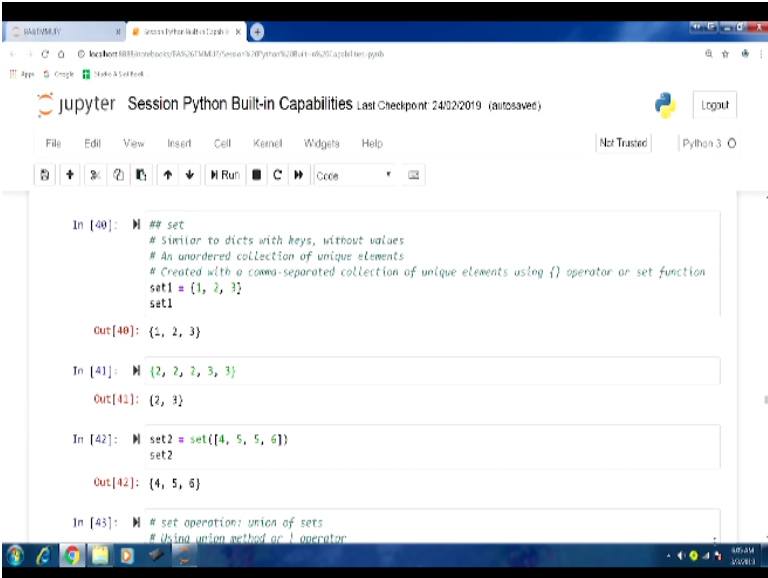


Business Analytics And Text Mining Modeling Using Python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology-Roorkee

Lecture-14
Built-in Capabilities of Python-VI

Welcome to the course business analytics and text mining modeling using python. So in previous lectures, we have been discussing data structures in python language. And specifically, we talked about the set objects in previous lectures. So let us do a recap of what we discussed related to set, and then we will just pick up from the point where we stopped in the previous lecture. So we talked about sets similar to dicts with keys.

(Refer Slide Time: 00:52)



```
In [40]: # set  
# Similar to dicts with keys, without values  
# An unordered collection of unique elements  
# Created with a comma-separated collection of unique elements using {} operator or set function  
set1 = {1, 2, 3}  
set1  
  
Out[40]: {1, 2, 3}  
  
In [41]: {2, 2, 2, 3, 3}  
  
Out[41]: {2, 3}  
  
In [42]: set2 = set([4, 5, 5, 6])  
set2  
  
Out[42]: {4, 5, 6}  
  
In [43]: # set operation: union of sets  
# Using union method or | operator
```

And you know, without values, or an order collection of unique elements. So created with the help of curly braces or set function. So what we will do, we will quickly run through some of you know these lines of code, so that we are able to refresh some of the things that we discussed in the previous lecture.

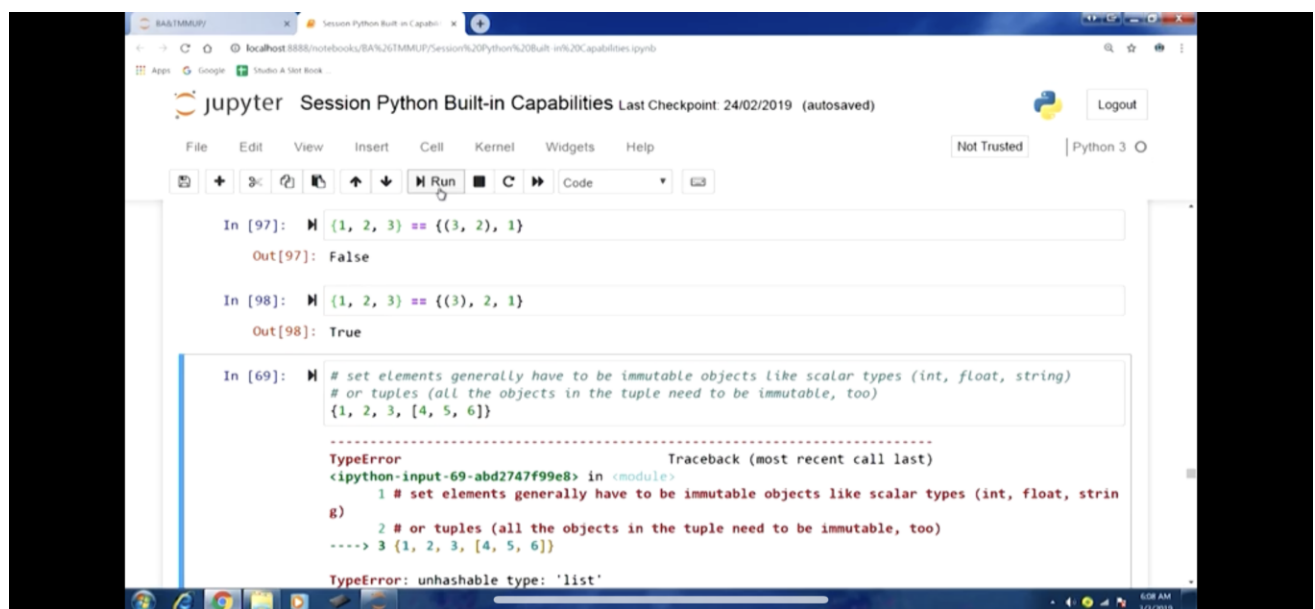
(Video Starts: 01:13)

So, this is how we can create just a second. So, this is how we can create this set object press on this, you can see unique elements 1 2 3 are part of this, then again, we pass it you know list of elements, only the unique elements are taken and set object is created. Then we talked about

few set operations, for example, union of set, so we talked about union method and or operator, few more examples that we had gone through.

