

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

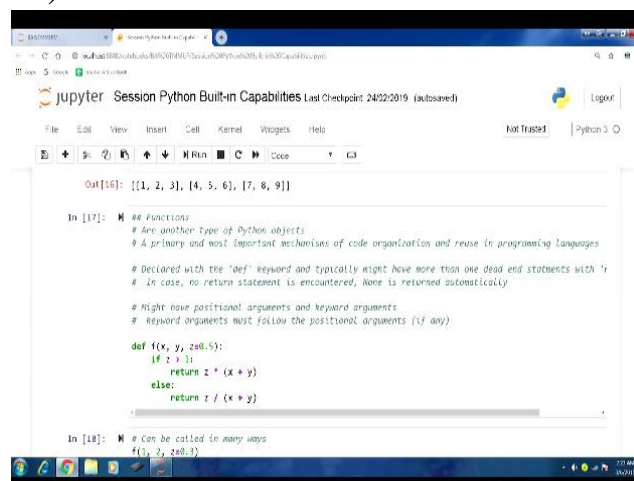
Lecture-15
Built-in Capabilities of Python-VII

Welcome to the course business analytics and text mining modeling using python. So, in previous few lectures we have been covering python language you know mainly in the context how you know, basics and how you know it could be really useful in the analytics context. So, from the analytics perspective we have been covering python. Now in the previous lecture we started our discussion on functions.

So, let us pick up from there. So, we talked about functions. So, functions are just any other type of python objects. However, as we discussed that they are going on the most important mechanisms in terms of code organization and reuse. Because many times a certain piece of code we would like to you know execute again and again for certain computations certain you know, other processing that we require.

So, it is better to have them you know in the functional form, so, we can use these functional objects by functional python object. So, in a sense, we will be able to reuse that part of the code and our code organization will also improve. So, for any programming languages function play an important role. Now, as far as python is concerned you know in terms of defining you know, function objects.

(Refer Slide Time: 01:46)



The screenshot shows a Jupyter Notebook window titled "jupyter Session Python Built-in Capabilities Last Checkpoint: 24/12/2019 (autosaved)". The notebook contains the following code and output:

