Computational Mathematics with Sagemath

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

Lecture – 04
Lists in Python

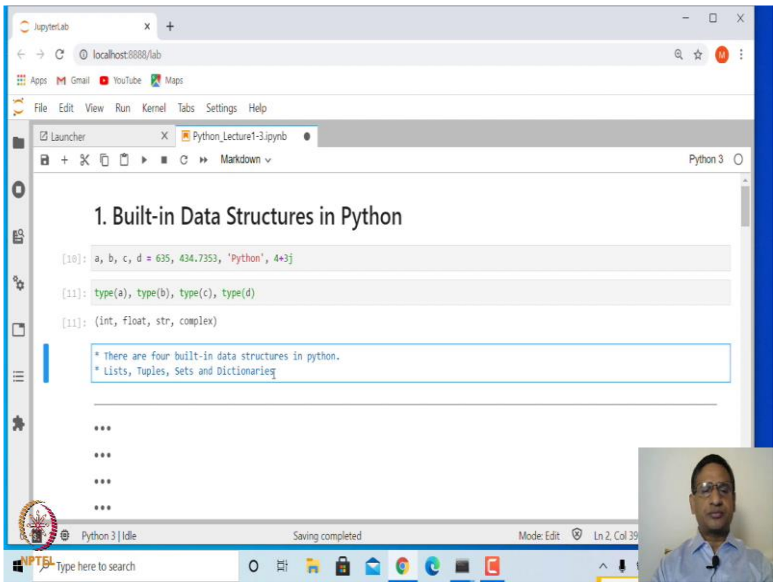
Good morning students and welcome to the third lecture on Computational Mathematics with SageMath. In the last two lectures we looked at how we can make use of Python as an advanced scientific calculator.

We made use of two modules, namely **math** module and **cmath** module and using these two modules we explored basic operations such as addition, multiplication, division, finding quotient, remainder, raising power and things like that.

So, we used basically variables which were floating point variable or integer variables, we also saw complex variables. We also saw that we can use round bracket and we can also use curly bracket. Curly bracket we used, for giving variable names inside print and using f string.

In today's lecture, we will look at other important built-in data structure types such as list, tuples, dictionaries and sets. We will also see use of other brackets such as square bracket and curly brackets. They will have different meaning.

(Refer Slide Time: 01:54)



1. Built-in Data Structures in Python

```
[10]: a, b, c, d = 635, 434.7353, 'Python', 4+3j

[11]: type(a), type(b), type(c), type(d)

[11]: (int, float, str, complex)
```

```
* There are four built-in data structures in python.
* Lists, Tuples, Sets and Dictionaries
```

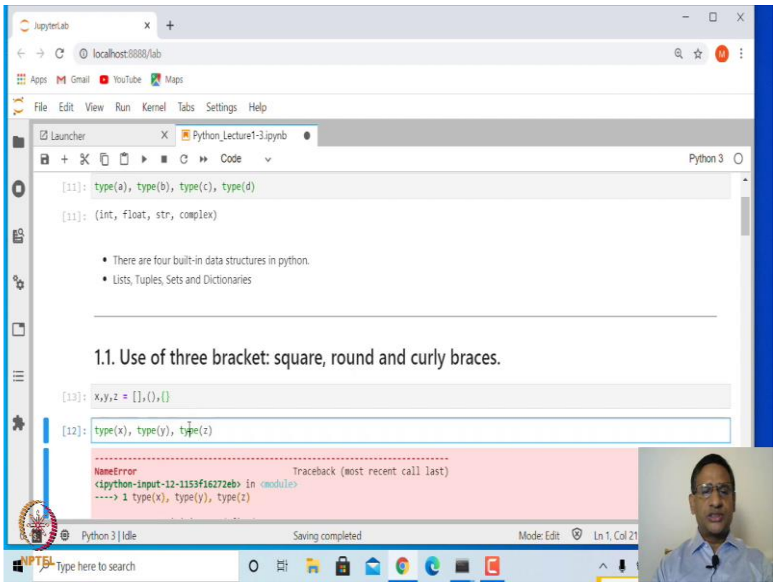
So, let us get started. Today actually, I am using Jupiter lab . You can start Jupiter lab by opening Anaconda Prompt and then type Jupiter space lab. In case you have installed

Anaconda as an administrator, as an admin you may need to run this using admin. In case you get error while starting this as an user then you can start as admin.

So, let us start . Suppose we have variables a, b, c, d and a is let us say, 635, b is 434.7353, c is python as a string and then d is 4 plus 3j, that is, imaginary number. You can find type of each of this variable by just typing a command type in the bracket a, type b, type c, type d.

So, if you execute, that is, run this it says that a is an integer variable , b is a floating-point variable, c is a string and d is complex . So, these are the data types that we have already used. But, apart from these, python has four important built-in data structures, namely, list, tuples, sets and dictionaries.   And in mathematics, we make use of all these things quite frequently. Dictionary is something which is very handy data type and we will see its use later.

(Refer Slide Time: 03:58)



```
[11]: type(a), type(b), type(c), type(d)

[11]: (int, float, str, complex)
```

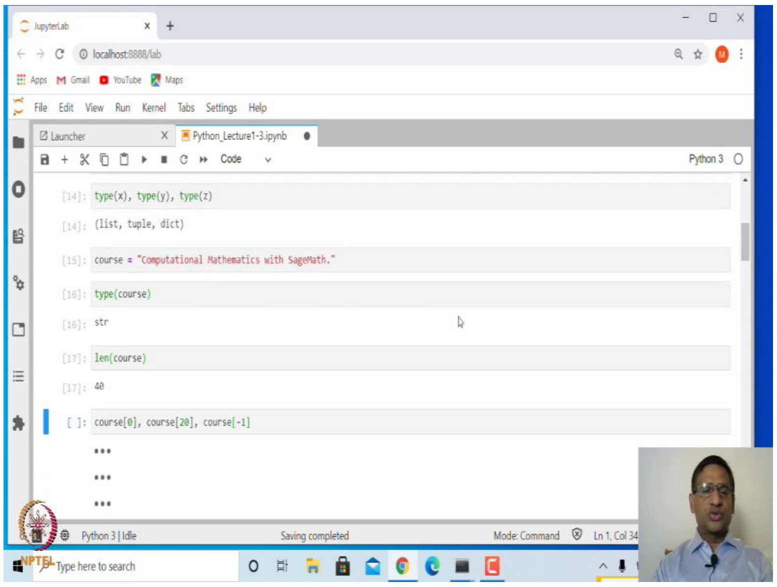
- There are four built-in data structures in python.
- Lists, Tuples, Sets and Dictionaries