And then we talked about intersection of sets, intersection method and ampersand operator. We gone through about these few more examples. And then we talked about updating a set. So we also said that these are in place counterparts of union method and or operator that we talked about. So let us go through this aspect. Then we did few more examples for the same. We talked about another set operation, adding elements talked about removing elements 2 method is specifically remove method and pop.

(Refer Slide Time: 03:47)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running and saving code. The notebook content shows three code cells:

```
In [97]: {1, 2, 3} == {(3, 2), 1}
Out[97]: False

In [98]: {1, 2, 3} == {(3), 2, 1}
Out[98]: True

In [69]: # set elements generally have to be immutable objects like scalar types (int, float, string)
# or tuples (all the objects in the tuple need to be immutable, too)
{1, 2, 3, [4, 5, 6]}

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-69-abd2747f99e8> in <module>
      1 # set elements generally have to be immutable objects like scalar types (int, float, string)
      2 # or tuples (all the objects in the tuple need to be immutable, too)
----> 3 {1, 2, 3, [4, 5, 6]}
```

The error message at the bottom indicates a `TypeError: unhashable type: 'list'`, which occurs because lists are mutable and therefore cannot be elements of a set.

And how they could be different that you can notice from the examples, here you can see the pass on the actual element that we want to remove using the method and the pop method, the arbitrary element is actually removed. So you can see through the examples also, as I am running, then clearing, if you want to clear set down, we can use the clear method. So you can see here, if we try to delete element from an empty set object.

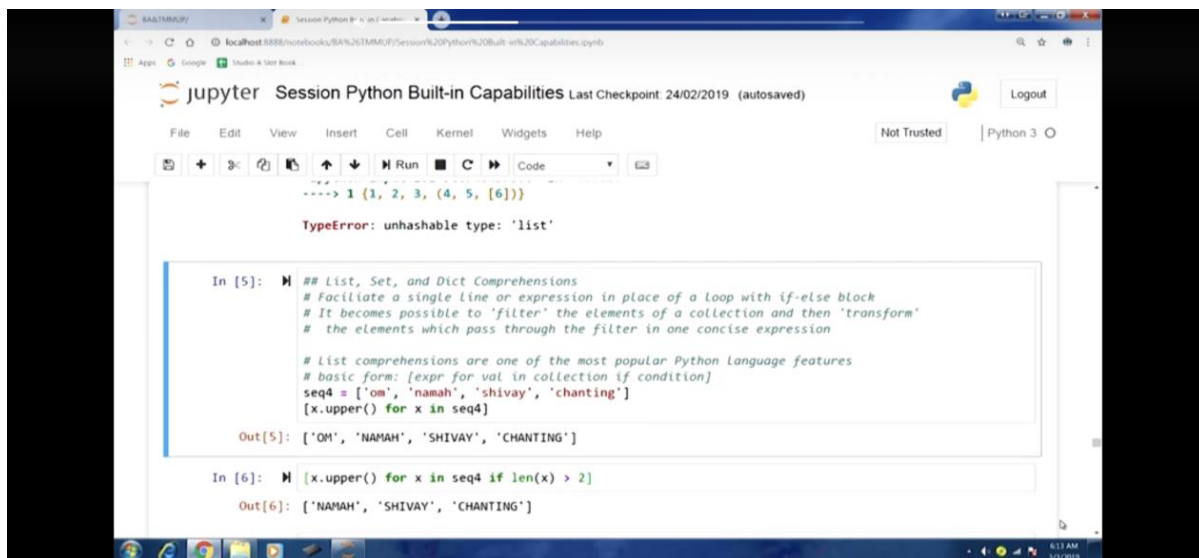
And we will get an error, you can see this. Now, we also talked about some other things operation like subset, superset, disjoint set. So we talked about this also, then we talked about if

we want to check whether the contents of 2 set objects, whether effectively, they are similar or not. So let us run this, you can see through these examples, the part that we discussed. So I think we stopped at this point.

So let us discuss from here. And a few more things about, you know, set objects are set elements that generally have to be immutable objects, just like you know, we talked about the keys, and they are supposed to be immutable objects. And we also said that set objects are all you know, similar to you know dict with keys without values. So here also the elements, set elements are supposed to be immutable objects, like a scalar types in float string, or tuples.

And when we are saying tuples, the elements, which are part of tuple they are also supposed to be immutable. So let us understand this through in a few examples. So we are creating a set of few elements here. First one is 1, then 2, then 3 are these 3 first the elements are scalar types, then the fourth element is a list 4 5 6. So because list is mutable, so therefore, we should get an error.

(Refer Slide Time: 08:25)



```
-----> 1 {1, 2, 3, (4, 5, [6])}

TypeError: unhashable type: 'list'

In [5]: ## List, Set, and Dict Comprehensions
        ## Faciliate a single line or expression in place of a loop with if-else block
        ## It becomes possible to 'filter' the elements of a collection and then 'transform'
        ## the elements which pass through the filter in one concise expression

        # List comprehensions are one of the most popular Python language features
        # basic form: [expr for val in collection if condition]
        seq4 = ['om', 'namah', 'shivay', 'chanting']
        [x.upper() for x in seq4]

Out[5]: ['OM', 'NAMAHA', 'SHIVAY', 'CHANTING']

In [6]: [x.upper() for x in seq4 if len(x) > 2]

Out[6]: ['NAMAHA', 'SHIVAY', 'CHANTING']
```

So if I run this, and try to create, you know, a set object, I will get an error, as you can see, and the error, you can clearly notice unhashable type list. So now a few more examples related to

this, and are in the next example, the first of the elements 1 2 3 or scalar types, then I am passing on a tuple. So tuple is also made of the elements, which are immutable, scalar types. So this is ok, if I run this, I get the output.

