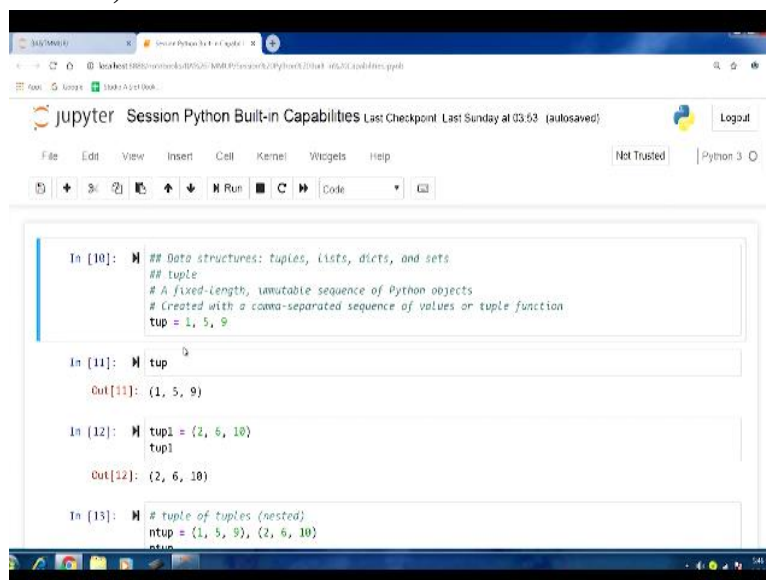


Business Analytics And Text Mining Modeling Using Python
Prof. Gaurav Dixit
Department of Management Studies
Indian Institute of Technology-Roorkee

Lecture-12
Built-in Capabilities of Python-IV

Welcome to the course of business analytics and tax money modeling using python. So, in previous you know lectures we have been discussing about data structures, you know, in the previous lecture and previous few lectures we talked about tuples list and we started a bit on dict.

(Refer Slide Time: 00:45)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". The notebook contains four code cells. The first cell (In [10]) contains comments about data structures and a tuple definition: `tup = 1, 5, 9`. The second cell (In [11]) prints the tuple: `tup`, resulting in `Out[11]: (1, 5, 9)`. The third cell (In [12]) creates a tuple: `tup1 = (2, 6, 10)`, resulting in `Out[12]: (2, 6, 10)`. The fourth cell (In [13]) creates a nested tuple: `ntup = (1, 5, 9), (2, 6, 10)`.

So, let us go through quickly what we have, you know, I understood in previous lecture, and then we will pick up from the point where we stopped in the previous lecture.

(Video Starts: 00:55)

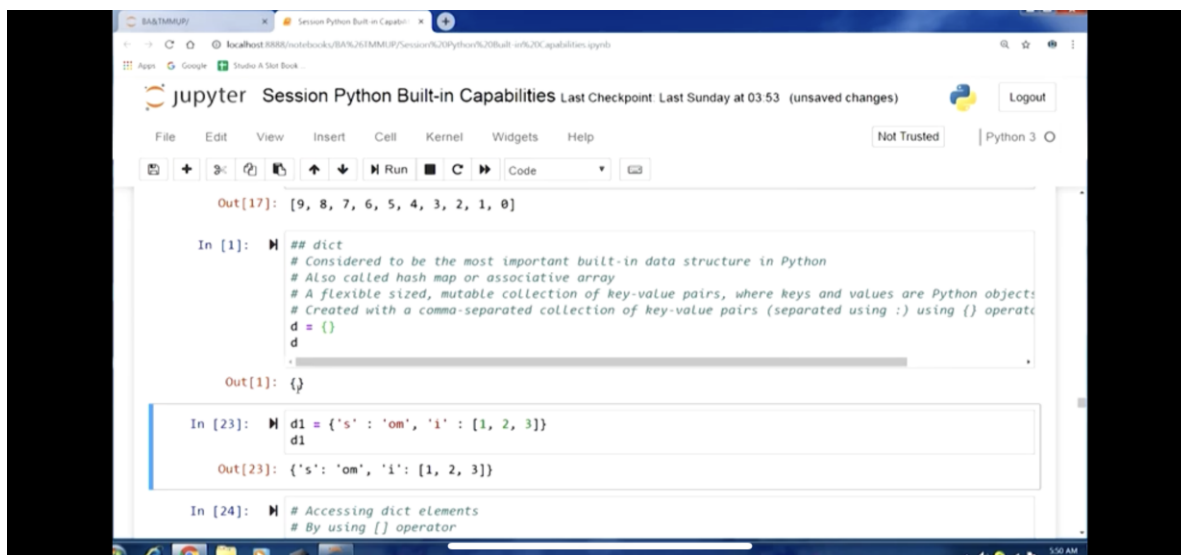
So, we talked about tuples, so, I will just skip this part and move to the next one. So, we talked about tuples and how they are created, how they can be work with, how they can be manipulated, and methods and other things. So, all these things we have talked about in the previous lecture. Then we talked about list and we also talked about what list is about, how it is defined, how to work with list, how to modify, what operations can be used.

All these aspects we have talked about in the you know previous lectures, let us move forward from here slicing also we talked about in the context of list. So, this particular aspect also we covered in the previous lecture, negatives index for slicing also we talked about in the previous lecture. So, I will just skip through this part, because we will be talking about a new you know data structure now.