1.1. Use of three bracket: square, round and curly braces.

```
[13]: x,y,z = [],(),{}

[12]: type(x), type(y), type(z)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-1153f16272eb> in <module>
----> 1 type(x), type(y), type(z)
```

Now, let us look at how we can make use of these other brackets like square bracket, round bracket and curly-brackets. We will start with a list, tuple and dictionary. Suppose, we define, let us say, x is equal to empty square bracket, y is equal to empty round bracket and z is equal to empty curly bracket. Now, if we ask for what are the types of these variables x, y, z, then what we get, x is a list, y is a tuple and z is a dictionary. Therefore, the list is written inside square bracket, tuple inside round bracket and dictionary inside curly bracket.

(Refer Slide Time: 04:39)



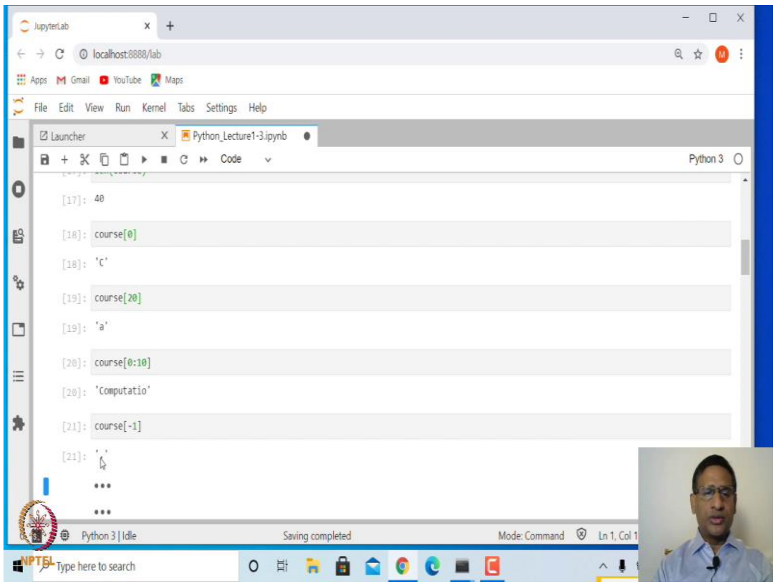
```
[14]: type(x), type(y), type(z)
[14]: (list, tuple, dict)
[15]: course = "Computational Mathematics with SageMath."
[16]: type(course)
[16]: str
[17]: len(course)
[17]: 40
[ ]: course[0], course[20], course[-1]
     ...
     ...
     ...
```

How do we define a list ? Before we define a list formally, we have already seen that we can have a string variable or you can type a sentence inside double quote or single quote. Suppose we store this in a variable called course which is inside double quote Computational Mathematics with SageMath.

This can also be thought of actually as a list, for example, you can find out what is the type of this variable course. It also says it is a string variable that is what we have already seen. Now suppose we want to find what is the length of this string variable course, you can use a command **len** and if you run this, you will get 40.

So, the length of this string variable is 40, that means, it has 40 characters including spaces and the full stop at the end. You can also take any slice of this variable. How do we take its slice? For example, if you want what is the first character in this string variable, you can say course and square bracket 0. So, that is capital C.

(Refer Slide Time: 06:42)

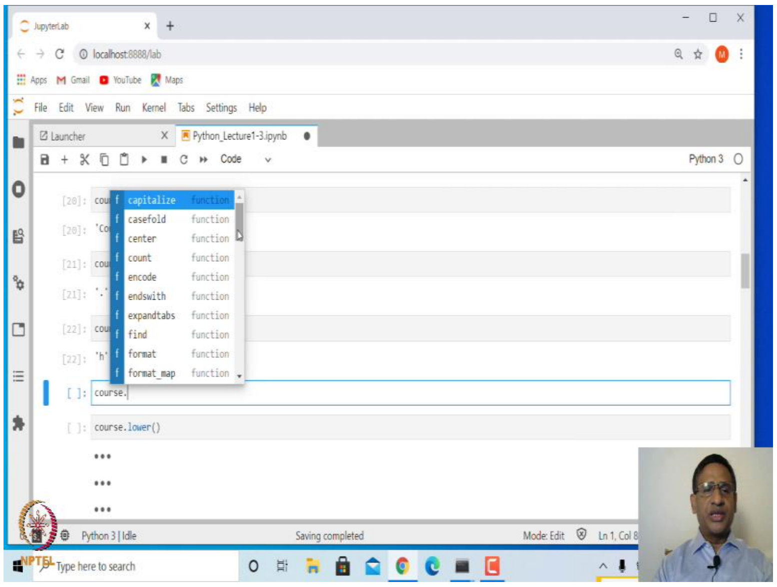


```
[17]: 40

[18]: course[0]

[18]: 'C'

[19]: course[20]

[19]: 'a'

[20]: course[0:10]

[20]: 'Computatio'

[21]: course[-1]

[21]: '.'
```

If I want to find out, what is for example, 20th character in this, we can say course and 20. Before that let me let me also say, I have said course 0. Here actually each character has got index and it starts from 0, course 0 is first character, course 1 will be the second character, which is o and so on.

