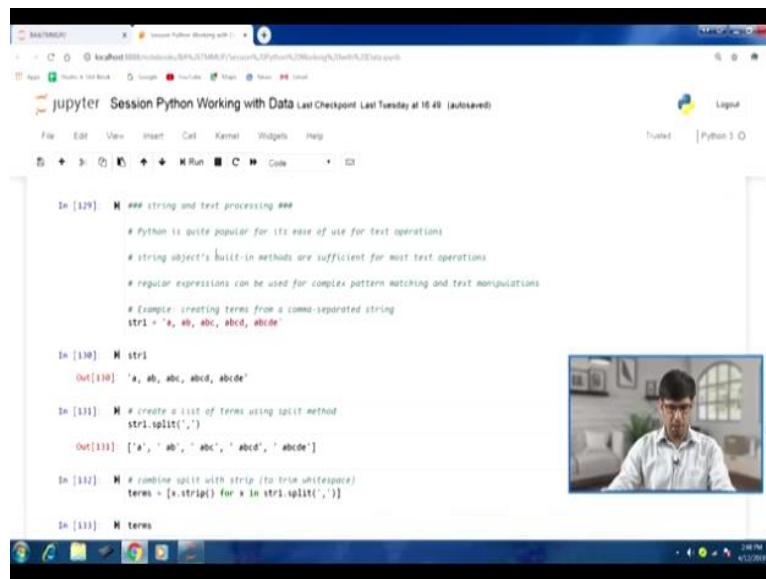


Business Analytics And Text Mining Modeling Using python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology Roorkee

Lecture-33
String and Text Processing-Part I

Welcome to the course business analytics and text mining modeling using python. So, in the last few lectures we have been focusing on how to work with data in the python environment. And now in this particular lecture we are going to start our discussion on another aspect which is string and text processing and quite relevant to the text mining modeling that we want to cover in this course.

(Refer Slide Time: 00:51)



```
## string and text processing ##

# Python is quite popular for its ease of use for text operations
# string object's built-in methods are sufficient for most text operations
# regular expressions can be used for complex pattern matching and text manipulations
# Example: creating terms from a comma-separated string
str1 = 'a, ab, abc, abcd, abcde'

In [129]: str1
Out[129]: 'a, ab, abc, abcd, abcde'

In [130]: # create a list of terms using split method
str1.split(',')
Out[130]: ['a', ' ab', ' abc', ' abcd', ' abcde']

In [131]: # combine split with strip (to trim whitespace)
terms = [x.strip() for x in str1.split(',')]

In [132]: terms
Out[132]: ['a', 'ab', 'abc', 'abcd', 'abcde']
```

So, python as we understand that this is quite popular for it is each of use for text operation. And much of the functionality that is present in the python platform and also the packages that we have covered. They are you know really they provide the ease of use the convenience and efficiency to be able to process the text and facilitate the text operations.

So, let us talk about the string object, so most of the text operations that we would typically like to perform, they can be covered by string objects built-in methods. However we can also use regular expression to perform some of the complex you know text manipulation python matching and text manipulation. So, we will talk about we would not go in to details of regular expressions

but you know briefly will touch upon this.

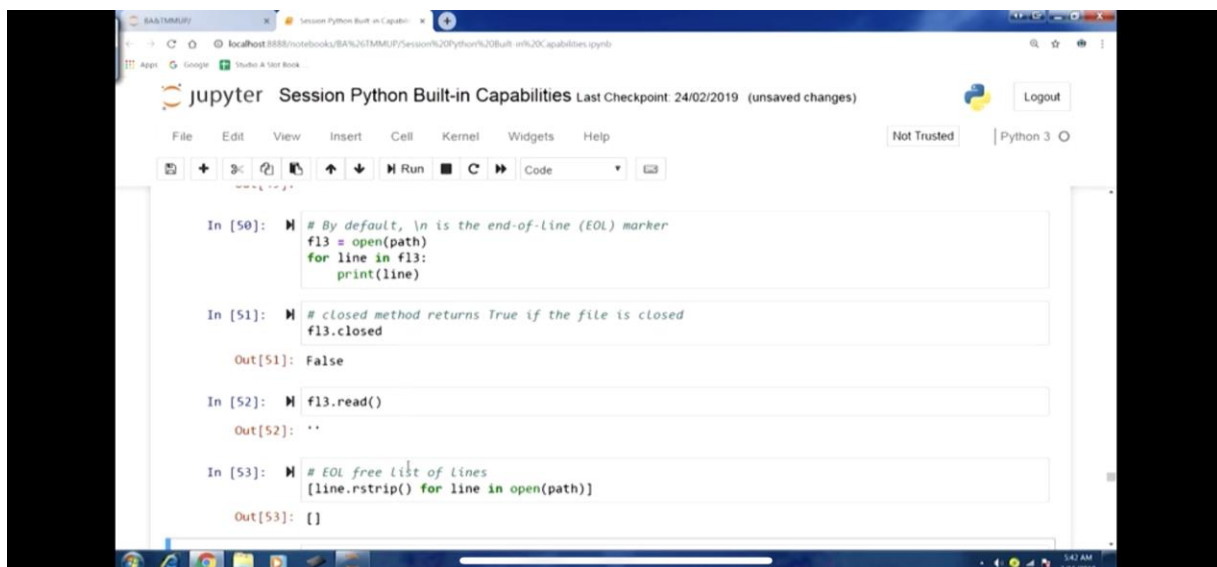
So, let us talk about discussion on string and text processing in python environment, so let us take a variable let us take a string variable here. So, you can see here we have a line of text here, so which we are going to define using a string variable string object str1.

