Business Analytics And Text Mining Modeling Using Python

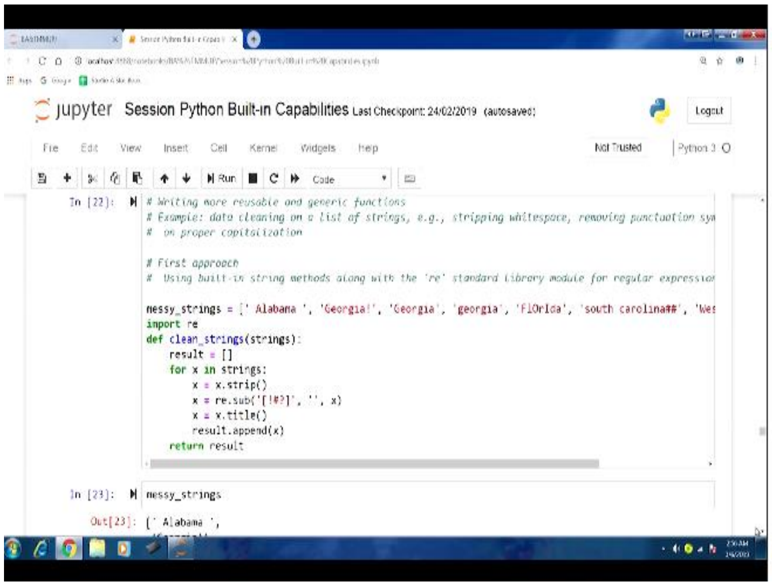
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology-Roorkee

Lecture-16
Built-in Capabilities of Python-VIII

Welcome to the course business analytics and text mining modeling using python. So in previous lecture we were talking about functions and we talked about various aspects and in the last part of the previous lecture we were specifically focusing on how to write more reusable and generic function. So we talked about you know way we took this data clean example where we have list of strings.

And how you know we can get it obviously you know wide spaces any punctuation symbols special characters or perform proper capitalization and all those things. So the main focus was you know the same lines of code same code block you know whether it can be written in a more reusable and more generic form. So that aspect we touched upon.

(Refer Slide Time: 01:21)



```
In [22]: # Writing more reusable and generic functions
         # Example: data cleaning on a list of strings, e.g., stripping whitespace, removing punctuation sym
         #  on proper capitalization

         # First approach
         #  Using built-in string methods along with the 're' standard library module for regular expressio

         messy_strings = [' Alabama ', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda', 'south carolina##', 'Wes
         import re
         def clean_strings(strings):
             result = []
             for x in strings:
                 x = x.strip()
                 x = re.sub('[!#?]', '', x)
                 x = x.title()
                 result.append(x)
             return result

In [23]: messy_strings

Out[23]: [' Alabama ',
```

So you can see the first approach as we discussed in the previous lecture we used fiddling string methods and each standard library. And in one function you know in a one loop we just went through all these string elements of this list string and just perform our operation in a sequential fashion and we had this one function.

(Video Starts: 01:39)

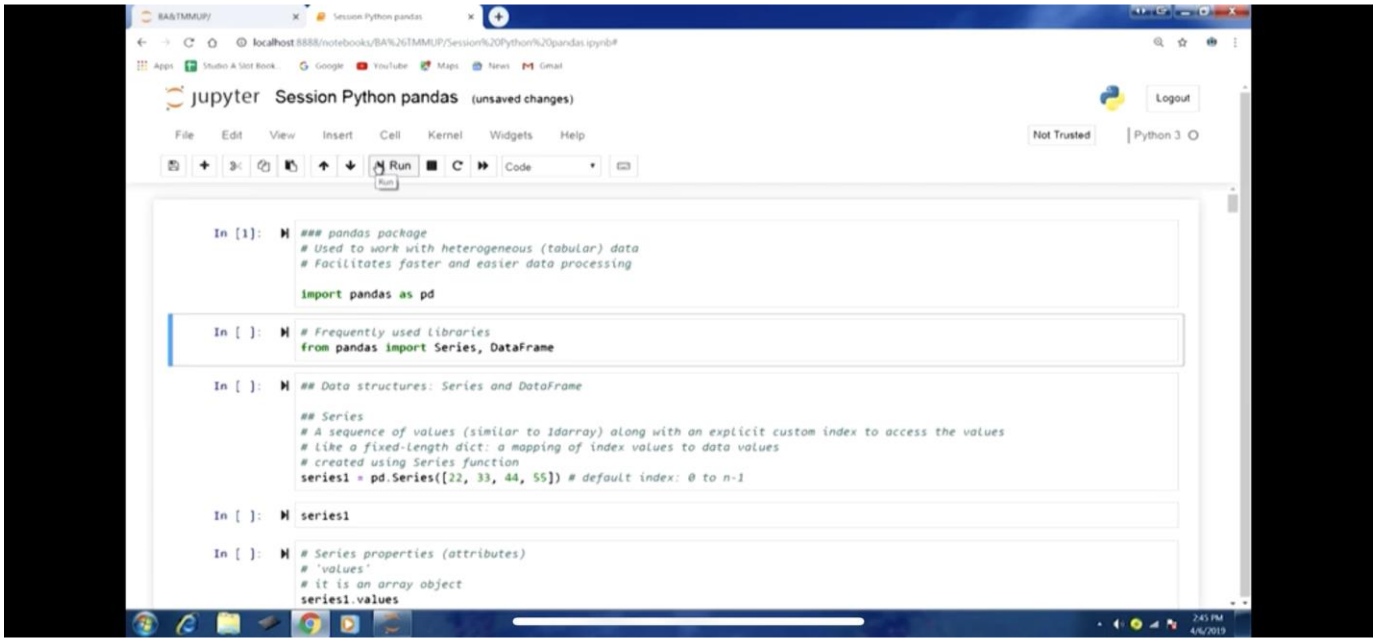
Now in the second approach we just changed the code organization a bit where we created a where we define another function for the V.sub function that we are using, so separate function for that and we created a list of you know operation the list of function that we want to apply on a list of a string. So in a sense we created a structure where we have a list of operations that we would like to apply and we have list of strings.