So, when I say course 20 this this gives me 21st character in this string variable, namely, course. So, this the 20th character is a somewhere here, it will come as 21st character. Similarly, you can even ask for all the characters starting from 0 to 10. Colon 0 to 10 means first 10 characters it will get. So, if I execute this, I will get 'Computatio'. These are the first 10 characters in this variable course. You can even have negative index. So, for example, if I say course minus 1, then what you see is this is a dot that is full stop, which was the last character in this variable course.

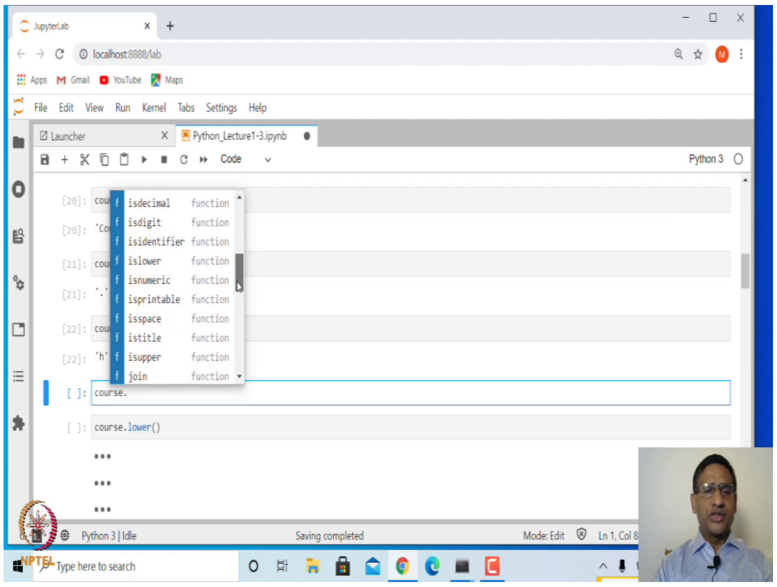
(Refer Slide Time: 08:23)



```
[20]: cou
[20]: 'Co
[21]: cou
[21]: '.'
[22]: cou
[22]: 'h'
[ ]: course.
[ ]: course.lower()
```

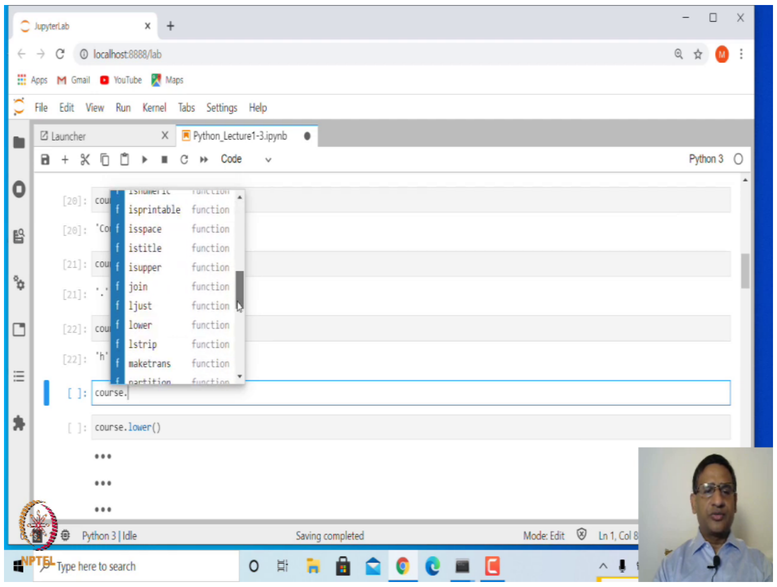
So, minus 1 gives you the last character, if I say minus 2, for example, if I say course in the square bracket minus 2, it will give me second last character, that is, h. You can also use other methods on this.

(Refer Slide Time: 09:02)



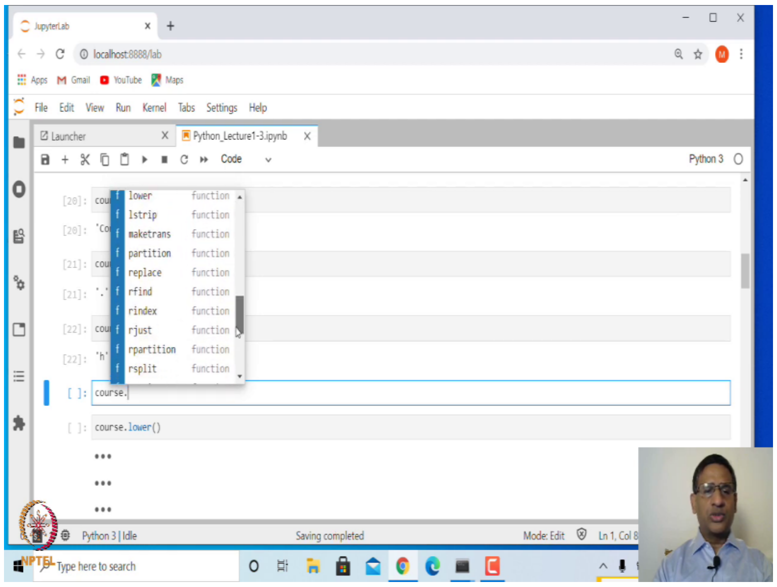
```
[20]: cou
[20]: 'Co
[21]: cou
[21]: '.'
[22]: cou
[22]: 'h'
[ ]: course.
[ ]: course.lower()
```

(Refer Slide Time: 09:06)