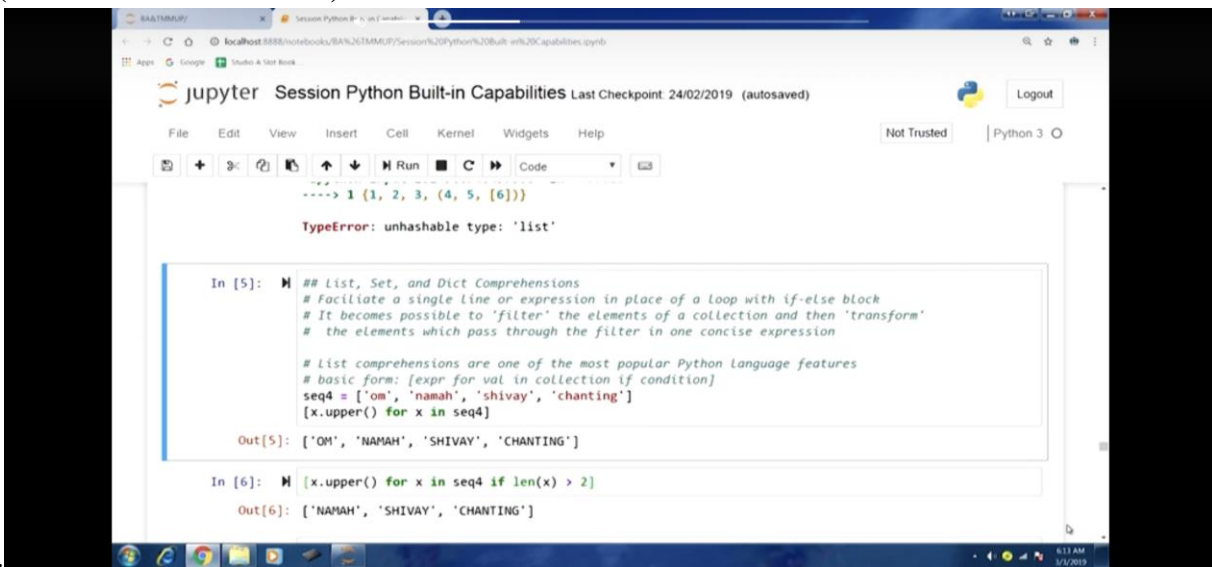
Now, next example that I am taking, you know, here I am creating a set. First 3 elements are again ok, and now the last element is a tuple however, you know first 2 elements of this tuple they are ok, the third element is again, I am using a list here. So again, this becomes a mutable object. So if I try to create this one, again, I will get an error. So you can see in unhashable type list.

So in this fashion, this will give you an idea again, about what can work in a set object and what cannot work. So sometimes, this is really useful while we are dealing with certain complex scenarios. Now, we will talk about one more aspect related to some of these data structures, specifically list set and dict. So there is a very popular language feature python language feature comprehension, so we call them list set and dict comparison.

We will talk them one by one however, let us discuss few points about these you know, comparisons and what they are about. So, these lists, set and dict, you know comprehension hasn't it a single line or expression in place of a loop you know, or and loop with else block. So, certain situation, when we are writing code will have to you know, use a loop and if else as you know, block in the sense conditional logic and all that, you know to complete our code.

Now, some of those situation where we are using for loop and you know, if else block, that can actually be replaced with a single line explanation, if you remember, in our previous session, we talked about ternary explorations. So, they are also the main idea was to use a single line or expression, here when we are dealing with list says you know, set ended comprehension, there also the idea is quite similar

(Refer Slide Time: 14:03)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The notebook content shows a Python session with the following code and output:

```
In [5]: ## List, Set, and Dict Comprehensions
        # Facilitate a single line or expression in place of a loop with if-else block
        # It becomes possible to 'filter' the elements of a collection and then 'transform'
        # the elements which pass through the filter in one concise expression

        # List comprehensions are one of the most popular Python language features
        # basic form: [expr for val in collection if condition]
        seq4 = ['om', 'namah', 'shivay', 'chanting']
        [x.upper() for x in seq4]

Out[5]: ['OM', 'NAMAH', 'SHIVAY', 'CHANTING']

In [6]: [x.upper() for x in seq4 if len(x) > 2]

Out[6]: ['NAMAH', 'SHIVAY', 'CHANTING']
```

At the top of the code cell, there is a line of code that has been executed, resulting in a `TypeError: unhashable type: 'list'` error:

```
--> 1 (1, 2, 3, (4, 5, [6]))
```

So, few more things are possible in this especially from the data analytics context, and also in the text mining modeling context, we use these comprehend sense, it becomes possible for us to filter the elements of a collection, and then transform the elements, which pass through the filter. So, given a certain number of elements, we can apply a filter, select the filter elements, and transform them.