We talked about some useful sequence functions as well. We talked about zip and zipping reversed and I think we were discussing dict in the last part of our previous lecture. So, let us pick up from there. So, we talked about you know, dict, you know, one of the most important building data structure in python and also called has map or associative added. So, this is a flexible sized, you know immutable collection of key value pairs.

So, this is more of a collection on order kind of collection, where we have key value pairs and with the help of key we can you know access values, and the key are supposed to be unique in nature and typically key are supposed to be immutable in a sense, generally but, overall, the collection is mutable, but the key part of it, you know, is expected to be are generally, it is immutable.

(Refer Slide Time: 04:14)



```
Out[17]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

In [1]: ## dict
        # Considered to be the most important built-in data structure in Python
        # Also called hash map or associative array
        # A flexible sized, mutable collection of key-value pairs, where keys and values are Python objects
        # Created with a comma-separated collection of key-value pairs (separated using :) using {} operator
        d = {}
        d

Out[1]: {}

In [23]: d1 = {'s': 'om', 'i': [1, 2, 3]}
        d1

Out[23]: {'s': 'om', 'i': [1, 2, 3]}

In [24]: # Accessing dict elements
        # By using [] operator
```

So, keys and values both are python objects. So, this is typically dicts are created with a comma separated collection of key value pairs. So, these key and value pairs they are separated by

colon, and the whole key value pairs and the multiple key value pairs that we might have in the collection, they are separated using comma and then you know, we use curly braces to actually indicate this data structure, that it is a dict type.

So, let us start with this. So, here as you can see, in the first example that we have, we are initializing a dict you know object here you can see we are just using the open curly braces just to you know define it you know, empty dict here. So, let us run this and you will see in the output just the curly braces are depicted there. Then, you know, once this is done, let us move to another example.

So, here you can see, we are initializing dict with the full form, where you can see the first key value pair that we have, we have the key that is in the string form that is a string object in single quotes s and then colon and then on that is the value. So, the value is in the strict form and the key is also in the district form. And then the second key value pair second element of the state object that we have is again key value pair.

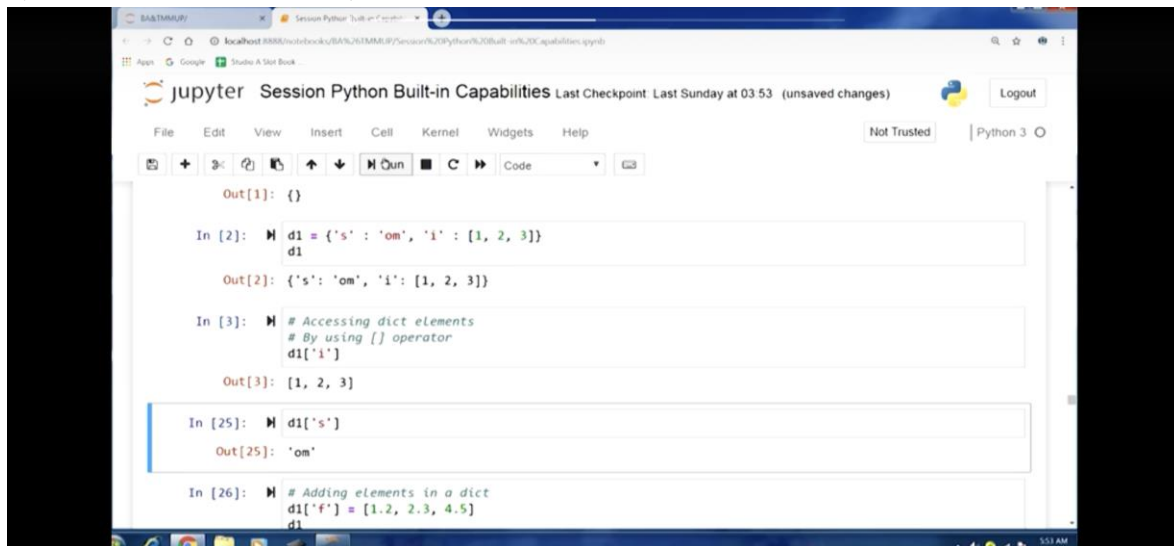
So key is again in the string form I and then coon separating the key with the you know value that is a value is the list object in the brackets, you know, we have 1, 2, 3. So, let us run this and you can see in the output within the curly braces, we have these key value pairs as elements of this particular list. Now in terms of accessing, you know dict elements, we can use brackets operator.

So, you know, just like we have been using other data structures like you know tuples or list. Similarly, this brackets operator can also be used to access elements of a dict. So, here d1 we have already initialized. So, you can see d1 and we are using brackets and there you can see the differences. Now in the tuples and list we are using the indices to access a particular element, here if you see that we are using the key actually to you know, access a particular you know, element.

So, that is why when I say it is you know kind of an order, because we are accessing a particular you know, key value pair using a key. So, therefore, irrespective of the order we

would be able to access it. So, therefore, order is not you know that, you know important in this particular data section, and if you look at tuples and list, it is with the help of a particular index that we access those elements.