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

```
In [17]: # Functions
# Are another type of Python objects.
# A primary and most important mechanisms of code organization and reuse in programming languages
# Declared with the 'def' keyword and typically might have more than one line of statements with ':'
# In case, no return statement is encountered, None is returned automatically.
# Might have positional arguments and keyword arguments
# Keyword arguments must follow the positional arguments (if any).

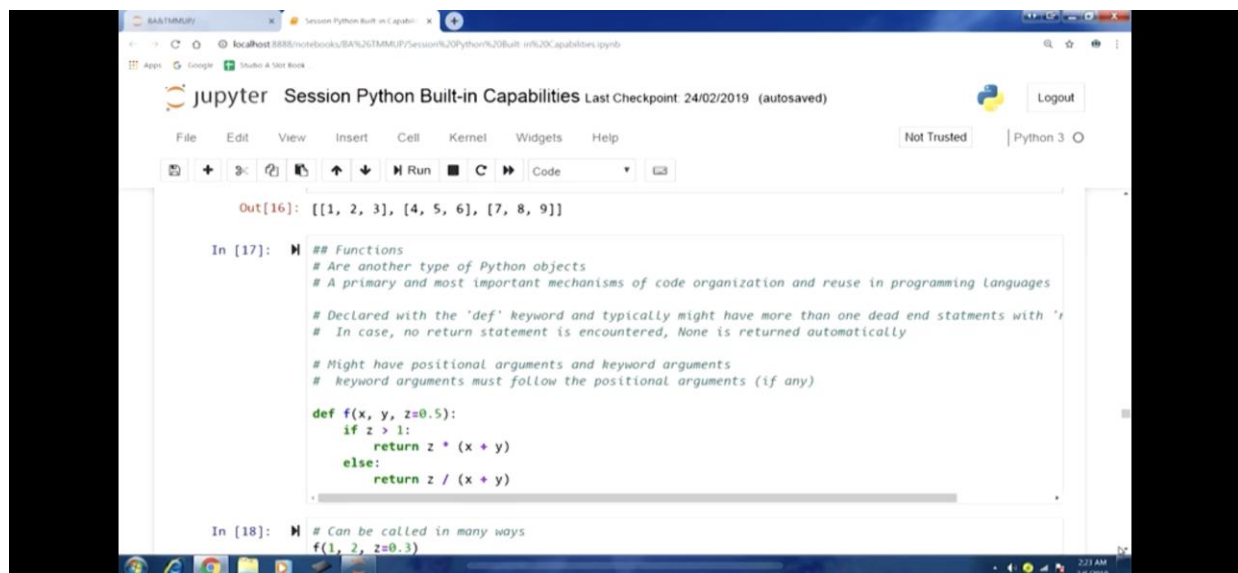
def f(x, y, z=0.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

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

We declare them with you know, using a def keyword and, you know typically them they might have more than 1, you know, a return statement. So, you know, as you can see here that in kind of statements. So, as we have discussed in the previous lecture that your flow of execution you know, can go to any of the multiple paths that could be there, and in each of parts that are possible, we are supposed to have return statement.

So that we are able to end the you know, a code of execution in the functional scope and able to return to the parent scope. So for that we need those returning statements to indicate the same and in case no return a statement is encountered, none is returned automatically. So, this is about how we are supposed to structure our function objects in python.

(Refer Slide Time: 01:56)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". The interface includes a top bar with the Jupyter logo, session title, and a "Logout" button. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with various icons for cell operations is visible. The main area displays a code cell with the following content:

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

In [17]: # Functions
# Are another type of Python objects
# A primary and most important mechanisms of code organization and reuse in programming languages

# Declared with the 'def' keyword and typically might have more than one dead end statements with 'return'
# In case, no return statement is encountered, None is returned automatically

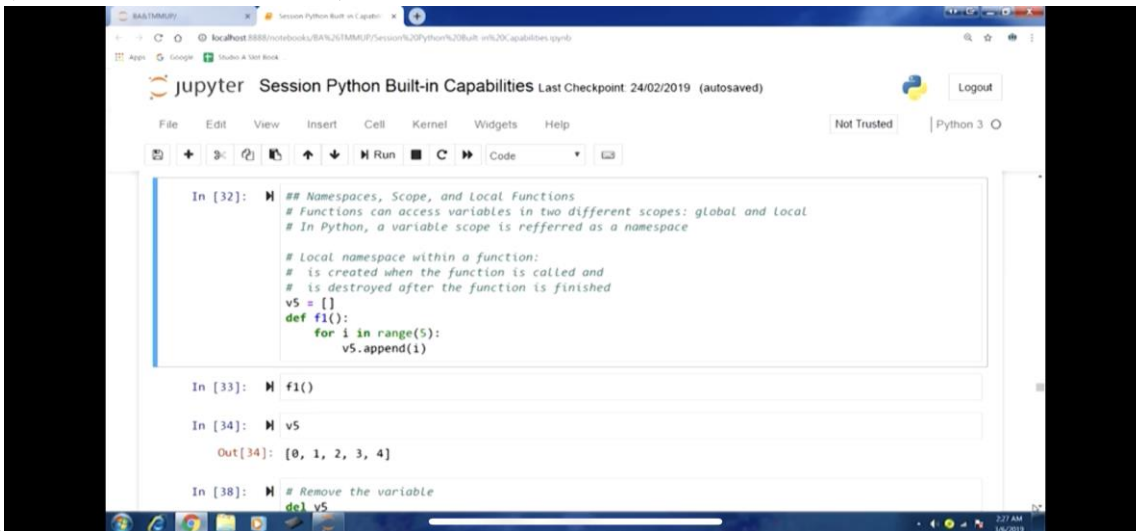
# Might have positional arguments and keyword arguments
# keyword arguments must follow the positional arguments (if any)

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

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

Now, as we have discussed in the previous lecture, they can have positional arguments as well as keyword arguments. And, you know, for obvious reason, keyword arguments must follow the positional arguments, because the positional arguments, they are to be accessed by their positions, however you know keyword argument, they can also be accessed by their names, the keywords, so, therefore, first positional arguments and then, you know, keyword arguments will follow them.

(Refer Slide Time: 05:34)

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 adding, saving, and running code. The code area contains a cell with the following text:

```
In [32]: ## Namespaces, Scope, and Local Functions  
        # Functions can access variables in two different scopes: global and local  
        # In Python, a variable scope is referred as a namespace  
  
        # Local namespace within a function:  
        # is created when the function is called and  
        # is destroyed after the function is finished  
        v5 = []  
        def f1():  
            for i in range(5):  
                v5.append(1)
```

Below the code cell, the execution history shows:

```
In [33]: f1()  
  
In [34]: v5  
Out[34]: [0, 1, 2, 3, 4]  
  