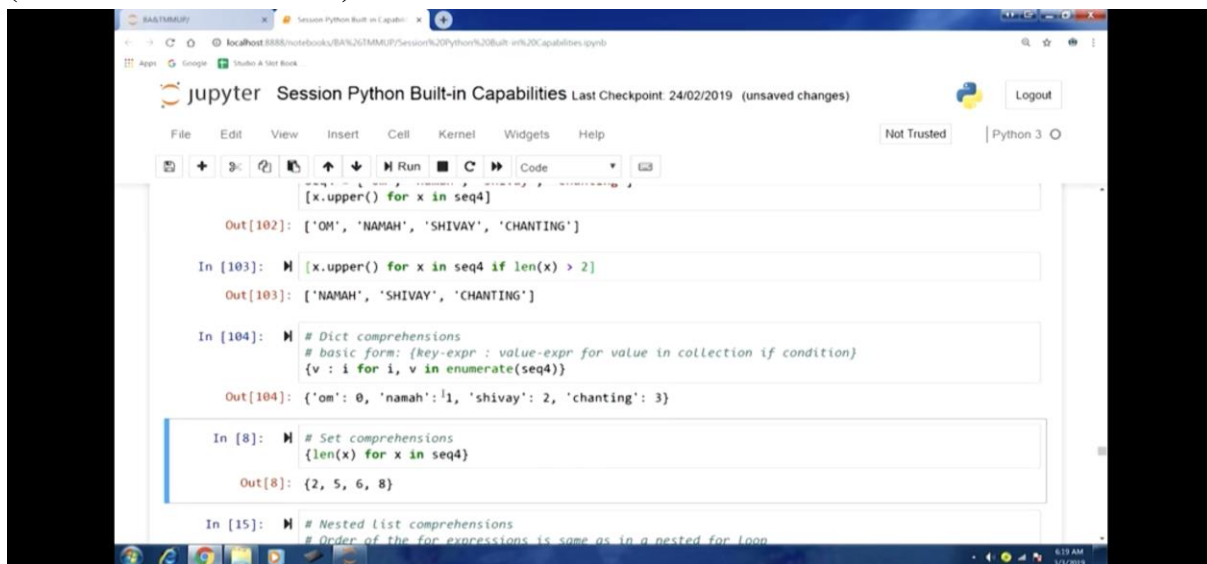
So, this process is, you know, which we can, you know, perform using a for loop and if else logic, so, you know, that we can do there that we can, you know, using these comprehensive, we can perform using a single line or expression. So, in especially in the data transformation, data processing, cleaning some of these steps that we are required to perform data analytics, including data mining, modeling, and text mining modeling.

So, you know, these language features could be really useful, so let us first start with list comprehension, they are one of the most popular python language features, and now the list of comprehension are actually coded. So, the basic form you can see the basic form within a list, that means, you know square brackets, first, you know, the exploration, this exploration is about transforming the filtered elements.

So, this exploration, we are using to actually whatever operation that we want to perform on the filter, you know, elements. So, first we you know, light our transforming expression, then, like for loop, we have this for keyword for value in collections, in essence, we are running for loop, so, for value in collection, and for you, and that means, for each of the elements in a collection, and we will run a loop and then filter.

So, filter is if conditions in a sense, when we are talking about loop and if else block, you know, where we would like to do filtering and whatever the filtered output is we would like to apply, you know, transform them, the same thing is, you know, can be coded in this fashion using a list comprehension. So, if condition is actually filtering, so we can, you know, select few elements from the collection that we have, and, you know, we can run a loop, select few, you know, filter few elements.

(Refer Slide Time: 19:50)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". 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 displays several Python snippets:

```
[x.upper() for x in seq4]

Out[102]: ['OM', 'NAMAH', 'SHIVAY', 'CHANTING']

In [103]: [x.upper() for x in seq4 if len(x) > 2]
Out[103]: ['NAMAH', 'SHIVAY', 'CHANTING']

In [104]: # Dict comprehensions
# basic form: {key-expr : value-expr for value in collection if condition}
{v : i for i, v in enumerate(seq4)}

Out[104]: {'om': 0, 'namah': 1, 'shivay': 2, 'chanting': 3}

In [8]: # Set comprehensions
{len(x) for x in seq4}

Out[8]: {2, 5, 6, 8}

In [15]: # Nested List comprehensions
# Order of the for expressions is same as in a nested for loop
```

And then transform those elements using the expression that we have. So, let us understand what we discussed with few examples. So, let us say we have the sequence 4 where we have the, this is a list of strings, and we have these 4 string elements, first one is om and then namah and shivay chanting. And so, you can see the next line is actually where we are using a list comprehension.

So again, this is a list comprehension. So we will start with you know square records and within a square brackets, the first thing is our exploration, where we are transforming, where we are

going to transform the filtered output. So, let us assume there is going to be some filtered output. So, what we are doing is we are taking that filtered output that is `x` and then applying this upper method. So, in a sense, we are transforming the filtered output.