(Video Starts: 02:06)

So, if I run this one, so we will have this str1 object here with these terms with these characters rather a, ab, abc and all that. So, this is the example that will take forward to demonstrate some of the built-in methods that we can use in string. So, first thing we can create a list of terms using a split method, so this is going to be really useful here.

So, we can call this object we can use this object str1 and call the method split here and here you can see we are passing a separator which is to be used to actually create this list of terms. So, we are calling str1.split and within parenthesis we are specifying the separator that is common here. So, if I run this in the output you would see that we have got we have got this you know list of these terms a, ab, abc, abcd, abcde.

(Refer Slide Time: 02:14)



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 viewing output. The notebook contains four code cells:

```
In [50]: # By default, \n is the end-of-line (EOL) marker
f13 = open(path)
for line in f13:
    print(line)

In [51]: # closed method returns True if the file is closed
f13.closed

Out[51]: False

In [52]: f13.read()

Out[52]: ''

In [53]: # EOL free list of lines
[line.rstrip() for line in open(path)]

Out[53]: []
```

So, if you really look at the output that ab, abc and other terms starting spaces appended there. So, separator was use to separate these terms but if you look at the term the space is also another

character and it is being used for those terms here. So, we have methods that can be actually use to strip this wide space, so we can combine this in this particular split method with another method strip.

So, that we are able to trim wide spaces that is coming in our list output the you know list of terms that we produced using a split method. So, for this we can use this list comprehension, so you can see the transformative expression is `x.strip`. So, essentially that is the later operation that we would be performing first will get the you know list of terms and then will you know strip of the wide space in each of those terms.

So, we are running a loop here and list comprehension for `x` and `str1.split` and again this is the same method we are calling here. Now this method for each you know `x` in this particular output in this particular output of this `str1.split` that means the list of terms that we have got into `l1`. So, in a loop will be done and for each of these terms we would be calling a strip here, so we would be getting rid of any wide space that is there.

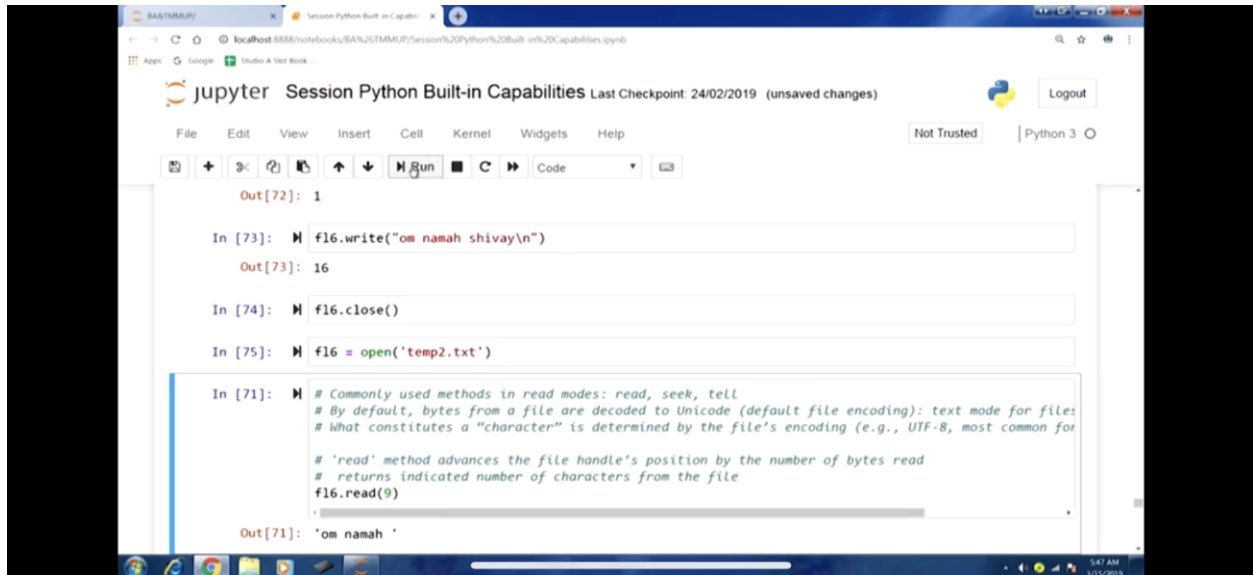
So, if I run this and then output you can have a look and you can compare the output to `l1` and output to `l3` and you can see that wide spaces that were there in those terms those are now gone. Let us move forward, now next aspect is about concatenation of a string with delimiters. So, sometimes if you want to you know if you want to create a text where we would like to you know append the terms using certain delimiters.