So we in a sense improved our you know clean strings function and we are running a loop for the string and then we are running a you know inner loop for different operations that we would like to perform. So this way of writing the same code you know is makes it more usable. Because in future whenever we require to add one more operation it would be easier to do the same in this one, because we will just have to add that you know operation in the clean of variables.

However, in case of first approach we will have to change the code, we will have to you know work that out and it change the function itself, here we do not have to change the function itself, we will just have to modify the list, in the first approach we will have to change the function. So in this lecture now we will focus few more aspects about functions as well. So functions sometimes they can be passed on as arguments to other functions. So to demonstrate that we will take this example of map function.

What it does it applies a function to a sequence so we might have a you know a list of elements and we might have a sequence and this you know this map function what it will do it will take any function that we pass as a you know first you know argument. And then it will apply it to the you know sequence that we might have. So here what we are doing is we are running a loop for X in map, so a map function it has 2 arguments in this case remove punctuation.

(Refer Slide Time: 03:32)


jupyter Session Python pandas (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

```
In [1]:  ### pandas package
         # Used to work with heterogeneous (tabular) data
         # Facilitates faster and easier data processing

         import pandas as pd

In [ ]:  # Frequently used libraries
         from pandas import Series, DataFrame

In [ ]:  ## Data structures: Series and DataFrame

         ## Series
         # A sequence of values (similar to 1darray) along with an explicit custom index to access the values
         # like a fixed-length dict: a mapping of index values to data values
         # created using Series function
         series1 = pd.Series([22, 33, 44, 55]) # default index: 0 to n-1

In [ ]:  series1

In [ ]:  # Series properties (attributes)
         # 'values'
         # it is an array object
         series1.values
```

So we would like to remove punctuation in the second element that is a list of strings. So here you can see the move punctuation this is a function this is being passed as an argument to another function which is math and in the one line that we have in this for loop we would just be printing out the elements of this particular sequence that we will have. So we will run this and you can see the messy strings the element that we have, remove function has been applied here using the map for remove punctuation function have been applied using the map function.

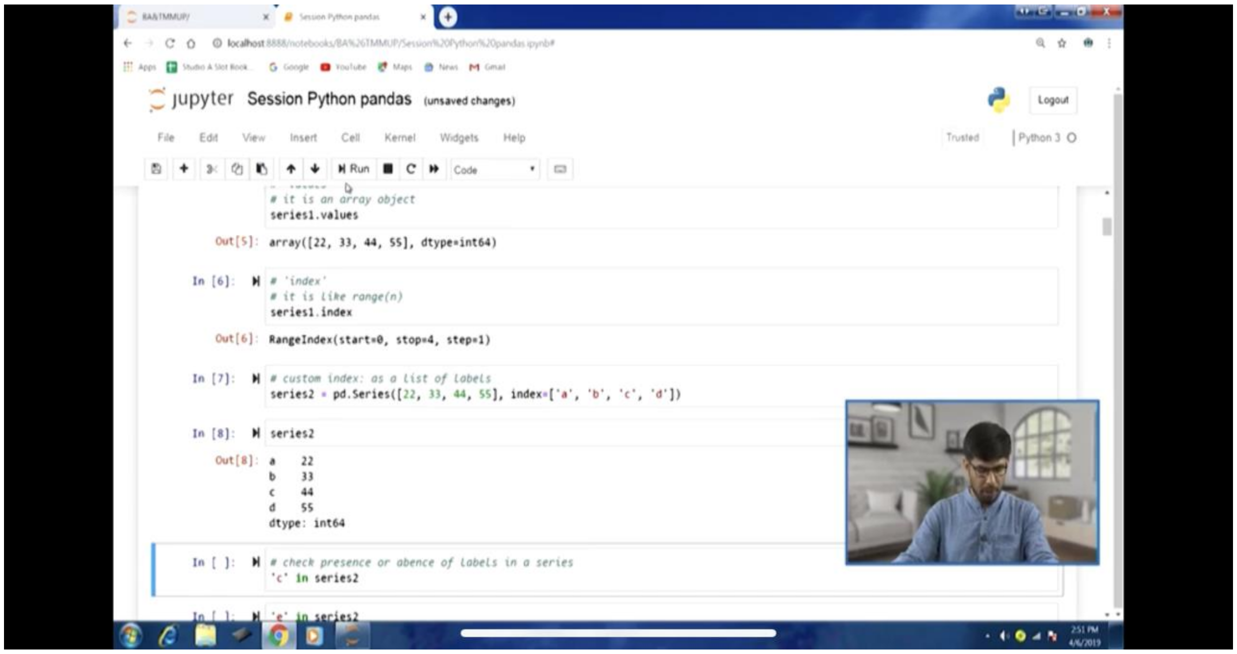
So if you **you** know compare this with the second approach that we had gone through it is quite similar to that we are applying in this fashion, you can see the messy string all the punctuation marks are gone. Let us move on. Now we will talk about another aspects related to function, so sometimes you know we might be required to write anonymous or lambda functions. So these anonymous on lambda function could be really useful when we are required to pass functions as arguments.

and those functions are very short function have just 1 or 2 lines of code. So for passing a function as an argument and which has just 1 or 2 lines of code you know we you know we would not prefer to have the full body of the function def keyword in the way we typically define it to create the ease of passing on the function as arguments we have these anonymous or lambda functions.

So this is lambda functions anonymous function there they are concise way of writing functions consisting of a single statement, so for this we use lambda keyword so a function object itself is never given a an explicit name attribute. So we would not be calling you know this lambda function by name, so that is why they are also called anonymous because they would not have any name.

So typically they are used when because they are supposed to have single statement, so they are typically used as function arguments, so function object, so convenient event functions as arguments are used. So in data processing context this is quite common where we might have a column of you know data column of observations and we would like to harm certain operation on that column.