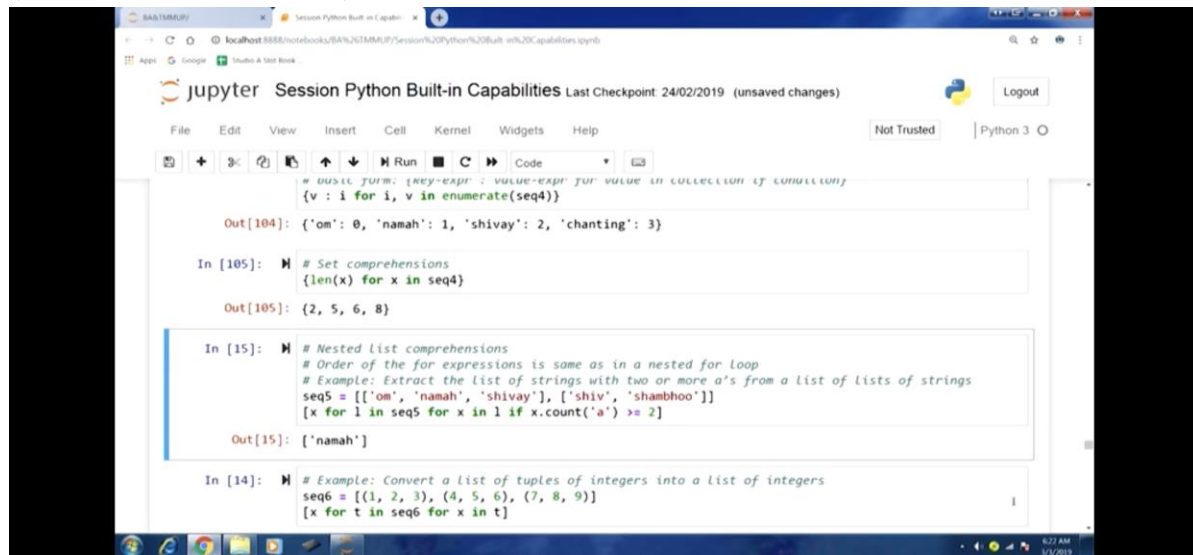
So, whatever filtered string elements that we get, we would like to you know, capitalize the new characters that are part of the string. So, this is the transformation. Now, we are running a loop for `x` in `sequence4`. So, for all the elements in this case, we have not included the conditional logic part that a part which is used to you know, filter the elements. So, that is actually optional. So, this is just to give you that indication.

So if I run this, sequence you can see and if I run this all the elements of the list that we had in `sequence4` we get a new list and a new list of strings and you know all the elements of string elements string values, they are now capitalized here. Now, similarly, in the next example, if you have a look here, expression is same, transforming expression is same `x.upper`, then we are running this loop for `x` in `sequence4`, so for, you know, each string element in the list.

And then if length of you know that the string element is greater than 2, so, all the values that we have, if their length is the number of characters in those strings are greater than 2, then you know, we would like to filter that out and then apply our transforming you know, this expression and that means, we would like to capitalize all those characters.

So, you can see out of the 4 string values that we have in the original strings `sequence4` `om` `namah shivay` `chanting`, `om` is the first string element that we have that is not satisfying the condition, so it will filter out. So the filter output will have `namah shivay` and `chanting`. Now all these 3 strings they will be capitalized, if I run this expression, this line of code, you can see in the output.

(Refer Slide Time: 26:06)

A screenshot of a Jupyter Notebook interface. The browser address bar shows a localhost URL. The notebook title is "Session Python Built-in Capabilities". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running, saving, and other actions. The code area contains several cells. The first cell shows a dict comprehension: `{v : i for i, v in enumerate(seq4)}` with output `{'om': 0, 'namah': 1, 'shivay': 2, 'chanting': 3}`. The second cell shows a list comprehension: `{len(x) for x in seq4}` with output `{2, 5, 6, 8}`. The third cell shows a nested list comprehension: `[x for l in seq5 for x in l if x.count('a') >= 2]` with output `['namah']`. The fourth cell shows a list comprehension: `[x for t in seq6 for x in t]`. The status bar at the bottom indicates "Python 3".

So from this you can really understand the kind of scenarios in data analytics context in the data mining and text planning context, where this could be really useful. Now, let us move on. Now we will talk about, you know, the next comprehension that is dict comprehension. So here, the basic form the way we are supposed to, you know, code, this is within curly braces, first, we indicate our key expression, and then colon and separate our value part.

And then we will have value expression, and then value expression, then we will run our loop for value in collection. So for each element, and then collection will run the loop and then the filtering part of it, if condition. So, in this fashion we can write. So, let us take an example now. So, for the sequence4 what we can do is we are using enumerate function here. So, in this example, again, we are not using the filtering part.

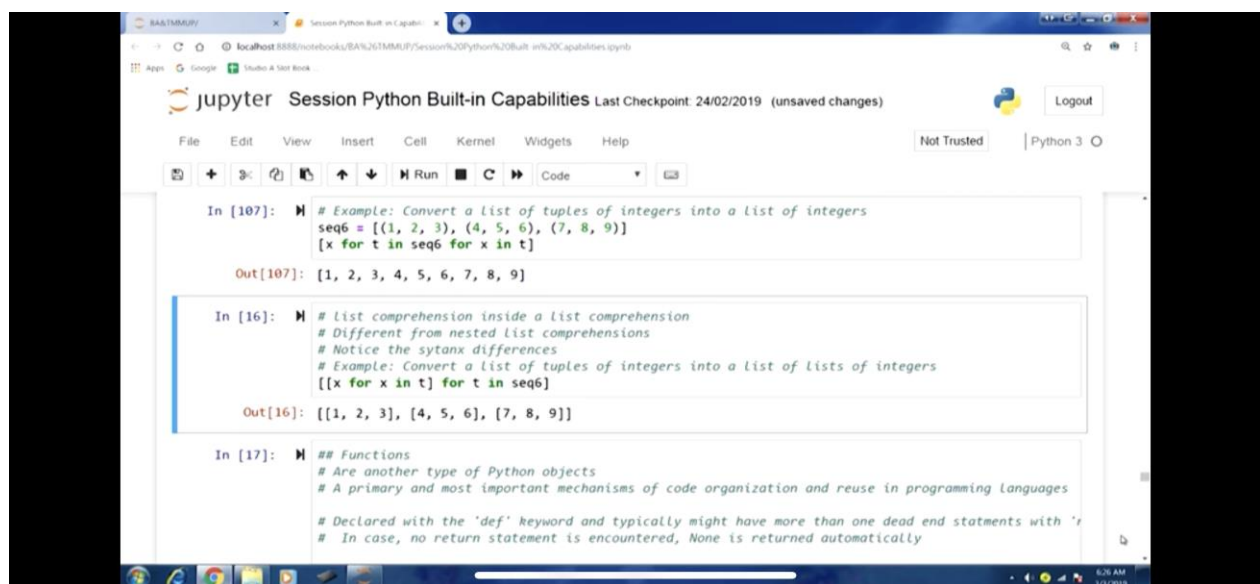
So we are running a loop and we are integrating the sequence. So what will happen for each of the you know, a string, that is part of the sequence for that, we are in essence, trying to create a mapping where we are trying to indicate the index for that particular, you know, string element. So you can see, indeed, expression that is to be used for the transformation, we have V:I though, V will store the string part, and then I will have the, you know, the index here.

