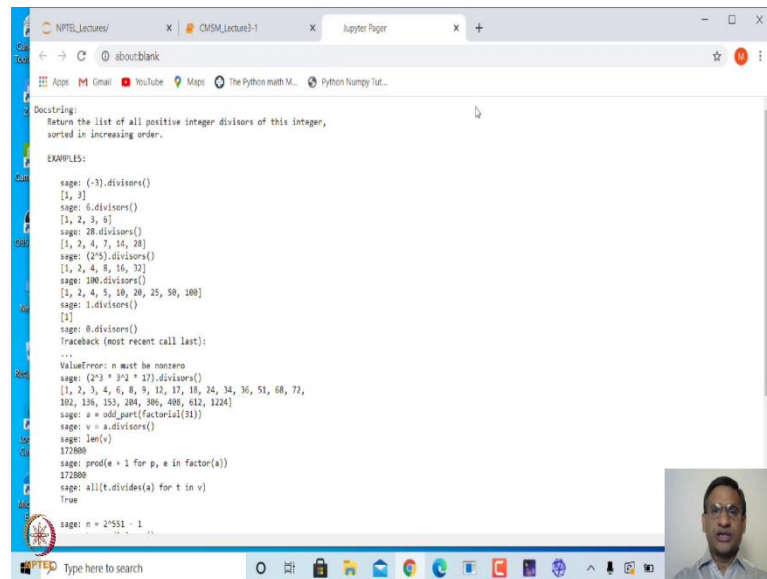
```

...
ValueError: n must be nonzero
sage: (2^3 * 3^2 * 17).divisors()
[1, 2, 3, 4, 6, 8, 9, 12, 17, 18, 24, 34, 36, 51, 68, 72, 102, 136, 153, 204, 306, 408, 612, 1224]
sage: a = odd_part(factorial(31))
sage: v = a.divisors()
sage: len(v)
172800
sage: prod(e + 1 for p, e in factor(a))
172800
sage: all(t.divides(a) for t in v)
True

```

And it gives you several examples and you can even copy paste any of this example.

(Refer Slide Time: 11:35)



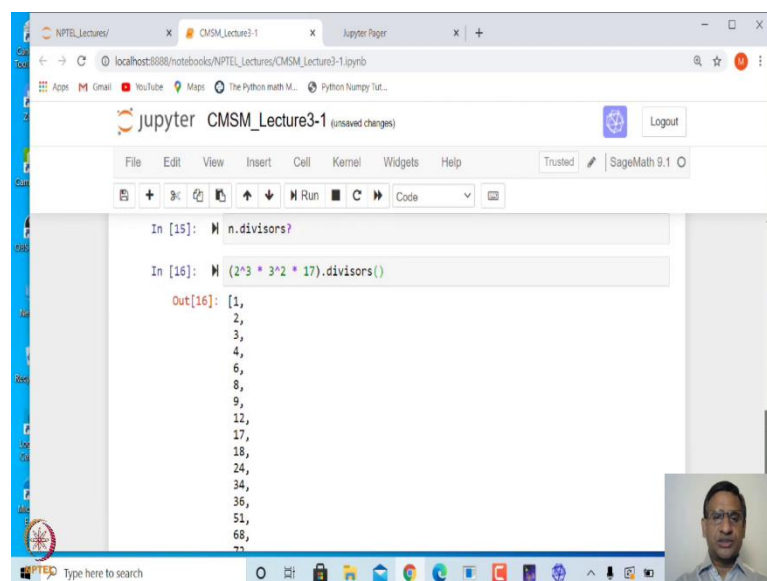
```
Docstring:
Return the list of all positive integer divisors of this integer,
sorted in increasing order.

EXAMPLES:

sage: (-3).divisors()
[1, 3]
sage: 6.divisors()
[1, 2, 3, 6]
sage: 28.divisors()
[1, 2, 4, 7, 14, 28]
sage: (2^5).divisors()
[1, 2, 4, 8, 16, 32]
sage: 100.divisors()
[1, 2, 4, 5, 10, 20, 25, 50, 100]
sage: 1.divisors()
[1]
sage: 0.divisors()
Traceback (most recent call last):
...
ValueError: n must be nonzero
sage: (2^3 * 3^2 * 17).divisors()
[1, 2, 3, 4, 6, 8, 9, 12, 17, 18, 24, 34, 36, 51, 68, 72,
102, 136, 153, 204, 306, 408, 612, 1224]
sage: a = odd_part(factorial(31))
sage: v = a.divisors()
sage: len(v)
172800
sage: prod(e + 1 for p, e in factor(a))
172800
sage: all(t.divides(a) for t in v)
True
sage: n = 2^551 - 1
```

For example, if I can just copy paste this let me make it bigger, so that it will fit in one of this tab and I can just copy this and I can paste it here.

(Refer Slide Time: 11:39)

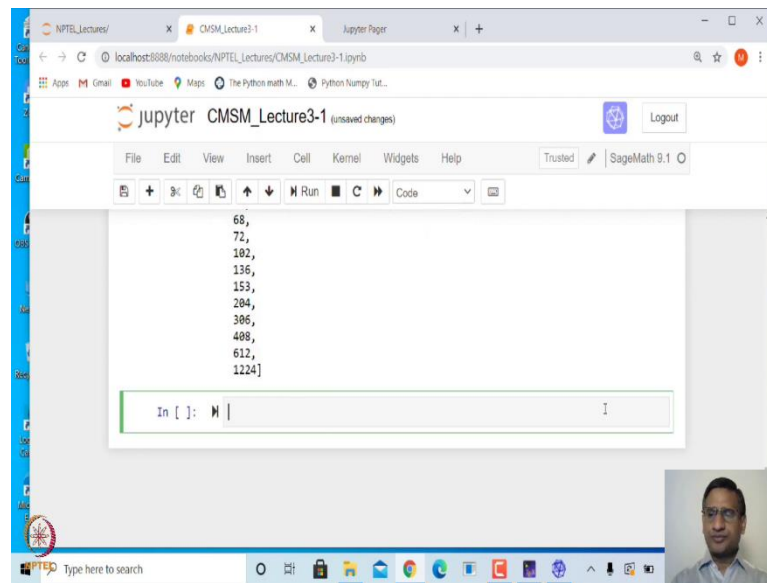


```
In [15]: n.divisors?

In [16]: (2^3 * 3^2 * 17).divisors()

Out[16]: [1,
2,
3,
4,
6,
8,
9,
12,
17,
18,
24,
34,
36,
51,
68,
72]
```

(Refer Slide Time: 11:41)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (unsaved changes)'. The notebook is running on a local host. The code cell contains a list of numbers: 68, 72, 102, 136, 153, 204, 306, 408, 612, 1224. The output cell shows the same list of numbers. The interface includes a menu bar with File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The status bar at the bottom shows 'SageMath 9.1' and a 'Logout' button. A small video feed of a person is visible in the bottom right corner.

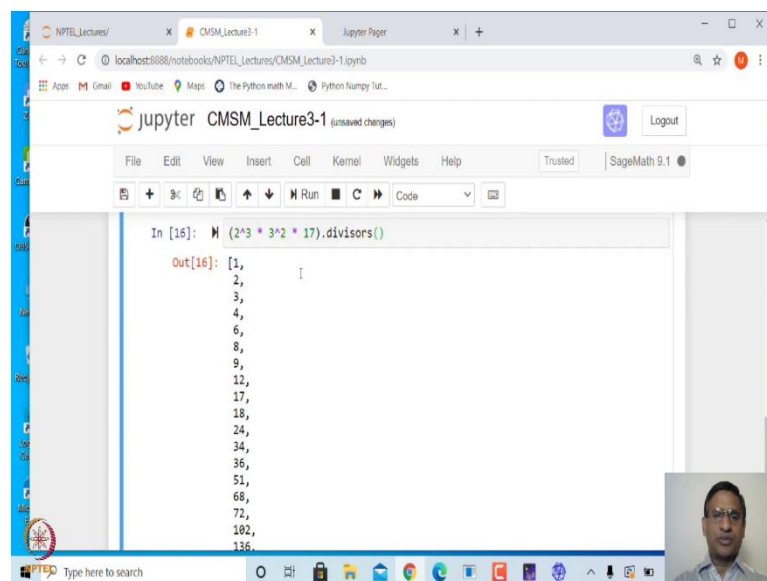
```
68,  
72,  
102,  
136,  
153,  
204,  
306,  
408,  
612,  
1224]
```

```
In [ ]: |
```

So, it will all these things when you open this help document you can go through this and then you can make use of any of the syntax which are there in this help document.

So, in the beginning it is a good idea to actually find out help or take help on every function that utilize it and you can go through it will tell you what is this function how it works with several examples.

