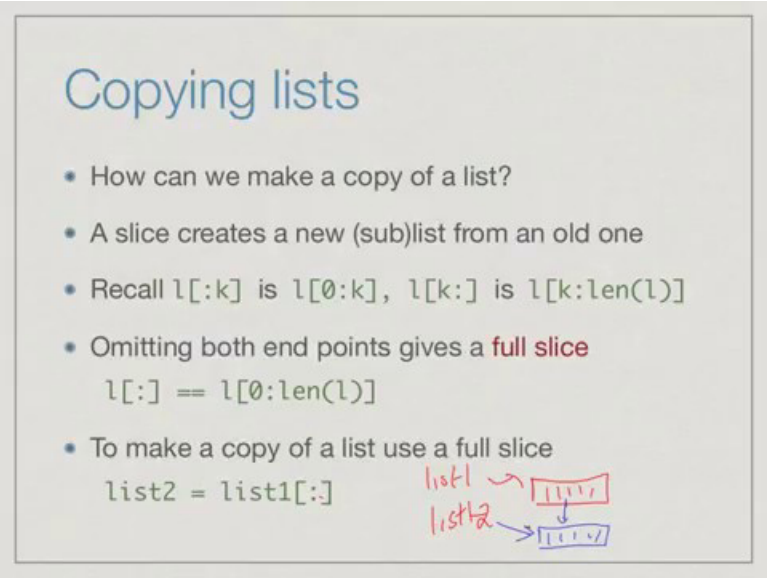
(Refer Slide Time: 17:10)



This is something which we will see is useful in certain situations, but what if we do not want this to happen what if we want to make a real copy of the list. So, recall that a slice takes a list and returns us a sub list from one position to another position. The outcome of a slice operation is actually a new list, because in general, we take a list and we will take a part of it for some intermediate position to some other intermediate position, so obviously, the new list is different from the old list.

We also saw that when we looked at strings that we can leave out the first position or the last position when specifying a slice. If we leave out the first position as this then we will implicitly say that the first position is 0, so we start at the beginning. Similarly, if we leave out the last position like this, then we implicitly assume that the last position the slice is the length of this list of the string and so it goes to the last possible value.

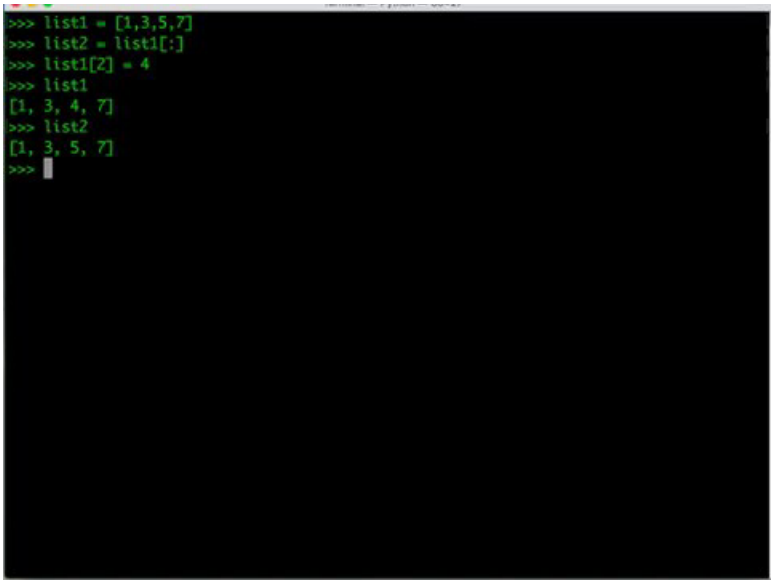
If we leave out the first position, we get a 0; if we leave out the last position, we get the length. If we leave out both position, we just put colon with nothing before nothing after logically this becomes 0 and this becomes the length. We have both characteristics in the same thing and we call this a full slice.

Now let us combine this observation which is just a short cut notation with this observation that each slice creates a new sub list. So, what we have is that l with just a colon after it is not the same as l it is the new list created from the old list, but it as every

value in l in the same sequence. This now gives us a simple solution to copy a list instead of saying list2 is equal to list1, which makes then both.