In [38]: # Remove the variable  
        del v5
```

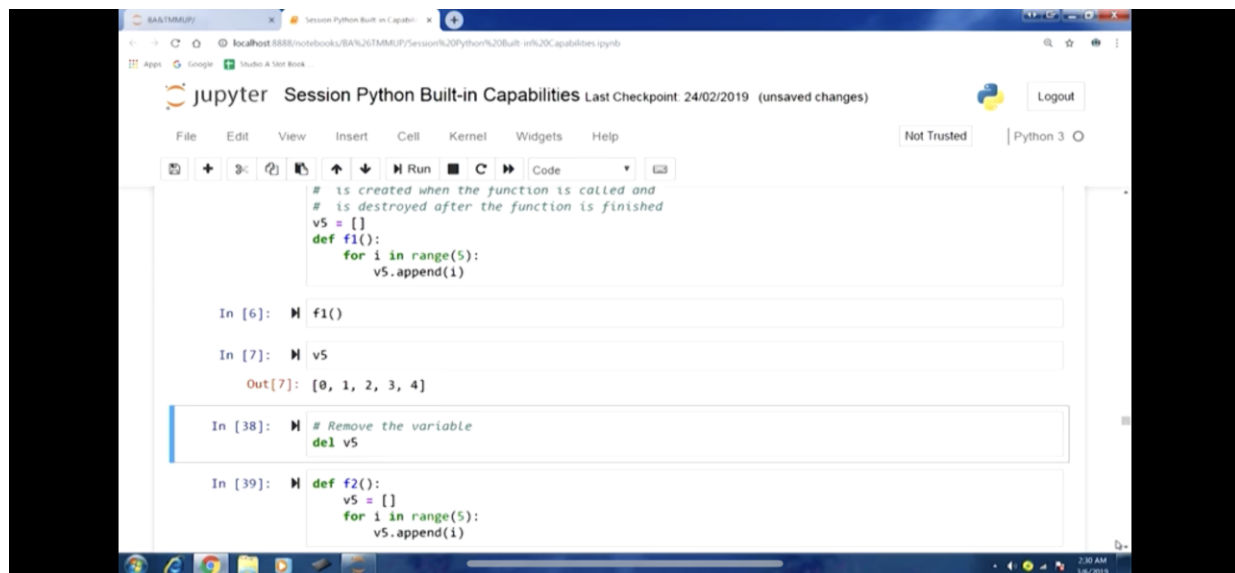
(Video Starts: 03:08)

So, that is the typical structure one example that you can see here. This part of the code, you can see, we have you know, default we are defining a function f and 3 arguments are there x y and z and the first 2 arguments are positional argument, third argument is the keyword argument, where we have given a value also. So, typically, as we have discussed before as well, keyword arguments they are typically used when we would like to have a default value.

So, and the nature of keyword argument also facilitates so different, you know, ways of calling that particular function. So, we will you know go through this, so, in the code block of this functional object, but we have we have conditional logic here, if else blocks here so z greater than 1 then will return this expression and otherwise else will return this expression, z you know, divided by $x+y$ if z rather than 1 then $z*x+y$. Now, we can call, you know, in many ways we can call this function in many ways.

So, first one first example that we're taking here is f first argument is 1, second argument is 2 and the third argument $z=0.3$, remember it the third argument is the keyword argument is also having it you know default value as well. So, if I run this so, you can see, we obtain a value here. And then similarly, second we are calling would be, you know, just be just pass the values.

(Refer Slide Time: 08:21)

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/BAN%20TMUP/Session%20Python%20Built-in%20Capabilities.ipynb'. The 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 running, saving, and other actions. The code cell contains a function definition and its execution. The output shows the function's return value.

```
# is created when the function is called and  
# is destroyed after the function is finished  
v5 = []  
def f1():  
    for i in range(5):  
        v5.append(i)  
  
In [6]: f1()  
  
In [7]: v5  
Out[7]: [0, 1, 2, 3, 4]  
  