(Refer Slide Time: 09:27)


jupyter Session Python pandas (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
         # it is an array object
         series1.values

Out[5]:  array([22, 33, 44, 55], dtype=int64)

In [6]:  # 'index'
         # it is like range(n)
         series1.index

Out[6]:  RangeIndex(start=0, stop=4, step=1)

In [7]:  # custom index: as a list of labels
         series2 = pd.Series([22, 33, 44, 55], index=['a', 'b', 'c', 'd'])

In [8]:  series2

Out[8]:  a    22
         b    33
         c    44
         d    55
         dtype: int64

In [ ]:  # check presence or abence of labels in a series
         'c' in series2

In [ ]:  'e' in series2
```

So just one line of code would suffice to perform that operation, so for that instead of writing the full definition the traditional the typical definition the way we do in python we can use lambda function. So **so** what happens this becomes feasible to write and apply a custom operator also we will see that, so you can see we are defining a short function here it has just one statement and that is also in the form of a return statement.

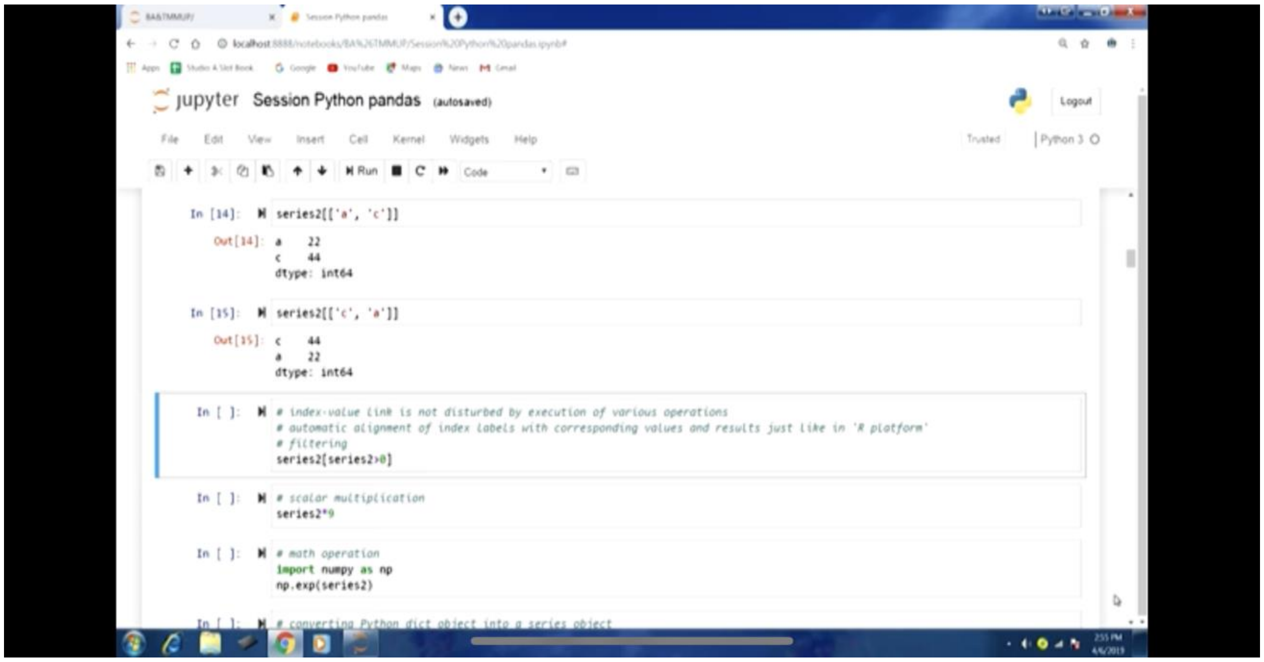
So what we are doing is we are taking x as an argument and just multiplying it with the 10 and returning the same. So this short function has just 1 statement that is the return statement. The same thing can be written as a lambda function where we can use the lamp key word here, you can see that 2 is the return value and the right hand side we had K we have created a lambda function here. So you can see lambda and the argument X:.

And then the single line of you know code that you would like to execute X *10. So this is you know this is the way of defining the you know lambda function. Now let us run this, so this is defined, now we will take example here you can see we are again defining a function apply short F. So it has 2 arguments a list and a and a short function. So the idea is so we would like to apply this short function on this list.

So what it will return, this short you know function this expression which is actually the you know return expression from Y applying the short function on this in this object X and this object X is you know a list element. So a loop we are running for X in a list. So let us define this as well. Now if we call this function a apply short f you can see you are passing a list point 1, point 2 to 2 elements in this list and we instead of you know short F we can use here a lambda function you can see lambda X and the X*10.

So this is a you know convenient way of writing single statement function, so we can use lambda keyword in this fashion if I run this you can see we got the output of 1 and 2 a list with the elements 1 and 2. Similarly let us take another example for to demonstrate this. So let us say we have the sequence 7 this variable and we have these 4 string values here shivay, chanting, om and namah and what we want to perform is we want to do a shorting which is based on the length of this string.

(Refer Slide Time: 13:22)



jupyter Session Python pandas (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [14]:  series2[['a', 'c']]

Out[14]:  a    22
          c    44
          dtype: int64

In [15]:  series2[['c', 'a']]

Out[15]:  c    44
          a    22
          dtype: int64

In [ ]:   # index-value link is not disturbed by execution of various operations
          # automatic alignment of index labels with corresponding values and results just like in 'R platform'
          # filtering
          series2[series2>0]

In [ ]:   # scalar multiplication
          series2*9

In [ ]:   # math operation
          import numpy as np
          np.exp(series2)

In [ ]:   # converting Python dict object into a series object
```

So you can see this short function that we have used before as well it requires a key which essentially is a function. So here we are using lambda function and that we are just taking the length of the you know string. So based on the length of this thing that would be treated as a key and that key would then be used for the shorting. So if I run this you can see in one go we are able to generate the key using lambda function length of the string and we are able to short this list of a strings here.