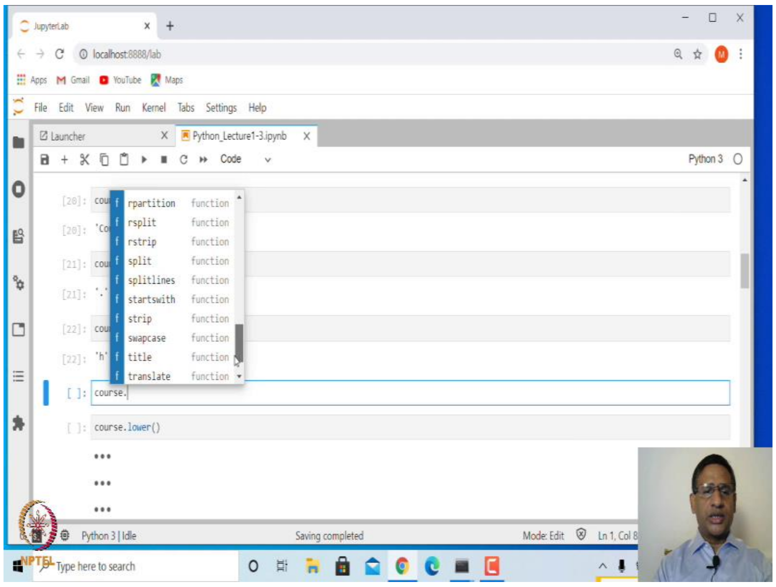
```
[20]: cou   isprintable   function
[20]: 'Co   isspace       function
[21]: cou   istitle       function
[21]: '.'   isupper       function
[22]: cou   join          function
[22]: 'h'   ljust         function
            lower         function
            lstrip        function
            maketrans     function
            partition     function
[ ]: course.

[ ]: course.lower()
```

(Refer Slide Time: 09:07)



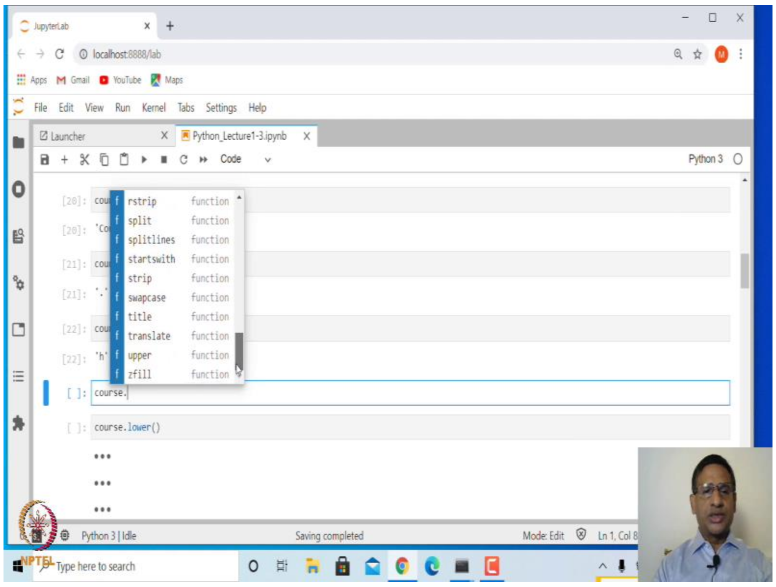
```
[20]: cou   lower         function
[20]: 'Co   lstrip        function
[21]: cou   maketrans     function
[21]: '.'   partition     function
[22]: cou   replace       function
[22]: 'h'   rfind         function
            rindex        function
            rjust         function
            rpartition    function
            rsplit        function
[ ]: course.

[ ]: course.lower()
```

(Refer Slide Time: 09:08)



```
[20]: cou  rpartition  function
[20]: 'Co  rsplit      function
[21]: cou  rstrip      function
[21]: '.'  split       function
[22]: cou  splitlines  function
[22]: 'h'  startswith  function
           strip       function
           swapcase    function
           title       function
           translate   function
[ ]: course.
[ ]: course.lower()
```

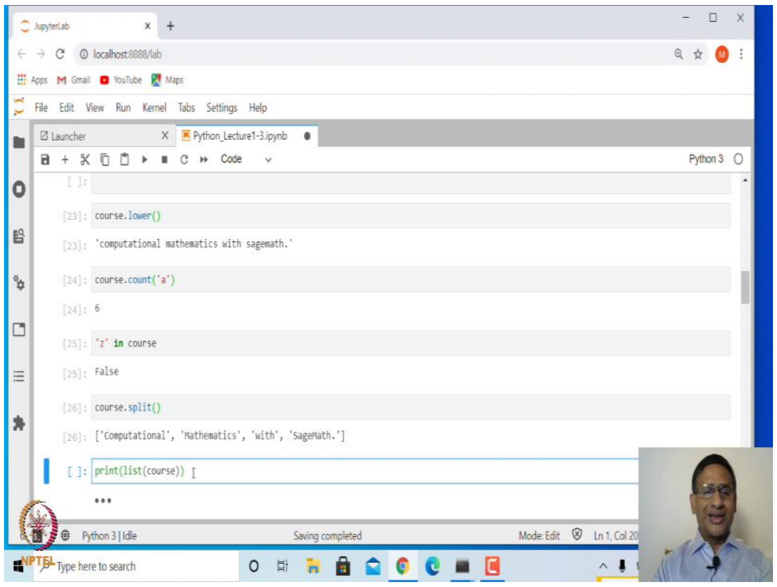
(Refer Slide Time: 09:11)



```
[20]: cou  rstrip      function
[20]: 'Co  split       function
[21]: cou  splitlines  function
[21]: '.'  startswith  function
[22]: cou  strip       function
[22]: 'h'  swapcase    function
           title       function
           translate   function
           upper       function
           zfill       function
[ ]: course.
[ ]: course.lower()
```

For example, you can you can look at course and then dot and then press tab, you will see several other methods that you can apply on this variable course. I encourage you to explore each of this function by taking help on this. So, for example, if I say, capitalize, or, if I say course dot lower it will convert everything in lower case.

