TUPLES & LISTS

TUPLES

Tuples, tuples are an ordered sequence. Here is a tuple ratings. Tuples are expressed as comma separated elements within parentheses. These are values inside the parentheses.

Tuple1 = ("disco",10,1.2)

Tuple1[0]: disco
Tuple1[1]: 10
Tuple1[2]: 1.2

0	"disco"	-3
1	10	-2
2	1.2	-1

0:26
In Python, there are different types, strings, integer, float. They can all be contained in a tuple. But the type of the variable is tuple. Each element of a tuple can be accessed via an index. The following table represents the relationship between the index and the elements in the tuple.

The first element can be accessed by the name of the tuple, followed by a square bracket with the index number, in this case, 0.

We can access the second element as follows, we can also access the last element. In Python, we can use negative index.

Example:
say_what=('say',' what', 'you', 'will')
say_what[-1]
= 'will'

The relationship is as follows, the corresponding values are shown here.

We can concatenate or combine tuples by adding them, the result is the following with the following index.

("disco", 10, 1.2)
tuple2 = tuple1 + ("hard rock", 10)
=("disco", 10, 1.2, "hard rock", 10)

1:14
If we would like multiple elements from a tuple, we could also slice tuples.

For example, if we want the first three elements we use the following command, the last index is one larger than the index you want.

("disco", 10, 1.2, "hard rock", 10)

tuple2[0:3]:("disco", 10, 1.2)

Consider the following tuple A=(1,2,3,4,5), what is the result of the following:A[1:4]:

A=(1,2,3,4,5)
A[1:4]
= (2, 3, 4)

Similarly, if we want the last two elements, we use the following command. Notice how the last index is one larger than the length of the tuple,

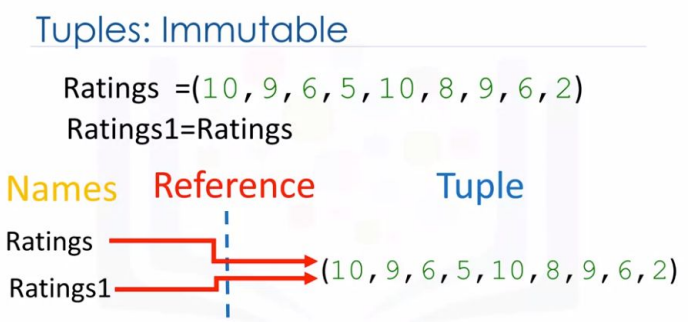
1:37
we can use the len command to obtain the length of the tuple. As there are five elements, the result is five.

len(("disco", 10, 1.2, "hard rock", 10))
= 5

1:45
Tuples are immutable, which means we can't change them.

Ratings =(10,9,8,7,6,5,4,3,2,1)
Ratings1 = Ratings

To see why this is important, let's see what happens when we set the variable ratings one to ratings. Let's use the image to provide a simplified explanation of what's going on. **Each variable does not contain a tuple but references the same immutable tuple object.**



See the objects and classes module for more about objects.

2:11
Let's say we want to change the element at index 2. Because tuples are immutable, we can't, therefore, ratings 1 will not be affected by a change in rating. Because the tuple is immutable, i.e, we can't change it. We can assign a different tuple to the ratings variable. The variable ratings now references another tuple.

Ratings =(2,10,1)

As a consequence of immutability, if we would like to manipulate a tuple, we must create a new tuple instead.

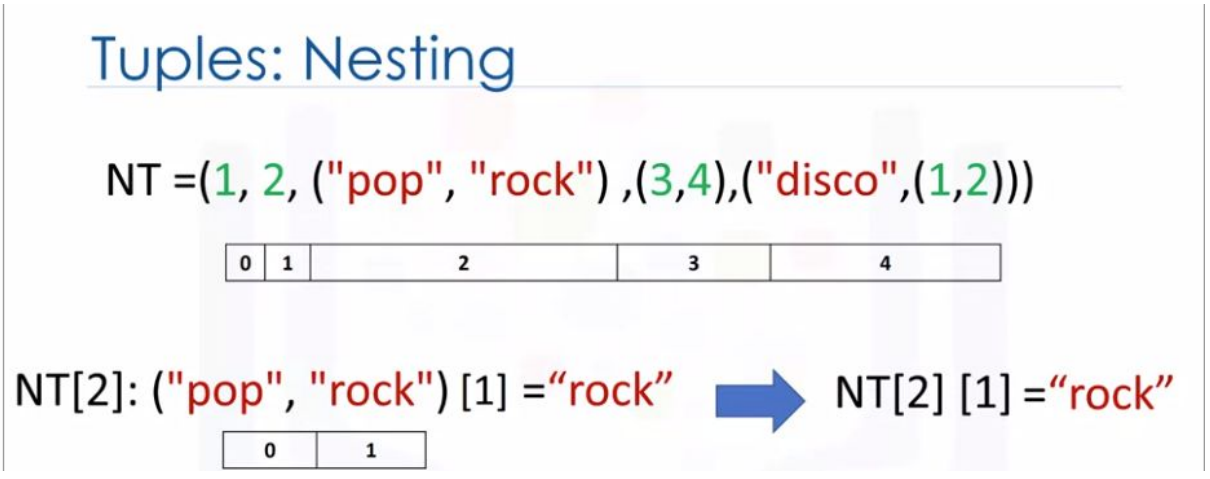
For example, if we would like to sort a tuple, we use the function sorted, the input is the original tuple. The output is a new sorted tuple, for more on functions, see our video on functions.