In [38]: # Remove the variable  
del v5  
  
In [39]: def f2():  
         v5 = []  
         for i in range(5):  
             v5.append(i)
```

So, f then 3, 4.5, 1.3, so if I run this I will get a value, then another way of calling is that we are just passing values first 2 arguments and the third argument is in a sense optional argument, because we have already indicated the default value in the definition itself. So, if we want, we do not need to specify that. So, therefore we can call this function using just 2 arguments. So, if I run this will get a value.

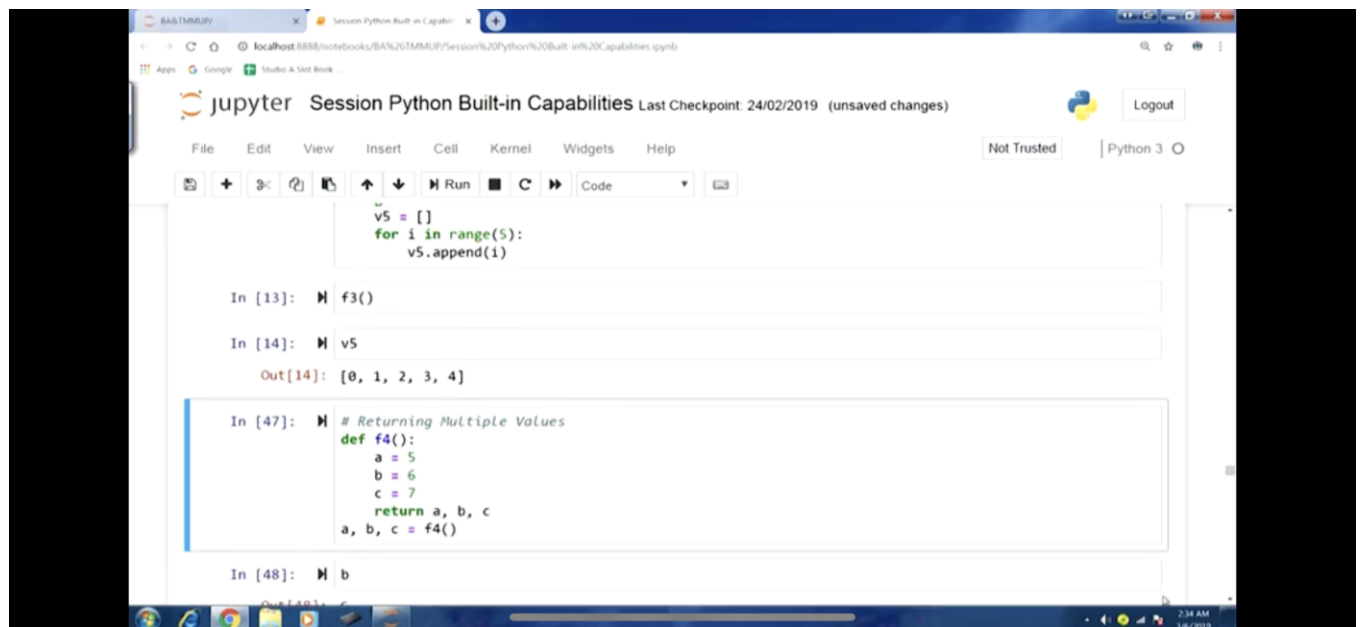
Now, this must we have discussed in the previous lecture, as well. So, now, let us move forward. Now, let us talk about some more aspects about functions, name spaces scope and local functions. So, functions can access variables in 2 different scopes, 1 is global, 1 is local. So, if variable scope, you know, whether it is global or local, it is also referred as a name is space in python terminology.

So, local name is space within a function, it is typically created when a function is called and it is destroyed, after the function is finish when we are in functional environment, when we are executing the code within function, then a local namespace for that functional, that functionality scope is created. And once the call to you know, that function is finished, then that local namespace is destroyed.

So all the variables that would be created inside the functional scope inside that local namespace, they would be destroyed once the execution comes out of the functional scope, the local namespace. So let us understand this through few examples. So here in the first example of what I have done is I have created available outside the functional scope V5 and it is an empty list here.

And then I am defining a function called f1. And in that I have this you know, just a for loop there for I in range 5, and V5 got append i. So we would be appending these elements one by one, as we run the loop into the list empty list that we have created. So let us run this. And what is the function is defined, you can see let us call this function f1. So in this fashion f1 and the parentheses because this function does not require any argument.

(Refer Slide Time: 12:10)

The image is a screenshot of a Jupyter Notebook running in a web browser. The browser's address bar shows a local host URL. The Jupyter interface includes a top bar with the title 'Session Python Built-in Capabilities' and a 'Logout' button. Below this is a menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with icons for running, saving, and other actions is also present. The notebook contains several code cells. The first cell defines an empty list 'v5' and a for loop that appends values from 0 to 4. The second cell calls a function 'f3()'. The third cell prints the variable 'v5', and the output below it shows the list '[0, 1, 2, 3, 4]'. The fourth cell defines a function 'f4()' that returns three values 'a', 'b', and 'c'. The fifth cell calls 'f4()' and assigns the results to 'a', 'b', and 'c'. The bottom of the screen shows a Windows taskbar with the time '2:14 AM' and date '1/6/2019'.

