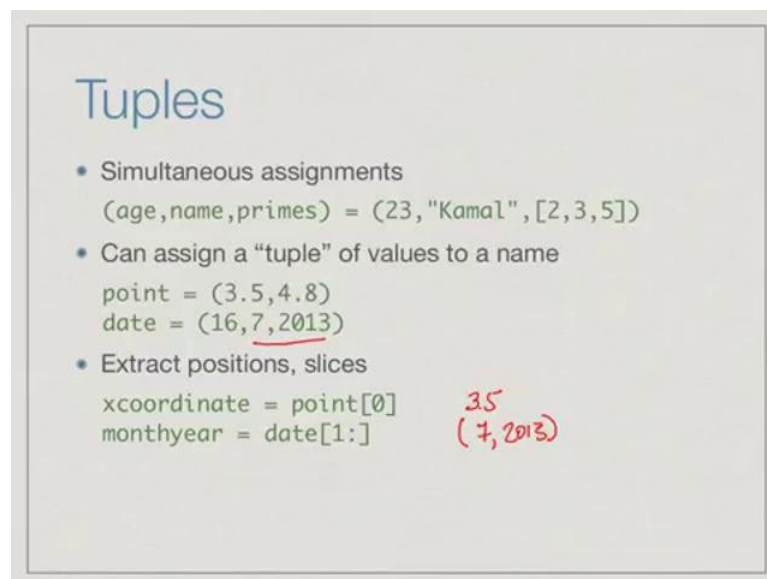**Programming, Data Structures and Algorithms in Python**
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Week - 04
Lecture - 05
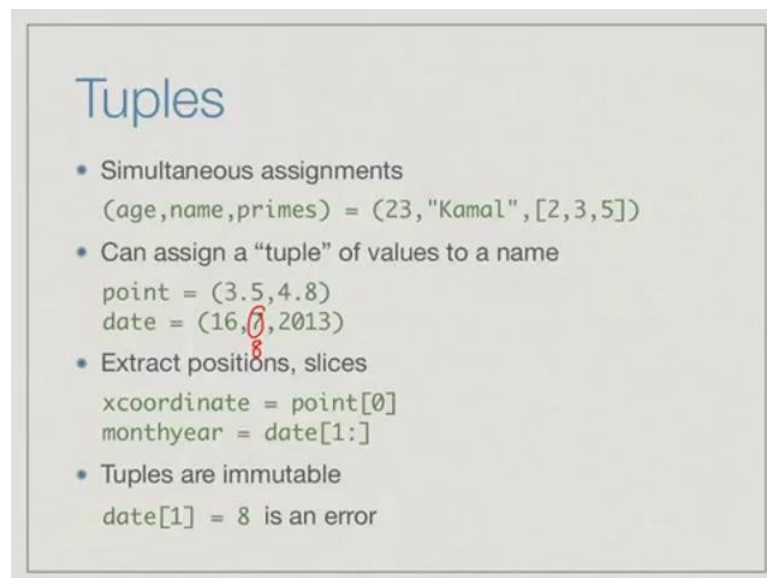**Tuples and Dictionaries**

(Refer Slide Time: 00:01)

We have seen this kind of simultaneous assignment, where we take three names on the left and assign them to three values in the right, and we enclose these in these round brackets. So, this kind of a sequence of values with the round bracket is called a Tuple.

Normally we talk about pairs, triples, quadruples, but in general when it goes to values of k we call them k tuples. On python, tuples are also valid values. You can take a single name; and assign it a tuple of values. For instance, we can take a two-dimensional point with x coordinates 3.5 and 4.8 and say that point has the value 3.5 comma 4.8, and this is not a list, but a tuple. And we will see in a minute what a tuple is.

Similarly, we can say that a date is made up of three parts a day, a month, and a year; and we can encloses into a three value or triple. So, tuple behaves like a list, so it is a kind of sequence. So, like strings and list, in a tuple you can extract one element of a sequence. So, we can say that the 0th value in point is the x coordinate. This would assign the value 3.5 to the value x to the name x coordinate, or we can take a slice we can say that if we

want only 7 and 2013, we take date and take the slice from one to the end then we will get 7 comma 2013. So, this behaves very much like a different type of sequence exactly like strings and lists we have seen so far, but the difference between a tuple and a list is that a tuple is immutable.