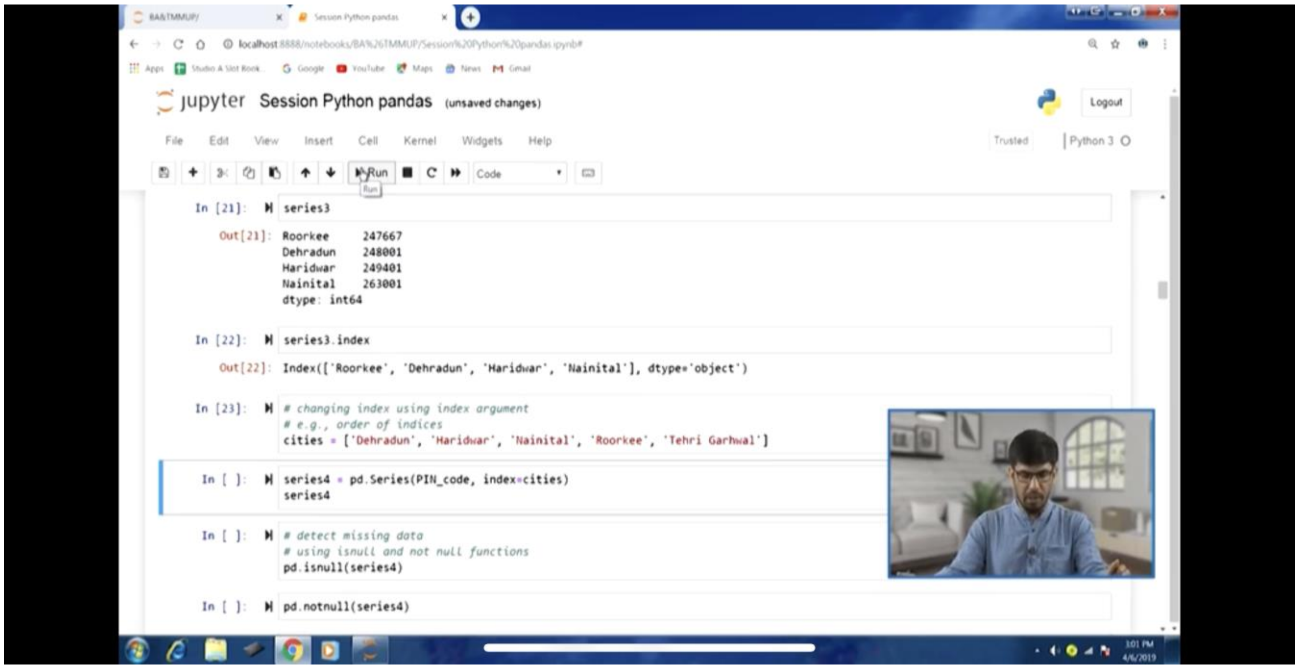
So you can see om comes first because we are having just 2 characters then namah, then shivay, then chanting. So in this fashion you can see a writing lambda function or anonymous function could be really useful in the context as you can see when we are passing function as arguments or in some situation like in the web use short method here. Now certain other situations also. Now there is another aspect related to function that we would like to discuss now is it is called currying.

So this is about defining a new function that calls an existing function with fewer arguments. So we might have a function with 2, 3 or more arguments now sometimes we might require to have a function which you know 1 or 2 arguments have you know fixed kind of value and only the variable other arguments are going to change so sometimes you would like to define a function for that because it might you know it might be used repeatedly in the future lines of code. So therefore we call this currying.

So let us say we have this add numbers function, so defined as def add numbers 2 arguments X, Y and we are just returning X+Y. So what we can do is we can use lambda or keyword here and define a new function that that will have fewer arguments. So you can see add_5, this is the new function and you are using lambda keyword access is the argument add numbers we are using the previously defined function add numbers which are 2 argument 5,X.

Essentially we are adding 5 to the argument single argument that will be passed in the add_5 function. So from a 2 argument function add numbers now we have created or defined a new function with just 1 argument and the second argument is a fixed number here 5. We can achieve the same thing using partial function from the building func tools module. So this func tools modules has several useful functions.

(Refer Slide time:19:11)



jupyter Session Python pandas (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [21]:  series3

Out[21]:  Roorkee     247667
          Dehradun    248001
          Haridwar    249401
          Nainital    263001
          dtype: int64

In [22]:  series3.index

Out[22]:  Index(['Roorkee', 'Dehradun', 'Haridwar', 'Nainital'], dtype='object')

In [23]:  # changing index using index argument
          # e.g., order of indices
          cities = ['Dehradun', 'Haridwar', 'Nainital', 'Roorkee', 'Tehri Garhwal']

In [ ]:   series4 = pd.Series(PIN_code, index=cities)
          series4

In [ ]:   # detect missing data
          # using isnull and not null functions
          pd.isnull(series4)