(Refer Slide Time: 07:08)

A screenshot of a Jupyter Notebook interface. The title bar says "Jupyter Session Python Built-in Capabilities". The notebook has several cells. The first cell shows an empty dictionary: `Out[1]: {}`. The second cell creates a dictionary `d1` with keys 's' and 'i': `In [2]: d1 = {'s': 'om', 'i': [1, 2, 3]}`, with output `Out[2]: {'s': 'om', 'i': [1, 2, 3]}`. The third cell accesses the value for key 'i': `In [3]: d1['i']`, with output `Out[3]: [1, 2, 3]`. The fourth cell accesses the value for key 's': `In [25]: d1['s']`, with output `Out[25]: 'om'`. The fifth cell adds a new key 'f' to the dictionary: `In [26]: d1['f'] = [1.2, 2.3, 4.5]`. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for running and saving, and a status bar at the bottom showing "Python 3".

So, therefore, elements the order is very much they are in the way they are created and then the way they are accessed, so, this is one difference. So, here let us run this `d1` and within the brackets, we are indicating the key that is `i` and if I run this you can see in the output I get the you know value that is you know a value part of key value pair that is this list the elements of 1, 2, 3.

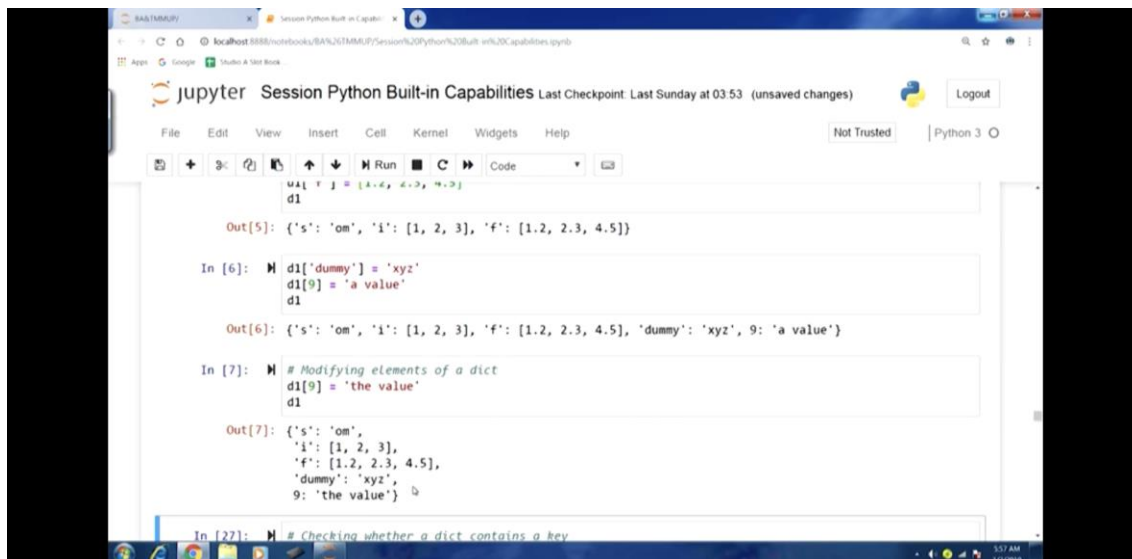
Now, similarly, let us do another example, this time the will use the another key that is `S` and let us run this again we can see we can access the value part that is on a string here. Now, adding elements in a dict, how that can be performed. So, you know here for adding any element, we can again use the brackets operator here and you know we will have to specify the key here. So, you can see `d1` and we are using a new key here in the 's'.

So, this is a string object based key and then we are assigning the other half other part of the you know element here that is a list value, you know based on list So, in the brackets we have 1.2, 2.3, 4.5. So, this is actually you know all the elements of this particular object that is list or floating point number. So, that is why we have also given the key as `f`. So let us execute this perfect example. So, let us run this part.

So, you can see in the output now, we have 3 elements. So, earlier we had you know 2 elements, key value pairs with the s and i as far as string are you for you know, you know, all the elements in the value part being the integer numbers. And then the third, which we have added with the key f that is you know, we are indicating that all the elements and that value part are floating point number.

So, you know, this is one way to add elements in a dict. Now, let us run few more examples. So, we are adding few more elements here for example d1 and the key is dummy string, you know object and then the value is also a string that is x y z, then we are adding another element where we are missing the one and the key is 9. So this time the key is integer value. And then we have the value part of this, you know thing is again a string.

So, let us run this. And in the output you can see 2 additional elements dummy and value x y z and 9 a value have been added. So, in this fashion, we can add elements into a list in to a dict and now let us talk about modifying elements of a dict. So, here you can see this, that you know, in this example, what we are doing is the last element that we added d1 9 with the key and now we are changing a value to d value here. **(Refer Slide Time: 10:18)**



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
Out[5]: {'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5]}
```

```
In [6]: d1['dummy'] = 'xyz'
        d1[9] = 'a value'
        d1
```

```
Out[6]: {'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5], 'dummy': 'xyz', 9: 'a value'}
```

```
In [7]: # Modifying elements of a dict
        d1[9] = 'the value'
        d1
```

```
Out[7]: {'s': 'om',
         'i': [1, 2, 3],
         'f': [1.2, 2.3, 4.5],
         'dummy': 'xyz',
         9: 'the value'}
```