So, that can be done, so in this case we are taking example where we are using double colon as a delimiter here. So, first approach could be that we can first segregate each term and then when then we can use the `+` operator for concatenation. So, let us take this approach, so here terms is something that is a list of a string values that we have already created. So, now we are going to segregate terms using you know this particular you know left hand side this particular notation here first, second, third, fourth, fifth.

So, if I run this we will have each of those terms in these objects first, second, third, fourth and fifth. And then we can use the `+` operator to obtain these terms and we are also using delimiter

here. So, first + and then delimiter double colon then second. So, in this fashion if I run the output would be something like this you can see a double colon ab then abc double colon abcd in this fashion.

(Refer Slide Time: 07:25)

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 cell contains the following Python code:

```
Out[72]: 1

In [73]: f16.write("om namah shivay\n")
Out[73]: 16

In [74]: f16.close()

In [75]: f16 = open('temp2.txt')

In [71]: # Commonly used methods in read modes: read, seek, tell
# By default, bytes from a file are decoded to Unicode (default file encoding): text mode for file:
# What constitutes a "character" is determined by the file's encoding (e.g., UTF-8, most common for
# 'read' method advances the file handle's position by the number of bytes read
# returns indicated number of characters from the file
f16.read(9)

Out[71]: 'om namah '
```

So, concatenation of a strings and delimiters can be performed in this particular fashion. Now there is another approach second approach that we can use this involves using calling join method. So, in this case you can see that we are you know calling this join method on this delimiter object. So, double colon within single course double colon we have this you know this is string in a sense this is the delimiter.

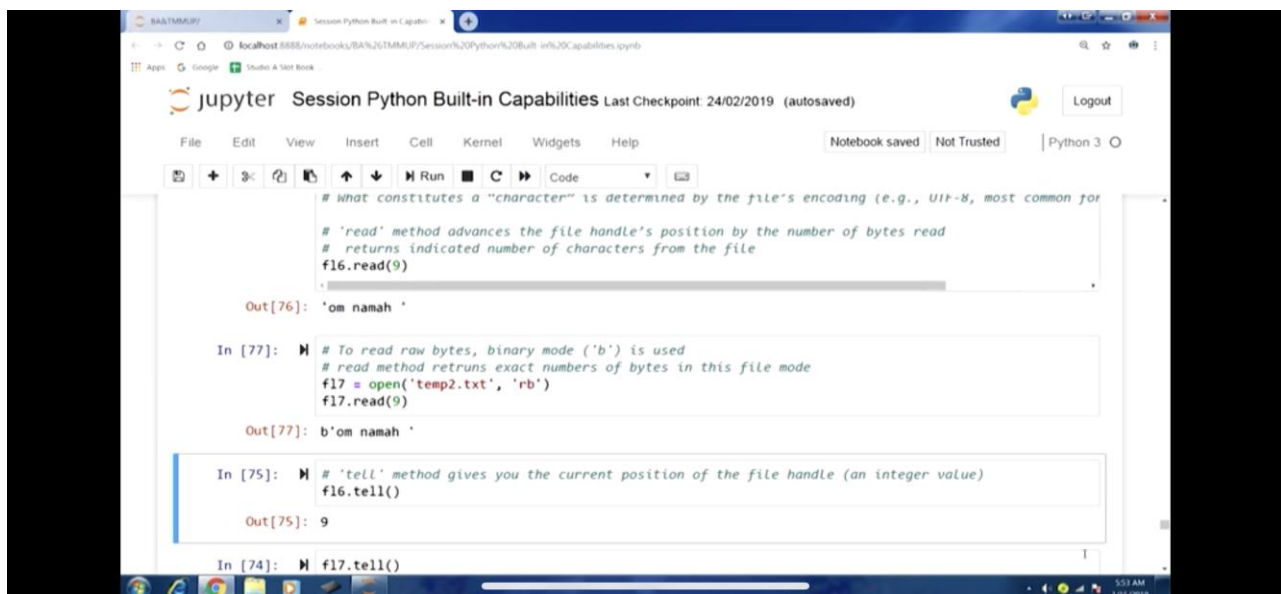
And we are calling join method and in the join method we are passing on terms this list, so we would like to join all the terms that are present in this list of terms using this particular you know string. So, if I run this you can see that effectively the outcome is same. So, these are 2 approaches which we can use to actually combine or you know concatenate terms string values with delimiters.

Now another example that will talk about it is checking presence of a term or finding it is index in the string . So, here let us take this example str1 again, so, these are the values string values in this str1. Now if you want to find out whether abcd this particular term is present in this string object or not, so we can use the in keyword, so abcd in str1 and will get true.

Similarly we can if you want to find out the index of you know this particular term abcd in this string str1 object. So, we can call this index method you know string index method that is str1.index and within the parenthesis we can pass on this term abcd. And this will actually give us the index where this particular term is there in the string, so you can see the output is 12.

So, you can really compare this output with the output 217 and you can see that if you count the term a, 3 then ab 4 5 then ,6 then space 7 then abc 10 and then ,11 then space 12. So, after 12 characters, so therefore this is 0 index, so abcd is starting at you know 12 index is 0 you know indexed. So, you can see that this index method is able to give us the index location in that string object for a particular term.