In [ ]:   pd.notnull(series4)
```

So partial function can actually be used to achieve the same thing, so first thing we will do is to import that module from func tools import partial this function and here also you can see we can achieve the same thing new function new_add in the score 5. So this is the new function and we are using partial function here and you can see the original function add number add_numbers is being passed as the you know first argument and comma 5.

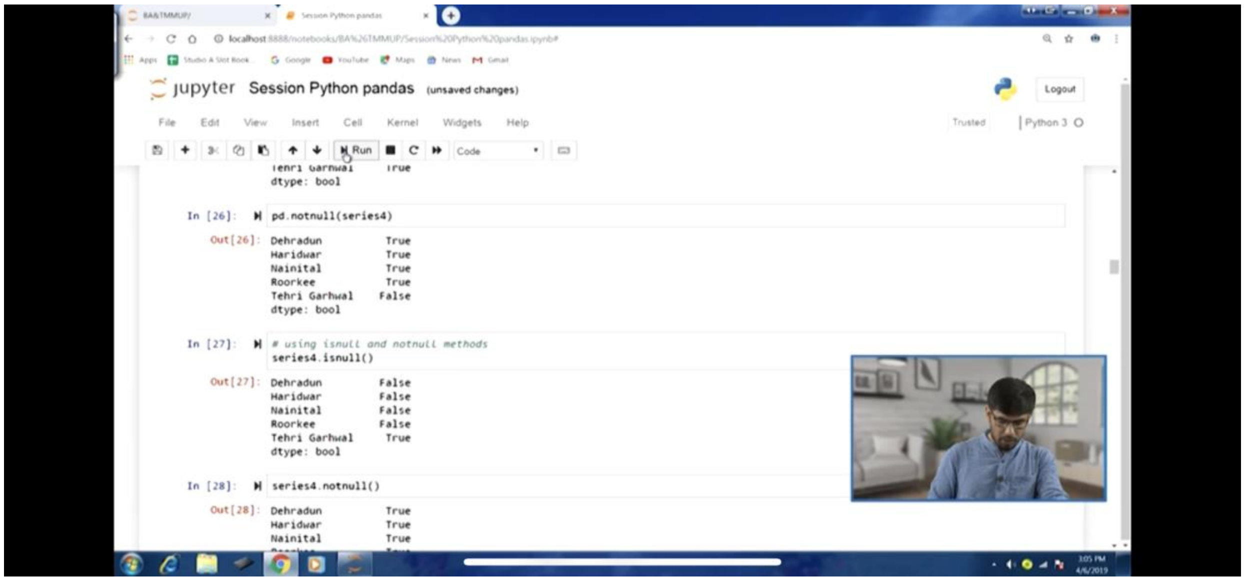
So 1 argument is 5 and we will just have to you know pass one more argument, so in this fashion also you would be able to you know define this new function with fewer arguments. So this is something that is called currying. So let us run this so now let us call add_numbers the full function that we had defined, we are passing 2 arguments here 2,4. If I run this we will get the answer 6.

Now the new function that we had defined using the lambda keyword add_5. So you can see I am just passing 1 argument to and the second one is actually 5. So we will get the output as 7. Similar **we also** we had also defined a new_add_5 function using the you know partial function. And here again we need to just pass on just one argument and you can see the output 8. So in these 2 arguments that we have in these 2 functions that we have just created.

They are actually based on the original 2 argument function and now they have become just one function. Now let us move forward. Now we will talk about another aspect related to functions this is referred as generators. So generators are also functions but typically they are used to construct iterable objects or iterators. As we have discussed before in the for-loop context typically we require an iterator.

So that we can whatever code block that we want to execute again and again in a repeated fashion. So through in an iterator it might also be required in the code block itself. So these are generators or functions to construct iterable objects. Now when we say an iterator, so iterator is again an object that will yield objects to the python interpreter when used in a context like a for loop. So whenever we are using a you know for loop kind of context this iterator will yield objects in a fashion.

(Refer Slide Time: 22:42)


jupyter Session Python pandas (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Trusted
Python 3
Logout

```
        Tehri Garhwal     True
        dtype: bool

In [26]:  pd.notnull(series4)

Out[26]: Dehradun          True
         Haridwar          True
         Nainital          True
         Roorkee           True
         Tehri Garhwal    False
         dtype: bool

In [27]:  # using isnull and notnull methods
          series4.isnull()

Out[27]: Dehradun         False
         Haridwar         False
         Nainital         False
         Roorkee          False
         Tehri Garhwal     True
         dtype: bool

In [28]:  series4.notnull()

Out[28]: Dehradun          True
         Haridwar          True
         Nainital          True
```

So that the iterations can be performed. So whenever we encounter a statement like you know the for loop statement like for I in some sequence the python interpreter typically it first attempts to create an iterator out of it you know iteratable object out of some sequence. So whatever sequence for I in some sequence, so from the some sequence the python interpreter will try to create an iterator.

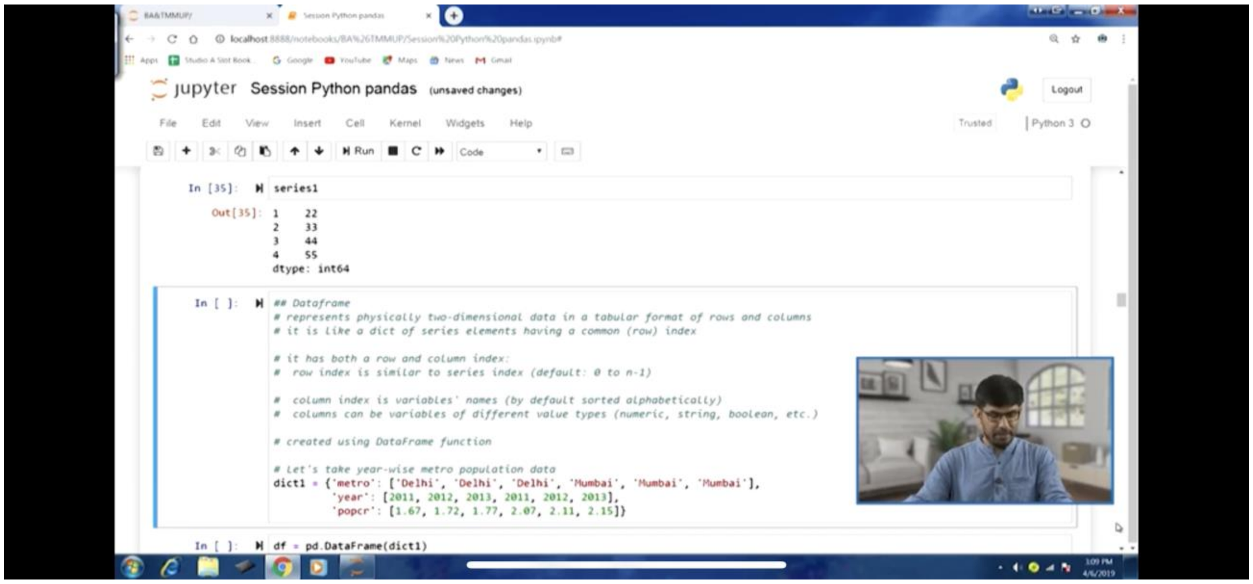
So the generators these functions can also help in constructing those iterators. So let us take an example and followed with few more examples to explain the whole idea here. So here you can see we have created this dict variable here dict object here d4 d key value you know pairs a1 b2 c3. And we are running this loop. So if I run this you can see a b c these have been printed, here if we look at this sequence that we have you know defined d4 if we apply ITER function to find out the iterator part of it.

This is something when we said python interpreter attends to you know create an iterator. So we will apply this ITR function on this sequence and see what we get. So if I run this and I run again so you can see IT this variable that we have it says dict_key iterator. So the type key iterator is actually there in the python platform and when we apply for loop it actually tries to perform that. Now if we use list function to understand the element off to see the elements of this key iterator you can see a b c right.

So the key part of the depth object that we have they are the element of this iterator so the Python interpreter it will actually create this kind of you know relatable object it will apply the type function to confirm the type here as you can see type it and we get dict_key iterator. So this is a you know iterator you know object that python interpreter you know tries to create the same thing we are trying to convey here.

Now another way to you know construct these iterators is using generators functions. So now we will talk about the same. So now what generators typically do they return a sequence of multiple elements in an orderly fashion as and when requested. Because in a for loop context you know once a loop runs through then we require the next element you know to iterate over. So these generator funds will actually do the same thing.

(Refer Slide Time: 26:49)


jupyter Session Python pandas (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [35]:  series1
Out[35]: 1    22
         2    33
         3    44
         4    55
         dtype: int64

In [ ]:   ## Dataframe
          # represents physically two-dimensional data in a tabular format of rows and columns
          # it is like a dict of series elements having a common (row) index

          # it has both a row and column index:
          #  row index is similar to series index (default: 0 to n-1)

          #  column index is variables' names (by default sorted alphabetically)
          #  columns can be variables of different value types (numeric, string, boolean, etc.)

          # created using DataFrame function

          # let's take year-wise metro population data
          dict1 = {'metro': ['Delhi', 'Delhi', 'Delhi', 'Mumbai', 'Mumbai', 'Mumbai'],
                   'year': [2011, 2012, 2013, 2011, 2012, 2013],
                   'popcr': [1.67, 1.72, 1.77, 2.07, 2.11, 2.15]}

In [ ]:   df = pd.DataFrame(dict1)
```

