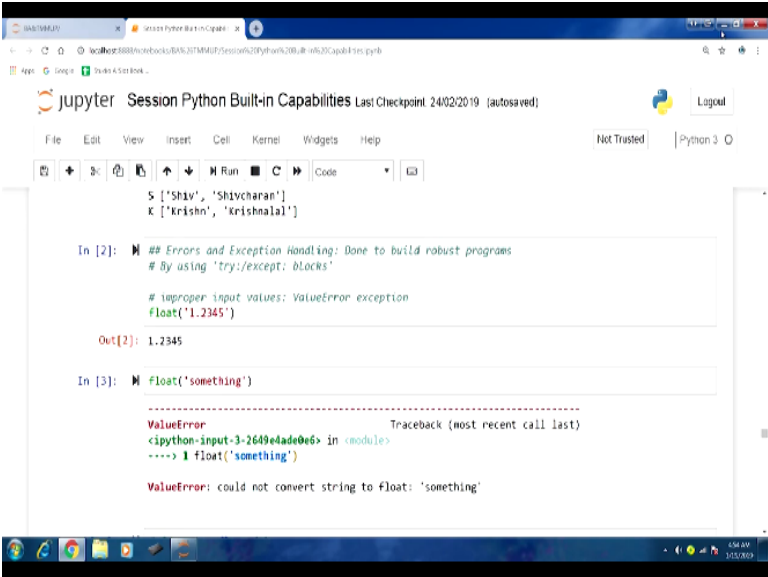


**Business Analytics And Text Mining Modeling Using Python**  
**Prof. Gaurav Dixit**  
**Department of Management Studies**  
**Indian Institute of Technology-Roorkee**

**Lecture-17**  
**Built-in Capabilities of Python-IX**

Welcome to the course business analytics and text mining modeling using python. So in previous few lectures we have been focusing on the you know built-in capabilities that are available in python platform and data structure part most of it we have been able to cover and now let us move to the next aspect that is errors and exception handling. So this is mainly required to be able to write robust programs.

**(Refer Slide Time: 00:56)**



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities Last Checkpoint: 24/02/2019 (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, cell execution, and code execution. The notebook content is as follows:

```
S ['Shiv', 'Shivcharan']
K ['Krishn', 'Krishnalal']

In [2]: ## Errors and Exception Handling: Done to build robust programs
        # By using 'try:except: blocks'

        # improper input values: ValueError exception
        float('1.2345')

Out[2]: 1.2345

In [3]: float('something')
```

The output for the third cell shows a `ValueError` exception with a traceback:

```
ValueError                                Traceback (most recent call last)
<ipython-input-3-2649e4ade0e6> in <module>
----> 1 float('something')

ValueError: could not convert string to float: 'something'
```

And for this we have this mechanism in python where we can use try except blocks and using these you know blocks we would be able to deal with certain errors, certain exceptions that you know are sometimes required to be handled. So let us start so first you know we will talk about the improper input values. So it might so happen that you know we are passing in our argument to a particular function which is not accepting a particular you know kind of input value.

So in that case it might show some error, so how do we, how can we use the try except blocks to handle that kind of error or exception. So let us start, so first thing improper input values, so when we are saying improper input values then we might get into this exception value error

exception that means the input values, the appropriate input values was not passed on.

**(Video Starts: 01:56)**

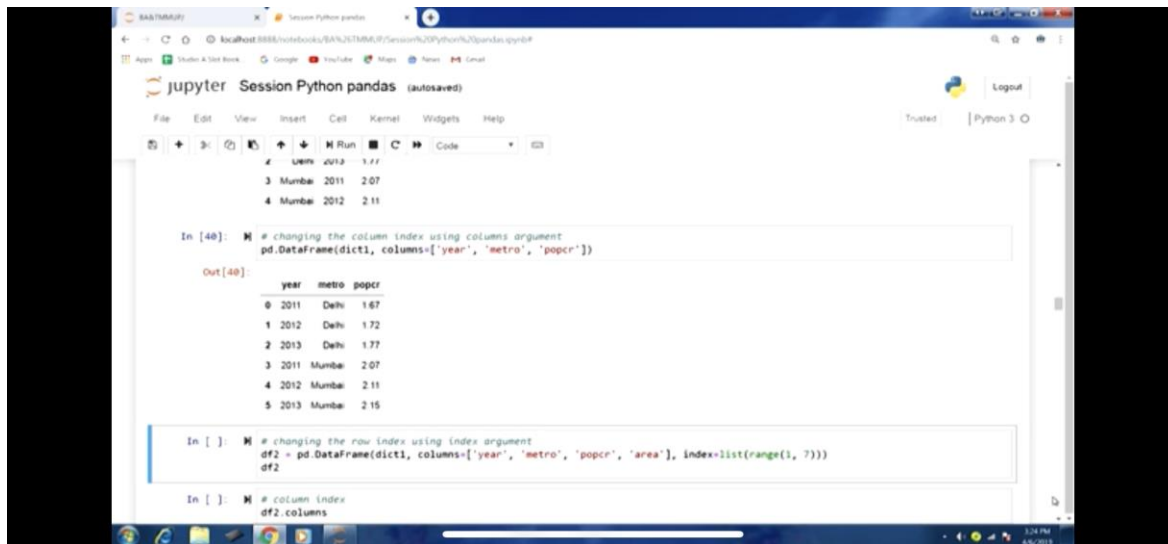
So let us run this, so what we are doing here we are calling the code function and 1.2345, we are trying to convert this value into a floating point number. So because this is a correct input, so immediately we get the output without any error or exceptions. Now if we try the same function float and pass on the value as you know a string value as an argument let us say something. So you can see in the next line that I am going to run the only argument that we have here we are passing a string value here.

However the string value cannot be appropriately converted into a float point number. So therefore this is an improper input value and therefore we will get into value error exception. So let us run this and you can see that we have got value error you know this exception and you can see could not convert string to float you know. So something was the string that we passed.

So it could not be you know convert into a floating-point number because it is an improper value. Now if we run into this kind of problem how do we handle this in our code, so for this we can use try except blocks. So in this we would like to define these kind of blocks, so we will you know use the def keyword and we are writing this our try except blocks, this is attempt float1  
attemptyfloat1.

So this value x is the argument that for which we are writing this and then we have this try keyword try colon and then return float x. So if the value is proper then we will be returning the the return value of the floatx function, if the value is not you know proper then we will run into the except block and we will be returning this message improper input value. So that the coder or the analyst will know that improper input value was passed on.