(Refer Slide Time: 01:44)

So, tuple behaves more like a string in this case, we cannot change for instance this date to 8 by saying date at position one should be replaced by the value 8. This is possible in a list, but not in a tuple. So, tuples are immutable sequences, and you will see in a minute why this matters.
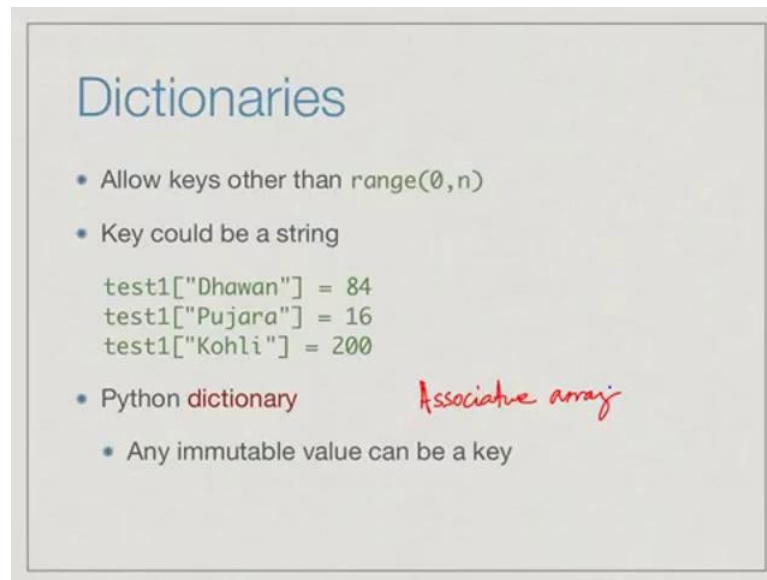
(Refer Slide Time: 02:10)

Let us go back to lists. A list is a sequence of values, and implicitly there are positions associated to this sequence starting at 0 and going up to the length of the list minus 1. So, an alternative way of viewing a list is to say that it maps every position to the value; in this case, the values are integers.

We can say that this list l is a map or function in a mathematical sense from the domain 0, 1, 2, 3, 4 to the range of integers; and in particular, it assigns l 0 to be 13, l 4 to be 72 and so on where we are looking at this as a function value. So, the program language way of thinking about this is that 0, 1, 2, 3, 4 are what are called keys. So, these are the values with which we have some items associated. So, we will search for the item associated with 1 and we get back 46. We have keys and the corresponding entries in the list are called values. So, a list is one way of associating keys to values.

(Refer Slide Time: 03:19)

We can generalize this concept by allowing keys from a different set of things other than just a range of values from 0 to n minus 1. So, a key for instance could be a string. So, we might want a list in which we index the values by the name of a player. So, for instance, you might keep track of the score in a test match by saying that for each player's name what is the value associated. So, Dhawan's score is 84, Pujara's score is 16, Kohli's score is 200, we store these all in a more generic list where the list values are not indexed by position, but by some more abstract key in this case the name of the player.

This is what python calls a dictionary, in some other programming languages this is also called an associative array. So, you might see this in the literature. So, here is a store of values which are accessed through a key which is not just a position, but some arbitrary index and python's rule is that any immutable value can be a key.

This means that you can use strings which are immutable. And here for instance you can use tuples, but you cannot use lists as we will see. And the other feature of a dictionary is that like a list, it is mutable; we can take a value with a key and replace it. So, we can change Pujara's score, if you want by an assignment to 72, and this will just take the current dictionary and replace the value associated to Pujara from 16 to 72. So, dictionaries can be updated in place and hence are mutable exactly like lists.