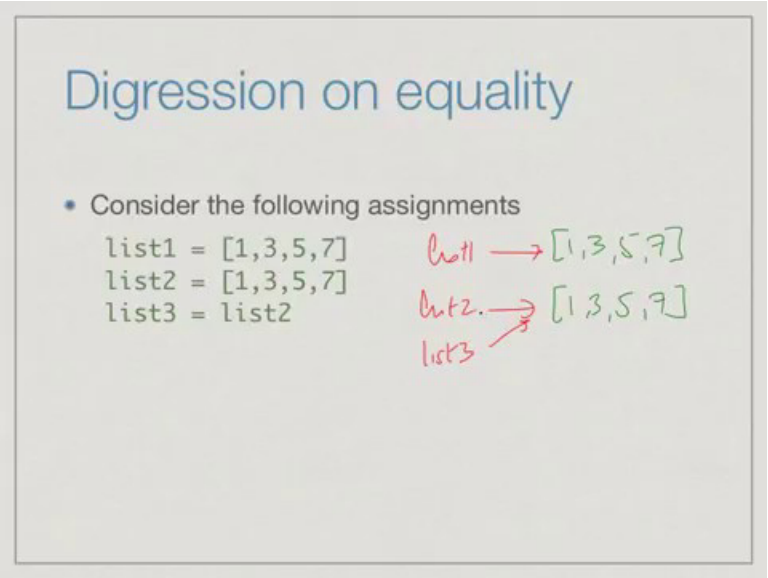
Remember if I do not have this then I will get list1 and list2 pointing to the same actual list. There will be only 1 list of values and will point to the same. But if I do there have this then the picture changes then what happens is that the slice operation produces a new list which has exactly the same length and the same values and it makes list2 point to that. Therefore, after this list1 and list2 are disjoint from each other any update to list2 will not affect list1 any update to list1 will not affect list2. Let us see how this works in the interpreter to convince ourselves this is actually the way python handles this assignment.

(Refer Slide Time: 19:45)



As before let us start with list1 is 1, 3, 5, 7 and list2 now let us say is the slice. So, now, if is update list1 at position 2 to be 4 then list1 looks like 1, 3, 4, 7. But list2 which was a copy is not affected right. When we take a slice we get a new list. So, if we take the entire list as a full slice we get a full copy of the old list and we can assign it safely to a new name and not worry about the fact that both names are sharing the value.

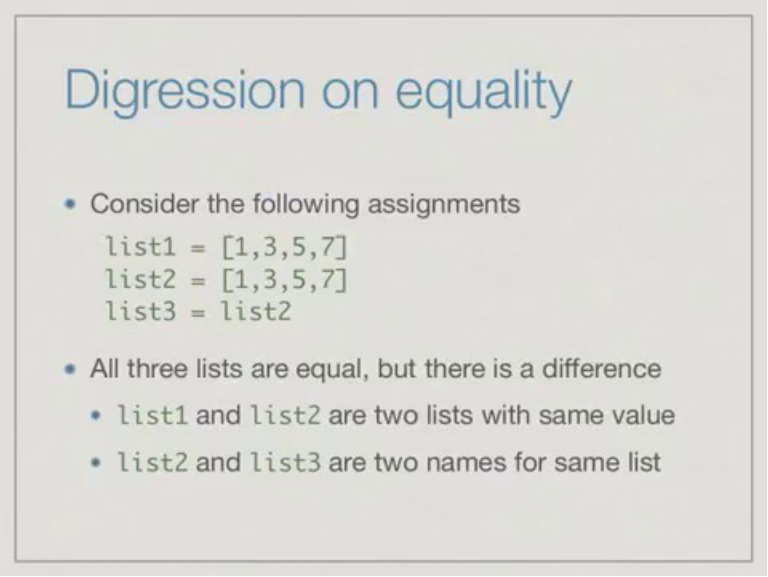
(Refer Slide Time: 20:19)



This leads us to a digression on equality. Let us look now at this set of python statements. We create a list 1, 3, 5, 7 and give with the name list1 and, when we create another list 1, 3, 5, 7, and give it the name list2.

And finally, we assign list3 to be the same values as list2 and this as be said suggest that list3 is actually pointing to the same thing. So, we have now pictorially we have two list of the form 1, 3, 5, 7 stored somewhere. And initially we say that list1 points to this and list2 points to this in the last assignment say that list3 also points to this.