(Refer Slide Time:03:35)



```
In [48]: # changing the column index using columns argument
pd.DataFrame(dict1, columns=['year', 'metro', 'popcr'])

Out[48]:
```

	year	metro	popcr
0	2011	Delhi	1.67
1	2012	Delhi	1.72
2	2013	Delhi	1.77
3	2011	Mumbai	2.07
4	2012	Mumbai	2.11
5	2013	Mumbai	2.15

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

In [ ]: # column index
df2.columns
```

So let us run this block, so now after defining this try except block `attempt_float1`, if we pass on this 1.2345 and there should be no problem here, so you can see we got 1.2345 as the output, however if we again try you know something that string value and we try to pass it on to this you know this new function `attempt_float1` and in this if we run this you can see you know improper input value as the output because that value was not found to be you know appropriate. So an exception was thrown.

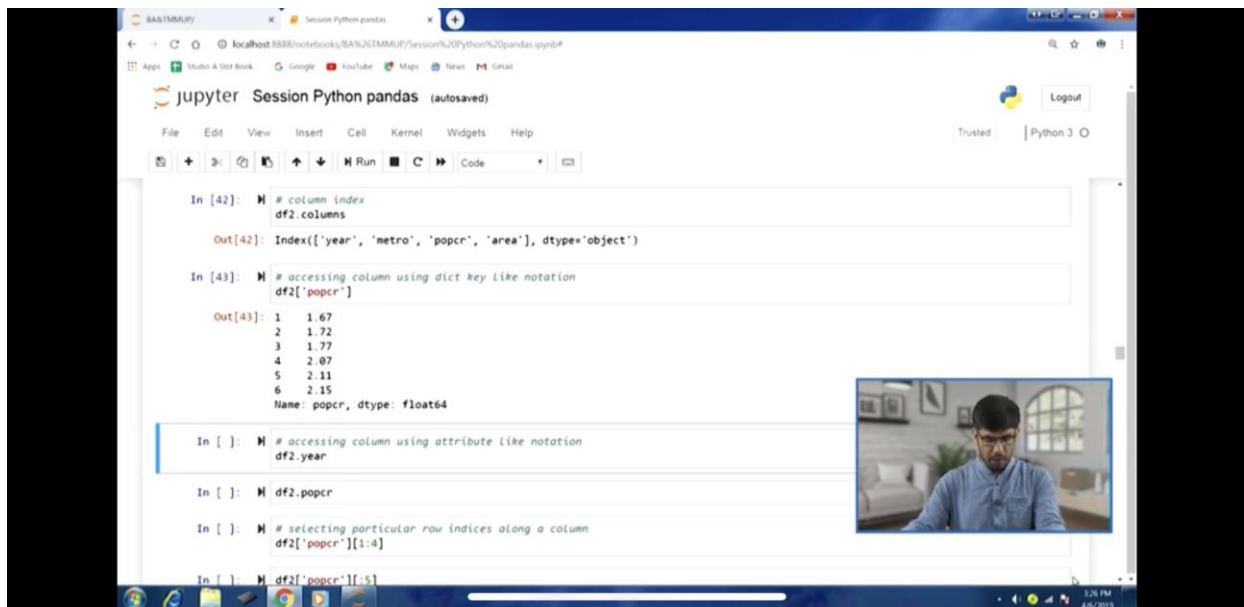
So to deal with that exception we have written this try except block within this function. So in a sense this is a wrapper to the original `attempt_float1` is a wrapper to the float function where if any exception is thrown we are handling that within this particular wrapper `attempt_float1`. So instead of using the inbuilt functions that are there in python float form we can write our own user-defined functions.

So that you know any exceptions and error handling could also be done. So let us move to the next example, now this is about second kind of exception that we might have to handle. So this is improper input types. So it might so happen that a particular input type a particular you know data section is not accepted by a you know particular function, but that is being passed on. So this is referred as type error exception where a wrong kind of type has been passed on.

So in this case again we are using the float function here and you can see in the parenthesis we

are passing just one element which is a tuple object here and having 2 elements 1,2. However the float is not supposed to accept this kind of type. So because of that we are getting type you know error exception. So the argument has to be string or number, you might think that string is also not something that should be acceptable.

However in the string format we might pass on a numeric value, so a numeric value within double quotes or single quote it could be a string value but it can be appropriately you know handled by the float function. So string again allowed type, so you would see that if we if I run this in the error you would see that type error float argument must be a string or a number. So because in a string there is a still possibility that a numeric value can be passed on within the double quotes or single course in value. **(Refer Slide Time: 05:32)**



```
In [42]: # column index
df2.columns

Out[42]: Index(['year', 'metro', 'popcr', 'area'], dtype='object')

In [43]: # accessing column using dict key like notation
df2['popcr']

Out[43]: 1    1.67
         2    1.72
         3    1.77
         4    2.07
         5    2.11
         6    2.15
         Name: popcr, dtype: float64

In [ ]: # accessing column using attribute like notation
df2.year

In [ ]: df2.popcr

In [ ]: # selecting particular row indices along a column
df2['popcr'][:5]