```
In [22]: # Checking whether a dict contains a key
```

And you will see that if we run this part and you can see that simply in the dict the last element 9 and the value part of it has been changed. So in this fashion, using simply the assignment operator, we would be able to do in place replacement of the value part of it. So this can be easily done in this fashion. So we can easily modify elements of a dict. Now, sometimes there might be situations sometimes where we would be required to check whether a dict contains a key.

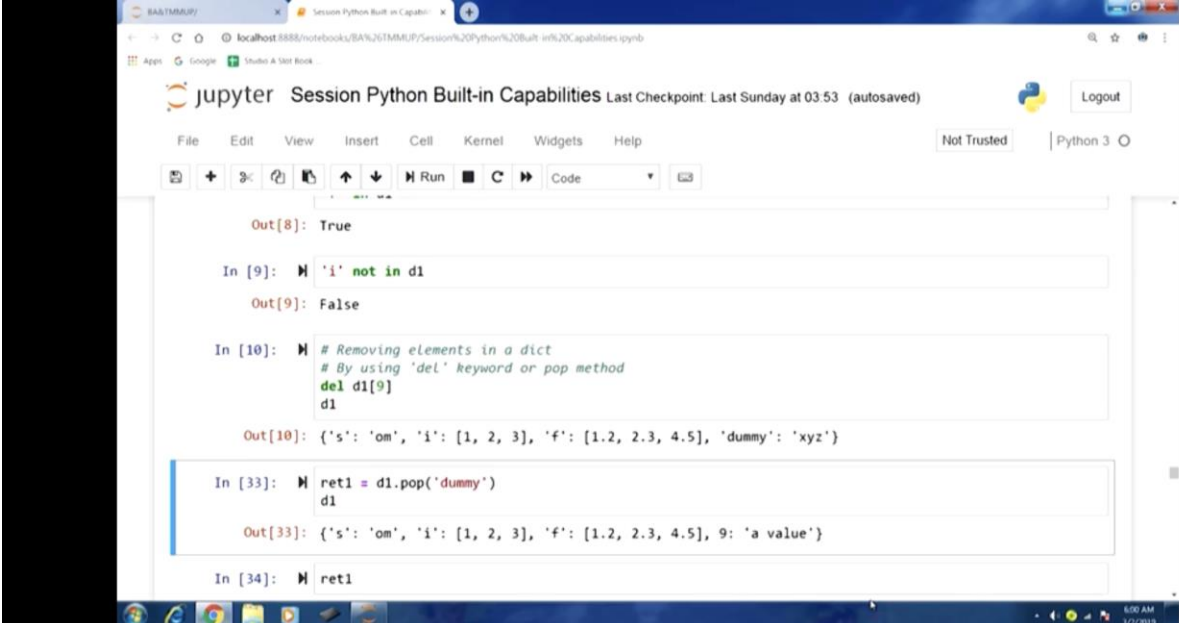
So for this we can use these 2 keywords in and not in, we can use in this fashion, for example, whether this f key is present in the dict object that is d1. So if I run this, I will get true because we know that this is present part of d1. Similarly, we can use the other the key word that is not in. So the example that I am using here is I this is the key not in d1. But we know that this is part of that. So we will get again the answer is false. So we will run this and you can see.

Now, let us move to the next part that is removing elements in a dict. So here we have, you know, 2 options here and 1 is del keyword and then other one is the pop method that we have. So we can either use our del keyword to delete a particular element in a dict, so we will have to just type del, and then we will have to indicate the element. So, here just like we access element using the key part here will have to indicate the element with a key part d1.

And within the brackets will have to type the key there is 9 this case that we want to delete. So if I run this you would see that particular element is out of this particular dict. So, you can see we have s i f and dummy and after that the last element which was 9 that is missing here and other way to remove the element is by using pop method. So, in the next example what we are doing here is that we are using pop method to this object d1 that we have created d1.pop.

And dummy is the key that we are passing as an argument. And so, we would like to delete this you know element this key value pair, where key is dummy. So, we are also recording, you know the return value here in the ret1, if I run this you can see the d1 now is only 3 elements are remaining s i and f and dummy is also gone. If I look at the you know the return value. So, you can see that one.

(Refer Slide Time: 17:00)

A screenshot of a Jupyter Notebook interface. The browser address bar shows 'localhost:8888/notebooks/BA%26TMMUP/Session%20Python%20Built-in%20Capabilities.ipynb'. 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 saving, undo, redo, and running code. The code area shows several cells. The first cell has 'Out[8]: True'. The second cell has 'In [9]: `'i' not in d1`' and 'Out[9]: False'. The third cell has 'In [10]: `# Removing elements in a dict
By using 'del' keyword or pop method
del d1[9]