(Refer Slide Time: 12:16)



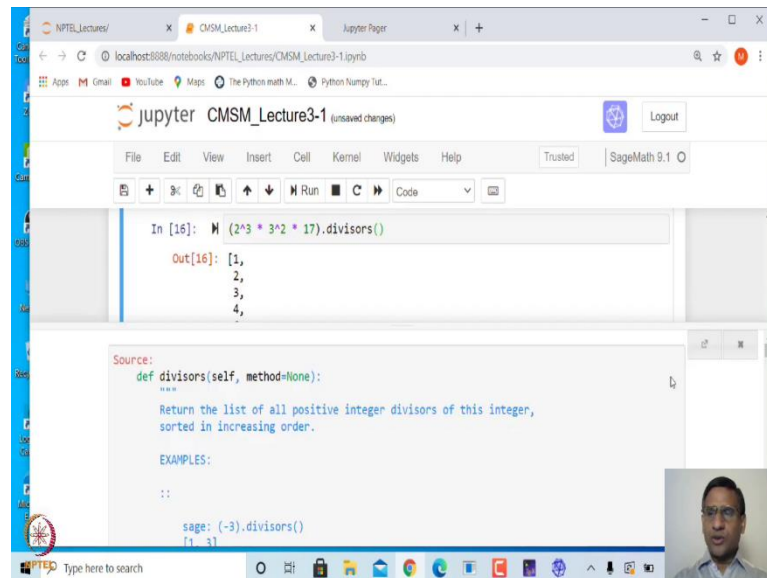
The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (unsaved changes)'. The notebook is running on a local host. The code cell contains the expression $(2^3 * 3^2 * 17).divisors()$. The output cell shows the list of divisors: [1, 2, 3, 4, 6, 8, 9, 12, 17, 18, 24, 34, 36, 51, 68, 72, 102, 136]. The interface includes a menu bar with File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The status bar at the bottom shows 'SageMath 9.1' and a 'Logout' button. A small video feed of a person is visible in the bottom right corner.

```
In [16]: (2^3 * 3^2 * 17).divisors()
```

```
Out[16]: [1,  
2,  
3,  
4,  
6,  
8,  
9,  
12,  
17,  
18,  
24,  
34,  
36,  
51,  
68,  
72,  
102,  
136]
```

Instead of one question mark if you write two question marks, so if I say n . *divisor*?? and double question mark then also it will open a help document, but it will be more detailed help document along with the source code.

(Refer Slide Time: 12:18)



The screenshot shows a Jupyter Notebook interface with a browser window. The notebook is titled "CMSM_Lecture3-1 (unsaved changes)". The code cell shows the command `(2^3 * 3^2 * 17).divisors()` and the output is a list of divisors: `[1, 2, 3, 4]`. Below the code cell, the source code for the `divisors` method is displayed, showing a function definition and examples of its usage.

```

In [16]: (2^3 * 3^2 * 17).divisors()

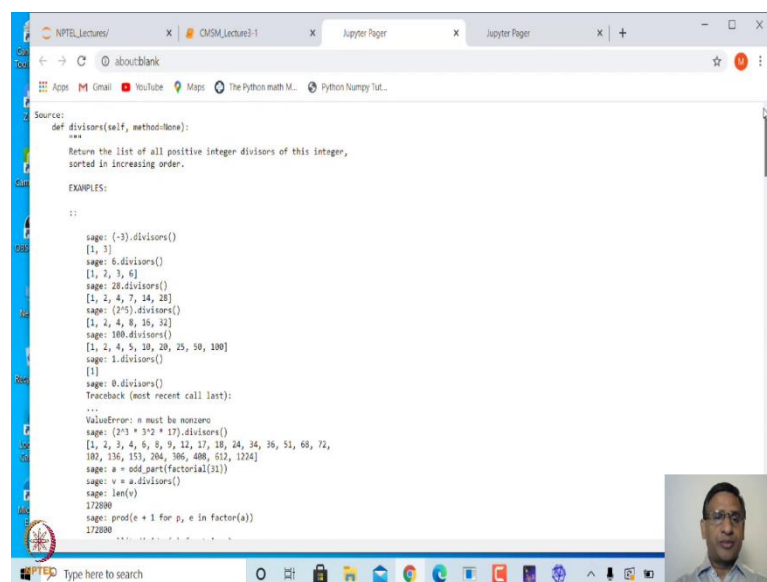
Out[16]: [1,
          2,
          3,
          4,
          ...]

Source:
def divisors(self, method=None):
    """
    Return the list of all positive integer divisors of this integer,
    sorted in increasing order.

    EXAMPLES:
    ::

    sage: (-3).divisors()
    [1, 3]
  
```

(Refer Slide Time: 12:20)



The screenshot shows a Jupyter Notebook interface with a browser window. The notebook is titled "CMSM_Lecture3-1". The code cell shows a series of commands and their outputs, including `sage: (-3).divisors()`, `sage: 6.divisors()`, `sage: 28.divisors()`, `sage: 2^5.divisors()`, `sage: 100.divisors()`, `sage: 1.divisors()`, `sage: 0.divisors()`, and a `ValueError: n must be nonzero` followed by `sage: (2^3 * 3^2 * 17).divisors()`. The output for the last command is a list of divisors: `[1, 2, 3, 4, 6, 8, 9, 12, 17, 18, 24, 36, 51, 68, 72, 102, 136, 153, 204, 306, 408, 612, 1224]`.

```

Source:
def divisors(self, method=None):
    """
    Return the list of all positive integer divisors of this integer,
    sorted in increasing order.

    EXAMPLES:
    ::

    sage: (-3).divisors()
    [1, 3]

    sage: 6.divisors()
    [1, 2, 3, 6]

    sage: 28.divisors()
    [1, 2, 4, 7, 14, 28]

    sage: 2^5.divisors()
    [1, 2, 4, 8, 16, 32]

    sage: 100.divisors()
    [1, 2, 4, 5, 10, 20, 25, 50, 100]

    sage: 1.divisors()
    [1]

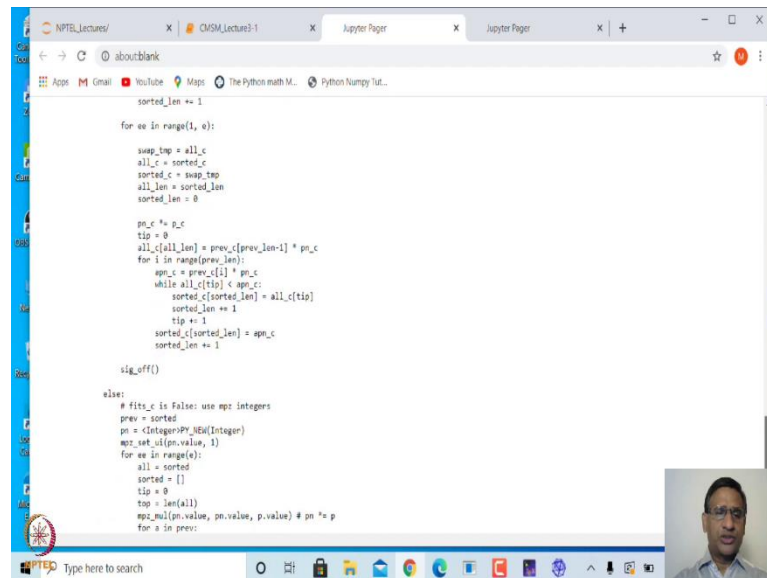
    sage: 0.divisors()
    Traceback (most recent call last):
    ...
    ValueError: n must be nonzero

    sage: (2^3 * 3^2 * 17).divisors()
    [1, 2, 3, 4, 6, 8, 9, 12, 17, 18, 24, 36, 51, 68, 72,
    102, 136, 153, 204, 306, 408, 612, 1224]

    sage: a = odd_part(factorial(31))
    sage: v = a.divisors()
    sage: len(v)
    172800

    sage: prod(e + 1 for p, e in factor(a))
    172800
  
```

(Refer Slide Time: 12:37)