(Refer Slide Time: 09:24)



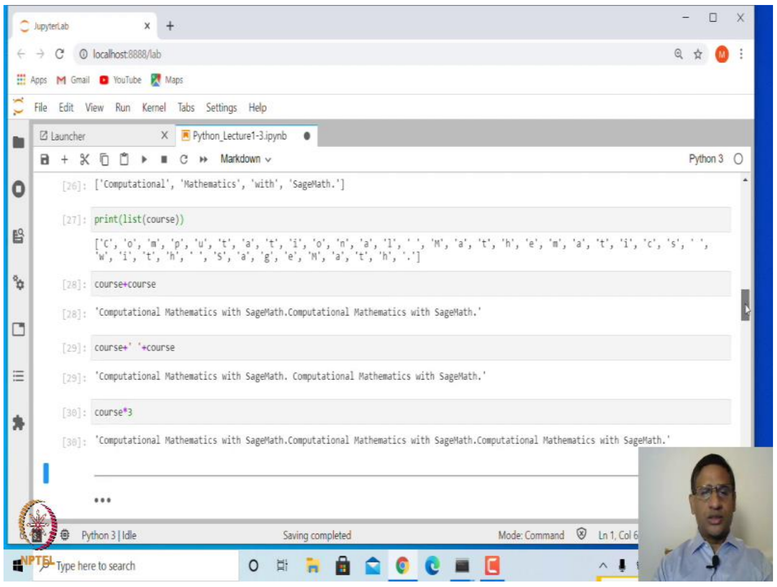
```
[ ]:

[23]: course.lower()

[23]: 'computational mathematics with sagemath.'

[24]: course.count('a')

[24]: 6

[25]: 'z' in course

[25]: False

[26]: course.split()

[26]: ['Computational', 'Mathematics', 'with', 'SageMath.']

[ ]: print(list(course))
```

If I say course dot count and character a , it will count how many times a has appeared in this variable course. And if I want to check whether some character is there in this variable course or not, we can simply use character z. You have to write inside single quote or double quote and then say in course. So, that is how you can check whether some character lies inside this course.

You can also use split command and let us see what it does? course dot split empty round bracket , what it does? it converts this particular sentence into a list of words. You can see here this is in square bracket and that is, how a list is defined.   A list is defined by writing the values, entries inside square bracket separated by comma.

Here course dot split has made this as a list which has four words, the first one is 'Computational', second one is 'Mathematics', third one is 'with' and the fourth one is 'Sagemath dot', full stop. So, you can also convert course into a list. This list in the bracket course will convert into a list of characters. Then you can print this and let us see what we get? So, this is a list of characters.

(Refer Slide Time: 11:11)



```
[26]: ['Computational', 'Mathematics', 'with', 'SageMath.']

[27]: print(list(course))

      ['C', 'o', 'm', 'p', 'u', 't', 'a', 't', 'i', 'o', 'n', 'a', 'l', ' ', 'M', 'a', 't', 'h', 'e', 'm', 'a', 't', 'i', 'c', 's', ' ',
      'w', 'i', 't', 'h', ' ', 'S', 'a', 'g', 'e', 'M', 'a', 't', 'h', '.']

[28]: course+course

[28]: 'Computational Mathematics with SageMath.Computational Mathematics with SageMath.'

[29]: course+' '+course

[29]: 'Computational Mathematics with SageMath. Computational Mathematics with SageMath.'

[30]: course*3

[30]: 'Computational Mathematics with SageMath.Computational Mathematics with SageMath.Computational Mathematics with SageMath.'
```

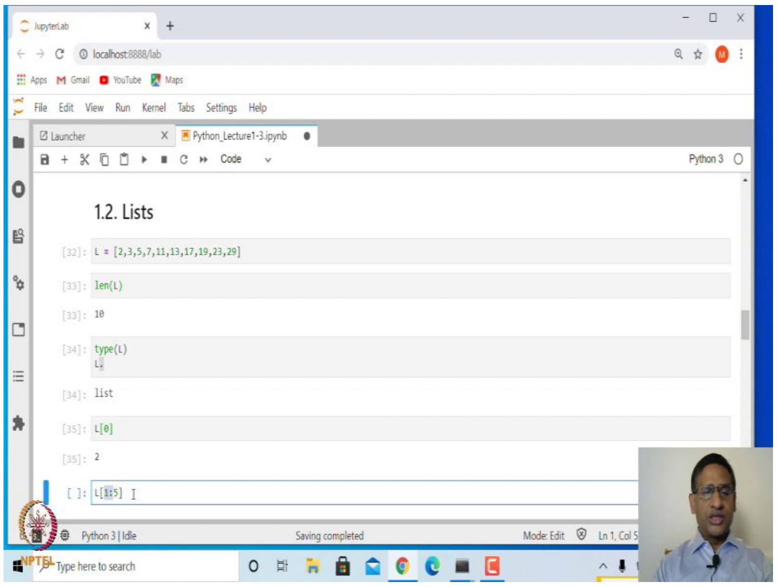
If you want to count for example, how many characters are there in this particular course variable, you can simply say length, len and then list and then course, you can even add two string variables.

If I say course plus course, what it is going to do? It will add the course to the existing course. It is not giving any space in between and that we possibly would have wanted.

We can add space in between. So, what we can do is, we can say course and in between we can say, empty space as a character plus this and now you will see it has added space in between.

You can even multiply this course by 3 or 5. So, what if I say course star 3 it will repeat this course 3 times. So, you will get Computational Mathematics with SageMath, Computational Mathematics with SageMath, Computational Mathematics with SageMath 3 times. If you can any string variable, you can think of that also as a list, where you can take any slice of that string variable.