d1`' and 'Out[10]: `{'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5], 'dummy': 'xyz'}`'. The fourth cell has 'In [33]: `ret1 = d1.pop('dummy')
d1`' and 'Out[33]: `{'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5], 9: 'a value'}`'. The fifth cell has 'In [34]: `ret1`'. The bottom status bar shows '6:00 AM 1/2/2019'.

So the value that was part of the key value pair that is with the key dummy. So the value is recorded in that one x y z. So, in this fashion, we can you know these are 2 ways typically which are used to remove an element in a particular dict object. Now we have 2 methods called keys and values methods. So sometimes you like to know the keys which are part of a particular you know, dict object, or values which are part of a particular, you know, dict object to obtain those results, you know, we can use these 2 methods keys and values methods.

So, let us run this `d1.keys` and with the parentheses, so we will call this function and in the output you can see all the keys that are part of this `d1` object `d1` dict object, they can be seen here `s` `i` and `f`. Similarly, because you know this is in this form, so we can use the list. So in the output if as you can see we use the list for the `d1.keys`, then in output we can get a list of with the key part of the elements only.

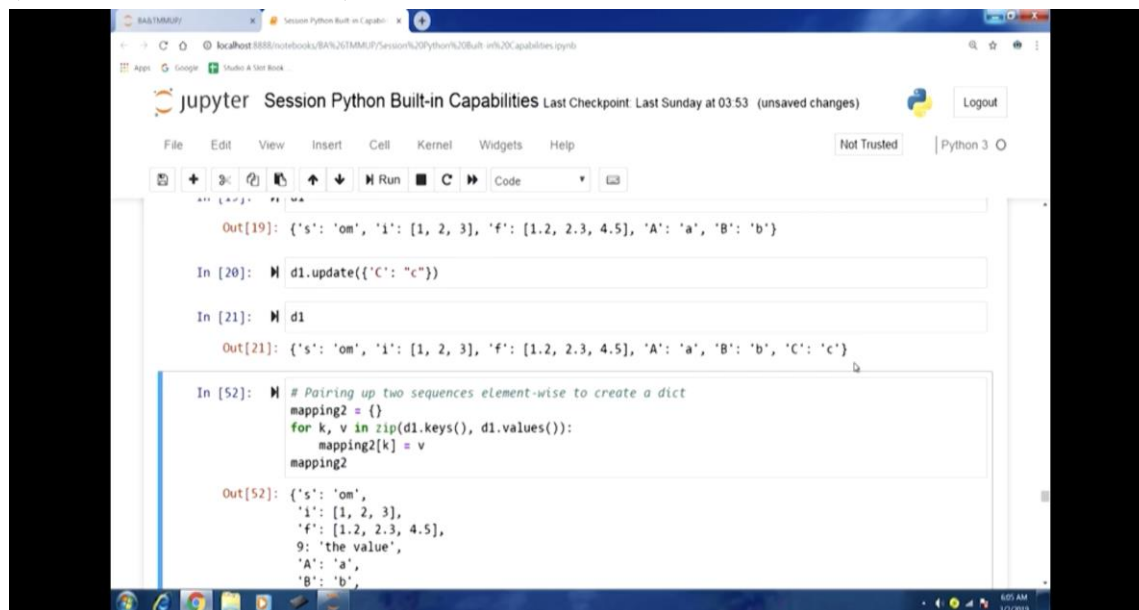
So `s i f` here you can see. Similarly, we can use the values method for this object dict object `d1.values`. If we run this again we will get the output. Here you can see the dict values. And you

can see the output contains only the values part of the dict object. So we just again, we want to have a just output as the values, then we can again, use the list function here. So list d1.values, if we run this, and you would see the output just contains the values part.

Now, let us move on to the next part where sometimes we might be required to much to dicts. So far, this, we can use update method. So what it does is it does in place update or modification. So let us have this example of this d2 object. So in this d2 we have within the curly braces we have you know 2 elements with you know, capital A as the key and the small a as the value than capital V as the key and the small v as the value.

So we have this d2 object. So let us initialize this on this. So this is there, now we can add, what we can do is d1 object that we had created earlier, we can use the update method to merge this. So we can call like this, d1.update, and within the parenthesis we can pass the argument d2 the other dict object and if I run this, try again, you know, access the values of the d1, you can see that apart from the original 3 elements that were part of d1 s i and f, we have 2 additional elements, you know, capital A and capital V.

(Refer Slide Time: 17:00)

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 code, saving, and other actions. The code cell shows the following:

```
Out[19]: {'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5], 'A': 'a', 'B': 'b'}

In [20]: d1.update({'C': 'c'})

In [21]: d1

Out[21]: {'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5], 'A': 'a', 'B': 'b', 'C': 'c'}

In [52]: # Pairing up two sequences element-wise to create a dict
mapping2 = {}
for k, v in zip(d1.keys(), d1.values()):
    mapping2[k] = v
mapping2

Out[52]: {'s': 'om',
'i': [1, 2, 3],
'f': [1.2, 2.3, 4.5],
9: 'the value',
'A': 'a',
'B': 'b',
```

 The output of the last cell is partially visible. The status bar at the bottom indicates the time is 6:05 AM on 3/2/2019.

So in this fashion we can merge 2 dicts. Now further if I want to add another element, so let us take another example. Here, instead of merging 2 already created dicts, I am just updating the parent dict with additional element. So, you can see `d1.update` and within the parenthesis I am passing and other in the form of you know just directly typing the other dict with just one element is capital C and the value is small c so if I can pass this.

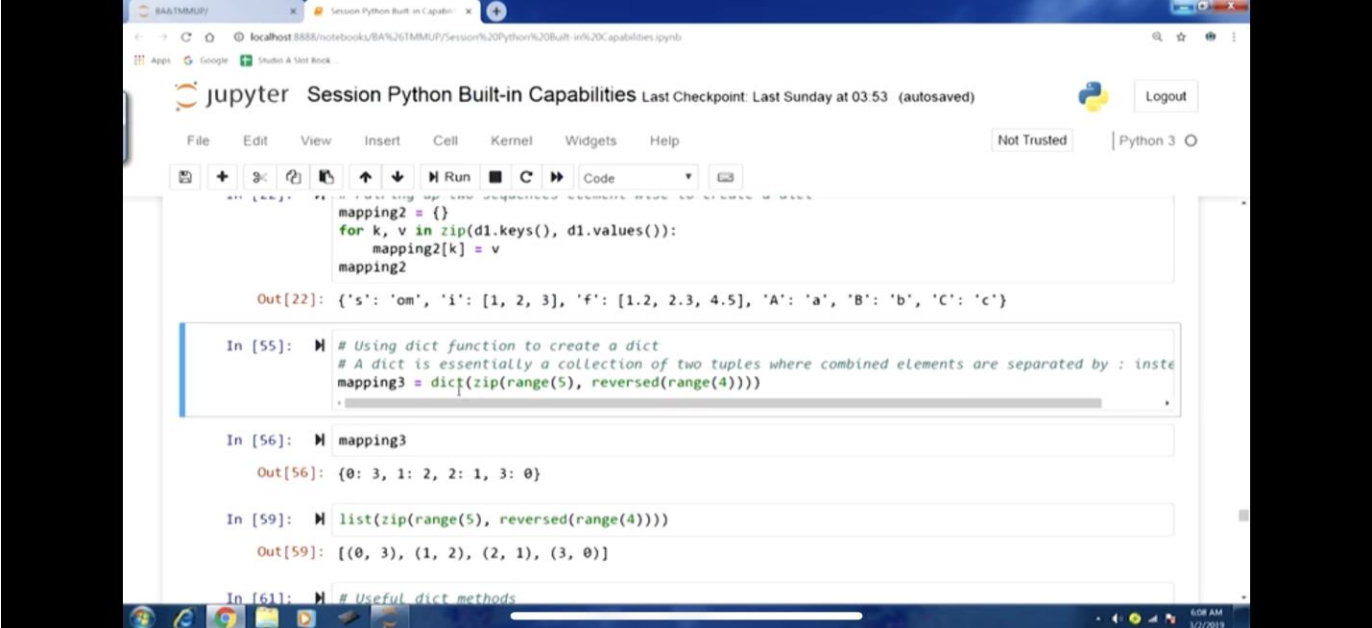
So, let us run this and now in the `d1` if I run this, you can see that another element has been added at the end of this particular dict. Now sometimes, you know when we are getting grading dict, we might be required to pair up 2 sequences element wise to create a dict. So far this is something how this can be done. So, let us focus on this. So, some of the previous you know, methods that we have discussed before we can use them here.

So, similar example we had done before as well. So, here we are you know, first defining a dict empathetic mapping to with just the curly braces as you can see, so the idea is that we have 2 you know, sequences you know and how we can pair them you know element wise. So, we have let us say `d1.keys`, this is 1 you know, 1 sequence, let us say and we have `d1.values` and this is another sequence and how we can pair them up in a sense a dict is created.

So, in a sense from 1 sequence, we are going to take the keys part and from the another sequence we are going to take the values part. So, in this for to generate this kind of thing we are running for loop here and you can see we have `k`, `v` and `v` in `zip` and in the `zip` we are passing you know these 2 sequences `d1.keys`, `d1.values` and the mapping part you can see the mapping 2 in case so far you know `k` is to indicate the you know index of this particular you know dict here.

So, mapping2 `k` and we are passing in the values here `v`. So, when this whole equation for loop runs, then we will have created a mapping2 will combining paring of these 2 sequences. So, let us run this. So, you can see in the output you can see `s` is taken from the key sequence and then the `om` which is a value for this you know first element of this dict and this is coming from the second sequence that is `d1.values`.

(Refer Slide Time: 20:17)



The screenshot shows a Jupyter Notebook window titled "Session Python Built-in Capabilities". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running code, and other functions. The notebook contains several code cells and their outputs:

```
mapping2 = {}
for k, v in zip(d1.keys(), d1.values()):
    mapping2[k] = v
mapping2
```

Out[22]: {'s': 'om', 'i': [1, 2, 3], 'f': [1.2, 2.3, 4.5], 'A': 'a', 'B': 'b', 'C': 'c'}

```
In [55]: # Using dict function to create a dict
# A dict is essentially a collection of two tuples where combined elements are separated by : inste
mapping3 = dict(zip(range(5), reversed(range(4))))
```

```
In [56]: mapping3
```

Out[56]: {0: 3, 1: 2, 2: 1, 3: 0}

```
In [59]: list(zip(range(5), reversed(range(4))))
```

Out[59]: [(0, 3), (1, 2), (2, 1), (3, 0)]