The screenshot shows a web browser window with multiple tabs. The active tab is titled 'Jupyter Pager'. The address bar shows 'about:blank'. The main content area displays a Python script. The script starts with a function definition for a sorting algorithm, likely bubble sort, using variables like 'sorted_len', 'swap_temp', 'all_c', 'sorted_c', 'prev_len', 'pn_c', 'tip', 'apn_c', 'sorted_len', and 'sig_off()'. It includes a loop 'for ee in range(1, e):' and a nested loop 'for i in range(prev_len):'. The script also includes a comment '# fits_c is False: use mpz integers' and a loop 'for a in prev:'. The bottom of the browser window shows a Windows taskbar with various application icons and a search bar.

```
sorted_len = 1
for ee in range(1, e):
    swap_temp = all_c
    all_c = sorted_c
    sorted_c = swap_temp
    all_len = sorted_len
    sorted_len = 0

    pn_c = p_c
    tip = 0
    all_c[all_len] = prev_c[prev_len-1] * pn_c
    for i in range(prev_len):
        apn_c = prev_c[i] * pn_c
        while all_c[tip] < apn_c:
            sorted_c[sorted_len] = all_c[tip]
            sorted_len += 1
            tip += 1
        sorted_c[sorted_len] = apn_c
        sorted_len += 1

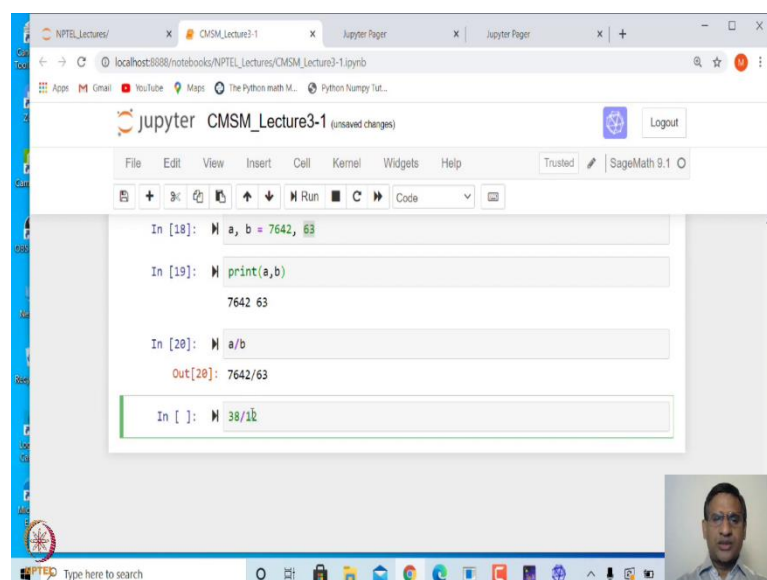
    sig_off()

else:
    # fits_c is False: use mpz integers
    prev = sorted
    pn = <Integer>PN_NMu(Integer)
    mpz_set_ui(pn.value, 1)
    for ee in range(e):
        all = sorted
        sorted = []
        tip = 0
        top = len(all)
        mpz_mul(pn.value, pn.value, p.value) # pn *= p
        for a in prev:
```

So, this is a function defined and it tells you how it has been defined along with the examples and you can see here how this program has been written. So, that is what I meant by saying that SageMath is open-source software.

So, not only you can use it freely, but you can also see the source code and you can even modify this source code. So, right, so this is how you can take help on any function or method. Now, suppose you want to have two integers.

(Refer Slide Time: 13:05)



The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs include 'NPTEL_Lectures/', 'CMSM_Lecture3-1', and 'Jupyter Pager'. The address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-1.ipynb'. The Jupyter interface has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu bar is a toolbar with icons for adding, deleting, and running cells. The main area shows a code cell with the following input and output:

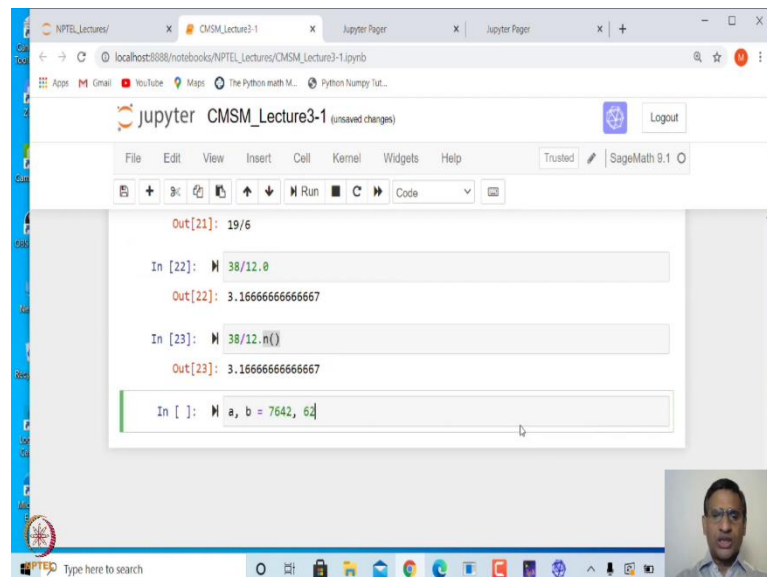
```
In [18]: a, b = 7642, 63
In [19]: print(a,b)
7642 63
In [20]: a/b
Out[20]: 7642/63
In [ ]: 38/11
```

The bottom of the browser window shows a Windows taskbar with various application icons and a search bar.

Let us say define two integers a and b. So, I can say a, b is equal to again let us say 7642 and another integer b is let us say 63. So, a and b are you can print what are a and b. It will tell you a and b are this.

Now, suppose we want to check whether this number a is divisible by 63 or not. So, one way is to look at a divided by b. When you say a divided by b it tells me that actually it gives me the same integer by this. So, therefore, it is not divisible. Actually this returns a rational number. In case there are some common factors it will get rid of this common factor.

(Refer Slide Time: 14:04)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The notebook contains the following code and output:

```
Out[21]: 19/6

In [22]: 38/12.0
Out[22]: 3.1666666666666667

In [23]: 38/12.n()
Out[23]: 3.1666666666666667

In [ ]: a, b = 7642, 63
```

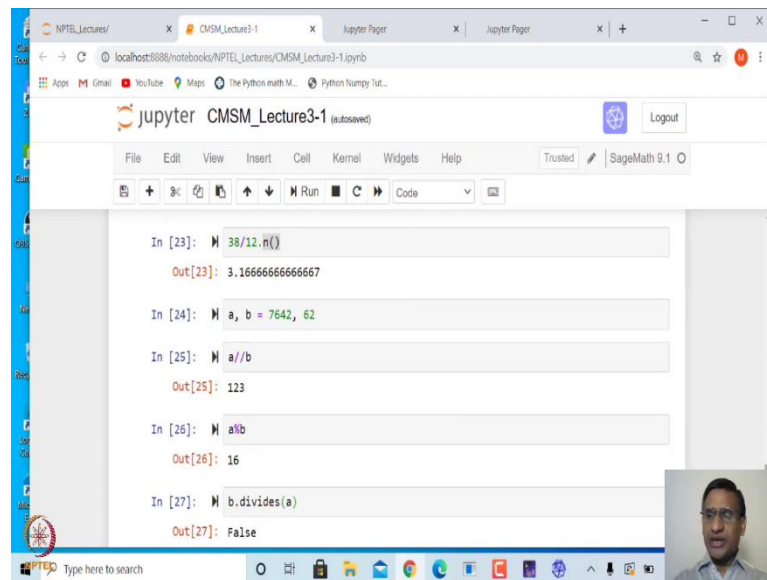
The bottom right corner of the notebook shows a small video feed of a person speaking.

So, for example, if let us say 38 divided by 12, then it will give me 19 by 6 because it will take out this factor 2 from the numerator and denominator. So, this is how you can define a rational number.

In case you wanted to find let us say a divided by b as a real number you could you could say for example, 38 divided by 12.0 this is one way or I can also say 38 divided by 12 and then press dot and n(). This dot n means it is going to give you numerical values.

So, there are two ways in which you can find the numerical values, but dot n we will keep using in order to get numerical value of any variable. So, we have defined two integers a and b like this and suppose we know now that this a is not divisible by b.

(Refer Slide Time: 15:09)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (autosaved)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, cell execution, and code execution. The notebook contains several code cells with their corresponding outputs:

```
In [23]: 38/12.m()
Out[23]: 3.166666666666667

In [24]: a, b = 7642, 62
Out[24]: (7642, 62)

In [25]: a//b
Out[25]: 123

In [26]: a%b
Out[26]: 16

In [27]: b.divides(a)
Out[27]: False
```

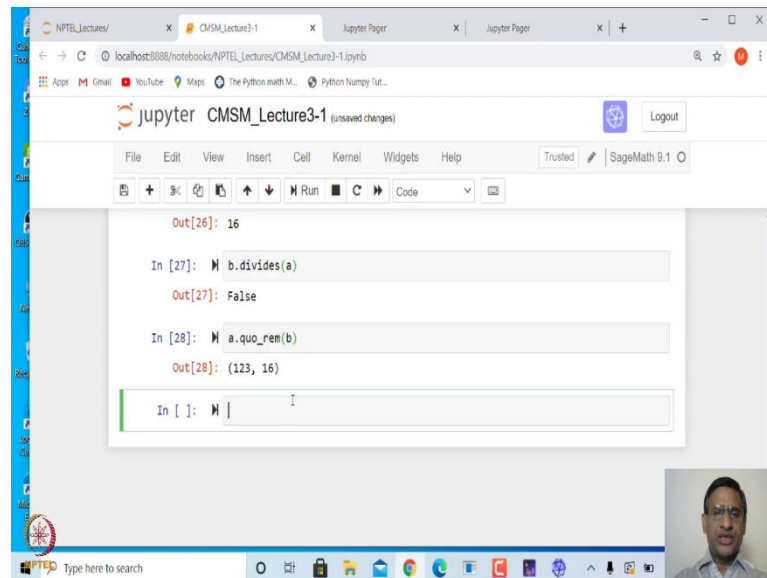
A small video feed of a person is visible in the bottom right corner of the notebook window.