In [ ]: df2['popcr'][:5]
```

So to be able to deal with that cases the float argument except the string also as an argument in python. However tuple is not accepted here, so for tuple we are getting type error exception. Now how do we manage this, so for this again we can write a you know wrapper function here def attempt\_float2 and in this function again we are defining certain you know try except blocks, So you can see we have 1 try block and the 1, 2 except blocks.

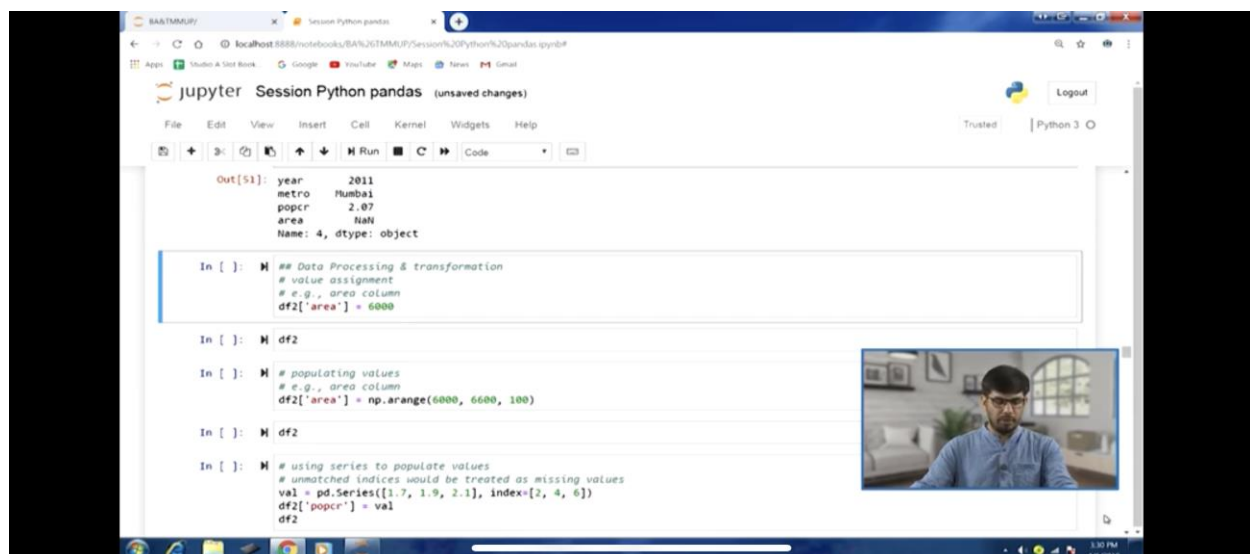
So here in the try block as before we are returning the you know floatx, so if there is no error no exception that has been thrown then we would like to return the whatever return value we get

from you know float function. Now the second except block is handling the value error which we have already discussed. So if the improper input value like you know a string value where the characters are you know they are like something that we did.

So to handle that we have this exception block here, except a value error and we are returning this improper input value. Now the second case that we are discussing in our type error, so we have the second except block here except type error and in this case we are returning the improper input type as the message. So that the user or the coder or the analyst will come to know that you know the improper input type was passed on as an argument to the function.

So attempt\_float2 in this second wrapper function that we have that I have written here you know we are handling both these types, we have 2 exceptions blocks a value of 1 for value error the another for type error. So if I define this if I run this line of code and now this function would be defined. Now if I pass on an appropriate value you know 1.2345 you can see this appropriate this is you know being passed in within these single quotes here.

So if I run this there is no problem in this string, this is a string but it can be appropriately converted into a numeric value and therefore convert into a floating-point number. So this is okay, now in the second example I am passing on something, so in this case this is an improper input values. So if I run this so you can see improper input value. So because something this particular string is made of characters. **(Refer Slide Time: 09:25)**



The screenshot displays a Jupyter Notebook window titled "Session Python pandas (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook content shows the following:

```
Out[51]: year      2011
         metro    Mumbai
         popcr    2.07
         area      NaN
         Name: 4, dtype: object
```

```
In [ ]: ## Data Processing & transformation
        # value assignment
        # e.g., area column
        df2['area'] = 6000

In [ ]: df2

In [ ]: ## populating values
        # e.g., area column
        df2['area'] = np.arange(6000, 6600, 100)

In [ ]: df2

In [ ]: ## using series to populate values
        # unmatched indices would be treated as missing values
        val = pd.Series([1.7, 1.9, 2.1], index=[2, 4, 6])
        df2['popcr'] = val
        df2
```

A small video inset in the bottom right corner shows a man with glasses and a blue shirt, likely the presenter, speaking.

So this is not acceptable, so we have already handled this exception, so we have caught the message improper input value. Third example is that where we are passing on in tuple, so `attempt_float2` and in the parentheses we have this one argument and that is a tuple made of 1 elements 1,2.

So if we force on this so a exception type error would be erased here and that we are handling in this function `attempt_float2`, so if I run this we will get this message improper input type. So in this fashion you can see depending on the kind of exception scenario that we might you know expect depending on the kind of you know data that we might be dealing with certain scenarios might be possible.

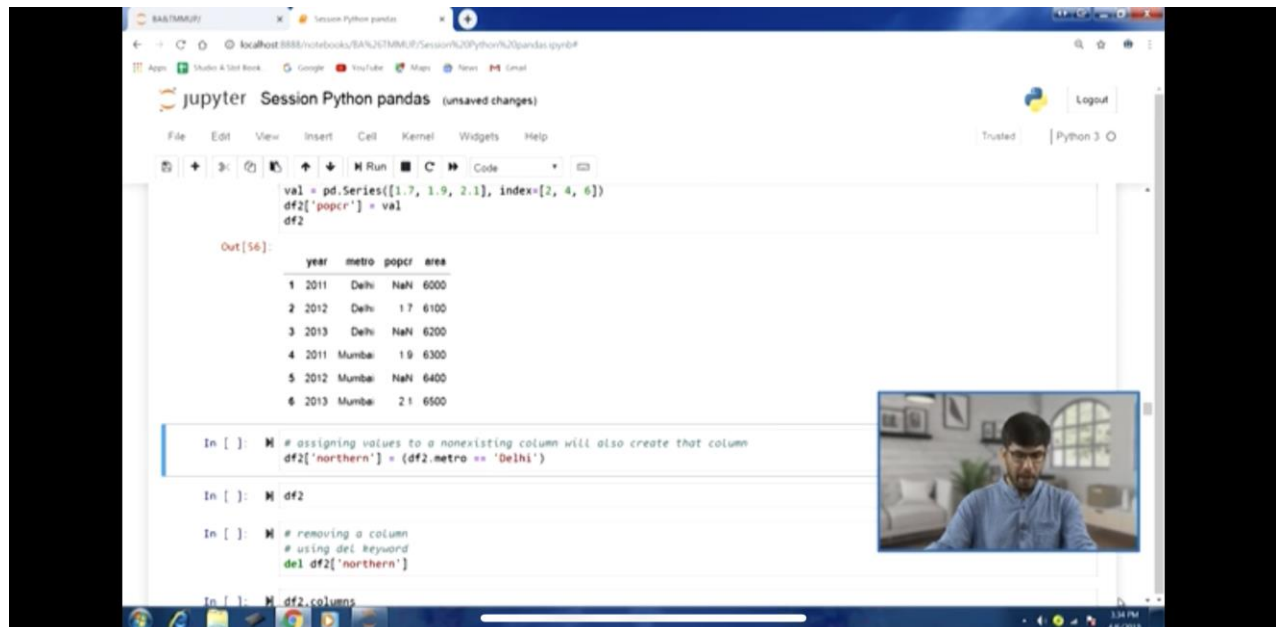
So we can always write these kind of wrapper kind of functions which will have try and except blocks and we will be able to handle different scenarios and in this sense we will be able to write and develop robust programs. So when we say you know robust, so we would be able to handle different kind of inputs different kind of scenarios errors and other things that may happen there.

Now let us move on to the next aspect, earlier example that we have just discussed, there we were handling these 2 there we had 2 except blocks and we were handling value error and type error you know separately. However we can use try and except block where we can write except block which can you know catch multiple exception types using just one statement. So how do we define that.

so this is the example you can see now we are writing another function here `attempt_float3` and in this case you can see that try block is same as before we are just returning the float value and then we have just one except block however we have in the parentheses we are you know mentioning both kind of exceptions value error, type error. So this is a tuple, so these 2 exceptions we are handling.

And if you look at the you know the return masses that we have here is improper input value or type. So it could be either value error exception or type error exceptions. So in one statement we

are trying to handle, we are trying to catch multiple exception types here. So in this fashion we can define this function and you know handle these kind of exceptions. So let us run this, now you can see a next example that we have at attempt\_float3 and we are passing a numeric in the string form a numeric value in the string form. (Refer Slide Time:12:54)



The screenshot shows a Jupyter Notebook titled "Session Python pandas (unsaved changes)". The code in the cell is as follows:

```
val = pd.Series([1.7, 1.9, 2.1], index=[2, 4, 6])
df2['popcr'] = val
df2
```

The output of the code is a DataFrame with 6 rows and 4 columns: year, metro, popcr, and area.

	year	metro	popcr	area
1	2011	Delhi	NaN	6000
2	2012	Delhi	1.7	6100
3	2013	Delhi	NaN	6200
4	2011	Mumbai	1.9	6300
5	2012	Mumbai	NaN	6400
6	2013	Mumbai	2.1	6500

Below the output, there are three more code blocks:

```
In [ ]: # assigning values to a nonexisting column will also create that column
df2['northern'] = (df2.metro == 'Delhi')
```

```
In [ ]: df2
```

```
In [ ]: # removing a column
# using del keyword
del df2['northern']
```

```
In [ ]: df2.columns
```

A video inset in the bottom right corner shows a man with a beard and glasses, wearing a blue shirt, speaking.

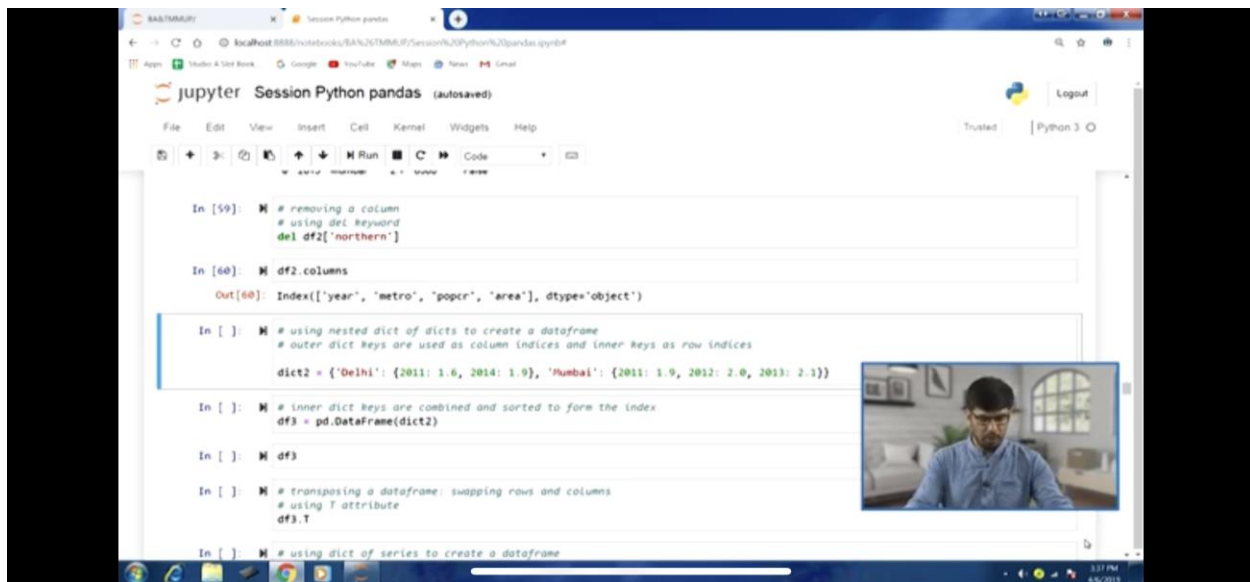
So if I run this we will get this floating point number and next example again that we are passing on something this string value to this attempt\_float3 function and if I run this you can see I am getting improper input value or type. So we are not you know it might not matter as to you know you know deal them separately rather we can club them in this fashion using 1 except block and we can adjust our messaging also that improper input value or type.

So one of these 2 things occurred and that we have handling 1 you know multiple exception have been handling 1 statement or 1 except block. Now if we pass on tuples then again we will get the same message. So this is the way to handle this, there might be certain scenarios where we would like to execute a code block regardless of whether try block succeeds or not. So to deal with these scenarios we have another you know another keyword and other block that can be used that is using finally block.

So in this case whether the try block succeeds or not this is code block is going to be you know

executed. So let us take this example here, so the example that we are taking here is about generating locks or cleaning of the python objects, given any scenario we might be creating and initializing and processing you know many my python object. So irrespective of the so you know respective of the code that we are writing we would always like to clean up the python objects that we might have created we might have used.

So this kind of situation we would like to this kind of code execution we would like to perform irrespective of whether a particular you know try block succeeds or not. So in such situation we might also prefer to not suppress exceptions. In this case we can use the finally block, so if you look at the code that we are writing is that I have written here. So first thing is I am opening this temp.txt file and in the file mode is write. **(Refer Slide Time: 16:04)**



```
In [59]: # removing a column
# using del keyword
del df2['northern']

In [60]: df2.columns
Out[60]: Index(['year', 'metro', 'popcr', 'area'], dtype='object')

In [ ]: # using nested dict of dicts to create a dataframe
# outer dict keys are used as column indices and inner keys as row indices
dict2 = {'Delhi': {2011: 1.6, 2014: 1.9}, 'Mumbai': {2011: 1.9, 2012: 2.0, 2013: 2.1}}

In [ ]: # inner dict keys are combined and sorted to form the index
df3 = pd.DataFrame(dict2)

In [ ]: df3

In [ ]: # transposing a dataframe: swapping rows and columns
# using T attribute
df3.T