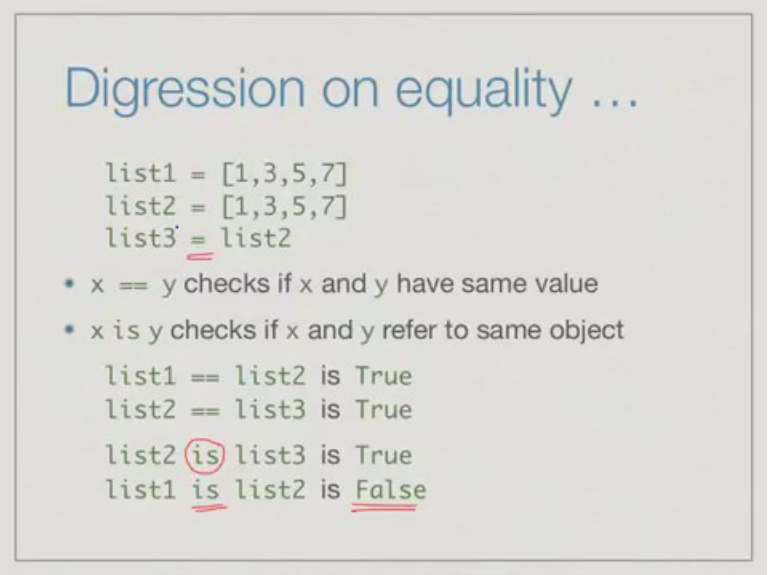
(Refer Slide Time: 21:10)



All three lists are equal, but there is a difference in the way that they are equal. So, list1 and list2 are two different lists, but they have the same value right. So, they happen to have the same value, but they are two different things and so, if we operate on one it need not preserve this equality any more.

On the other hand list2 and list3 are equal precisely because they points to the same value, there is exactly one list in to which they are both pointing. So, if we update list3 or we update list2 they will continue to remain equal. There are two different notions of equality whether the value is the same or the actual underline object that we are referring to by this name is the same. In the second case, updating the object to either name is going to result in both names continuing to be equal.

(Refer Slide Time: 21:57)



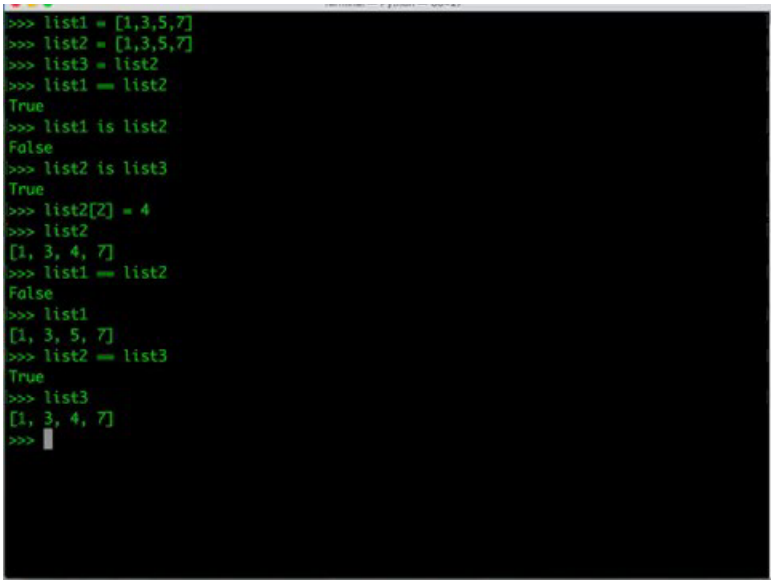
Python has we saw this operation equal to equal to, which is equivalent or the mathematically equality which checks if x and y as names have the same value. So, this will capture the fact that list1 is equal to list2 even though they are two different lists they happen to have the same value.

To look at the second type of equality that list3 and list2 are actually the same physical list in the memory. We have another key word in python called 'is'. So, when we say x is y what we are asking is, whether x and y actually point to the same memory location the same value in which case updating x will effect y and vice versa. We can say that x is y checks if x and y refer to the same object.

Going by this description of the way equal to equal to and is work; obviously, if list2 list3 are the same object they must always be equal to - equal to. So, x is y then x will always equal to equal to y, because there are actually pointing to the same thing. But in this case although list1 list2 are possibly different list they are still equal to - equal to, because the value is the same.