(Refer Slide Time: 12:48)



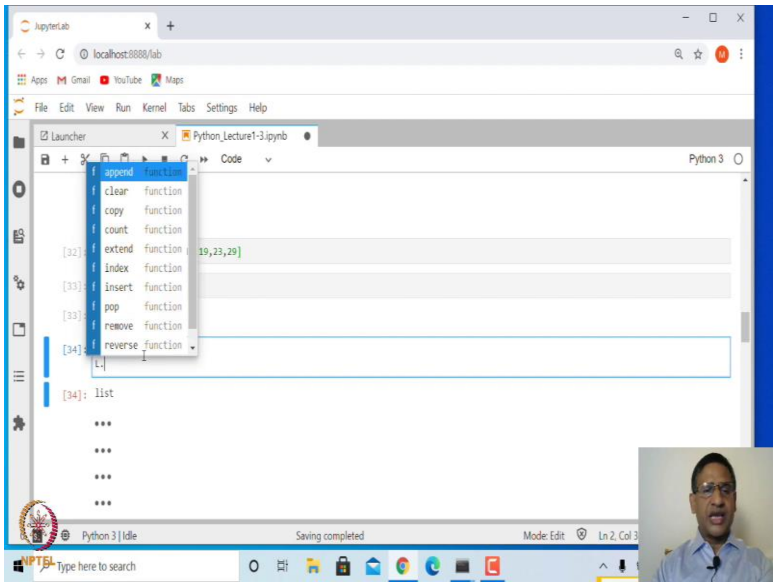
1.2. Lists

```
[32]: L = [2,3,5,7,11,13,17,19,23,29]

[33]: len(L)

[33]: 10

[34]: type(L)
      L.

[34]: list

[35]: L[0]

[35]: 2

[ ]: L[1:5]
```

Now, let us see how we can define a list. We have already seen when I converted this course into a list it gave me list of character. And course dot split also gave me list of words. So, we already know that list can be created by using square bracket and by giving its entries separated by commas.

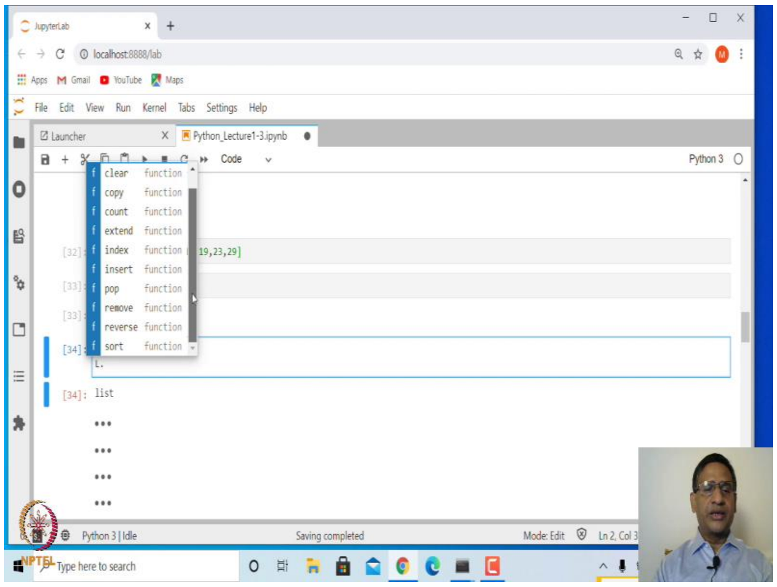
Suppose we have a list, we create a list called L, which is 2, 3, 5, 7, 11, 13, 17, 19, 23 , 29, that is a list. Let us execute this and then we can find out what is length of this list. And as I said you can find out all the methods that you can apply on this variable L. We can ask what is type of L. This is a list.

(Refer Slide Time: 13:58)



```
[32]: ...19,23,29]
[34]: L.
[34]: list
```

(Refer Slide Time: 14:02)

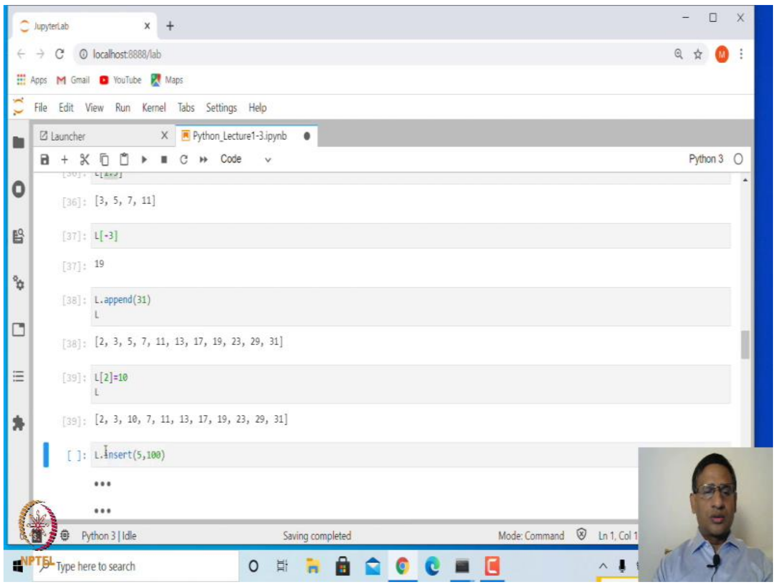


```
[32]: ...19,23,29]
[34]: L.
[34]: list
```

We can say, L dot and then press tab, then again you will see the list of methods that can be applied to L. For example, you can append something in the list, you can clear, copy, count, extend, index, insert, pop, remove, reverse. I again encourage all of you to go through this and try to explore each one of these method.

Similarly, you can again find the first entry in this list by giving the index 0, which is L square bracket 0, this is the first entry, which is 2, if I want to find out the second entry to 5th entry, this will be L 1 colon 5. 3 is the second one entry.