In [ ]: # using dict of series to create a dataframe
```

So this is write only a file mode, we will talk about file handling in detail in later lectures, however in this example we have just opened a file and we get file object fl and the file is going to be opened in the write only mode. So any message that we would like to write in this file, so you can consider this like a logging thing. So we are trying to create a log here, generate log here. So this file log file has been opened you know in the write mode.

So then in the try block you know we are writing fl.write and done, so in a sense in the you know try block succeeds then we you know will have done message there and then we have the except



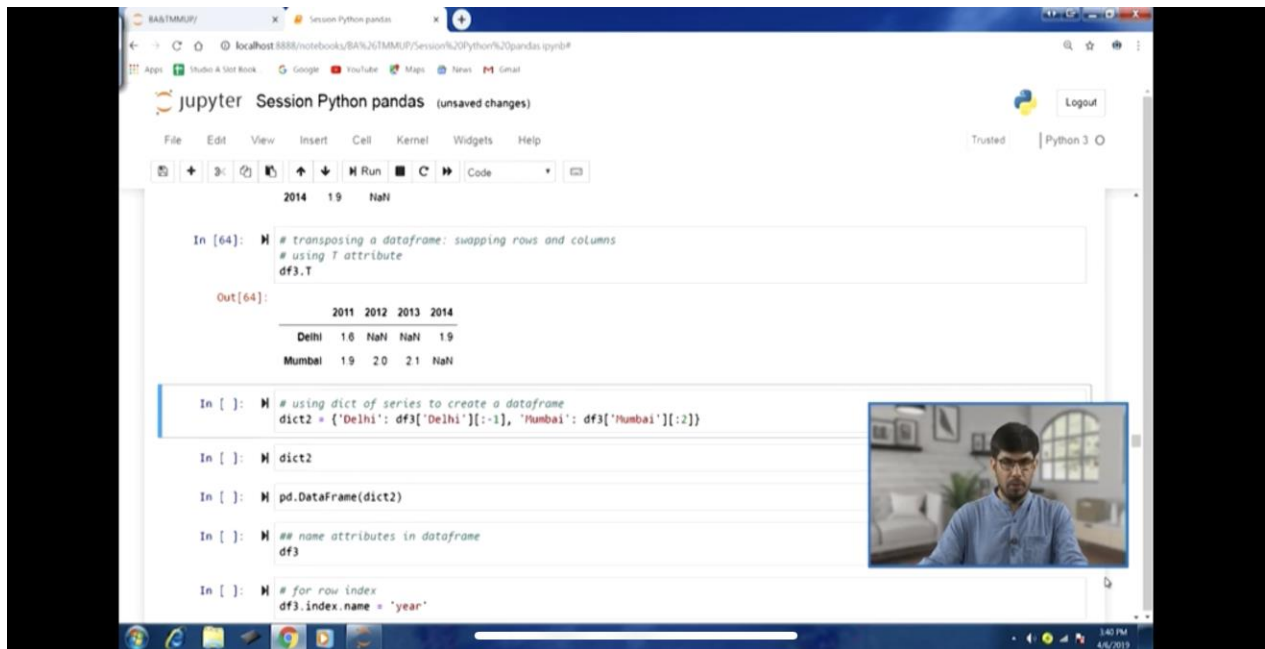
log . So in that case you know some exception is there then we would be writing in the log not done. So that would be written there. Now we have the finally block here, so again finally and then colon and then we have this block here.

So this particular block is going to be executed regardless of whether the try block succeeds or not. So what we have in this block `fl.write` and `log1` `fl.write log2` and `fl`. you know `close`. So we will be writing these 2 statements there in that file you know file object `fl` and that means this file `temp.txt` and then we will flows this file object. So in a sense we have recorded our you know `log` statements and then we have cleaned up the existing you know a python objects which is in this case you know this `fl` this file object.

So let us run this, so once we run this we can you know always you know have you look at the log that we have just generated. So for this we will have to use this `read` method. So `fl.read`. So if I run this you see I get an error here `io operation on closed file` right. So this error we have received, so actually what is required is that that the file was not opened in the read mode and the file was actually closed.

So first we need to open the file in the read mode and then we will be able to read the contents of the file. So by default in python platform the files are open in the text mode. So we can first open this file `fl1`, this new file object open `temp.txt` this file object `fl1` would be created in the read mode and then you know `fl1`.then we will be able to read the you know contents of that file. So let us run this and you can see in the output we have got done `log1+log2`.

So you can see why we have got these messages because if we go back to the block that we had run you can see that you know first the try block has succeeded, so we got the masses done here and then once you know after that finally block is going to be executed regardless of whether try block succeeds or not. But in this case try block succeeded.(Refer Slide Time:18:41)



So then we will have log1 and log2. So those 3 you know messages are actually written in this log file done log1 and log2. Now let us close this file f11, there might be certain scenarios that we would like to execute a code block only if try block succeeds. So only you know the try log is goes through only then we would like to execute this certain piece of code. So for that we can use else block.

So how to define this, so again we are taking this the same example here the temp.txt file we already have we also have some data written in this file. So let us open this file f12 file object, let us open this temp.txt, in another file mode write a which is for you know appending in the file appending any content in the file. So first we have the try block here and f12.try. So we are typing you know done.

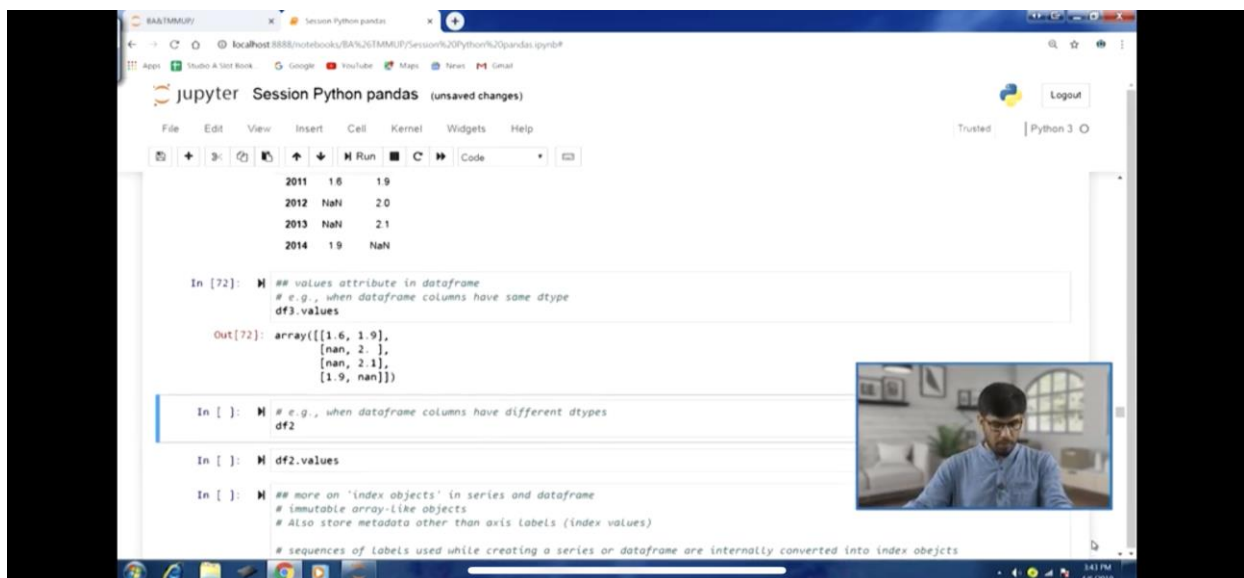
And only if this block succeed then we want if we want to execute the code then you can see we have the else block here and in this we are printing this statement succeeded. So this particular else block is only going to be executed, if the try block succeeds. If the try block fails then the you know this else block is not going to be you know executed and the except block might be executed where we have the you know print failed statement.

And the finally block is also there, so finally block will also be you know be executed. So we will be closing this you know file object. So let us run this, if I run this you can see here that we have caught this message succeeded here, you can see the succeeded message we have got here. So that means the try block that was successful that succeeded and therefore else block was also executed.

And therefore we have got this we have printed this message succeeded. So 2 additional block you know in addition to the try and except block that we typically have we can have you know these 2 additional blocks else and finally. Now let us move to the file management aspects in the python object. So how do we work with files in python that is quite simple in this particular language platform and python is particularly very popular for text and file managing.

So we will talk about the file management aspects now. So one of the important thing that we one of the first thing that we do in file management is to open a file and there are various operational mode to open a file it could be read-only mode, it could be right only mode, it could be read and write and you know various other modes are also there we will be discussing them. So first thing is opening a file in a particular operational mode.

**(Refer Slide Time:21:53)**

The image shows a Jupyter Notebook window titled "Session Python pandas". The notebook contains a pandas DataFrame with 4 rows and 3 columns. The first column contains years (2011, 2012, 2013, 2014), the second column contains the word "Nah", and the third column contains numerical values (1.9, 2.0, 2.1, NaN). Below the DataFrame, there is a code cell with the following text: 