```
In [61]: # Useful dict methods
```

Similarly, the next one i is coming from keys and the value part is coming from the second sequence. Similarly, third, f is coming from the you know first sequence, the value part is coming from the second sequence. So, so on so forth. So, in this fashion given any you know 2 sequence using them we can pair up elements wise and create a dict. So, once you can scan, feed the keys part and another sequence can feed the value part, we can also use dict function to create it.

So, till now we have been mainly using the curly braces, but we have a dict function as well which can be used. So, dict can also be you know defining in few other words also for a dict is essentially a collection of 2 tuples where combined elements are separated by colon instead of comma. So dict can also be in a way understood using this representation a collection of 2 tuples you know 2 tuples where the elements are separated by colon.

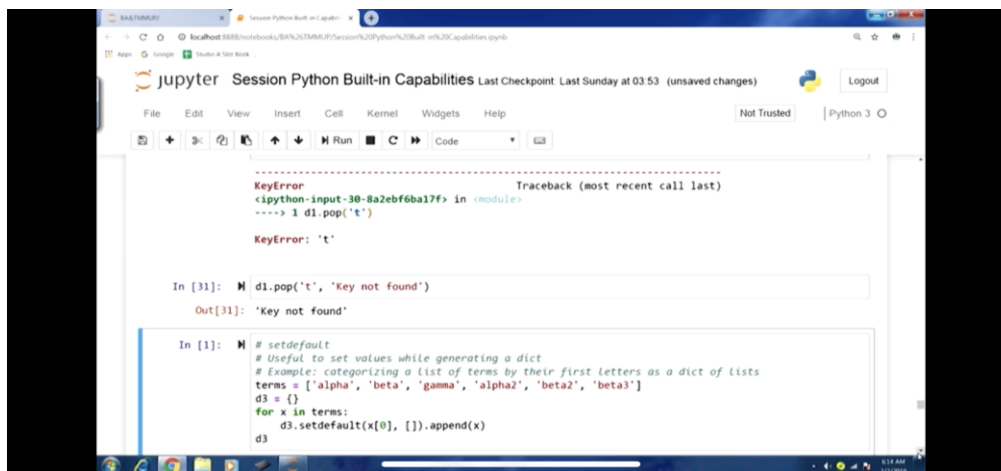
So, let us understand this through an example. So, we are now this time we are creating a you know another dict called mapping3 and we are using dict function this time and you can see we

are you know zipping these 2 sequences range 5 and reverse things 4 here and you can see these 2 are actually tuples. So, in a sense we are you know combining and let us run this. So, you can see the key part of this is you know coming from here.

So, you can see 0 here first key value pair, first element of this dict is 0:3. Now this can be considered as a tuple where you can see 0 and 3 are separated by colon instead of comma. So, in a sense this is tuples, this is followed by another tuple which is 1:2 so 1 is key 2 is value and they are separated by colon. So in tuple they would have been separated by comma. So, this could be another tuple, so if you really look at it, you know the dict this is what we meant.

So they can be considered a collection of you know, tuples. Well, the elements of separated by colon instead of comma. So 0 3 1 tuple and 1 2 another tuple then 1 3 another tuple then 2 1 another tuple and 3 0 another tuple. So in that sense. Now , if we just, you know, whatever we had used whatever we had passed on to this dict function. If we pass on the same thing to a list function you can see here.

(Refer Slide Time: 24:52)

A screenshot of a Jupyter Notebook interface. The top bar shows 'Jupyter Session Python Built-in Capabilities' and 'Last Checkpoint: Last Sunday at 03:53 (unsaved changes)'. 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 running, saving, and other actions. The main area shows a code cell with a traceback for a 'KeyError: 't''. The traceback shows the error occurred in a cell with the code 'd1.pop('t')'. Below the traceback, the input and output of the cell are shown: 'In [31]: d1.pop('t', 'Key not found')' and 'Out[31]: 'Key not found''. Below this, another code cell is shown with the code: 'In [1]: # defaultdict\n# Useful to set values while generating a dict\n# Example: categorizing a list of terms by their first letters as a dict of lists\nterms = ['alpha', 'beta', 'gamma', 'alpha2', 'beta2', 'beta3']\nd3 = {}\nfor x in terms:\nd3.setdefault(x[0], []).append(x)\nd3'. The output of this cell is not visible.

And in the list function, you can see an output we have pass first sequence zip range 5 and the is another sequence is reversed range 4, you can see this is a list of tuples were 0 3 1 2. So if you compare the mapping3, the dict object that we have created earlier and this one the list object that we have created just now by using the same sequences. And you can see that you know, in a sense, they are like tuples collection of tuples.

Let us move on. Now we will talk about some of the useful dict methods. For example, get and pop. So here, let us start with the get. So d1 is the one you know dict object that we have already created. So if we run this d1.get, and we are using this key s, so we will be able to access the value of it, so d1.get as an om and d1.get the And this is so because we know that t is not part of this d1 object.

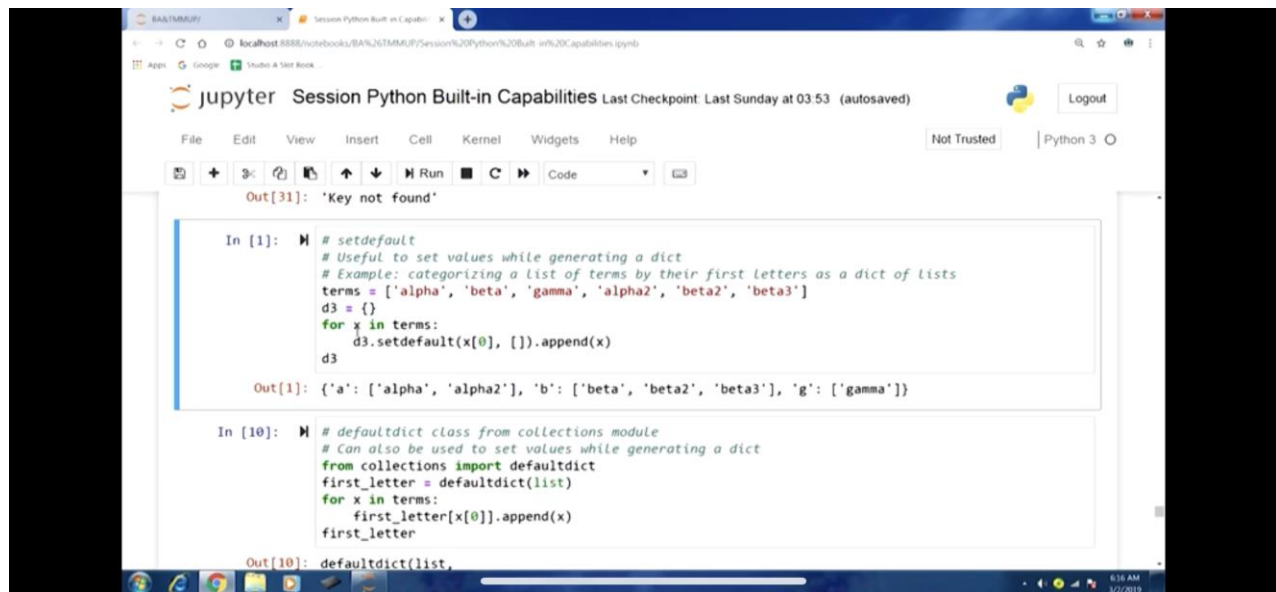
So if I run this, you can see that nothing is returned here. Because that object, that particular element is not part of this d1 object. Now, when we run this particular line of code, we did not get anything in return because it is the you know, this non was return. So we can change this default return value. So for this the same function get we can use here, in the next line of code, what we are doing is d1.get.

And in the parentheses we are passing a second argument as well, where it says key not found. So instead of the default are returned value that is not we would be if that particular, you know, element is not part of that object will be printing message 1key not found. So if I run this, you can see you will get you know, if I run this here, you can see we get this key not found message.

Now, if I, you know, run and other, you know, matter d1.pop. So here again we do not have any element with the key t in the d1 object. So again I will get error, so if I run d1.pop t, so we will get error and so what we can do is again we can, you know, change this, you know, default rate and value. So here, you can say d1.pop t and the second argument that I am passing on to the pop you know pop method that is key not found.

So, because you know t1 is not part of the object now will get this message as the printed outputs, if I run this you can see key not found. Now let us move on to the one important function there is another method that is set default. So, this particular function is quite useful to set values when we are generating a dict. So for to demonstrate this will take this example where we are supposed to categorize a list of terms by their first letters as a dict top list.

(Refer Slide Time: 27:01)