On the other hand if I look at the is operation then list1 list2 is list3 happens to be true, because we have seen that this assignment will not copy the list it will just make list3 point to the same thing is list2. On other hand list1 is list2 is false that is because they are two different list. So, once again its best to verify this for ourselves to convince ourselves that this description is actually accurate.

(Refer Slide Time: 23:41)

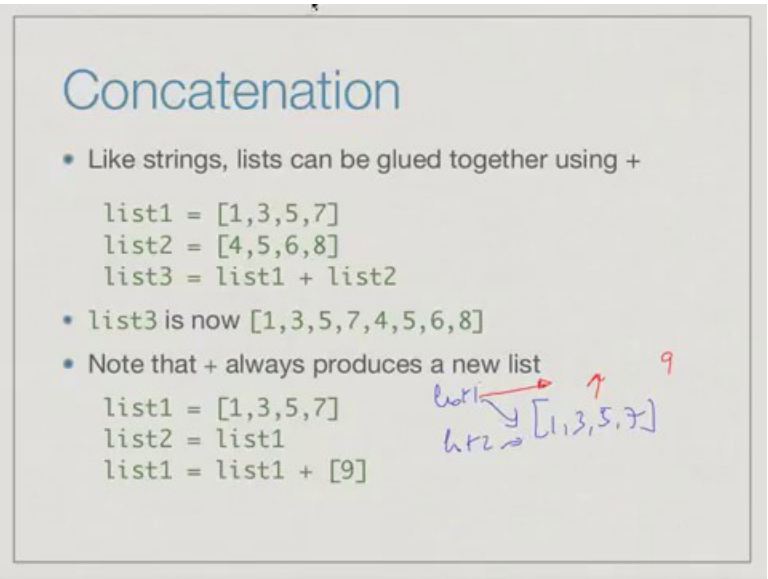


Let us type out those three lines of python in the interpreter. So, we say list1 is 1, 3, 5, 7 list2 is also 1, 3, 5, 7 and list3 is list2. Now, we ask whether list1 is equal to list2 and it indeed is true, but if we ask whether list1 is list2 then it says false. So, this means that list1 and list2 are pointing to the same value physically. So, we update one it will not update the other.

On the other hand, if we ask whether list2 is list3 then this is true. If for instance we change list2, 2 to be equal to 4, like we are done in the earlier example then list2 has now become 1, 3, 4, 7. So, if we ask if list1 is equal to list2 at this point as values that's false. Therefore, because list1 continues to be 1, 3, 5, 7 and list2 has become 1 3 4 7; however,

if we ask whether list2 is equal to list3 is true that is the case, because list3 is list2 in the sense if they both are the same physical list and so when we updated list3 list2 will also be updated via list3.

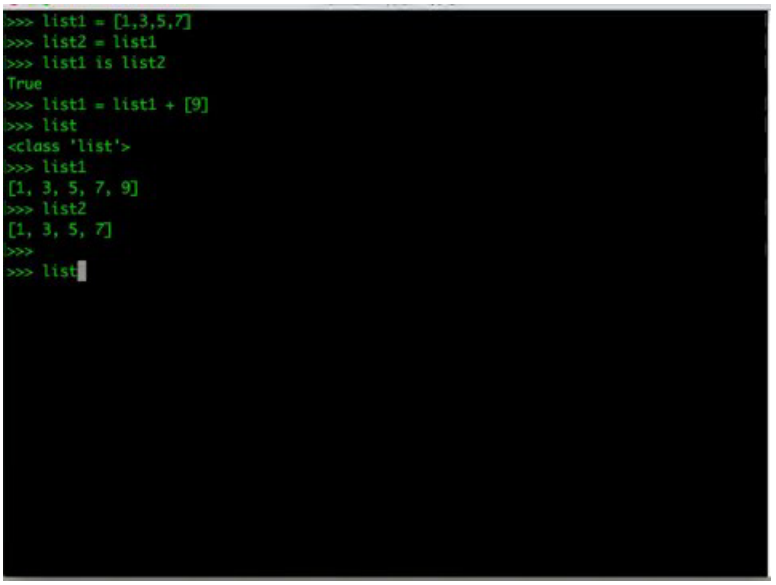
(Refer Slide Time: 24:54)



Like strings, we can combine lists together using the plus operator. So, plus is concatenation. So, if we have list1 is the value 1, 3, 5, 7 list2 is the value 4, 5, 6, 8. Then list3 equal to list1 plus list2 will produce for us the value 1, 3, 5, 7, 4, 5, 6, 8. One important thing to recognise in our context of mutable and immutable values is that plus always produces a new list. If we say that list1 is 1, 3, 5, 7 and then we copy this list as a name to list2. We saw before that we have 1, 3, 5, 7 and we have two names list1 and list2.