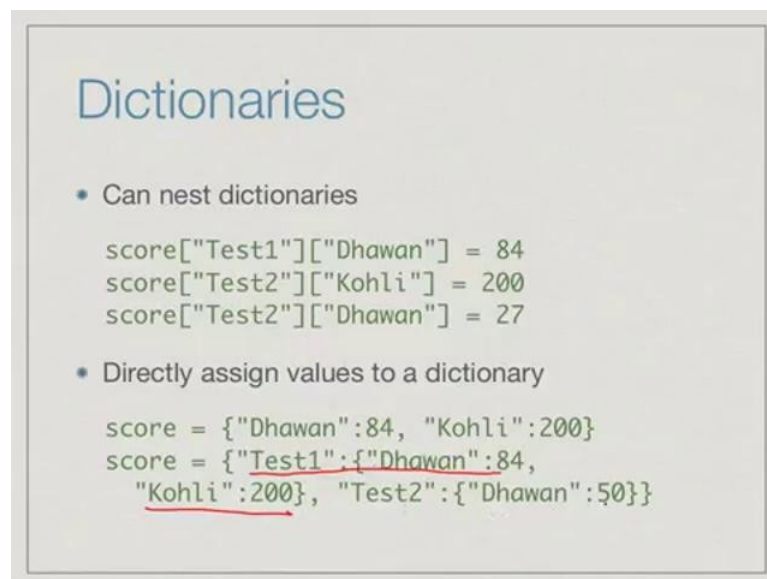We have to tell python that some name is a dictionary and it is not a list. So, we signify an empty dictionary by curly braces. So, remember we use square brackets for list. So, if you want to initialize that dictionary that we saw earlier then we would first say test 1 is the empty dictionary by giving it the braces here and then we can start assigning values to all the players that we had before like Dhawan and Pujara and so on. So, notice that all these three sequences and types of things that we have are different, so for strings of course, we use double codes or single codes; for list we use square brackets; for tuples, we use round brackets; and for dictionary, we use braces.

So, there is an unambiguous way of signaling to python what type of a collection we are associating with the name, so that we can operate on it with the appropriate operations that are defined for that type of collection. So, once again for a dictionary, the key can be any immutable value; that means, your key could be an integer, it could be a float, it could be a bool, it could be a string, it could be a tuple, what it cannot be is a list or a dictionary. So, we cannot have a value indexed by a list itself or by a dictionary.

(Refer Slide Time: 06:21)

So, we can have multiple just like we have nested list where we can have a list containing list and then we have two indices take the 0th list and then their first position in the 0 list, we can have two levels of keys. If you want to keep track of scores across multiple test matches, instead of having two dictionaries is we can have one dictionary where the first key is the test match test 1 or test 2, and the second key is a player.

With the same first key for example, with the same different first key for example, test 1 and test 2; you could keep track of two different scores for Dhawan. So, the score in test 1 and the score in test 2. And we can have more than one player in test 2 like we have here; we have both Kohli and Dhawan this one.

If you try to display a dictionary in python, it will show it to you in this bracket in this kind of curly bracket notation, where each entry will be the key followed by the values separated by the colon and then this will be like a list separated by commas. And if we have multiple keys then essentially this is one whole entry in this dictionary, and for the key test 1, I have these values; for the key test 2, I have these values. And internally they are again dictionaries, so they have their own key value.

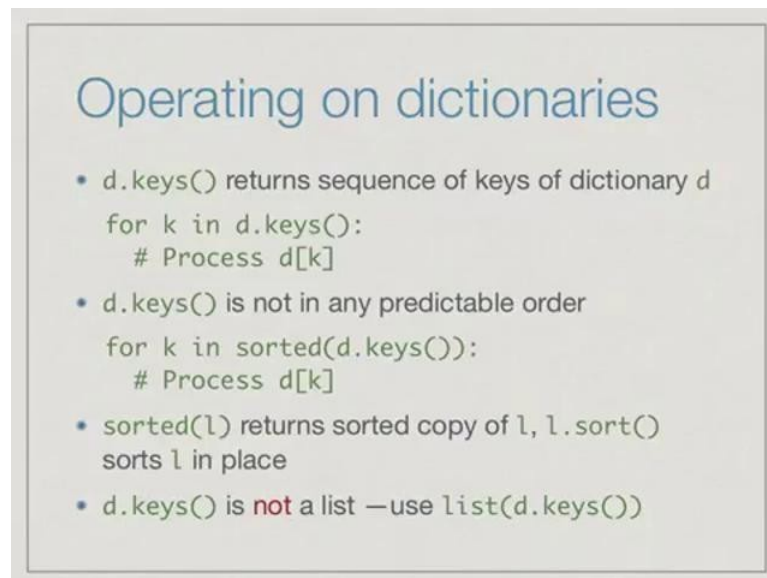(Refer Slide Time: 07:39)


```
>>> score = {}
>>> score["Test1"]["Dhawan"] = 76
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Test1'
>>> score["Test1"] = {}
>>> score["Test2"] = {}
>>> score["Test1"]["Dhawan"] = 76
>>> score["Test2"]["Dhawan"] = 27
>>> score["Test1"]["Kohli"] = 200
>>> score
{'Test1': {'Dhawan': 76, 'Kohli': 200}, 'Test2': {'Dhawan': 27}}
>>>
```