Let us change instead of 63, let us say b is 62 and now we want to check whether a is divisible by b or not. So, how do we check? We can say of course, we can still say $a // b$ and then we can check what is the quotient.

So, quotient is 123 and we can find out remainder $a \% b$ that will give me the remainder. So, remainder is non zero. Therefore, this is not divisible. You could also find this by a command let us say $b.divides(a)$. So, it will tell you whether b divides a or not.

So, either you can find quotient remainder or you can check whether b divides a using this function. You could find quotient and remainder by double slash and percentage, but there is also an inbuilt function.

(Refer Slide Time: 16:11)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and saving. The code area contains the following cells:

```
Out[26]: 16

In [27]: b.divides(a)
Out[27]: False

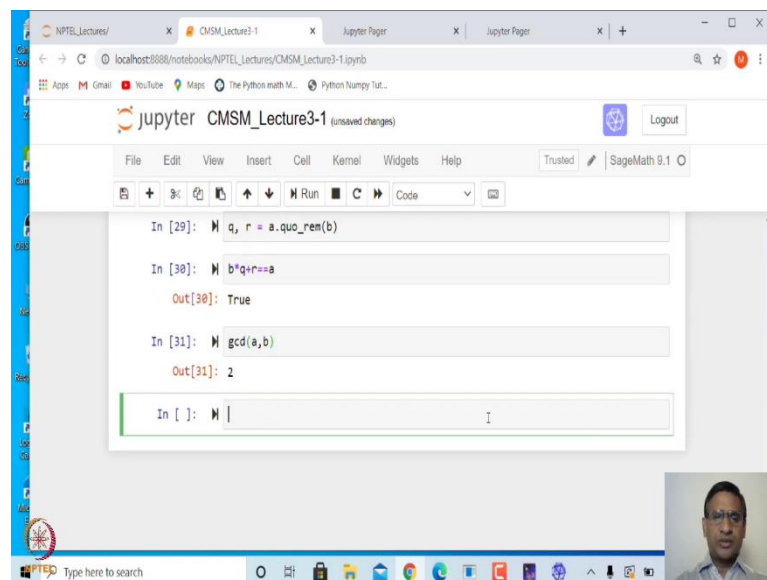
In [28]: a.quo_rem(b)
Out[28]: (123, 16)

In [ ]: 
```

A small video feed of a person is visible in the bottom right corner of the notebook window.

So, if I again for example, if I say a dot and press tab and let us just explore this list and is there something to find quotient and remainder. So, if you go down to q, so, q here there is one called *quo_rem()* and in the bracket if I say b (*'a. quo_rem(b)'*) it tells me the quotient is 123 remainder is 16.

(Refer Slide Time: 16:38)



The screenshot shows the same Jupyter Notebook window with additional code cells:

```
In [29]: q, r = a.quo_rem(b)

In [30]: b*q+r==a
Out[30]: True

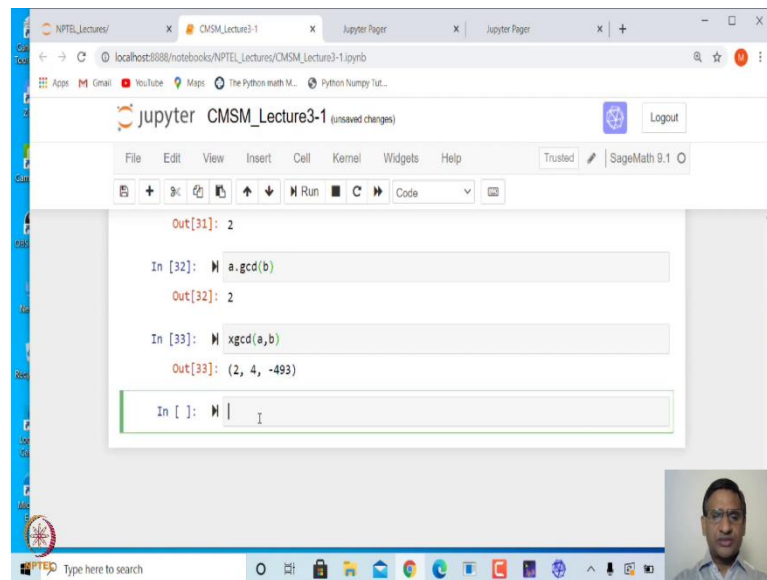
In [31]: gcd(a,b)
Out[31]: 2

In [ ]: 
```

The video feed of the person remains in the bottom right corner.

So, I can store this into let us say q, r is equal to this and we can even verify b into the quotient plus remainder r is it equal to a (i.e. $b*q+r == a$), the answer is yes. So, this is how you can find quotient and remainder. If you have two integers you can also find for example, gcd of these two integers. So, if I say gcd of a, b it will tell me the gcd is 2.

(Refer Slide Time: 17:07)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1'. The notebook contains the following code and output:

```
Out[31]: 2

In [32]: a.gcd(b)
Out[32]: 2

In [33]: xgcd(a,b)
Out[33]: (2, 4, -493)

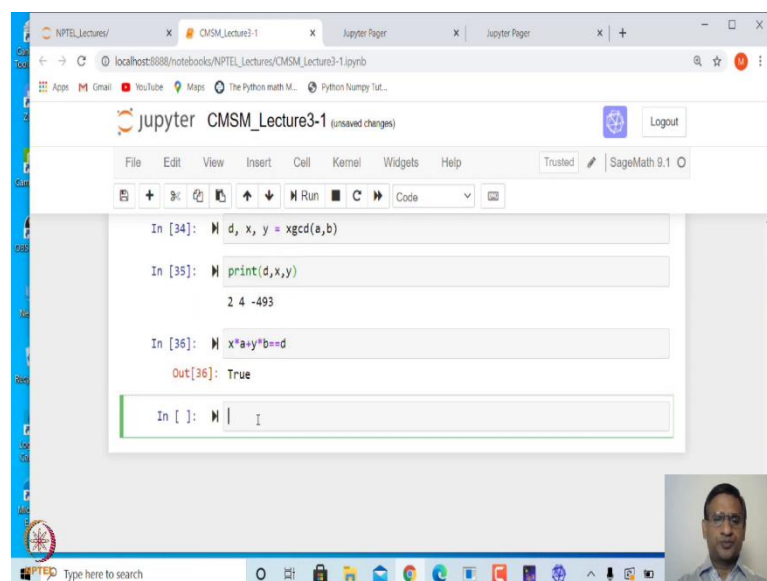
In [ ]: 
```

The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing 'SageMath 9.1'.

You could also have said $a.gcd(b)$ both are same. So, it can be used gcd can be used as a function or as a method. Instead of finding gcd if you say $xgcd(a,b)$ it will return three things. One first one is gcd and this is let me call this as x and this is y.

So, these two integers x and y have the property that x times a plus y times b should be equal to this gcd. So, gcd can be written as linear integer linear combination of a and b. This is what is called Bezout's identity given two integers a and b and d if d is gcd of a and b then there exist integers x and y such that $x * a + y * b = d$ where d is called gcd.

(Refer Slide Time: 18:07)



The screenshot shows the same Jupyter Notebook window with additional code and output:

```
In [34]: d, x, y = xgcd(a,b)

In [35]: print(d,x,y)
2 4 -493

In [36]: x*a+y*b==d
Out[36]: True

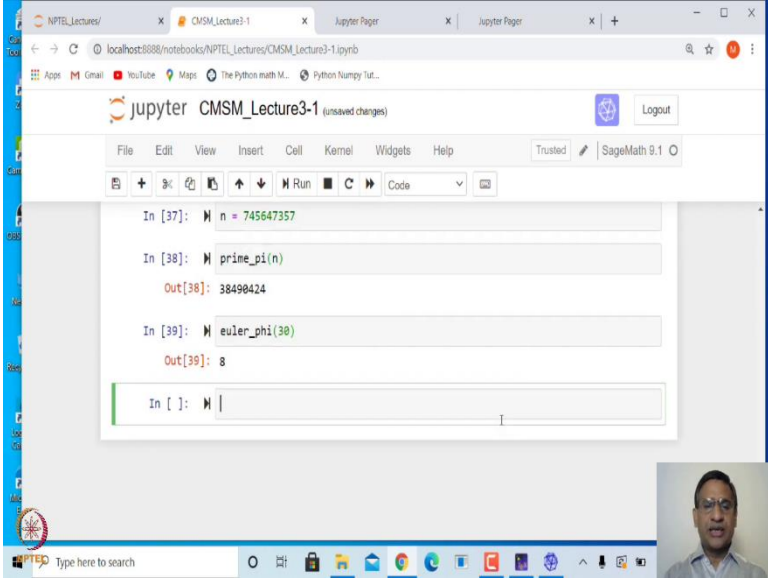
In [ ]: 
```

The interface remains the same as in the previous screenshot, showing the Jupyter Notebook environment with the SageMath 9.1 kernel.