So typically a function will return once and you know that call would be over, however a generator as an when requested it will keep on returning multiple elements one by one as and when request the same thing we have mentioned here. So for this generators they typically use the yield keyword instead of the return keyword. So let us take an example, so here we are defining a squares function which has 1 argument and default value 10.

And we are printing this statement generating squares from 1 to you know whatever the limit is there and we are formatting is and to the n square and for i in range 1,n+1 you can see will be we are using written yield statement instead of return statement. So yield i square i to the power 2. So that would be written. So this is squares generator would actually be yielding these you know a square numbers right starting from 1.

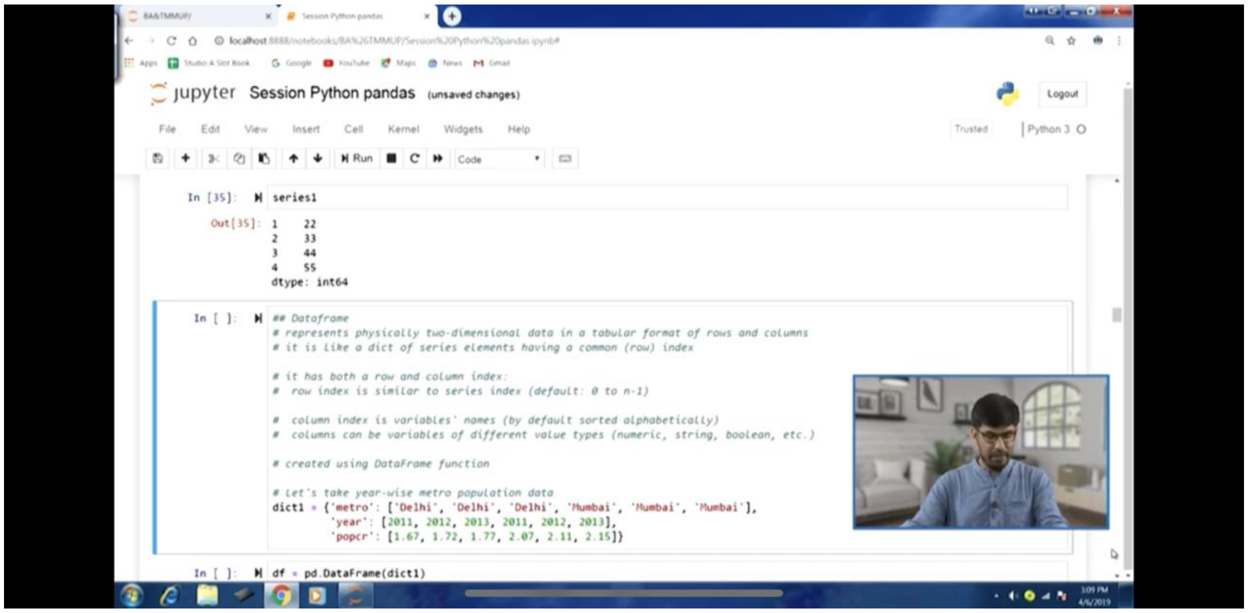
So let us call this a generator function. So we are using g to store this the return you know elements. So that is done, now let us run g you can see this is a generated object if we run the type function on this G object you can see generator. So this type of this is a generator type of function object, now if I run list function on this g you can see generating is squares for 1 to 100. So these are the elements 1 4 9 up 200 because default value was 10.

So starting from 1 to you know 10 it was run and we have got 1 4 9 and up to 100. So this is the iteratable object that has been iterator that has been created using the this generator function squares, we can have another example for x in g and we can print this, so if I run this and you can see here and then once we have gone through this we can again you can see for x in g we did not get any output.

So because all the elements that were part of g they have been yielded. So therefore in this run we did not get it, so we will have to call it again. So you can see g1 we are calling it again g1 squares and we are running a loop, so the same code you can see the same code is there just to convey this to that you know generators are also function. So in a normal function in a regular function you know get a return value.

And then the call is over and the in the generator function you once you get all the elements happy yielded then that call is also over, that is why in the line number 47 then the from that group that we have we did not get any output. However once we call it again using the g1 you know variable here and if I run this you can see again because this has been called again. So we get the full iterator information starting from 1 4 9 16 up to 100.