Let us see how it works we start with an empty dictionary say score. And now we want to create keys, so suppose we will say score test 1, Dhawan equal to 76. Now this is going to give us an error, because we have not told it that score test 1 is suppose to be a dictionary. So, it does not know that we can further index with the word Dhawan. So, we have to first tell it that not only score is a dictionary, so is score test 1 and presumably since we will use it, so is score test 2.

Now we can go back and set Dhawan's score in the first test to 76 and may be you can set the second test to 27 and maybe we can set Kohli's score in the first test to 200. Now, if you ask me to show what scores looks like, we see that it has an outer dictionary with

two keys test 1, test 2 each of which is a nested dictionary. In a nested dictionaries, we have two keys Dhawan and Kohli with scores 76 and 200 as the values. In test 2, has one dictionary entry with Dhawan as a key and 27 is the score.

(Refer Slide Time: 08:52)

If you want to process a dictionary then we would need to run through all the values; and one way to run through value all the values is to extract the keys and extract each value by turn. So, there is a function d dot keys which returns a sequence of keys of a dictionary d. And the typical thing we would do is for every key in d dot keys do something with d square bracket k. So, pick up all the keys.

This is like saying for every position in a list do something the value at that position. This is something for every key in a list do something with a value associated to that. Now one thing we have to keep in mind which I will show in a minute is that d dot keys not in any predictable order. So, dictionaries are optimized internally to return the value with a key quickly. It may not preserve the keys in the order in which they are inserted. So, you cannot predict anything about how d dot keys will be presented to us. One way to do this is to use the sorted function.

We can say for k in sorted d dot keys, process d k, and this will give us the keys in sorted order according to the sort function. So, sorted l is a function we have not seen so far; sorted l returns a sorted copy of l, it does not modify. What we have seen so far is l dot

sort, which is the function which takes a list and updates it in place. So, sorted l takes an input list, leaves it unchanged, but it returns a sorted version.

The other thing to keep in mind is that though it is tempting to believe that d dot keys is a list, it is not a list; it is like range and other things. It is just a sequence of values that you can use inside of for, so we must use the list property to actually create a list out of d dot keys.
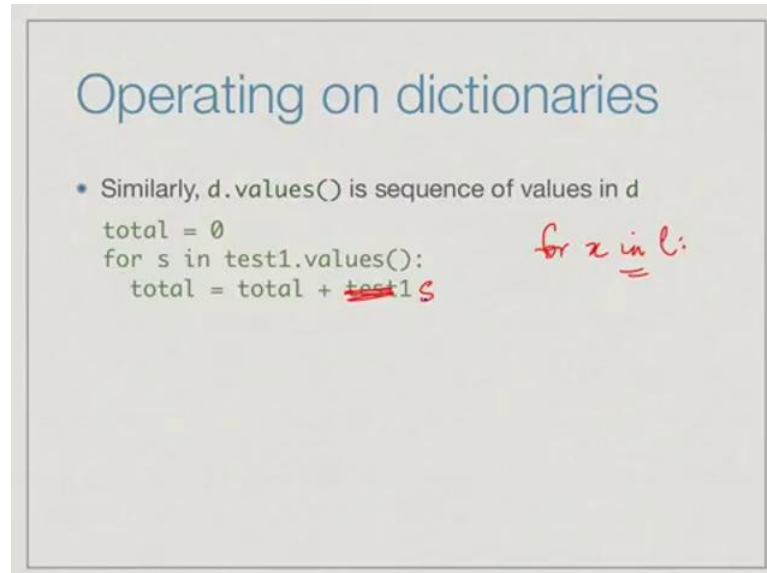
(Refer Slide Time: 10:46)