So, let us verify that. How do we verify that? Let me store this the output in d which is gcd and x that is the first integer and y the second integer. Now, you can print what are d , x , y it will tell you this. Now, if I verify $x*a + y * b$ is it equal to d we should get answer true.

So, this we have verified Bezout's identity. And the beauty of the sage is that we have used for example, gcd function and you know that you can also find gcd of polynomials. So, the same function can be applied to find gcd of polynomials and in this case gcd of integers. So, that is another advantage.

(Refer Slide Time: 19:02)

A screenshot of a Jupyter Notebook interface. The browser tabs show 'NPTEL_Lectures/' and 'CMSM_Lecture3-1'. The address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-1.ipynb'. The notebook title is 'jupyter CMSM_Lecture3-1 (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code area shows three input cells:

```
In [37]: n = 745647357
```

```
In [38]: prime_pi(n)
```

```
Out[38]: 38498424
```

```
In [39]: euler_phi(30)
```

```
Out[39]: 8
```

```
In [ ]: 
```

A small video feed of a person is visible in the bottom right corner of the notebook window.

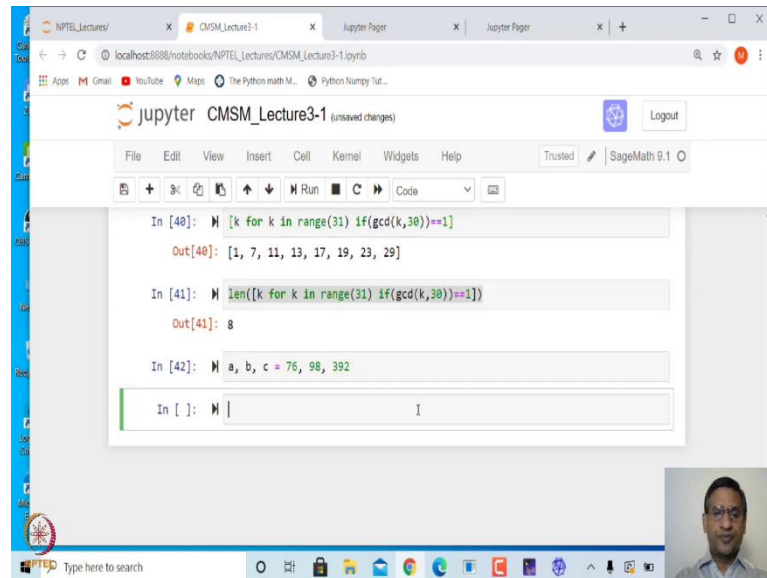
Let us also look at let me again define an integer n which is somewhat let us say big integer let us say this n . And suppose we want to find out how many primes are there between 2 and n . So, if I say there is a function called `prime_pi()` which will find the number of primes and if I say in the bracket n this tells me that it has so many primes between two and n . So, you can count how many primes are there.

For example, if I say tell me the integer n such that between 2 and n there are 1000 primes. So, what is that integer that's very simple Python code will tell me how to find this? So, for example, you can even find out how many integers are there from 1 to n which are co prime to n .

So, there is a function called `euler_phi()` and if I say `euler_phi(30)`, then it will tell me it is 8 which means that there are 8 integers between 1 and 30 which are co prime to 30.

And how do we find this? So that means, there are 8 integers such that integers and gcd of each of these these eight with 30 is one that is what we mean by saying co prime.

(Refer Slide Time: 20:34)

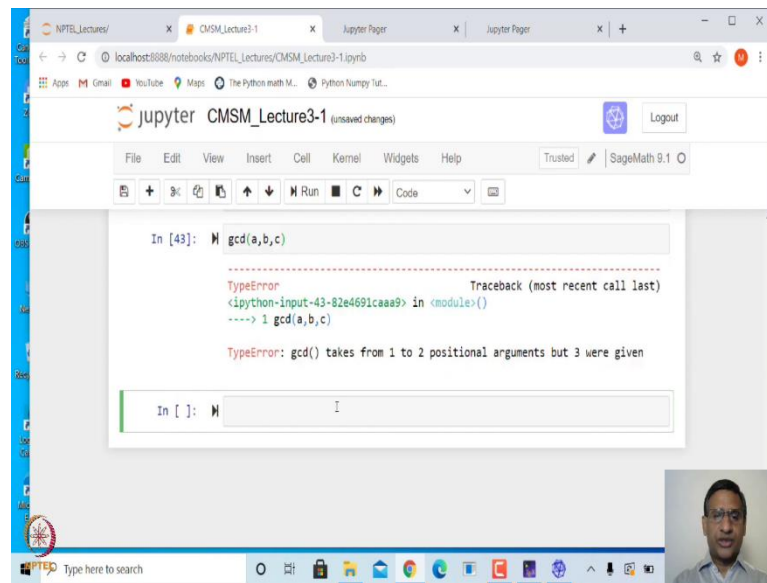
A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/NPTEL_Lectures/CMSM_Lecture3-1.ipynb'. The notebook title is 'CMSM_Lecture3-1 (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and other functions. The code area shows three input cells:
In [40]: `[k for k in range(31) if(gcd(k,30))==1]`
Out[40]: `[1, 7, 11, 13, 17, 19, 23, 29]`
In [41]: `len([k for k in range(31) if(gcd(k,30))==1])`
Out[41]: `8`
In [42]: `a, b, c = 76, 98, 392`
The bottom of the screen shows a Windows taskbar with various application icons and a search bar. A small video feed of a person is visible in the bottom right corner of the Jupyter interface.

So, for example, I can say `[k for k in range(31)]`, so, it will give 0 to 30, I want including 30. So, I will say 31 and when do you take this k? If $\text{gcd}(k, 30)$ is equal to 1 i.e. `'[k for k in range(31) if(gcd(k,30))==1]'`. Then these are the integers.

Now, we can count what is the length of this? So, we can say `len()` of this, it will tell me this is 8. So, we have not only we have found out this number of integers which are co-prime to 30 using inbuilt function `euler_phi()` but we have also verified this using `'len([k for k in range(31) if(gcd(k,30))==1])'`.

So, these are all Python codes. So, you can see here any Python code we can run as it is. So, you can also find for example, we have found gcd of two integers a and b. Suppose I want to find gcd of three integers, let us say a b and c. So, let us define a, b, c is equal to let us say 76, 98 and 392.

(Refer Slide Time: 21:56)



The screenshot shows a Jupyter Notebook interface with the title 'CMSM_Lecture3-1 (unsaved changes)'. The code cell contains the following text:

```
In [43]: gcd(a,b,c)
```

The output shows a `TypeError` with a traceback:

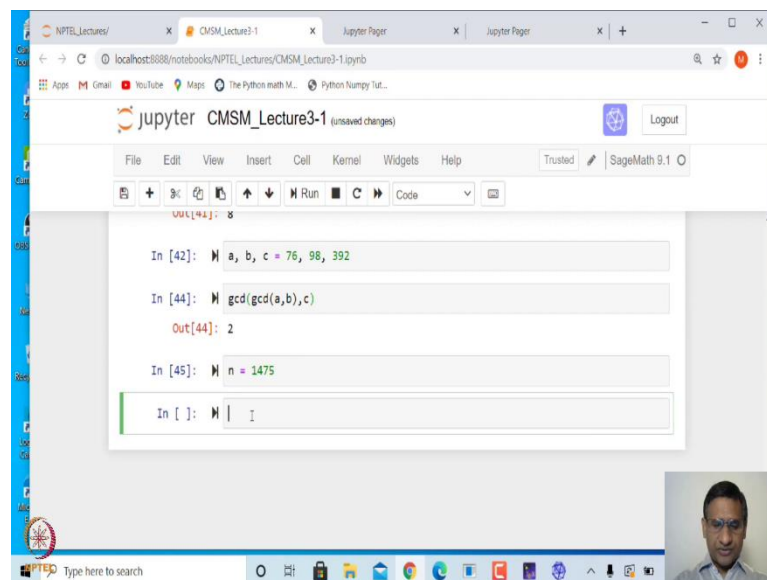
```
Traceback (most recent call last)
<ipython-input-43-82e4691caaa9> in <module>()
----> 1 gcd(a,b,c)

TypeError: gcd() takes from 1 to 2 positional arguments but 3 were given
```

The input prompt 'In []: ' is visible at the bottom of the code cell.

And if I want to find out gcd of a, b and c let us run this. It says that we cannot find because it takes only 1 to 2 position. Therefore, we cannot find gcd of three integers at a time.