(Refer Slide Time: 30:17)



```
In [35]:  series1

Out[35]:  1    22
          2    33
          3    44
          4    55
          dtype: int64

In [ ]:   ## Dataframe
          # represents physically two-dimensional data in a tabular format of rows and columns
          # it is like a dict of series elements having a common (row) index

          # it has both a row and column index:
          #  row index is similar to series index (default: 0 to n-1)

          #  column index is variables' names (by default sorted alphabetically)
          #  columns can be variables of different value types (numeric, string, boolean, etc.)

          # created using DataFrame function

          # Let's take year-wise metro population data
          dict1 = {'metro': ['Delhi', 'Delhi', 'Delhi', 'Mumbai', 'Mumbai', 'Mumbai'],
                   'year': [2011, 2012, 2013, 2011, 2012, 2013],
                   'popcr': [1.67, 1.72, 1.77, 2.07, 2.11, 2.15]}

In [ ]:   df = pd.DataFrame(dict1)
```

So if I list to this g1 again you can see this is an empty list because all the elements have already been yielded. However in the you know previously as I had shown that output 46 as you can see list of g there you know all the elements that were part of the traitor they are displayed. So it is very much like a function but the only difference is all the elements that are there are to be yielded once that is done then you would not you know see any output.

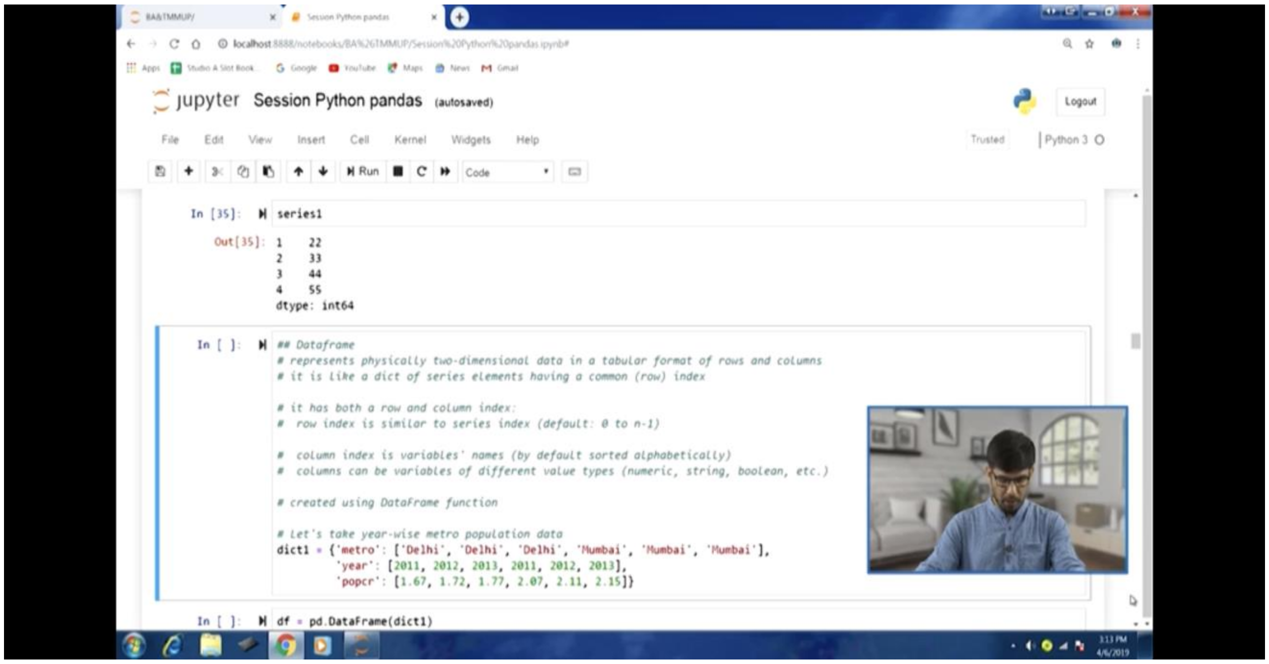
So you will have to call it again, so you can see how this particular generators they actually work. Now there is something called generator expressions, so they are similar to list comprehension that we have discussed in previous lectures with parentheses instead of brackets. So list comprehension they use you know in list formulation we use brackets because they are list and here in generator expression their way semantically they are very similar to list comprehension but the only thing is we use parentheses.

So let us look at one example so here you can see g2 and this is our generated expression we have the this transformative expression that we have X to the power 2 and we are running a low for X in range 10. So let us run this and we get g2 this is generated object as you can see, so from the generator expression we get generated object if we look at the list of g2 you can see starting from 0 because for z in range 10 was there, so it is started from 0.

So the elements you can see g start it is not starting from 1 it is starting from 0 1 4 because it is you know 0 indexed. So this is the output so generator expressions can also be really useful to perform the same thing you know to get the iterator. Now they can also be you know this instead of list comprehension we can also use the generator expression as functional arguments. So this is one example that we are taking we are using some function.

And you can see here we are passing on generator expression as function arguments here you can see x to the power 2 and then for x in the in this range. So if I run this you can see the output, so instead of list comprehension we can pass them generator expression as well. Let us take another example here you can see I am using dict function and inside the dict function you know again we have generated expression here.

(Refer Slide Time: 32:45)


jupyter Session Python pandas (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [35]:  series1

Out[35]:  1    22
          2    33
          3    44
          4    55
          dtype: int64

In [ ]:   ## Dataframe
          # represents physically two-dimensional data in a tabular format of rows and columns
          # it is like a dict of series elements having a common (row) index

          # it has both a row and column index:
          #  row index is similar to series index (default: 0 to n-1)

          #  column index is variables' names (by default sorted alphabetically)
          #  columns can be variables of different value types (numeric, string, boolean, etc.)