Now if we update list1 by saying list1 plus nine this will actually generate a fresh list which has a nine at the end and it will make list1 point their and list2 will no longer be the same right. So, list1 and list2 will no longer point to the same object. Let just confirm this.

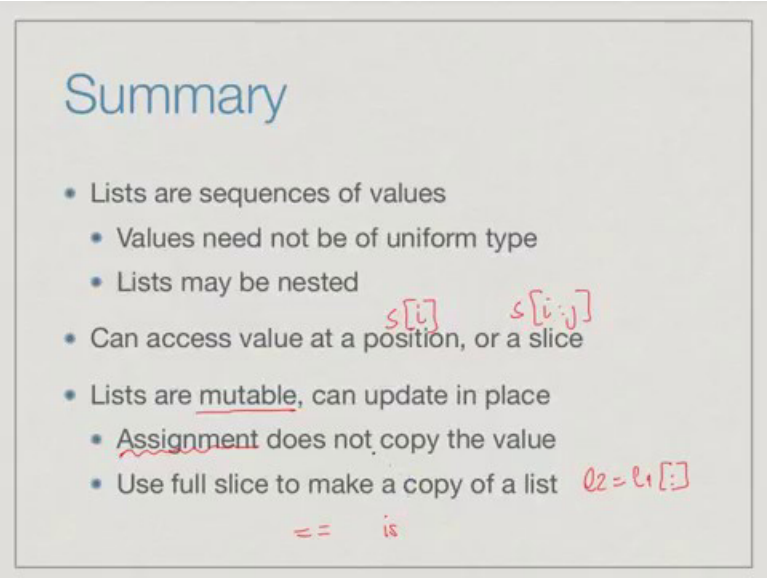
(Refer Slide Time: 26:00)



In the python interpreter let us set up list1 is equal to 1, 3, 5, 7 and say list2 is equal to list1. Then as we saw before if we say list1 is list2 we have true. If on the other hand we reassign list1 to be the old value of list1 plus a new value 9.

This extends, list1 to be 1, 3, 5, 7, 9. Now we will see the list2 is unchanged. So, list1 and list2 have become decoupled because which time we apply plus it is like taking slice. Each time we apply plus we actually get a new list. So, list1 is no longer pointing to the list it was originally pointing to. It is pointing to a new list constructed from that old list with a 9 appended to it at the end.

(Refer Slide Time: 26:49)



To summarise we have now seen a new type of value called list. So, list is just a sequence of values. These values need not be of a uniform type, we can have mixed list consisting or list, Boolean, integers, strings. Although almost always we will encounter list, where the underline content of a list is of a fixed type. So, all position will actually typically have a uniform type, but this is not required by python and we can nest list. So, we can have list of list and list of list of list and so on.

As with strings, we can use this square bracket notation to obtain the value at a position or we can use the square bracket with colon notation to get a sub list or a slice. One new feature of python, which we introduced with list, is a concept of a mutable value. So, a list can be updated in place we can take parts of a list and change them without effecting the remaining parts it does not create a new list in memory. One consequence is this is that we have to look at assignment more carefully.

For immutable values the types we have seen so far, int, float, bool and string when we say x equal to y the value of y is copied to x. So, updating x does not affect y and vice versa. But when we have mutable values like list we say l2 is equal to l1 then l2 and l1 both point to the same list. So, updating one will update the other, and so we be have little bit careful about this.

If we really want to make a copy, we use a full slice. So, we say l2 is equal to l1 colon with nothing before or after, this is implicitly from 0 to the length of l1, and this gives us

a fresh list with exact same contents as l1. And finally, we saw that we can use equality and is as two different operators to check whether two names are equal to only in value or also are physically pointing to the same type.