```
In [72]: # values attribute in dataframe  
# e.g., when dataframe columns have same dtype  
df3.values
```

 The output of this cell is displayed as: 

```
Out[72]: array([[1.6, 1.9],  
               [nan, 2. ],  
               [nan, 2.1],  
               [1.9, nan]])
```

 Below the output, there are two more code cells. The first one is: 

```
In [ ]: # e.g., when dataframe columns have different dtypes  
df2
```

 The second one is: 

```
In [ ]: df2.values
```

 At the bottom of the notebook, there is a comment: 

```
# more on 'index objects' in series and dataframe  
# immutable array-like objects  
# Also store metadata other than axis labels (index values)  
# sequences of labels used while creating a series or dataframe are internally converted into index objects
```

 In the bottom right corner of the notebook window, there is a small video inset showing a man with glasses and a blue shirt speaking.

So by default the file is open in the read-only mode or the open function we have already discussed and used in the previous examples as well and as far as the file path that is the first

argument that is to be passed in the open function it could be relative or execute. So we can give either the full path in an absolute sense or if we are working if we have all our files in the working directory itself within the current directory itself.

Then we can just use the relative addressing and just give the name of the file that we want to open. So here I am defining a path variable here path temp1.txt and I would like to open this file open this path here file path object here. So if I just try to run this I will get an error here that no such file or directory temp1.txt. So this example I have written actually to tell you that if the file does not exist we are going to run into this kind of file not found error.

So first we need to ensure that the file is there, so temp.txt is the file that we already have there, so we will run this again so next file object fl3 and we will run this and this file is already there. So we will just use it for you know to discuss other aspects. So now let us talk about the read method, the use of read method is that whatever data that we want to read we can indicate in the arguments or we if we do not indicate all the data that is part of that file that would be returned in the form of is you know string.

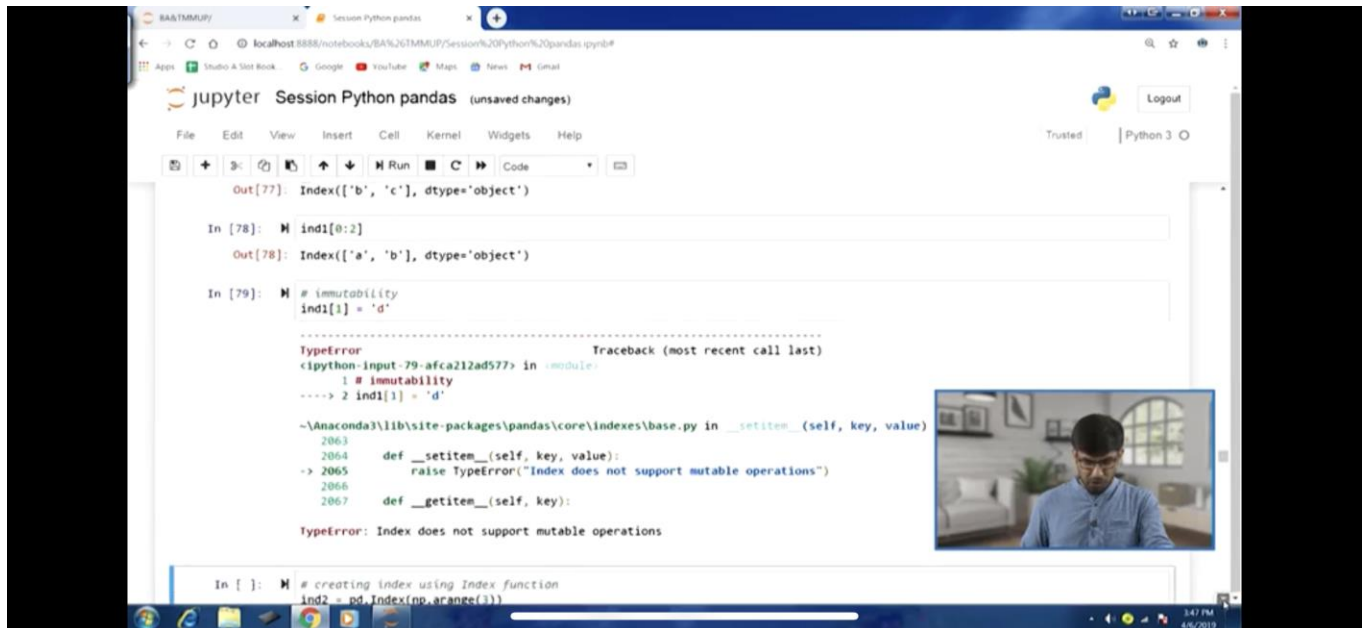
So read method returns data from file as a string, so in the fl3.read, in this if I run this you know if I make this method call. So you can see we have got this string as an output done then slash and then log1 slash and log2, then session done and done. So this is the string that we have in this file. So this is how we can read it. Now once we read a file we might reach to the end of the file so in that case if we again read the file then an empty bytes object is going to be return.

So if I run the next line again if I repeat fl3 dot read again if I execute this you can see an empty object has been created because we have already these the end of the file, had we passed on you know some integer value as an argument to the read method, then those many characters would have been you know read and would have been you know given in the output and the file position will also change and the next you know read will start from those positions.

So in this case we had you know returns of all the data in the previous call therefore we these the end of file and therefore an empty bytes object is returned in this case. Now we have also used

the close function, so the main purpose of this particular function is to closing the file objects that are typically created with the open function. So this is an important cleanup operation in the sense the resources that are involved in you know in a file object.

**(Refer Slide Time: 25:19)**

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/BA%26TMMUP/Session%20Python%20pandas.py?b4'. The notebook title is 'Session Python pandas (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The code area shows the following:

```
Out[77]: Index(['b', 'c'], dtype='object')

In [78]: ind1[0:2]
Out[78]: Index(['a', 'b'], dtype='object')

In [79]: # immutability
ind1[1] = 'd'

-----
TypeError                                Traceback (most recent call last)
<ipython-input-79-afca212ad577> in <module>:
      1 # immutability
----> 2 ind1[1] = 'd'

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in _setitem_(self, key, value)
    2063
    2064     def _setitem_(self, key, value):
-> 2065         raise TypeError("Index does not support mutable operations")
    2066
    2067     def _getitem_(self, key):

TypeError: Index does not support mutable operations
```

Below the error, a new code cell is partially visible:

```
In [ ]: # creating index using Index function
ind2 = pd.Index(np.arange(3))
```

A small video inset in the bottom right corner shows a man with glasses and a blue shirt speaking.

They would be returned back to the operating system, so this is a cleanup operation, so it is recommended to always close all the file objects. So if I run this `flt.close` then this worker file object is going to be closed. Now there are going to be certain scenarios where we would like to iterate over the lines in a file. So in a file we might be having you know text data and we would like to iterate over the lines of you know lines of data that we might have in that file.

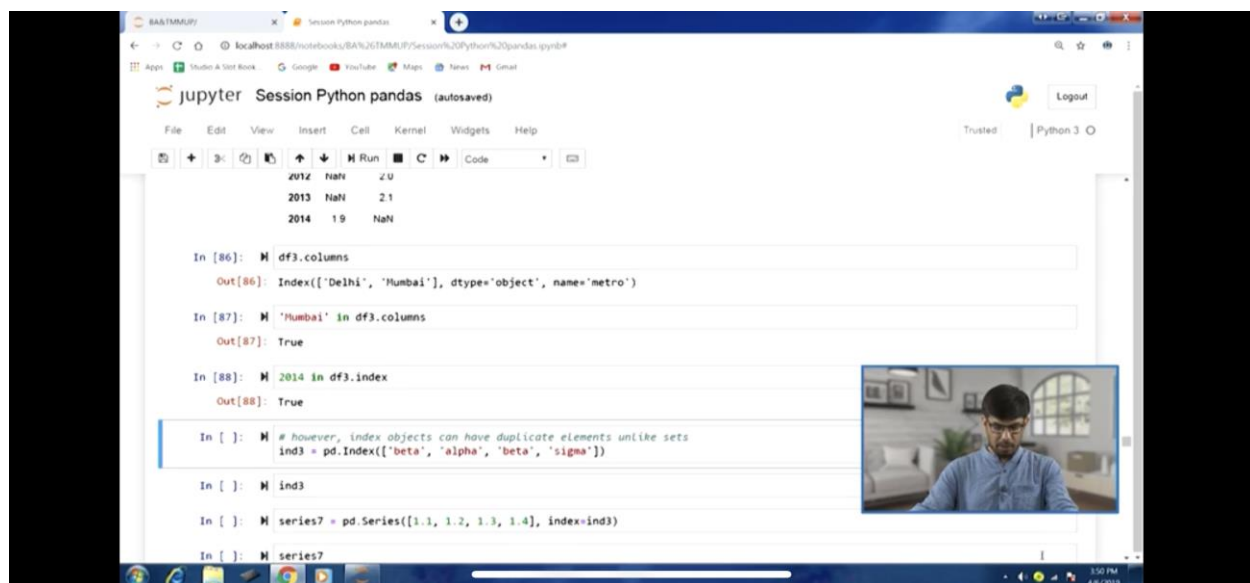
So how this can be done, so we can treat file handle as a list and then this iteration over that you know handle can be perform. So here is an example, so first thing we are creating this file object `fl3` open path. So that `temp.txt` that we have created that file is going to be open in the read-only mode and then we are running a for loop, so far line in `fl3` and then we have passed remember we had discussed the `paths` keyword before.

So we do not want to do anything any operation then we can use the `past` keyword we have talked you know previously that because we do not have any you know braces or other you know

was kind of that kind of syntax that is typically used in other languages therefore we require this kind of a statement if we do not want to execute anything. So in this case we just wanted to demonstrate that we can iterate over the lines in a file.

So this is how we can do it, so if I run this so because we have iterated over all the lines in this file therefore you must have least at the end of the file. So therefore if I try to read this file then in the next line that I have `fl3.read` then I should be expecting an empty bytes object. So that is the case here you can see in the output number 26 you can get the empty bytes object because we have already iterated over all the lines in the file.

Now there is something called end of line marker, so by default slash n is the end of line marker. So let us take an example to understand this aspect as well, so let us open the file `temp.txt` open path path is indicating that file name `temp.txt` and then we are running a loop for line in `fl3` and the only statement that we have in the for block is print that line. So we would be iterating over the lines in this file and we would be printing each line. **(Refer Slide Time: 28:23)**