So if I run this, I will get this dict object where each of the string, they have their indexes, they have their dices map to them in a dict format, key value pair format, so you can see in the

output. Now, next type of comparison is the set comprehension. So the form is, you know, quite similar to the other 2. So you can see again, you know, directly I am using example here. So, this is also within curly braces, and the transformative exhibition that we have is the length of x.

So whatever filtered output that we can you like to compute the length of, you know, that there was a string values, and we are running a loop here for action sequence for so for each of the string values that we have in this list of strings, we would like to compute the length, and we will get a set of those length values. So, if I run this, you can see 2 5 6 8 the length of those strings have been computed.

(Refer Slide Time: 28:49)



```
In [107]: # Example: Convert a list of tuples of integers into a list of integers
seq6 = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
[x for t in seq6 for x in t]

Out[107]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [16]: # List comprehension inside a list comprehension
# Different from nested list comprehensions
# Notice the syntax differences
# Example: Convert a list of tuples of integers into a list of lists of integers
[[x for x in t] for t in seq6]

Out[16]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In [17]: ## Functions
# Are another type of Python objects
# A primary and most important mechanisms of code organization and reuse in programming languages
# Declared with the 'def' keyword and typically might have more than one dead end statements with 'r'
# In case, no return statement is encountered, None is returned automatically
```

Now, let us talk about a few more aspects about list comprehension. So, the next one is a nested list comprehension. So, just like we can have, you know, nested loops. Similarly, because here also a loop is involved and that is the core component of the comprehensions. So here also we can have you know nested comprehension, sometimes we require this also. So, here in terms of order of the for expression, this is going to be very similar to what we have and we you know, write our nested loops.

So, let us take an example here. So, suppose, we want to extract the list of strings with 2 or more is that means a character a letter from a list of strings. So, here you can see the sequence 5

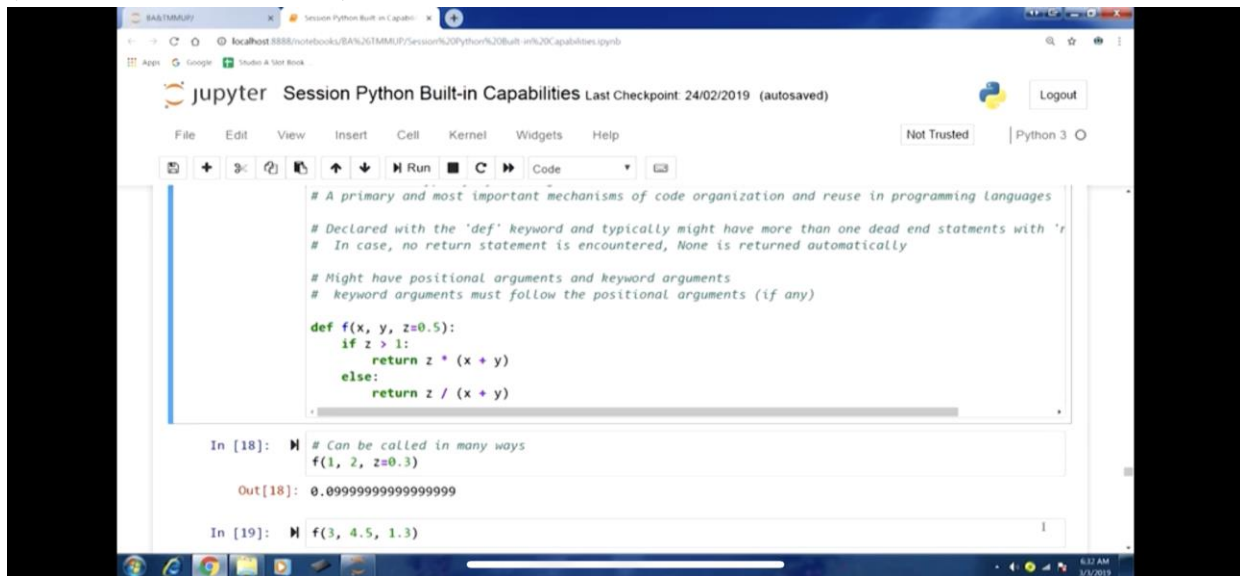
as an example, I have, you know, defined here, this is a list of list which are again list of strings. So, first list element is om namah shivay and these 3 element of string, the second list is shiv and Sambo.

So, these are 2 you know, strings and 2 elements of the second string, these 2 list are part of another list. So, let us look at the nested list comprehension that I have written. So, here you can see x for l in sequence5, so, in sequence 5, that is a list of list. So, that means each element is a list. So, for each of the list, we will use this transformative exploration that is nothing but that element itself x and then the second loop is for x in l.