(Refer Slide Time: 12:32)



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 viewing output. The notebook content consists of three code cells. The first cell contains a comment about file encoding and a code snippet using `f16.read(9)`, with the output `Out[76]: 'om namah '`. The second cell contains a comment about binary mode and a code snippet using `open('temp2.txt', 'rb')` and `f17.read(9)`, with the output `Out[77]: b'om namah '`. The third cell contains a comment about the `tell()` method and a code snippet using `f16.tell()`, with the output `Out[75]: 9`. A fourth code snippet `In [74]: f17.tell()` is visible at the bottom but has no output yet. The status bar at the bottom indicates the time is 5:51 AM on 3/15/2019.

Now if I want to find out something like abcdef in the index method if I run this you will see that because this particular string was not present. So, you can see that we have run into this error value errors of the string not found. However you know we have another method fine method which can also achieve the same thing. But in case substring is not present it would not produce an error instead it will return a value -1.

So, let us use this method fine method, so the again same example str1.find and within the

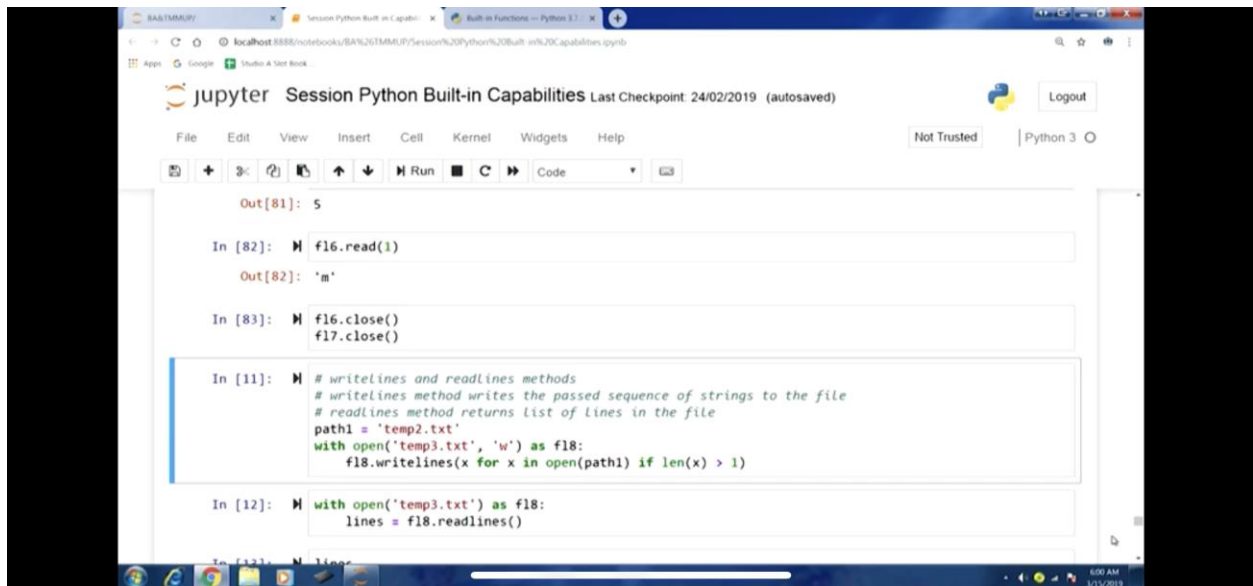
parenthesis we have passing this particular term within single code abcd. If I run this you can see the same output 12 is the index location for this term in this string 1. And if I again call abcdef this is the term is not present in the str1 object, so if I run this I will get a - value.

Because that is to indicate that this particular substring has not been found in the str1 object, so these are the 2 methods index and find which can be use to find the index location of a term in a string. Now let us move to other aspects of you know string and text passing, so next one is about the counting the occurrences of a term and replacing with the substitute. So, let us first discuss the count method here, so for a string object str1 you can call this count method here within parenthesis abc.

If I run this you know in the string if you compare the output you can see the frequency as count has come out to be 3. So, abc as a term is present thrice, so you can see in the original string that we have. So, in the output 217 you can see abc is coming thrice, so this count method as after we produce the count of occurrences of this particular term abc in the string object. Now if you want to replace this with a substitute, if you want to replace abc with something else then we can use replace method.

So, to demonstrate method we are going to replace with you know empty space. So, we will call str1.replace and you can see we are going to replace the first argument is specifying the (()) (11:27) the term that we want to replace in this case comma.

(Refer Slide Time: 19:35)

A screenshot of a Jupyter Notebook interface. The browser address bar shows a localhost URL. The notebook title is "Session Python Built-in Capabilities" with a last checkpoint of "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, running, and code execution. The code area shows several input cells. The first cell has the output "Out[81]: 5". The second cell contains "In [82]: f16.read(1)" with output "Out[82]: 'm'". The third cell contains "In [83]: f16.close()" and "f17.close()". The fourth cell contains a multi-line comment and code: "# writelines and readlines methods", "# writelines method writes the passed sequence of strings to the file", "# readlines method returns list of lines in the file", "path1 = 'temp2.txt'", "with open('temp3.txt', 'w') as f18:", "f18.writelines(x for x in open(path1) if len(x) > 1)". The fifth cell contains "In [12]: with open('temp3.txt') as f18:", "lines = f18.readlines()". The bottom status bar shows "Python 3" and a "Not Trusted" warning.