```
In [86]: df3.columns
Out[86]: Index(['Delhi', 'Mumbai'], dtype='object', name='metro')
```

```
In [87]: "Mumbai" in df3.columns
Out[87]: True
```

```
In [88]: 2014 in df3.index
Out[88]: True
```

```
In [ ]: # however, index objects can have duplicate elements unlike sets
ind3 = pd.Index(["beta", "alpha", "beta", "sigma"])

In [ ]: ind3

In [ ]: series7 = pd.Series([1.1, 1.2, 1.3, 1.4], index=ind3)

In [ ]: series7
```

So if I run this particular you know code here you can see I have got done log1 log2 done done, so you can see that each of these lines have been printed as a new line in the output. So that is itself is indicating that slash n is the you know end of line marker here is the shall n being used as the l marker here. Now sometimes you might know you might not you know you might want

to have a check or you know you might forget to remember that whether a particular object is open or not.

So for that also we have a method called `closed`, so this particular `closed` method is going to return `true` if the file is closed otherwise if the file is open then it will return `false`. So in this case you can see that `fl3` file object we have we there are still open we have just printed all the lines in that you know `fl3` in that file. So if I run this `fl3.close` because this file is open I should get `false` as the return value.

So you can see this, so this can be really useful when we want to put a check somewhere you know to make sure whether a file is open or not. Now if I again if I try to you know read this file here you can see that you know that empty bytes object is returned here. Now sometimes you know you might want the you know list of lines which are free from end of line markers so that can also be done for that we can use the a list comprehension where we have this transformative expression `line.rstrip` you know method.

So it will strip the line of any end of line markers and we are iterating over all the lines in that file. So you can see line in this list comprehension in the within the brackets we have `line.rstrip` and then for line in open paths. So if I run this you can see all the lines that we had in this file. So we get a list of all those you know all those statements, all those lines done log1 log2 done done. So in this end of line markers is marker is gone that has been stripped off.

Now I can close the file here `fl3.close`, now another aspect is cleaning up open files so though we can we can use the `close` method to close any file object that might be there. However sometimes might be difficult to track all the file objects that we might have created. So for that to you know **to** ease that that aspect cleaning of aspect we have another you know another keyword and blocks that can be used.

So this is with keyword, so this is actually used to perform automatic cleaning of resources like file objects after the execution of with block. So whatever is there in the with block once that part is executed, all the file objects all the sources they are going to be cleaned up. So let us take

an example here, so with keyword then with open path as fl4. So essentially this is you know we are creating this fl4 file object with this function open path.

And we have just a one line in this with block where we would like to again we have the list comprehension. So `line.rstrip()` or `line` in `fl4`, so essentially we will get the list of you know these statements in the lines that you have in the file. So if I run this you can see that you know `fl4` that will we can have a check whether `fl4` file object was closed or not. So now perform this `fl4.close` if I run this you can see `true` as the output.

So the output you know that this list comprehension that was performed and once that is executed if we check whether the file object is still whether the file is still open or not you can see `fl4.closed` that is `true`. So cleaned up automatically enough of resources has happened. So we can always write about file management file processing that we want to perform you know using the `with` keyword you know.

So this will ensure the automatic cleanup of this sources. Now once the file is closed if I try to read this file here `fl4.read()` you can see I will get this value error `io operation on closed file`. Now let us move on to the other aspects here. So there are other you know python file modes, so let us talk about the file modes which are typically used for writing. So one thing is this would be used carefully because purpose of sometimes for different we might have different writing different scenarios for writing content in the file.

So we have different file modes that can be used to cover all of those scenarios. So first one is the write mode write only mode, so for this we have within single course we have to specify `'w'`, so this right only more this is used to create a new file and once we use this write mode if there is any file and having some data with the same file name then that is going to be erased. So if we have a `temp.txt` and some data is there and if we create a new file object.

And we open the same file in the right mode using `W` write only mode using `W` file mode then the previous all the data that would be gone because we would essentially be creating you know a new file with the same file name, on the next file mode is small `x` within single quotes a small `x`



this is also write only mode only the operational part of is file mode is slightly different in the sense we create a new file if the file path does not already exist.

So this is to address the scenario where we might already have and you know existing file and therefore if the file is already there we might not like to create you know another file. So in that case you know we can use this file mode x. So it will open the file in the right only mode and however if the file is file with the same name is already existing. So opening of that file in that right only mode will actually fail.

Now there is certain scenarios we might like to you might like to add content you know in an existing file we would like to write content into an existing file, so for that we have another file mode appending. So this is within single course so we can type 'a' so this is to append to existing file and if in case the file does not exist already then it is going to be created and whatever content we want to write that would be written there.

If the file is there if the file is existing and we want to write something, so at the end of the file the particular content is going to be written. So let us take an example here, so we will take the example of this file mode a which is to which is for appending to exist file. So we will have fl5 and other file object here and open function and we are passing this file temp.txt that we have already created.

So we are using a as a file mode, so that means whatever we want to write that is going to be added at the end of the file. So we call the file function fl5.write and if you want to write this string append it you know. So this string is going to be this write method will write this pass the string to the file. So if I done this you can see the output you have got 9 that is actually number of characters that are to be written into the file.

So that is return here let me close this file now fl5.close and now if I want to deed the lines and that are part of this file. So I can use this with block with open temp.txt where we have just now written appended as fl6 and I will record I am using another method lead lines to read all the lines that are in that file and I am storing that in the lines you know variable here. So if I run this

I will have lines and again if I execute this.

So you can see this is this list of strings the last element is appended, so you can see at the end of all the you know of the file this particular string has been written, if I try to open this file in another file mode writew w mode, then if I try this fl6 open temp.txt w because you know whatever content that is there this file is already existing. So whatever data that is there that is going to be erased here. So if I run this now here if I try to just you know read this file ok this is open in the write only mode.

So I would not be able to read here, so you can see not readable because I need to open the file in the read mode. So first I will close this and then I will open this file open path and then I will lead this and you can see we have got the empty bytes object.

**(Video Ends: 38:14)**

Because we had used the write mode and for an existing file the same file name. So therefore all the data all the content that was there in the file that was erased. So we would like to stop at this point and we will continue our discussion on file management in the next lecture, thank you.

**Keywords: file managing, file management, float, python pandas.**