(Refer Slide Time: 22:14)



The screenshot shows the same Jupyter Notebook interface. The code cell contains the following text:

```
In [42]: a, b, c = 76, 98, 392
```

```
In [44]: gcd(gcd(a,b),c)
```

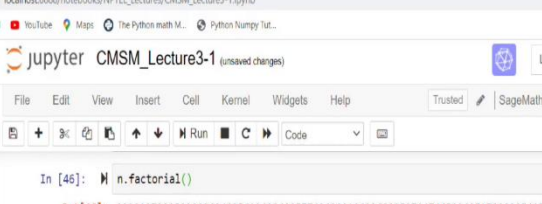
The output shows the result of the nested gcd function:

```
Out[44]: 2
```

The input prompt 'In []: ' is visible at the bottom of the code cell.

But, what you can do is you can find first gcd of a and b and then find gcd of c along with gcd of a and b. So, we will say gcd of a and b with c is 2. So, 2 is gcd of these three integers. So, this is how you can explore any integer and look at various properties.

Next let us say we want to look at some other function that we know how to deal with. So, for example, if I say let us say n is equal to let us say 1475 and I want to find factorial of this number.



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1'. The address bar indicates the file is located at 'localhost:8888/...'. The top navigation bar includes tabs for 'NPTL_Lectures/' and 'CMSM_Lecture3-1', and buttons for 'upyter Leger' and 'upyter Pager'. Below the navigation bar is a toolbar with icons for file operations, running, and code execution. The main area contains a code cell with the following text:

```
In [46]: n.factorial()
```

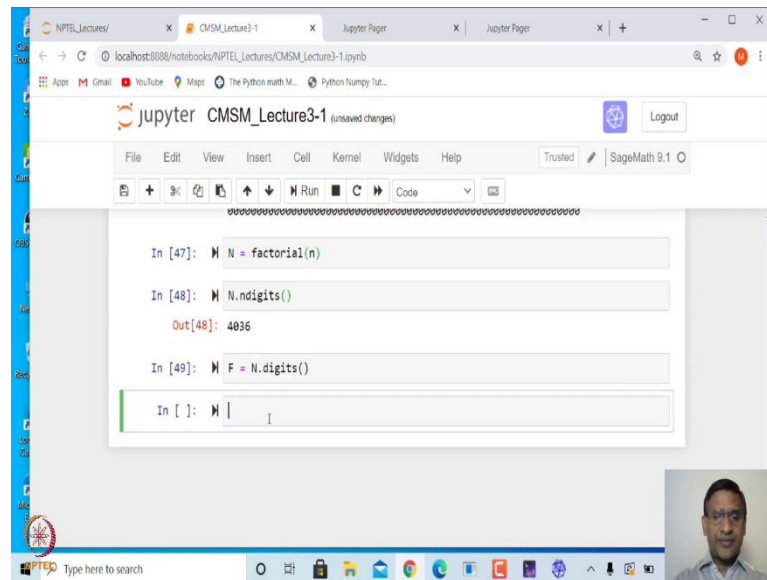
The output of the cell is displayed below the code:

```
Out[46]: 23301272395228836348256134984095574969891863268995271474560965179803954050  
558074614722826392798580917392375431176608934127593944888165832290780382  
3628351121281456206825956184087764638471496712385449297598524105683476406  
9974223164718619436599063667462967092131727419113739478689762386393984083  
6530812827626186748672985647424258098391081747323991754027551096230606119  
75608889968575556764483690987889529272985823997375548143842799552197533939  
17667249432794594581437178832774902520273258679202821494541476728138521043  
903186497228546758102787612999468464188184344721978377669493161717473561771  
82941616474893926181341405424262121082432228408391478858705204463885185449  
637772366376129909598322854164136443768112244246563627162323232727142074  
5109944266269687421551639667237960766136606814723519779669296055555555931  
4722818176046208679847876586949174505912983310594305185162428636115138403  
692691062383194253958510590753921314319140080818872816508642387788831518  
524984714202440514204036789256128100909149731766719956784894942758910325  
548114559571728552919693713785821282893837421454002558729883163139335206  
59529856377453969527783202890073908504121289819471379519476582784232694725  
17443921598027235297091582811619587310930567609124813647521433621549896633
```

The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1'. The interface includes a top bar with file management icons, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a toolbar with execution and navigation buttons. The main area displays a list of 100 zeros, suggesting a large array of zeros has been created or displayed. The bottom status bar shows the current cell is an 'In' cell.

So, if I say `n` dot factorial (`'n.factorial()'`) it will tell me the factorial of this 1475 and there are so many digits in fact. So, you can even let us say let us store this what we have found out in a variable as number `N`.

(Refer Slide Time: 23:47)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (unsaved changes)'. The notebook contains the following code cells:

```
In [47]: N = factorial(n)
```

```
In [48]: N.ndigits()
```

```
Out[48]: 4036
```

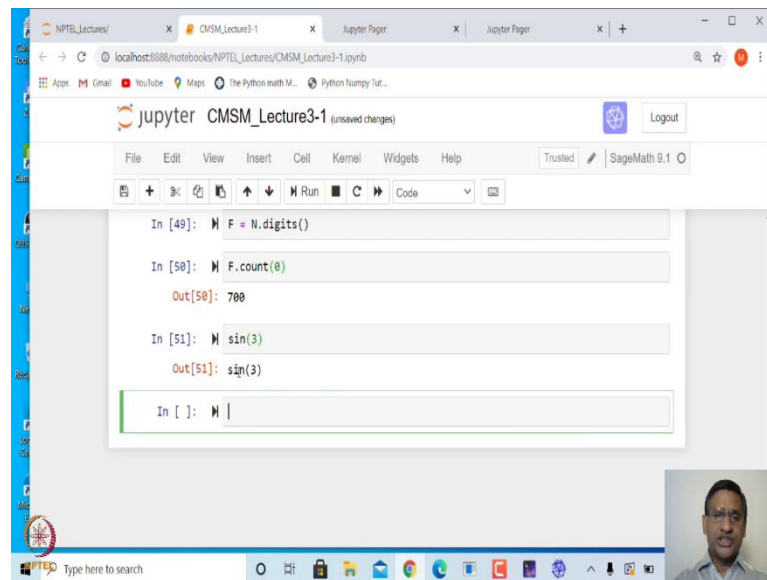
```
In [49]: F = N.digits()
```

```
In [ ]: 
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom. A small video feed of a person is visible in the bottom right corner.

Let me call this as number as `N`. `N` is equal to you could also say `factorial(n)` instead of `n.factorial()`. And suppose we want to find out number of digits I can simply say capital `N.ndigits()`. It tells me the number of digits are 4036 and if I want to find out how many zeros are there, then what we can do is we can say `n.digits()`. So, it will separate all the digits and then let us store this in `F`.

(Refer Slide Time: 24:26)



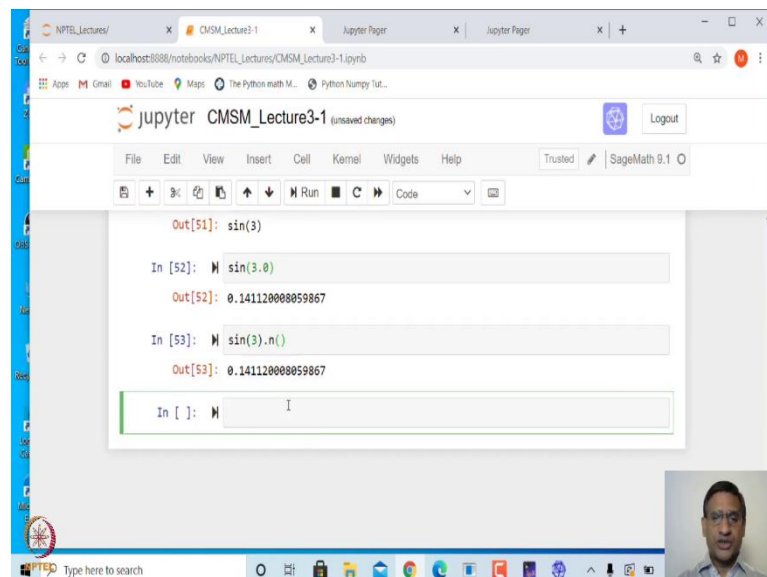
The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (unsaved changes)'. The notebook is running on a local host. The code cells show the following:

```
In [49]: F = N.digits()
In [50]: F.count(0)
Out[50]: 700
In [51]: sin(3)
Out[51]: sin(3)
In [ ]: |
```

The output for `F.count(0)` is 700, and the output for `sin(3)` is `sin(3)`. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, code execution, and help.

And then we can say `F.count(0)` there are 700 0s in 1475 factorial. So, that is quite easy. You can also apply various scientific function like sine, cosine, exponential, logarithmic and all these things are inbuilt. So, for example, if I say sin of 3 radian it just returns sin 3. If I want to find the numerical value there are two ways in which we can do.

(Refer Slide Time: 25:00)



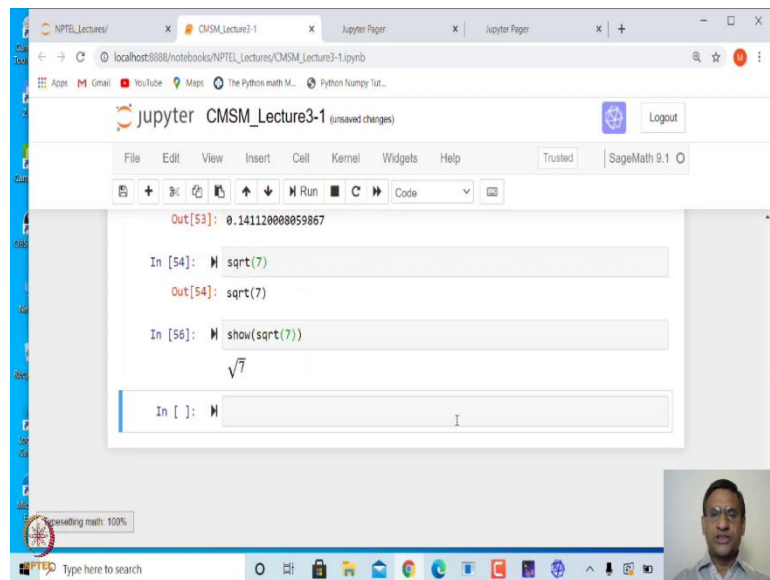
The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (unsaved changes)'. The notebook is running on a local host. The code cells show the following:

```
Out[51]: sin(3)
In [52]: sin(3.0)
Out[52]: 0.141120008059867
In [53]: sin(3).n()
Out[53]: 0.141120008059867
In [ ]: |
```

The output for `sin(3)` is `sin(3)`, the output for `sin(3.0)` is 0.141120008059867, and the output for `sin(3).n()` is 0.141120008059867. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, code execution, and help.

We can say sin of 3.0 or we can say sin 3 in the bracket and then `.n()` ('`sin(3).n()`'), this is to find numerical value.

(Refer Slide Time: 25:15)



```
Out[53]: 0.141120008059867

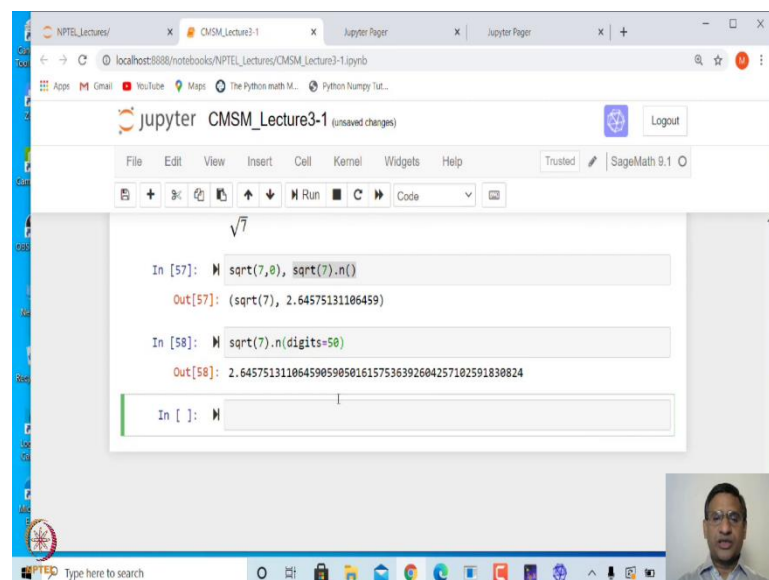
In [54]: sqrt(7)
Out[54]: sqrt(7)

In [56]: show(sqrt(7))
Out[56]:  $\sqrt{7}$ 

In [ ]:
```

So, that is how we can find it. Suppose, if I want to find out square root of let us say 7, this returns just square root 7 ('sqrt(7)'). In fact, you can ask it to show in symbolic notation. If I say show sqrt of 7, then it will just print $\sqrt{7}$. If I want to find numerical values of this, by now we know what is to be done.

(Refer Slide Time: 25:42)



```
Out[57]: (sqrt(7), 2.64575131106459)

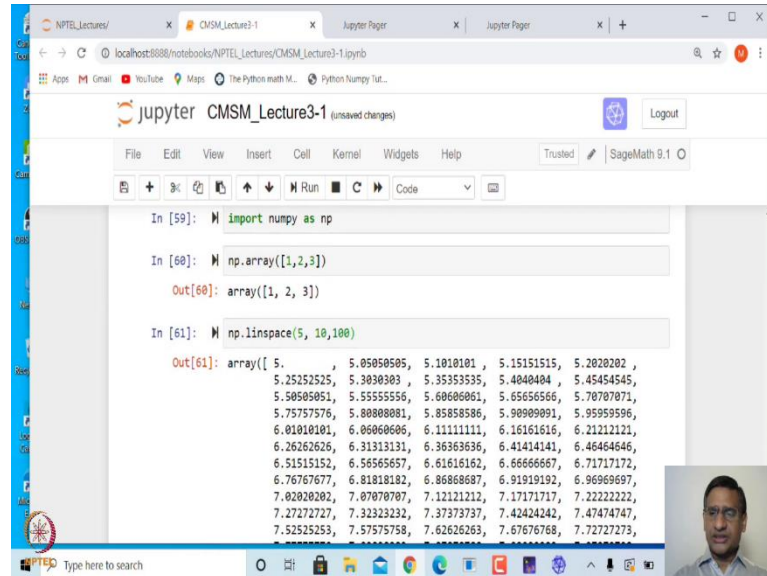
In [58]: sqrt(7).n(digits=50)
Out[58]: 2.6457513110645905905016157536392604257182591830824

In [ ]:
```

This you can say 7.0 or you can say sqrt in the bracket 7 and then dot n. You can even print how many digits do you want. So, for example, if I say this and in the bracket if I put digits is equal to let us say 50 ('i.e. sqrt(7).n(digits=50)') then it will give you so many

digits. So, that option is also there. And as I said this any Python program you can run inside this sage notebook or sage worksheet as it is.

(Refer Slide Time: 26:19)



The screenshot shows a Jupyter Notebook window titled 'CMSM_Lecture3-1 (unsaved changes)' running on SageMath 9.1. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook content shows three input cells:

```
In [59]: import numpy as np
```

```
In [60]: np.array([1,2,3])
```

The output for the second cell is:

```
Out[60]: array([1, 2, 3])
```

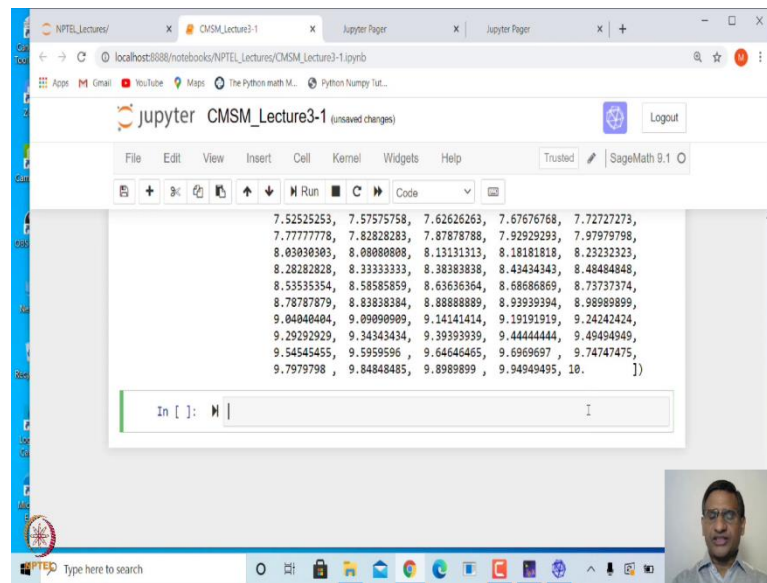
```
In [61]: np.linspace(5, 10, 100)
```

The output for the third cell is a large array of 100 values, starting with 5.00000000 and ending with 7.72727273. A small video inset of a person is visible in the bottom right corner of the notebook window.

So, for example, if I say import numpy as np, then it will import numpy. And after that.. so, it is still doing right. It has done if you see a star ('*') here; that means, the kernel is still running. Generally, I find SageMath in Window is a bit slower compared to SageMath in Linux.

Now, for example, in numpy you have a function array. So, if I say np.array and then I say 1, 2, 3, it will create an array or if I say np.linspace all these things we have used between let us say 5 and 10 I want 100 points, then this can be also used.

(Refer Slide Time: 27:10)



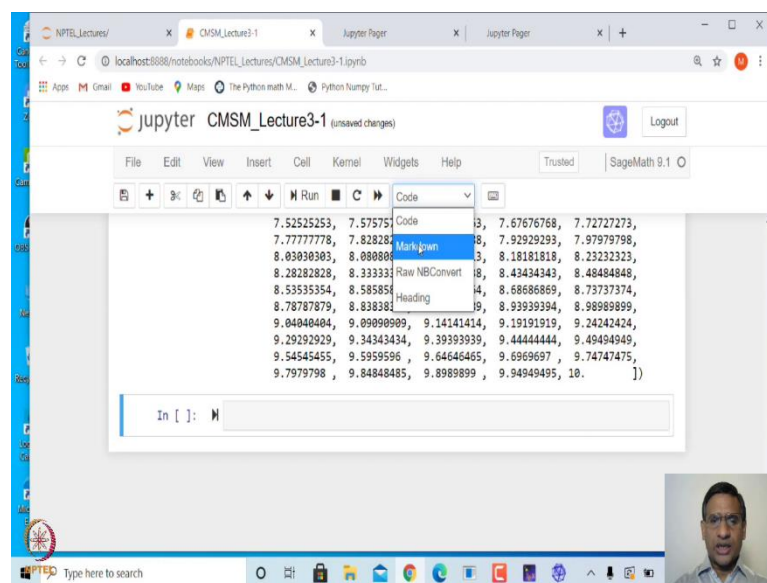
The screenshot shows a Jupyter Notebook window titled "CMSM_Lecture3-1 (unsaved changes)". The notebook contains a single code cell with a list of numbers. Below the code cell, there is a video feed of a man.