And the second argument is specifying the term that is going to substitute the first arguments. So, in this case you know null so empty string, so that we have passed in the second argument.

So, if I run this you would see that str1 the output has changed commas are gone there, so you can see how a replace method can actually be use to transform a string object. Similarly if we want we can replace comma with colon also, so another example str1.replace you know comma first argument, colon second argument. And you can see in the output 225 that you know colon is there in trace of comma.

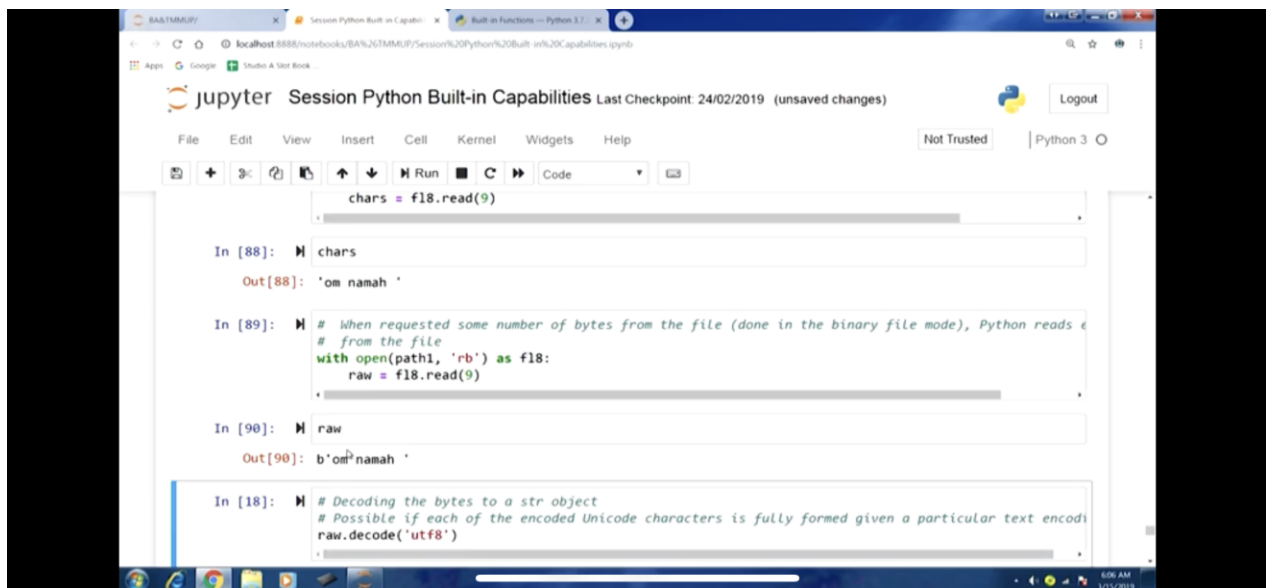
Now let us move forward to another important aspect of string and text processing, so this is about regular expressions, so simple text and string operation can be perform using string methods. However if you know certain complex you know pattern matching and text manipulation is required then probably regular expression or the you know best solution to achieve that.

So, let us talk about a few details about you know regular expression, so for this to use regular expression we would require python's build-in re module. And there are typically 3 types of function that we perform using regular expression is pattern matching, substitution and splitting, so these are the 3 you know operation that we typically perform. Now a regular expressions in

sort is also referred as regex, so a regex describes a pattern to locate in the text.

So, in this particular section while we are discussing regular expression, we would be creating regex objects also. So, that regex object would actually you know the regular expression that we would be defining that would be to describe a pattern, that we would like to match in the you know string object. So, let us talk about how typically we can implement you know the codes related to regular expression.

(Refer Slide Time: 25:09)