```
Out[31]: 'Key not found'
```

```
In [1]: # defaultdict
# Useful to set values while generating a dict
# Example: categorizing a list of terms by their first letters as a dict of lists
terms = ['alpha', 'beta', 'gamma', 'alpha2', 'beta2', 'beta3']
d3 = {}
for x in terms:
    d3.setdefault(x[0], []).append(x)
d3

Out[1]: {'a': ['alpha', 'alpha2'], 'b': ['beta', 'beta2', 'beta3'], 'g': ['gamma']}
```

```
In [10]: # defaultdict class from collections module
# Can also be used to set values while generating a dict
from collections import defaultdict
first_letter = defaultdict(list)
for x in terms:
    first_letter[x[0]].append(x)
first_letter

Out[10]: defaultdict(list,
```

So we might have a number of terms and you know those terms would be you know, essentially English word. So, some of those words will start with the you know, same first later, so, we would like to club them together. So, in this fashion, we would like to create a you know a dict of list in a sense, where all the elements which are shading the same first letter they let us consider that as a list.

So, similarly, will have given a set of terms will have a number of such list where all the elements of start with the same letter. Now that first letter can be treated as a key. So , if that is the kind of requirement that we have, so how we can use set default method to perform this. So one thing is to perform this, we can also follow the conditional logic we can use if else statements to perform this in a very easily and all support, but that would require writing a more number of lines of code, and applied probably slightly more computational time.

So instead what we can do is, we can use the set default method to set this particular kind of, you know, key value combination while generating this state. So let us say we have terms like this alpha beta gamma alpha2 beta2 beta3. So here you can see that alpha and alpha 2, they can be club together, then similarly beta and beta 2 and beat 3, they can be club together, and then gamma you know, separately.

So, let us define, you know, empathetic d3 with the, you know, just the curly braces, and we can run this for loop here for x in terms. So for every time we will run this loop, and will, you know, d3 object, and we will call this method set default. So, what we can do in the argument side is that the x0 which is actually the first one to be the first letter in that, you know, any particular time and so that will take as the key part, and the second argument is going to be you know, a list.

So, that is our default function will put all the terms which are starting with this, you know first letter into that list form, and keep on appending to you know, that list as we iterate through all the elements of the terms. So, if I run this, this is what I get you know, a you know, and then the value part is a list with alpha alpha2 to you know elements, similarly b and the value part is beta beta2 beta 3, so, in this fashion.

So, this set default is quite useful when we are trying to generate a dict. So, similarly, there are other ways to perform the same thing. So, this is something that will discuss in the next lecture.

(Video Ends: 28:18)

So at this point would like to stop, thank you.

Keywords: Swapping, Concatenation, Scrapping, Slicing, tuple, dummy