So just empty parentheses here and we have called the function, now we check the value of V5. So, we would see that even though the we have added elements in this p5 list inside the functional scope, but since V5 was no defined in the global namespace, therefore, the values have been added to this variable. And you would see in the output that 0 1 2 3 4, all the elements that we have added inside the functional scope, inside the local namespace of the function.

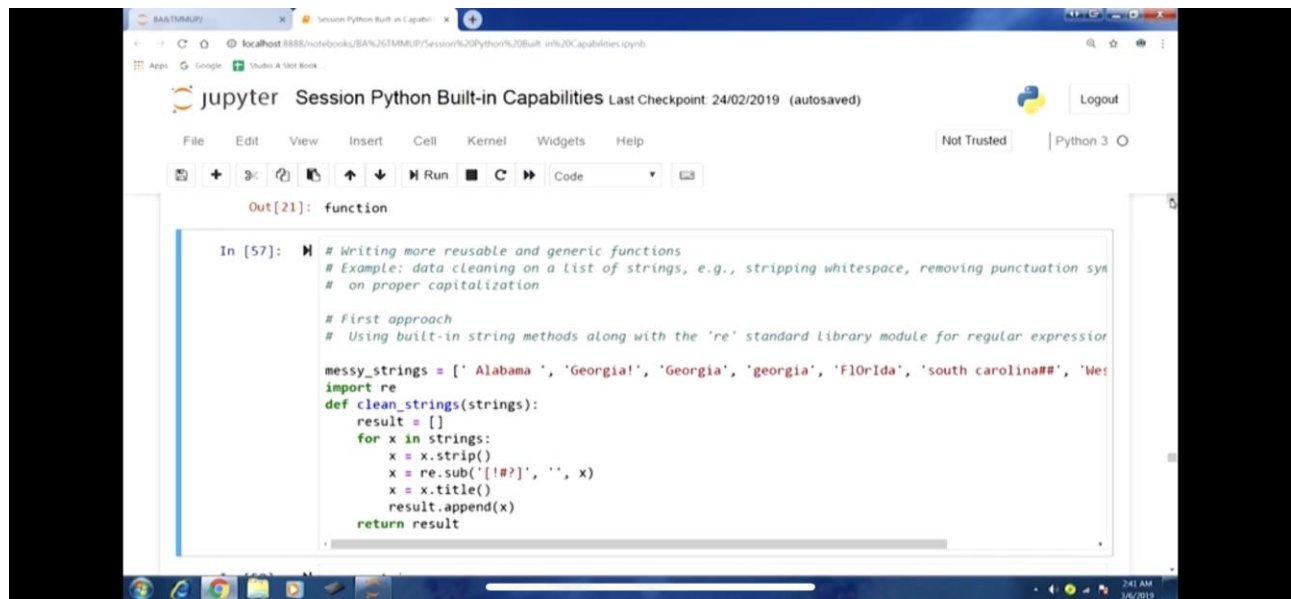
They can be access, they can be seen in the global name is space as well, because the variable was you know, define and created in the global namespace. So, it would not be destroyed once we come out of the functional local name space of the function. So, that is one aspect. Now, let us remove this variable. So, we can use the del command here and del V5. So, let us remove this one.

Now, we are defining another function here to again, you know, understand the name space and the scope aspects. So, here in this function, the V5 variable is defined in the functional scope itself in the local namespace of the function itself. So, def f2, and the first line is the one where we are you know, defining this variable creating this variable V5 empty list, and then we are running the loop for i in range 5.

And you can see them we are appending the element, you know, to this particular sequence this program list. So, in this case, this variable V5 this list object, V5 has been, you know, created in the functional scope or local namespace of the function. So, therefore, once the call to that function is, you know, finished, then this particular variable is going to be destroyed. So, if you try to access this protocol variable, or its elements, we would not be able to do so.

So, let us run this, let us define the function f2, let us call this function and once we are able to call this function and let us try to access variable V5. So, you can see name V5 is not defined. So, because the ones we call the function that local namespace gets destroyed, so along with that, any variable any other objects that are part of that local name space they would also get destroyed.

(Refer Slide Time: 18:40)



So, V5 is destroyed in that sense and therefore, we cannot see the elements of V5, we are not able to access because as far as global namespace is concerned, the global scope is concerned or the parent scope is concerned, this variable does not exist, this variable is not defined. Now, let us understand few more aspects about a scope local and global scope. Now, sometimes, you know, we might like to create a variable within the function.