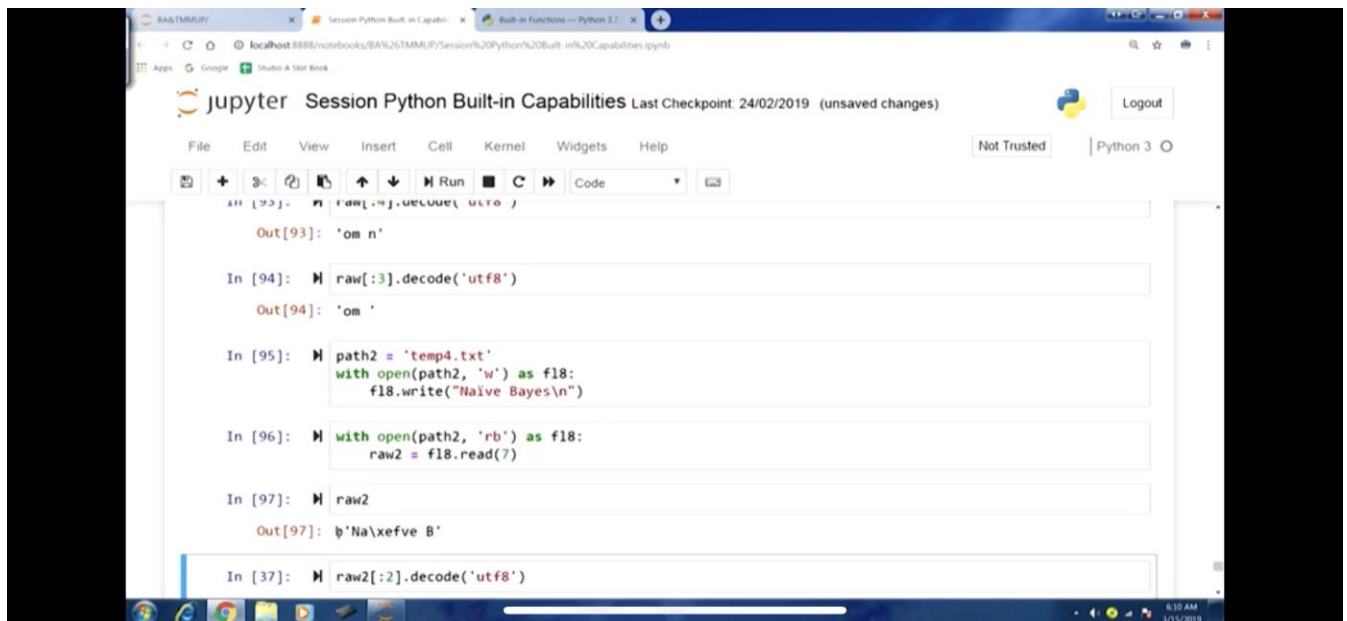
A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/BA%26TMMUJ/Session%20Python%20Built-in%20Capabilities.ipynb'. The Jupyter Notebook title is 'Session Python Built-in Capabilities' with a 'Last Checkpoint: 24/02/2019 (unsaved changes)' and a 'Logout' button. 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 contains several code cells. The first cell shows 'chars = f18.read(9)'. The second cell, labeled 'In [88]:', shows 'chars' and the output 'Out[88]: 'om namah ' '. The third cell, labeled 'In [89]:', shows a comment about binary file mode and the code 'with open(path1, 'rb') as f18: raw = f18.read(9)'. The fourth cell, labeled 'In [90]:', shows 'raw' and the output 'Out[90]: b'om namah ' '. The fifth cell, labeled 'In [18]:', shows a comment about decoding bytes to a string object and the code 'raw.decode('utf8')'. The bottom status bar shows '6:06 AM 1/13/2019'.

So, 1 approach could be first approach could be this is suitable when we are you know just processing 1 or 2 text or a strings. Then you know we do not need to create a regex object, so for a given line of code which involves you know some regex, some pattern we can just process the text. And so internally you know the that regex that we would might be passing in a particular argument that would be compare internally and the text would be you know process there.

So, if once in a while we require to apply regular expression we can just go with this approach, the another approach is which is suitable when we are processing many text or a string. Then we can use the same regex so for that we would like to compile the regex and reuse this compile regex as a regex object. So, will be calling you know regex is specific methods and you know will be applying and processing text and string.

So, let us take few examples, so first example that we are taking here is now before we move ahead and take example let us first import this package re, so import re. So, then let us talk about this example, now first thing that will take split a string having different types of wide space characters. So, these wide space character could be tabs, spaces and new lines so for that how do we you know split such a string.

(Refer Slide Time: 29:19)



```

In [93]: raw[0].decode('utf8')
Out[93]: 'om n'

In [94]: raw[:3].decode('utf8')
Out[94]: 'om '

In [95]: path2 = 'temp4.txt'
with open(path2, 'w') as f18:
    f18.write("Naive Bayes\n")

In [96]: with open(path2, 'rb') as f18:
    raw2 = f18.read(7)

In [97]: raw2
Out[97]: b'Na\xefve B'

In [37]: raw2[:2].decode('utf8')

```

So, now here as we saw while we were using these strings built-in methods. If the same kind of you know character being used comma or colon and then it is quite easy to use built-in strings method to actually process the text or a string. However, if the different wide space character are mixed up tie ups in some places tabs spaces new lines. Then it might be difficult to manipulate using the built-in methods however regular expressions can actually achieve this quite easily.

So, the appropriate regular expression to achieve this could be this as you can see within single codes back slash s+, s is for a space s+ is actually for you know removing different you know space characters that could be there. So, we can use this regex here, so let us first define this string text and you can see if you look at this string we have you know different types of a space wide space characters om namah then slash t then space then shivay then slasht.

So, tab and spaces have been used here, so I can define this text first then you can use this split

function that is available in this package `re.split`. So, in this the first argument I can pass on this regular expression that we have justifying back slash `s+`. So, we would not go into detail of how to write a appropriate regular expression to you know perform pattern matching. For that is out of is scope for this particular course.

However we are just giving you demonstrating you using you know certain regular expression how certain you know processing relative to string and text can be achieved. So, you can have a look at the regular expression at a later time, so this is the back slash `s+` is the regular expression here and we are going to apply this on the text that we have just defined. So, if I run this, this particular regular expression would be applied on the text string that we have a list of string value where we have got each of the terms all the wide spaces all those characters have been stripped off you know.