```
7.52525253, 7.57575758, 7.62626263, 7.67676768, 7.72727273,  
7.77777778, 7.82828283, 7.87878788, 7.92929293, 7.97979798,  
8.03030303, 8.08080808, 8.13131313, 8.18181818, 8.23232323,  
8.28282828, 8.33333333, 8.38383838, 8.43434343, 8.48484848,  
8.53535354, 8.58585859, 8.63636364, 8.68686869, 8.73737374,  
8.78787879, 8.83838384, 8.88888889, 8.93939394, 8.98989899,  
9.04040404, 9.09090909, 9.14141414, 9.19191919, 9.24242424,  
9.29292929, 9.34343434, 9.39393939, 9.44444444, 9.49494949,  
9.54545455, 9.59595959, 9.64646465, 9.69696969, 9.74747475,  
9.79797979, 9.84848485, 9.89898989, 9.94949495, 10. ]]
```

So, all these things are very easy to explore and generally it is actually much better than using Python. Of course, behind the scene Python works, but many things have been simplified in SageMath and that is what we will see as we go along.

So, at the end let me leave you with some simple exercises. So, you can try these exercises which will which will actually kind of test whatever things that we have done in this particular session.

(Refer Slide Time: 27:59)

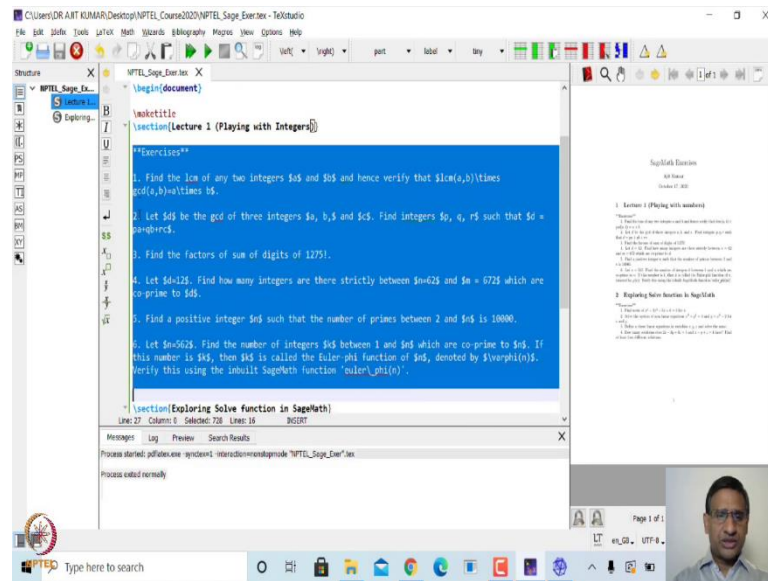


The screenshot shows a Jupyter Notebook window titled "CMSM_Lecture3-1 (unsaved changes)". The notebook contains a single code cell with a list of numbers. Below the code cell, there is a video feed of a man.

```
7.52525253, 7.57575758, 7.62626263, 7.67676768, 7.72727273,  
7.77777778, 7.82828283, 7.87878788, 7.92929293, 7.97979798,  
8.03030303, 8.08080808, 8.13131313, 8.18181818, 8.23232323,  
8.28282828, 8.33333333, 8.38383838, 8.43434343, 8.48484848,  
8.53535354, 8.58585859, 8.63636364, 8.68686869, 8.73737374,  
8.78787879, 8.83838384, 8.88888889, 8.93939394, 8.98989899,  
9.04040404, 9.09090909, 9.14141414, 9.19191919, 9.24242424,  
9.29292929, 9.34343434, 9.39393939, 9.44444444, 9.49494949,  
9.54545455, 9.59595959, 9.64646465, 9.69696969, 9.74747475,  
9.79797979, 9.84848485, 9.89898989, 9.94949495, 10. ]]
```

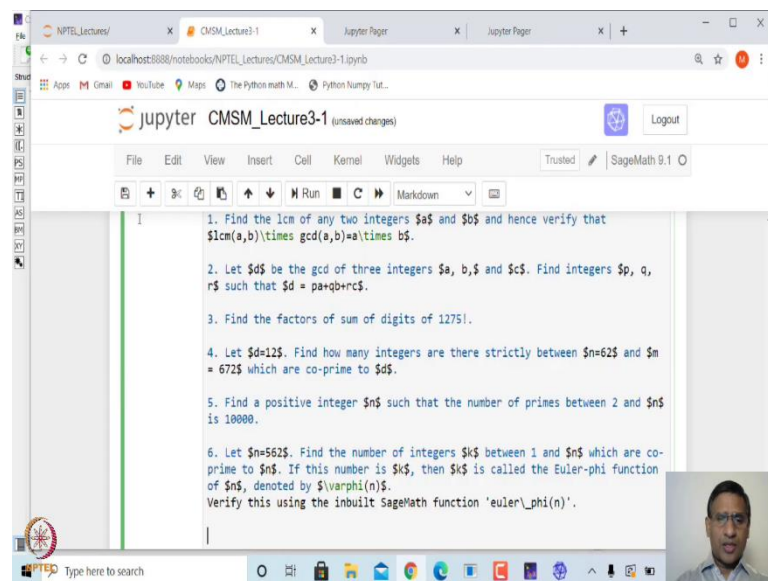
So, let me just convert this into markdown.

(Refer Slide Time: 28:06)

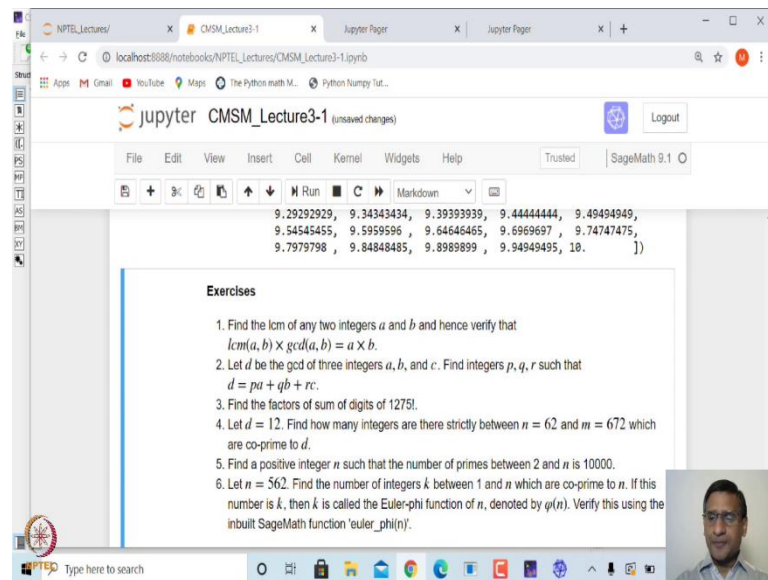


And let me just bring the list of exercises which just copy paste this, yeah.

(Refer Slide Time: 28:12)



(Refer Slide Time: 28:13)



So, these are some exercises you should try. So, for example, find gcd and lcm of two integers and check that multiplication of gcd and lcm is the product of the two numbers. You can just check whether the Bezout's identity can be extended to three integers a, b, c, d . You can find what are the factors of sum of digits in 1275 factorial and the next one is to find out number of co primes between two integers 62 and 766.

And then similarly this I already said you can find integer n such that the number of primes between 2 and n is 10000. And the last one is to deal with `euler_phi()` function which I already told you, but this is you can explore about 562. And you can look at what are the properties of this euler phi function. If you are interested in case you have not done some course in number theory you can look at let us say Google help about euler phi function and try to verify some of the property.

So, thank you very much. In the next lecture we will see some more things in SageMath.