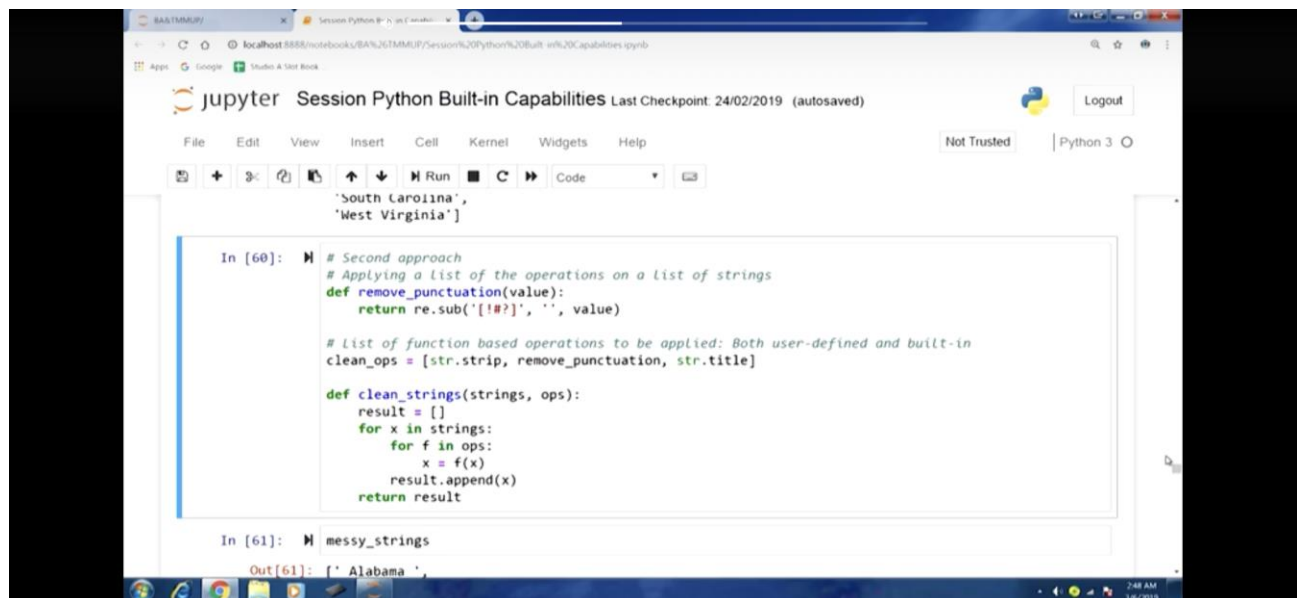
But the global scope. So, that is also possible for that we can use global keyword. So, what we can do is you can see in this example here, we are defining a new function f3 here with no arguments. So, to define a variable inside the body of the function, but with global scope, we need to use global keyword. So, the variable we are defining global V5, once we have you know, define the type of the scope of this variable, then we you know, again, you know, initialize it as an empty list V5 as an empty list.

And then we run the loop. So, because you have used the global keyword or we have you know, in a sense define this variable in the global namespace or the parent scope. So, therefore, if we call this function and in the code block of the function, you know, elements are being added. So, we will be able to access even the you know once the call to the function is finished.

So, if I run this, the function will be defined. Now, if I call this function f3 here, so, call has happened. Now if I try to access this variable V5, you can see 0 1 2 3 4 now, because the function was defined using global keywords, so, therefore, it is in the global part of the global namespace. So, therefore, even though we added the elements inside the local name space of the function, we are able to access this particular element because it is in the global namespace.

So, this is about the you know a local and global scope and the name is space aspects in the python language. Now let us move forward, now let us talk about some you know different aspects about functions. So, some of these things we have touched upon before as well, for example, returning multiple values. So, this aspect also we have discussed before. So, let us do one more exercise here, while we are discussing, you know functions in more detail.

(Refer Slide Time: 25:18)

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/BAN/26TMMUP/Session%20Python%20Built-in%20Capabilities.ipynb'. The notebook title is 'Session Python Built-in Capabilities' with a 'Last Checkpoint: 24/02/2019 (autosaved)' and a 'Logout' button. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for saving, undo, redo, and running code. The code area shows two input cells. The first cell contains a list of strings: ['South Carolina', 'West Virginia']. The second cell, labeled 'In [60]:', contains a Python function definition and a list of operations. The function 'remove_punctuation' uses 're.sub' to remove punctuation. The 'clean_ops' list includes 'str.strip', 'remove_punctuation', and 'str.title'. The 'clean_strings' function iterates over a list of strings and applies the operations in 'clean_ops'. The output cell, labeled 'In [61]:', shows 'messy_strings' and the output 'Out[61]: f' Alabama '.