So, in this fashion we can split a term which is having different types or special characters and however regular expression can be really useful in this context. Now another approach could be as we discussed using `compile` function, so first we can compile this you know regular expression. So, for that we can use this `re.compile` and pass on this regular expression and create this regex object.

Now using this regex object we can call this the `split` method pass on the text string in the first argument, so if I run this will get the same output. Now if I want to apply this regex to multiple such text or strings then this particular process is going to be really useful. Now let us talk about another important method here list of all patterns you know matching a regex. So, how do we process such patterns, so we can use `findall` methods.

So what it does it will return all the matches here. So, for example `regex.findall text`, so the all the you know patterns which are being matched by a regex they are going to be written as using this particular method `findall`. So, if I run this you can see how many you know matches we had you can see this space then tab and then you can see another you know tab addition this space. So, you can see different you know combination of space characters and all of them have been matched with this returned.

Now let us take a slightly bigger text here, so the in the next example what we are trying to achieve is we are trying to identify e-mail at this is from a text block. So, if you can look at this text1 here certain email addresses have been you know of there in this particular text. So, let me first define this text 1 and what would be the appropriate regular expression to actually extract you know these email ids from this text.

So, that is written in the next line of code, so you can see regular expression to identify most email addresses. So, this is going to work for most email addresses and the way they are typically you know the way they are typically define. So, you can see pattern and you can see here within single codes r and then within single codes we are defining this you know raw form. And within single codes we have the regular expression that could be use to capture different components of email addresses though we are not in this particular pattern.

We are not trying to you know separate segregate these you know different components of an email address rather than we are trying to identify it as in one go. So, but you can clearly see different components of email addresses and the appropriate you know regular expression in those parts here you can see which will be able to identify. So, essentially it is based on how the email addresses are design, what kind of characters could be allowed in the part before ampersand and the part after ampersand and how they are concatenated using dot you know characters.

So, all those things are being expressed in this regular expression to create a pattern and then will be using that to match in that text. So, let me define this pattern, now will compile this re.compile pattern and then another argument we are setting here flask ignore case. So, we would like to ignore any cases that is that are there while you know compiling this pattern. Now once this we have this regex we can call the findall method and we can apply this on text 1.

So, that we are able to find all the matches all the matches which look like email addresses. So, if I run this on text 1 you can see we have got a list of string values and these string values are nothing but you know email addresses. So, the pattern that the regular expression that we had

define is actually able to extract or find or match all the email addresses from the text here. Now let us move to the next part, now here in this one we can use this search method.

So, search method has different functionality will be covering them you know as we go along here. So, right now we will focus on the functionality where it the search method will return a match object with only these start and end position of the first match of the pattern in this string. So, let us say `regex1` and we can call the search method on `text1`, so if I run this will this search method will written a match object `m1`.

And this will have the start and end of the you know there is a match it will have the start and end of that particular pattern. So, I can use that to actually construct the email address, so in the text 1 the text 1 the string object that we have call this `m1.start`. So, this will give me the you know start of the start index of the start position or index of the email and then colon `m1.end`. So, using the this match object `m1` I would be able to you know get the end position of or end index of a particular email address.

So, if I run this we will get you can see the first email address, so even though we have to write this to actually produce this output even after that we will get only the first one. Now there is another method `match`, so this will check for matches only at this part of the string. So, there the search method that will was processing only the first match, so whatever first match is there it was returning appropriate you know match object and that could be used to construct the output. Now in the `match` method only at this start of the string the matching will happen.

So, check for those pattern matching would actually be perform only at this start of the string. So, we can call like this `regex1` part `match` and on `text1`, so if I run this you do not see anything in the output because if you look at the text that we are using here for text 1. There at the start of the text we did not have email address, so we did not get any output, so we can use different function to actually see that none was found.

So, because at the start of the text we do not have email address, so none is return when we use the `match` method. Now let us talk about the now talk about another you know `regex` method

there is some method. So, what it can do it can find and replace all the occurrences of a pattern with a substitute. So, we can specify a particular substitute that substitute is string that could be actually used to you know replace all the matched patterns in the text.