(Refer Slide Time: 14:50)



```
[36]: [3, 5, 7, 11]

[37]: L[-3]

[37]: 19

[38]: L.append(31)
      L

[38]: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

[39]: L[2]=10
      L

[39]: [2, 3, 10, 7, 11, 13, 17, 19, 23, 29, 31]

[ ]: L.insert(5,100)
```

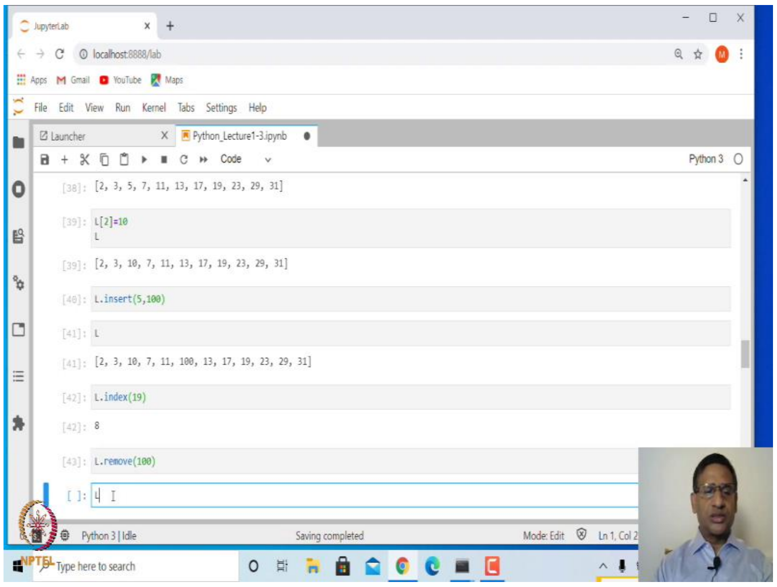
So, it has index 1, because as I said, the starting index of a list is 0 not 1, and this is the second entry, third entry, fourth entry and fifth entry.

So, when we say 1 colon 5 it generates a list 1 2 3 4 and L of that gives, L of 1, L of 2, L of 3, L of 4. Similarly, you can ask for L of minus 3, we already know what it will give the third last entry in this list, which is minus 19.

If I say L dot append and in the round bracket if I give 31, then let us see what it does? It will append 31 at the end. So, earlier it was stopping at 29, now we have 31 which is appended at the end. Similarly, you can say L of 2 is equal to 10, what it will do? It will replace the third entry which is here 5 by 10.

So, you can change any entry inside this list by this command, you can say L of let us say k which is the k plus 1-th entry and change that value. So now, if I say, what is L? you can see here the third entry had changed to 10 earlier it was 5.

(Refer Slide Time: 16:52)

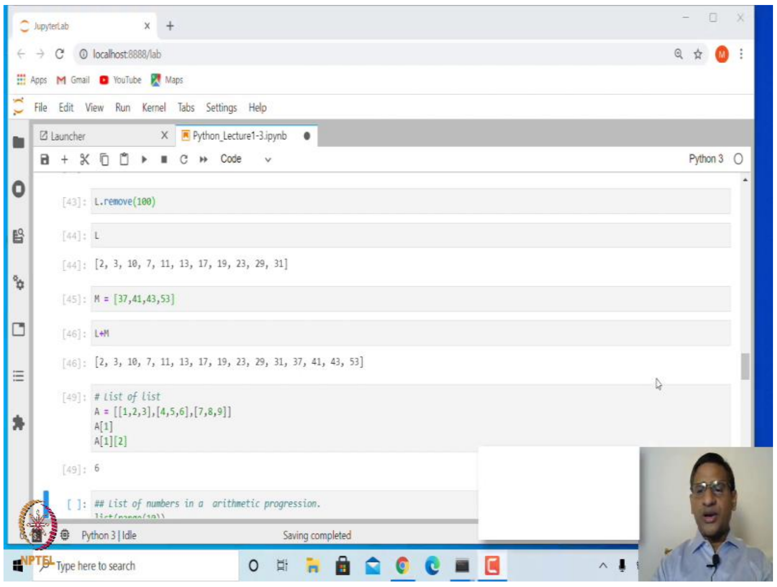


```
[38]: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

[39]: L[2]=10
      L

[39]: [2, 3, 10, 7, 11, 13, 17, 19, 23, 29, 31]

[40]: L.insert(5,100)

[41]: L

[41]: [2, 3, 10, 7, 11, 100, 13, 17, 19, 23, 29, 31]

[42]: L.index(19)

[42]: 8

[43]: L.remove(100)

[ ]: L
```

You can also insert an entry at any index in between. Suppose, if I want to insert 100, and I give the index 5 here,   then what it it will do? You can just check what L is now? Now, 100 has come at which place? first, second, third, fourth and fifth at fifth index, sorry at sixth index because we have put 5 this means the sixth entry sixth entry has become 100.

So, you can insert some value at the end or you can insert in between. You can also find out index of a particular entry inside a list. For example, if I say L dot index 19. Here 19 is eighth index, that means, ninth entry. The ninth entry in this list is 19. We can also remove a particular entry. For example, we inserted 100 and suppose we want to remove and so, we can simply say L dot remove 100, now if you ask what is L? Then you do not see 100 .

(Refer Slide Time: 18:03)



```
[43]: L.remove(100)

[44]: L

[44]: [2, 3, 10, 7, 11, 13, 17, 19, 23, 29, 31]

[45]: M = [37,41,43,53]

[46]: L+M

[46]: [2, 3, 10, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 53]

[49]: # List of list
      A = [[1,2,3],[4,5,6],[7,8,9]]
      A[1]
      A[1][2]

[49]: 6

[ ]: ## List of numbers in a  arithmetic progression.
```