```
'South Carolina',  
'West Virginia']  
  
In [60]: # Second approach  
# Applying a list of the operations on a list of strings  
def remove_punctuation(value):  
    return re.sub('[!@?]', '', value)  
  
# List of function based operations to be applied: Both user-defined and built-in  
clean_ops = [str.strip, remove_punctuation, str.title]  
  
def clean_strings(strings, ops):  
    result = []  
    for x in strings:  
        for f in ops:  
            x = f(x)  
            result.append(x)  
    return result  
  
In [61]: messy_strings  
Out[61]: f' Alabama '
```

So, I am defining another function, def f4, parentheses, no arguments here. And in the body of the function, I am creating these variables a=5, b=6, c=7, and I am returning you know, a,b,c. So, you can see, when we return in this fashion, in a sense, we are returning a tuple here. So using the tuple data centre that we have in python language, we are able to return all these values in one go.

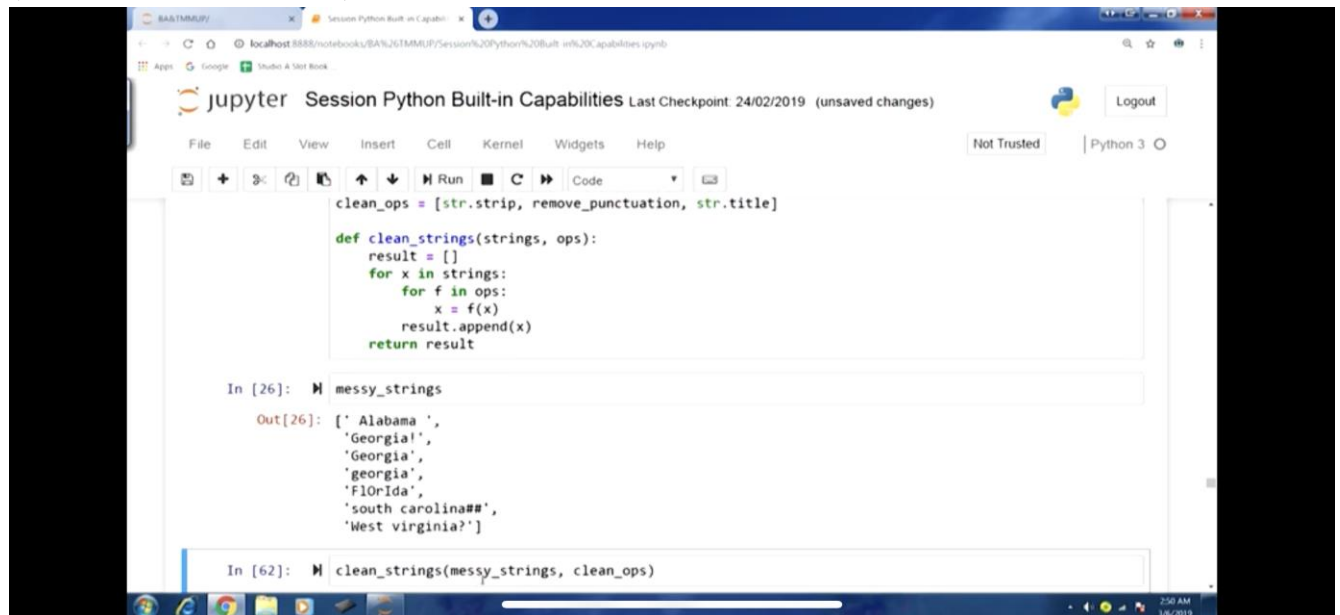
And you can see in the next line of this, you know, code block, I am recording this return value from f4 by calling f4 and storing in the this tuple a b c outside the you know functional name space. So, if I run this, so here you can see first thing that you should notice that a b c that were created inside the you know functionally scope in the local name space of the function, they will be destroyed you know, the a b c that will access they are the new a b c that we have created in the you know, global namespace by you know, recording the return value of the function f4.

So, if I try to access the value individual element of this tuple. So if I try to access if I run this b you can see we got the value 6. So, you can see in this fashion, we can return multiple values, and, you know, then, you know, access them in the, you know, global namespace as well. Now, another way to do the same thing is I can you know, record the variable instead of you know tuple with you know instead of mentioning each element of the tuple, I can just, you know, record this value, store this value in a valuable V6.

So, it is now stored, now if I access this variable V6 you can see we got a tuple here 5 6 7. So, this is another way to perform the same thing. Now, instead of maintaining the element of the tuple you can have the you know tuple variable and whenever you want to access will get the tuple as the output. Now, the same kind of thing, you know, we can also do with the dict data structure.

So, let us do the same exercise again once more. So, again, we are defining another function def f4 and you can see again 3 variables a=5, b=6, c=7. Now, we are returning in the dict format. So, the dict format is about key value pair combination. So, you can see a is the key and then colon a then b colon b, colon c and in this format, we will be returning, you know, the value that we have, you know, created in the functional, you know, scope.

(Refer Slide Time: 26:45)

A screenshot of a Jupyter Notebook interface. The browser address bar shows a local host URL. The Jupyter header indicates 'Session Python Built-in Capabilities' and 'Last Checkpoint: 24/02/2019'. The notebook has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu is a toolbar with icons for file operations and execution. The main area contains a code cell with the following Python code:

```
clean_ops = [str.strip, remove_punctuation, str.title]

def clean_strings(strings, ops):
    result = []
    for x in strings:
        for f in ops:
            x = f(x)
        result.append(x)
    return result
```

Below the code cell is an input cell with 'messy_strings' and an output cell showing a list of strings: ['Alabama ', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda', 'south carolina##', 'West virginia?']. At the bottom, there is an input cell with 'clean_strings(messy_strings, clean_ops)'. The Windows taskbar is visible at the bottom of the screen.

So, then I am recording the return value for by calling this function f4 in a valuable V7. So, if I run this part of the code, and if I access this value, V7 this variable, you can see we have got a dict object here. Now with the values of those elements, you know, value part of you know those key value pairs as well in that dict object, if I look at the type of f4 because we have discussed that functions in python are also objects so if I look at the type of output.

So you can see function is the output. So, here these are function type of these objects is function here. Now, let us move forward. Now, we would like to discuss a particular scenario you know, so, we can all the time as we discussed, you know, writing lines of code, which could be repeated later on. So, we as we discussed, we can use functions to make them you know, to organize that part of the code and use that part of the code.

So, that is one thing what another aspect is writing functional in such a way that they are more generic and more useful. So, the same function, you know, we can write in a fashion that it can be, used in only one particular situation, but the same function, the same lines of code, same code block, we can, you know, reorganize that code of law into more than one functions or in some fashion.

So, that it can be used in more than one scenarios. So, there can be those situation where the same code block can be run in the form of function can be used in very few scenarios, and the same code block can be written in the form of one or more functions and can be used in many scenarios. So, that aspect also, you know, I would like to emphasize here. So, this is about writing more reusable and generic functions.

So one example that we are taking here is data cleaning on a list of strings. So when we talk about data cleaning, it is about you know, stripping, wide spaces, removing punctuation, and proper capitalization, you know consistent, you know, capitalization that we might require typically in the analytics context. So, let us just talk about the first approach, which is not that much usual and generic.

But nevertheless, we are using function here itself, take the second approach, which is slightly more usable and more generic in nature. So, let us start. So, in this first of all, we are using building string methods along with the re standard library module for regular expression. So, in this we have this example, we have this messy strings. So, this is a list, this is a sequence.

This is a list object and having a string elements and strings if you see Alabama it has the wide space before it and the wide space after the characters that are part of Alabama and Georgia, we have extra exclamation marks and you know capitalization is also not consistent, we have Georgia multiple times with different variations and Florida, South Carolina with extra you know has characters there.

So, this is the kind of a string that we have. So, this is the messy string that we have, we would like to clean this particular data, clean this particular you know, sequence of a string values. So for that we would like to because this is something that you know, cleaning or particular string we might require quite often. So, why not write a function to perform this task, because that would be to better code organized and better, you know, reuse of the code.

So, we are importing this module because we will be using certain functions, which are part of this module re. And then we are you know, writing a function called clean_strings. So, what

objective is to deal with such kind of strings to be able to strip them of wide spaces, punctuation symbols and capitalization, we would like to have consistent capitalization.

So, in this code block of the function, we are first defining the result the empty list that is there and because in this empty list, we would like to return the clean destroying once we have performed all our cleaning operation, we would like to return that clean the strings using this results object result variable. So, we are running a loop here. So for x in strings, so, this is strings is you know, list of you know, a string element.

So, what each of the string, we would like to run the loop and we would like to perform certain operation in each of the string element here. So, first thing we are doing is we are calling this function x dot for each string will call this strip function. So, what it will do is it will remove all the you know wide spaces that are part of a particular you know string. So, all the wide spaces trailing or otherwise they would be stripped off, then the next line of code we are calling this function as part of this module V re.sub.

So, this will again you know, substitute if there are any punctuation mark or any characters, they would be substituted. So, you can see in the first argument that we are using here we have you know, these within the you know, this list that we have you know exclamation has question mark, so these are some of the punctuation marks are some special characters. So, we would like to substitute them with the, you know, this character, you know non character here.

So, this is something that we would like to perform to get rid of the punctuation symbols, then next thing is x.title, we are calling this you know, title method here. So, in this all the strings that we have, they will be converted into the title format. So, in a sense, they would be in the consistent with the capitalizes would be consistent, and as per the title format. So, let us define this, how the function is created.

Now, let us look at the messy string that we have now, to run this. So, you can see, you know, this list messy string that we have, and you can see the problem with the string values and the

cleaning that might be required now, will call the function cleaning string, string that we have just defined here, if I call this function can see clean string I am passing this string in this as an argument and you can see the whole string has been cleaned.

I can see Alabama, Georgia, Georgia and Georgia all the string elements that we have only the first letter is capitalized for all of them, and any extra spaces that we had, they are gone, and the any punctuation symbols or special characters that we had, they are also gone. So, in this first approach, but we had done in we had we have defined a function and in the code of that function, we have you know, done certain number of operations in one go in a you know, for a particular list of strings.

So, this is one approach, now, can we make, can we write a function in slightly more reusable, and slightly, you know, more organized way, so, that it is more generic. So, it could be, you know, used in multiple scenarios. So, let us, you know, focus on the second approach. So, what we are doing here is, what we are trying to achieve here is here will apply a list of the operations on list of strings.

So, instead of, you know, in the first approach, in the function, we just kept applying operations, here, we are trying to, you know, if further identify another structure, like we have a list of operations that we want to perform on a list of strings. So, in that sense, we will create a list of operation that we would like to do, and you would like to apply that on the list of strings.

So, we will create, you know, 2 different functions here. So, first function is about remove punctuation. So, in this, we are just using that the re.sub function here, and any punctuation mark or a specific character they would be removed by calling this function. Now, we are focusing on the list of function based operations to be applied. So, it could be user defined or built in. So, in this, we have created a list of operations, list of function based operations.

So, you can see first one is string.str.string and then second one is removed punctuation, which we have already defined, and then str.title. So, you know, we would like to perform, you know, these, you know, 3 function based operations and in a sense we upgraded more structure, where

we are listing all the functional operation that we want to perform. Now, what is left in the cleanest strings function.

So, in this function, again we are creating an empty list. And we are running a loop just like before, and now for f in ops. So, now, we have 2 you know, for loops, first for loop is to rate over the you know strings. So, you know, for each string would like to operate something, we would like to perform something, then second loop each operation in the ops, we would like to know, the list of operation we have created.

So, we would like to iterate over that list. And then we are just going to applying those operations you can see, for each element x, we are calling that function fx. So, in a iterative fashion, we are calling each of those functions. And this is more useful, more structures. So this is more scalable in the sand more generic kind of function. In some other scenario, we would like to, we would be required to perform another cleaning function, another cleaning you know operation, then we would just have to add that particular functional object in the clean ops llist.

And you can see the cleanest string function, we would not have to make any change. However, in the first approach that we have the in the clean string functions, we would have to make changes. However in this, we would have to just add that particular function in the list of functions that we would like to form. So this is slightly more usable, slightly more generic way of writing the same code block.

So let us run this. And now let us again have a look at the messy string that we had. Now you can see we are calling the clean string functions. First argument is the messy string. The second argument is clean ops that means list of functions that we would like to apply to clean the string that we have in the first argument.

(Video Ends: 26:52)

So if I run this, the same, you know, result is achieved as you can see. So you can clearly notice the differences in the sense the same function, same code lines, you know, more or less similar thing. But just the way we organize the code you know, the second approach made it more

usable, more generic in that sense. So with that, we would like to stop here, and we will continue our discussion on functions in the next lecture, thank you.

Keywords: Punctuation, Cleaning operation, Block operation, Execution.