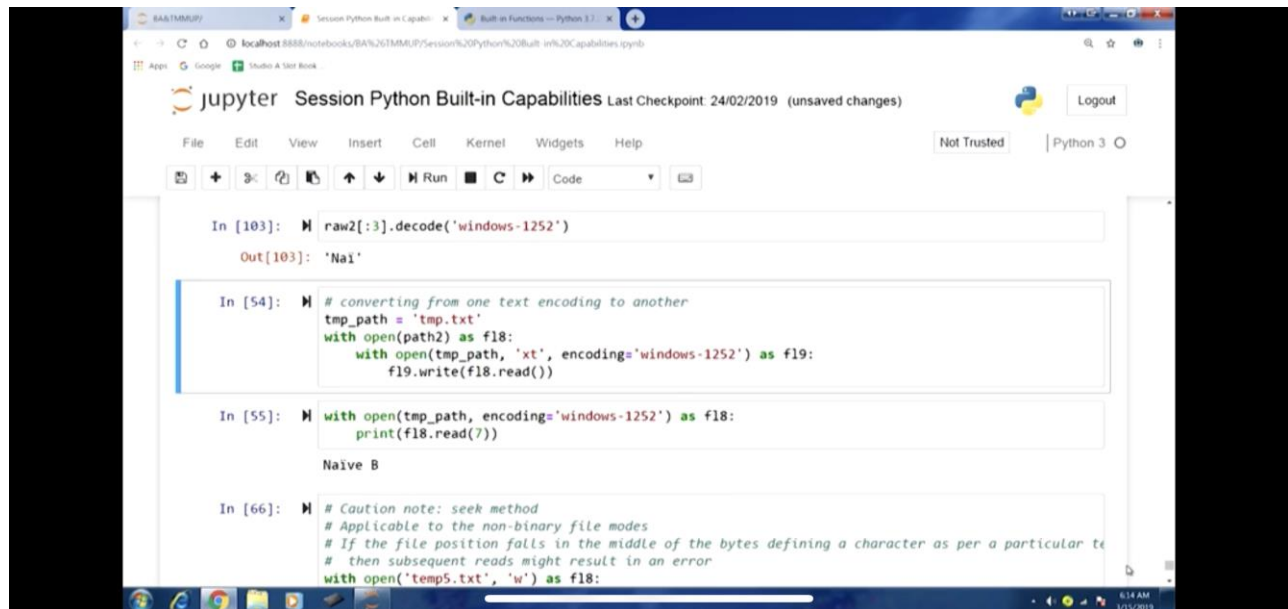
So, let us say `regex1.sub` and our substitute string is being passed in the argument `found` and we are processing `text 1`. So, if given the `regex1` regular expression that we have defined using some method will find all the email addresses in the `text 1`. And replace those email addresses with this particular string `found`. So, if I run this you can see in the output 2, 4, 0 ramesh found mukesh found.

So, you can see wherever we had the email address in the text that has been replaced with the `found`. So, `sub` method will actually substitute any pattern that is matched with the substitute string we can use `print` function to actually get a nicer output here and you can see for different email addresses this substitute replacement has actually happen. Now let us take another example related to the same you know data itself.

So, in this case we will find email address and segment each address into it is 3 component. So, typically as we talked about email addresses have component, so first one is going to be user name then domain name and then domain suffix. So, we would like to find those email addresses and then segment them into their 3 components. So, for that expression is same only thing is now we have use the parenthesis around different parts of the pattern.

So, you can see before ampersand the pattern that we have now it is within parenthesis similarly for the second part just after the ampersand and then the last part just after the dot character the last part is the domain suffix. So, we can use this pattern to perform segregation, so let us compile another `regex` object `regex 2` and then we can use this object as well to perform the similar kind of text processing.

(Refer Slide Time: 32:42)



The screenshot shows a Jupyter Notebook titled "Session Python Built-in Capabilities" with a last checkpoint of 24/02/2019. 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 contains four code cells:

```
In [103]: raw2[:3].decode('windows-1252')
Out[103]: 'Naï'
```

```
In [54]: # converting from one text encoding to another
tmp_path = 'tmp.txt'
with open(path2) as f18:
    with open(tmp_path, 'xt', encoding='windows-1252') as f19:
        f19.write(f18.read())
```

```
In [55]: with open(tmp_path, encoding='windows-1252') as f18:
        print(f18.read(7))
Naïve B
```

```
In [66]: # Caution note: seek method
# Applicable to the non-binary file modes
# If the file position falls in the middle of the bytes defining a character as per a particular te
# then subsequent reads might result in an error
with open('temp5.txt', 'w') as f18:
```

So, again let us use the find all method and to produce all list of tuples having segregate components. So, earlier when we ran this when we call this method we were trying to find all email addresses. Now we will have list of all email addresses now we will have list of you know tuples where each of the components of email addresses would be you know segregate. So, if I run this you can see in the output that we have a list of tuples instead of list of string values.

And in the tuples we have a string values where you know 3 components for each of the email address we can see that the regular expression that we are using it is able it can be used to actually segregate these component. So, now through this these examples you would be able to see the power of regular expression and how pretty complex text manipulations can be perform using them.

Now let us talk about another example, so in this case we are again using some method and you would like to print segregated component. So, some method can also perform this, so this is a additional functionality of some method. So, this is like a symbol component mapping, so here we have back slash 1 that will refer to the first component back slash 2 refers to the second and so forth. So, different symbols and you know different components.

So some method will actually create that kind of mapping and we can use that mapping to print the output. So, I can call a method I can call `regex2.sub` here and you can see I am passing these you know this is string that I want to print here user name, colons, slash 1, slash 2 domain, slash 2 suffix, slash 3 and the processing text1. So, if I run this we will have an output like this where we are able to use this symbol component mapping and able to produce different components of email addresses using sub method.

(Video Ends: 29:50)

So, with this we will like to stop here and will continue our discussion on regular expression in the next lecture, thank you.

Keywords: Concatenation, text processing, string, text manipulation, encoding, expression.