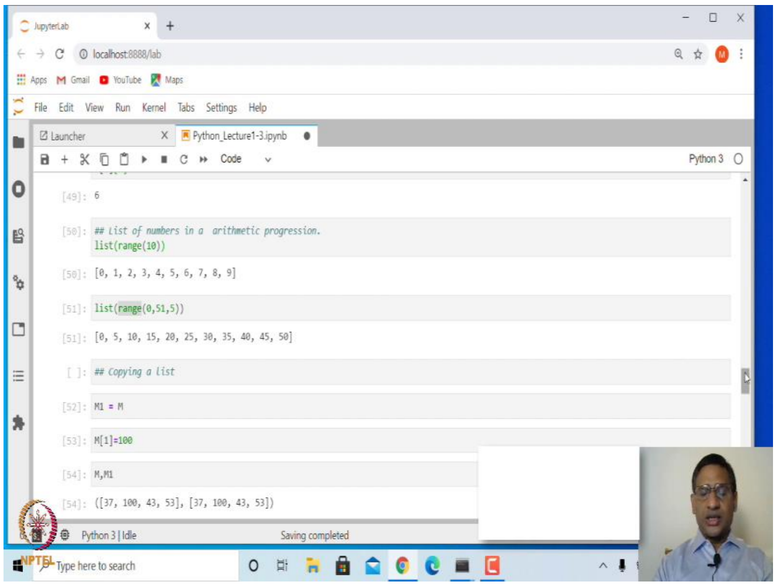
You can also do various other things. You can explore other methods by using lot tab. So, it also means that you do not actually need to remember all these commands. At any stage you can simply type dot and then press tab, all the methods will appear in drop down menu and from that you can choose.

So, suppose we create another list M which is 37, 41, 43 and 53, and we can add L and M. So, if I say L plus M, what will it do? It will append the entries of M in entries of L. So, you can see here after 31 now 37 to 53 has been appended. You can even create a list inside a list. For example, if I have A and which has 3 list namely 1, 2, 3, and then 1, 2, 3, 4, 5, 6 and 7, 8, 9. So, you can create nested list or a list inside a list. This you can think of as a matrix in fact.

We will look at how to define matrix and do various computations with matrices. Let us define this, and now suppose we want to find out what is A of 1, that means, the second entry in the list which is again a list. This is 4, 5, 6.

If I want to find out what is A of 1 and then inside this let us say 2. So, what will it give? A of 1 is 4, 5, 6 and the 2 will be the third entry which is 6. So, that is what you will get. That is how you can access any entry inside a nested list.

(Refer Slide Time: 20:21)



```
[49]: 6

[50]: ## List of numbers in a  arithmetic progression.
      list(range(10))

[50]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[51]: list(range(0,51,5))

[51]: [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

[ ]: ## Copying a list

[52]: M1 = M

[53]: M[1]=100

[54]: M,M1

[54]: ([37, 100, 43, 53], [37, 100, 43, 53])
```

Apart from creating a list by giving the entries in square bracket, Python also has one inbuilt command known as **range**.

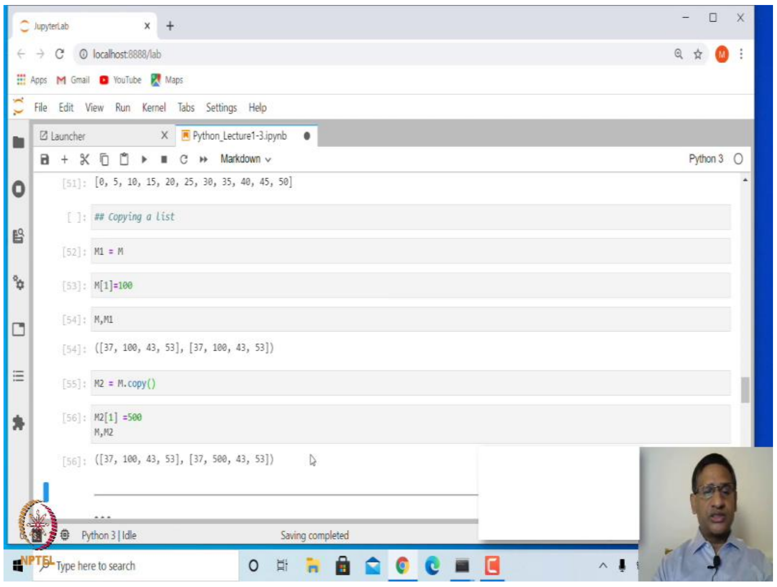
So, if I say range 10, it will create a list starting from 0 up to 9. So, 0, 1, 2, 3, 4 up to 9. So list of range 10 has created 0, 1, 2, 3 up to 9 as a list. In case you want a list for example, as an arithmetic progression say 0 to 51 and increment is 5, you will get 0, then 5 and then 10 up to 50, you will get.

So, that is what you can you can see here, it is taking time, 0, 5 up to 50. You can create a arithmetic progression using range command and where the increment has to be integer . Later on, we will see how to have non integer increment for example, floating point increment or decimal value.

Another very important thing is, suppose you have two lists. Suppose, let us say you have created a list M just now, which is a which is actually 37, 41, 43, 53 and suppose we want to store M in M1, let us say M1 is equal to M .

So, let us store that. Now suppose we change the second entry in M as 100. Now let us see what happens to M and M1? We know that have changed the second entry as 100, now what happens to M1.

(Refer Slide Time: 22:49)



```
[51]: [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

[ ]: ## Copying a list

[52]: M1 = M

[53]: M[1]=100

[54]: M,M1

[54]: ([37, 100, 43, 53], [37, 100, 43, 53])

[55]: M2 = M.copy()

[56]: M2[1] =500
      M,M2

[56]: ([37, 100, 43, 53], [37, 500, 43, 53])
```

So, you can see here for both M and M1, second entries have changed to 100. I whether we wanted that. In case, we did not want the entry of M1 to change, then this is not a good idea.

So, actually what it does is that, when you say M1 is equal to M, then it is just renaming that variable, it will have the same address. So, any changes that you make to M will be reflected in M1 and vice versa.

So, this is not a good idea. Many times we create a list, store that list in in some other list. So, that we do not lose the original one. So, in this case this M1 equal to M is not going to help. In that case, what we can do is, instead of just equality, we can say M dot copy.

So, let us say M dot copy and store it in M2. Now if say M2 of 1 is equal to 500, and then let us say what is M and M2?.Now you can see that the second entry of M is 100 whereas, second entry of M2 has changed to 500.

So, this is how you can copy a list to another list and you can play with the second one so, that the original list, there will not be any change in the original list.

Thank you, very much next time we will look, at how to deal with tuples and how to deal with dictionary.