So, let us validate the claim that keys are not kept in any particular order. So, let us start with an empty dictionary. And now let us create for each letter and entry which is the same as that letter. So, we can say for l in a, b, c, d, e, f, g, h, i, d i, d l is equal to l. So, what it is this saying, so when you say for l in a string it goes to each letter in that string, so want to say d with key a is the value a, d with the key b is the value b and so on right. So, now, if I ask you what is d a, you can a, what is d i, it is i.

Now notice that the keys are inserted in the order a, b, c, d, e, f, g, h, i but if I ask for d dot keys it produces it in some very random order. So, e is first and a is way down and so on. There is no specific order that you can get from this. So, this is just to emphasize that the order in which keys are inserted into the dictionary is not going to be the order in which they are presented to through the keys function. So, you should always ensure that if you want to process the keys in a particular order make sure that you preserve that

order when you extract the keys you cannot assume that the keys will come out in any given order.

(Refer Slide Time: 12:06)

In other way to run through the values in a dictionary is to use d dot values. So, d dot keys returns the key is in some order, d dot values gives you the values in some order. So, this is for example like say for x in l. So, you just get the values you do not get that positions. Here you just get the values you do not get the keys. So, if you want to add up all the values for instance from a dictionary, you can start off by initializing total to 0, and for each value, you can just add it up yes right. So, you can pick up each s in test 1 dot values and add it to the total.

So, you can test for a key being in a dictionary by using the in operator, just like list when you say x in l for a list it tells you true if x belongs to l the value x belongs to l, it tells you false otherwise. The same is true of keys. So, if I want to add up the score for individual batsmen, but I do not know, if they have batted in each test match. So, I will say for each of the keys, in this case, Dhawan and Kohli, initialize the dictionary which i have already set up not here I would have set that total is a dictionary. So, total with key Dhawan is 0, total with key Kohli is 0.

Now for each match in our nested dictionary, if Dhawan is entered as a batsman in that match, so if a name Dhawan appears as the key in score for that match then and only then you add a score, because if it does not appear it is illegal to access that match. So, this is one way to make sure that when you access a value from a dictionary, the key actually exists, you can use the in function.

(Refer Slide Time: 14:00)

Here is a way of remembering that a dictionary is different from the list. If I start with an empty dictionary then I assign a key, which has not been seen so far, in a dictionary there is no problem it is just equivalent to inserting this key in the dictionary with that value, if d 0 already exists it will be updated. So, either you update or you insert. This is in contrast with the list, where if you have an empty list and then try to insert at a position which does not exist, you get an index error.

(Refer Slide Time: 14:42)

In a dictionary, it flexibly expands to accommodate new keys or updates a key depending on whether the key already exists or not.

To summarize, a dictionary is a more flexible association of values to keys than you have in a list; the only constraint that python imposes is that all keys must be immutable values. You cannot have keys, which are mutable values. So, we cannot use dictionaries or list themselves as keys, but you can have nested dictionaries with multiple levels of these.

The other thing is that we can use d dot keys to cycle through all the keys in the dictionary, and similarly d dot values, but the order in which these keys emerge from d dot keys is not predictable. So, we need to sort it to do something else if we want to make sure to process them in a predictable order.

So, it turns out that you will see that dictionaries are actually something that make python a really useful language for manipulating information from text files or tables, if you have what are called comma separated value tables, it is taken out of spreadsheet because then we can use column headings and accumulate values and so on. So, you should understand and assimilate dictionary in to your programming skills, because this is what makes python really a very powerful language for writing scripts to manipulate it.