          # created using DataFrame function

          # Let's take year-wise metro population data
          dict1 = {'metro': ['Delhi', 'Delhi', 'Delhi', 'Mumbai', 'Mumbai', 'Mumbai'],
                   'year': [2011, 2012, 2013, 2011, 2012, 2013],
                   'popcr': [1.67, 1.72, 1.77, 2.07, 2.11, 2.15]}

In [ ]:   df = pd.DataFrame(dict1)
```

So you can see we have a tuple of i and i to the power 2 and for i in range 5. So this tuple is then going to be passed on to the dict and they will create a dict you know object out of it. So if I run this you can see 0 and 0, 1 and 1, 2 and 4 second element of the tuples that we generated using the you know generator expression that has become value part, the first element is the key part. So in this fashion generator expression can also be useful not just to you know construct a iterator.

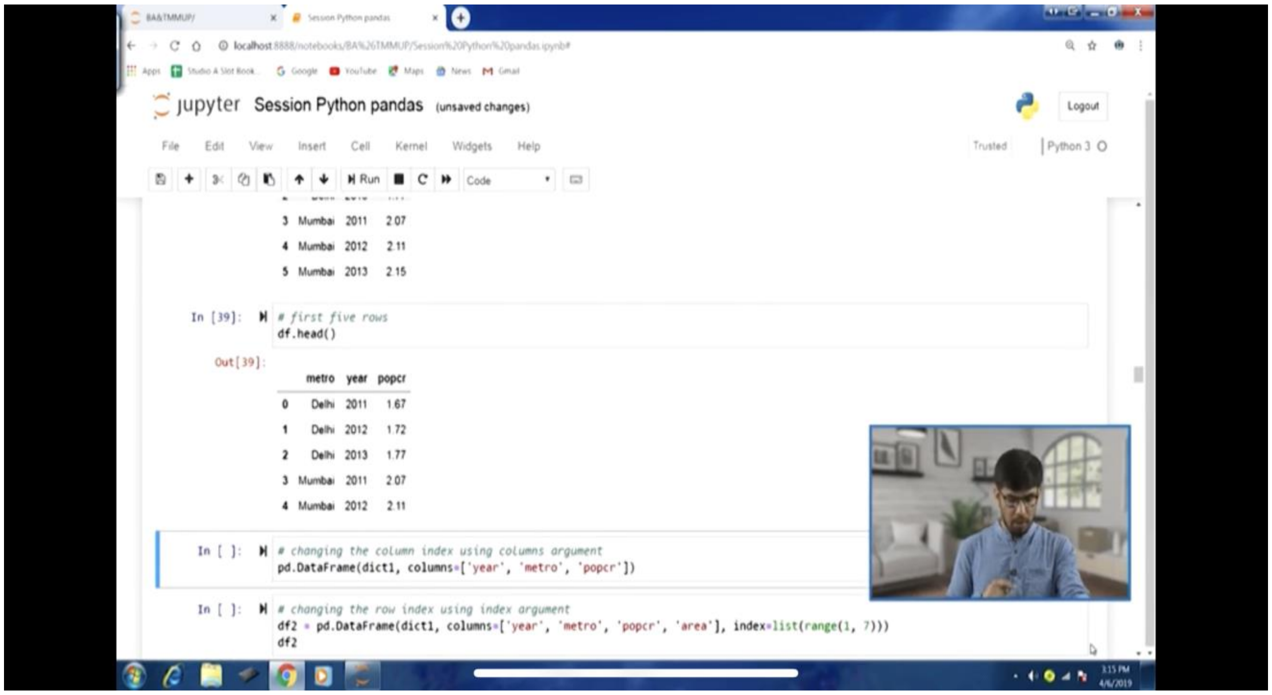
But also you know just you know is also in place of list comprehensions. Now let us talk about few more ways of you know creating generators so we have something called standard library iter tools. So this perfect reliability has a collection of generators which can be used. So there are a number of them there as part of this module all, however will talk about this one function group by which can be really useful in certain scenarios.

So what this group by function does it takes a sequence and a function as arguments. So 2 arguments as sequence and a function and it groups consecutive elements in the sequence by return value of the function. So 1 argument is the sequence and the another argument is a function is applied to each of the element of the sequence and the return value of that function and the sequence element they are combined they are grouped by this function.

So let us you know understand that through an example, so first we need to import this module iter tools, then let us take an example of this you know list of strings terms1, so we have Ram, Ramsevak, Shiv, Shivcharan, Krishn, Krishnala. So what we will do is we have written if lambda function here where we are returning the first letter of a given string. Now we are running a loop so in that we have for later,terms and the iter tools.roup by this is the function that we talked about and it is taking 2 argument term1.

This is a list of strings and then the another function first later which is a lambda function. Now it will group the elements you know of the terms1 and the return value that we get after applying first later on the elements of terms1 and then we would be printing the same. So if I run this you can see we got r that is the first later and you can see you know all the elements that share that first letter they have been grouped into the list and then capital S.

(Refer Slide Time: 32:45)



```
   3  Mumbai  2011  2.07
   4  Mumbai  2012  2.11
   5  Mumbai  2013  2.15

In [39]:  # first five rows
          df.head()

Out[39]:
      metro  year  popcr
   0  Delhi  2011  1.67
   1  Delhi  2012  1.72
   2  Delhi  2013  1.77
   3  Mumbai 2011  2.07
   4  Mumbai 2012  2.11

In [ ]:   # changing the column index using columns argument
          pd.DataFrame(dict1, columns=['year', 'metro', 'popcr'])

In [ ]:   # changing the row index using index argument
          df2 = pd.DataFrame(dict1, columns=['year', 'metro', 'popcr', 'area'], index=list(range(1, 7)))
          df2
```

And then the all the you know all the elements string elements that had you know first later they have been you know grouped. So you can see group by so whatever the return value of the function that is used as a key to group the elements of the sequences and the same thing is being has been returned. So you can see this group by function this is also in a way working as a generator here, now let us move on.

Now let us talk about certain other aspects which are related to errors on exception handling. So errors exception handling is typically required when we are looking to build or write robust programs, because certain situation might demand this because due to human errors or some other reasons you know there might be certain bugs or errors in the code that we might have. So therefore you know how do we manage how do we understand whether we have run into some error we have run into some exception.

So for that you know try accept blocks can be used to write some robust programs, because they in essence will help you in terms of identifying with dealing with those errors and exceptions.
(Video Ends: 27:52)

So we will talk about try and accept blocks in the next lecture where we will talk about when we do not get input proper input value or proper type. So different scenarios and how to handle those different scenarios. So we would like to talk about this in the next lecture. So at this point we would like to stop here itself, thank you.

Keywords: Punctuation, Pandas, Text mining modelling, iteration, Python Pandas.