So, within that list, you know, each of the you know, the string elements that we have, so, the inner loop would be about that, and then we have this filtering component also here. So, here each of the string that will have will count the number of characters a if it is greater than 2, then those strings are to be filtered out. And you know, they would be you know, the output will be presented in the list format, because you can see the outer because you can see the outer list comprehension.

And the transformative expression is that x is there is actually a list. So, if I run this, you can see I get namah because out of you know, this list of lists of strings, you know, only the namah string is there were 2 characters are there. So, that is our output to understand this, let us take another example. So, in this example, what we are trying to achieve is to convert a list of tuples of integers into a list of integers, so, how that can be performed.

(Refer Slide Time: 29:00)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". The notebook contains a Python function definition and its execution. The function is defined as follows:

```
# A primary and most important mechanisms of code organization and reuse in programming languages
# Declared with the 'def' keyword and typically might have more than one dead end statments with 'r'
# In case, no return statement is encountered, None is returned automatically
# Might have positional arguments and keyword arguments
# keyword arguments must follow the positional arguments (if any)

def f(x, y, z=0.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

The function is then called in two ways:

```
In [18]: # Can be called in many ways
f(1, 2, z=0.3)

Out[18]: 0.09999999999999999
```

The function is also called with positional arguments:

```
In [19]: f(3, 4.5, 1.3)
```

So, let us look at the sequence, this sequence is you know, a list of tuples. So, this list has 3 tuples 1 2 3 first tuple 4 5 6, second tuple and then 7 8 9 third tuple. So, what we are trying to achieve is we are trying to achieve a list which is just comprising of the scalar types. So, we would like to get rid of the tuple structure that is there. So, again, you know, we can achieve this easily with the nested list comprehension.

So, you can see first you know the outer comprehension x for t in sequence, so sequence 6 is what list of tuples. So, t means tuples, so for each tuple in that sequence is the transformative expression is x in nested comprehension that we had is for x in t . So in you know in the tuple tuple is made of you know, 3 scalar types here. So, for each all of them, we would like to you know, present in them in the list format.

So, if I run this you can see in the output, we have been able to in a sense transform the list of tuples of integer into just a list of integers. So, just using you know, these nested list comprehension and just like nested for loops, we could have used otherwise, this can be achieved using list comprehension 1 single sentence you know, concise way we can achieve this. Now, let us talk about another aspect that is related to this comprehension itself.

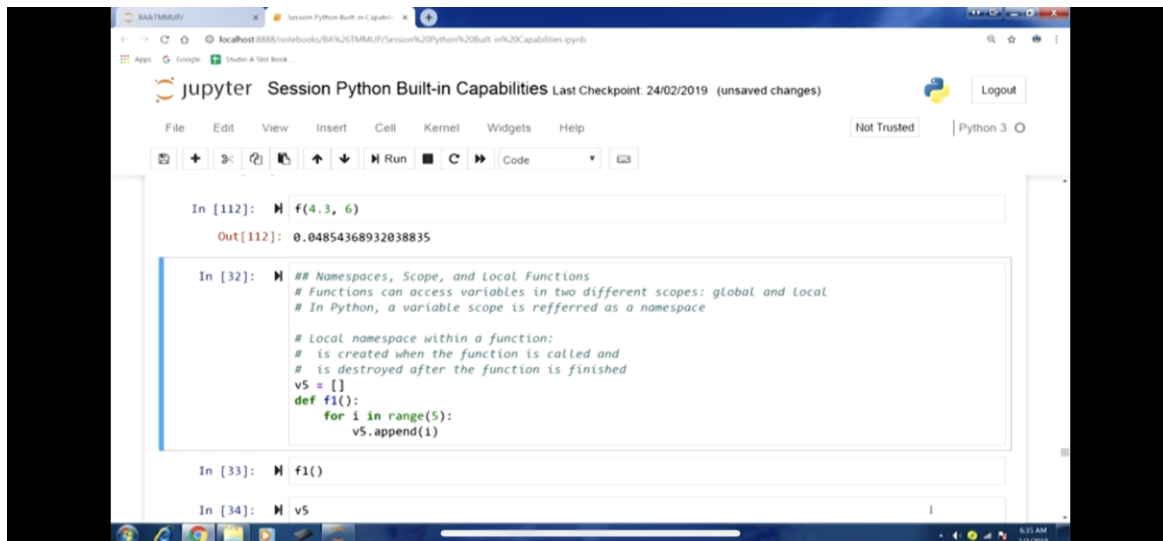
So, sometimes we might have list comprehension inside a list comprehension. So, this is slightly different from when we talk about nested list comprehension. So, in the sense

transformative expression that we might have that itself could be a you know comprehension. So, we need to focus on the syntax differences here. So for example, if we want to convert a list of tuples of integer to a list of lists of integers.

So, you can see here in the first we are running this you know, comprehension for t in sequence5, sequence6, so sequence6 is you know, list of tuples. So, for each tuple will run this loop, and, you know, for each tuple when we run this loop, and you can see the what is the transformative expression that is also a list comprehension. So, for each element in the tuple we are just you know presenting that you know, in the form of you know, list itself.

So, if I run this you can see in the output is so from list of, you know, tuples of integers, we have been able you know transform it into a list of lists of integers. So, this was about list comprehension. So, we have covered most of the data structure aspect, you know, tuples list, dicts and sets are now the comprehensions as well. Now, we will move on to the next important aspect in python language that is functions.

(Refer Slide Time: 28:06)

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/BAN%20TMM%20Python%20Built-in%20Capabilities.ipynb'. The Jupyter header includes 'Session Python Built-in Capabilities' and a 'Last Checkpoint: 24/02/2019 (unsaved changes)' message. The interface has tabs for File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the tabs are icons for adding new cells, undo, redo, and running the code. The code area shows three input cells. The first cell, 'In [112]:', contains 'f(4.3, 6)' and its output 'Out[112]: 0.04854368932038835'. The second cell, 'In [32]:', contains a multi-line comment about namespaces and a function definition:

```
## Namespaces, Scope, and Local Functions
## Functions can access variables in two different scopes: global and local
## In Python, a variable scope is referred as a namespace

# Local namespace within a function:
# is created when the function is called and
# is destroyed after the function is finished
v5 = []
def f1():
    for i in range(5):
        v5.append(i)
```

. The third cell, 'In [33]:', contains 'f1()' and the fourth cell, 'In [34]:', contains 'v5'. The bottom status bar shows the time as 8:31 AM on 1/2/2019.

So, we have been using in our previous lectures, we have been using our functions a lot you know, so, now, this time, we are slightly more detail will cover with some of the aspects related to function, you know, that has already been discussed. Now, functions are another type of

python object. So, in this programming language in python programming language functions are also defined as just another type of you know python objects.

So, because of this kind of you know, definition functions, you know, it is easier to code certain situation which involved functions. So, this is the primary and most important mechanisms of code organization and the using programming language. So, not just python programming language and almost all the programming language function play an important role in terms of organizing the code and to be able to reuse the code you know, whenever we expect that certain part of code you know will be used to in a repeated sense.

So, that is when we realize it is better to code that as a function rather than having the standalone code block. So, that will help us in a sense in code organization can be use you know, the use of that, you know, particular code block further. Now, how the functions that, you know, created. So, this we have already understood, so, we use the def keyword. And typically, we have more than one, you know, there in the statements with the return key words.

So, typically, it is like just now, you can consider a function or for that matter, any, you know, code that can be many pathways, so, the execution flow, if we talk about execution flow, it can go along, you know, if we want to reach from point A to point B, and, you know, rather point A, let us say it is there are various pathways, so, the flow of execution can take any of those pathways.

And so therefore, you know, there could be requirement of ending that product code blog in different ways. So, we need to use different you know, written keywords return keyword is something where we are indicating that the, you know, the code law that you have in the function that has finished and we are supposed to return to the parent block, the scope is going to change there.

So, multiple written statements might be there. And in case there is no return a statement, then by default, none is written automatically. So, multiple code of you know, flow of execution can go in along, you know, multiple pathways. So, for each one, we might have return, you know,

statements to end it, and then if not there, then none would be returned. Now, function as we have discussed before, also they can have positional arguments as well as keyword arguments or named arguments.

So, you know, both these kind of, you know, arguments could be there. And we talked about that this aspect before as well, that keyword argument must follow, if I must follow the positional arguments, if any, so, if there are any positional argument that are part of a function, they will, come first because you know, they have to they are defined by the position, and so, therefore did not bottom that they come first, and then the keyword arguments because they can also be identified with the keyword.

So, here, you can see, they have given one example. So, because of the, you know, these types of arguments that we might have a function can we call in many ways, so, we will see that aspect as well. So let us look at this function definition. So we have, using keyword def, and then the function in which is f. And in the parentheses, we define our, our arguments here. So x, y.

So these are 2 positional argument then third argument is the keyword argument that is z. And we are giving a default value to this argument as well. So another aspect related to keyword argument is that typically, when we want to give you know, default value to a particular argument, then keyword arguments could be really useful, then we have our code block, within the functional scope.

So if z greater than 1 and return this, so this is one possible flow of execution. So if the value of z happens to be greater than 1, then we will you know, execute this statement return z in multiplied x+y in the parenthesis. So, this is one possibility that we have, and once we reach the return statement, the will come out of the functional scope, there is another possibility if z is you know, not greater than 1 then will lead to the else block.

And then again, we have another returning statement there. So as I said, just in the overall code, you know, that we that we right or in the function, we might have multiple, you know,

possibilities. So therefore, especially in the functional, we have to, you know, explicitly indicate that using the return keywords, or, by default, none would be return. So, let us run this, and can we call in many ways, as we discussed because of the positional arguments and the keyword arguments.

So you can see, first example, that we are taking function f, and in the parentheses, first argument, value we are passing 1 then 2, and then you can see we are using the named argument z is equal to 0.3, the default values 0.3 0.5. So now this value that we are passing this is to be used, if I run this will get the output. Now another way of calling you can see here, first argument is t then second argument 4.5.

Now, third argument is 1.3 is I am not using the third argument, which is actually as a keyword argument. Rather, here I am using in a positional sense. So I am passing on the value here. So this can also be executed. This is another way of calling this function. Then, the third one is that we are I am just passing 2 arguments 4.3 and 6. So you can see whenever we have the key arguments in a sense that argument can also be an optional arguments because we have given default value.

So in that sense, you know, we do not need to pass that argument all the time. So the mandatory arguments will become the positional arguments, and the key word argument, you know, in a sense optional arguments because they will have default value anyways. So, we can call in this fashion also, f in the parenthesis 4.3 and 6 just 2 arguments, however remember the function takes 3 arguments. So, if I run this will also run fine. Now, the next aspect about the function is the name spaces, the scope and local function.

(Video Ends: 28:51)

So, some of these aspects are also related to how we know deal with variables that are that are to be defined within the functional scope and how we deal with the variables that we might be accessing the functional scope but define outside of it. So some of these aspects will discuss. So at this point would like to stop here and we will take it up in the next lecture, thank you.

Keywords: scope and local function, Float, Dict, Data structure.