RatingsSorted=sorted(Ratings)

Nesting:
A tuple can contain other tuples as well as other complex data types, this is called nesting. We can access these elements using the standard indexing methods.

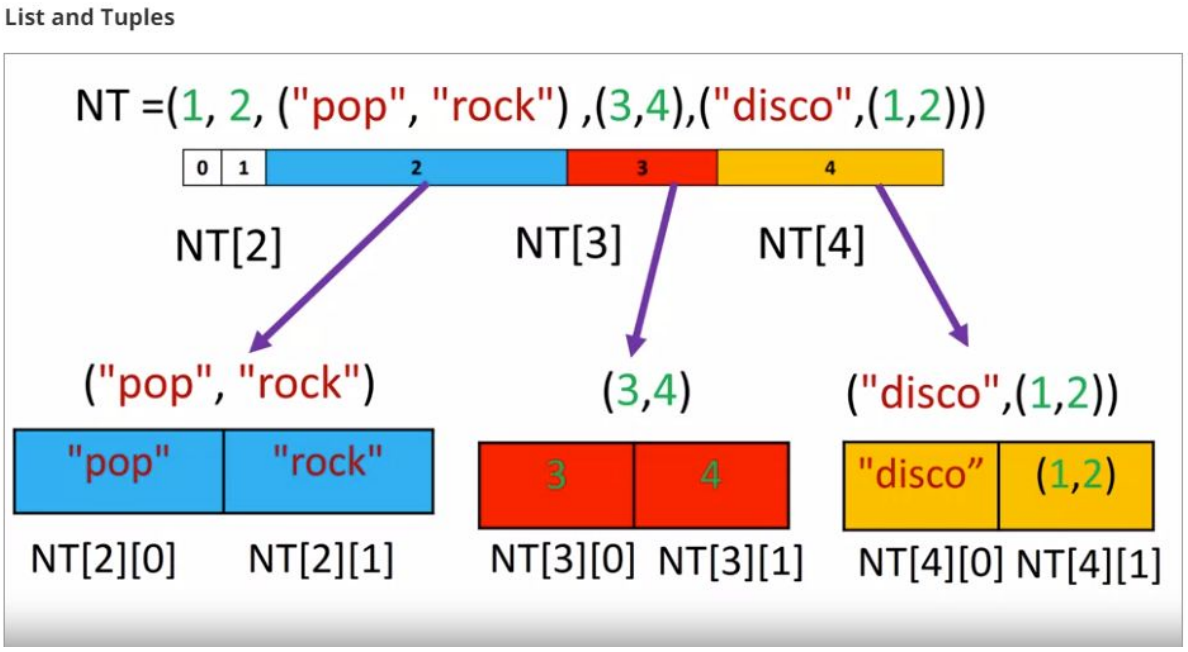
If we select an index with the tuple, the same index convention applies.
As such, we can then access values in the tuple.

For example, **we could access the second element**, we can apply this indexing directly to the tuple variable NT.



3:22
It is helpful to visualize this as a tree.

See Diagram below:



We can visualize this nesting as a tree. The tuple has the following indexes.

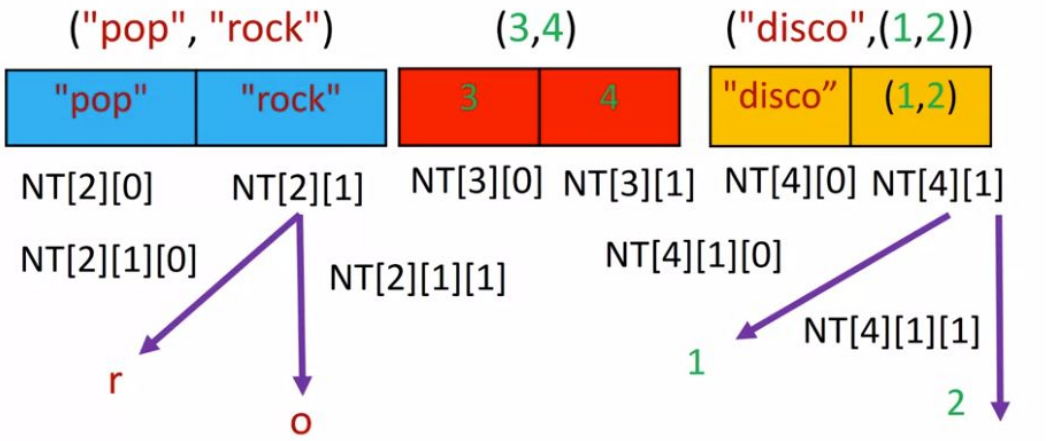
```
NT=(1,2,("pop","rock"),(3,4),("disco",(1,2)))
       0 1         2             3         4
```

```
len(NT) = 5
NT[0] = 1
NT[1] = 2
NT[2] = ("pop","rock")
NT[3] = (3,4)
NT[4] = ("disco",(1,2))
```

If we consider indexes with other tuples, we see the tuple at index 2 contains a tuple with two elements.

We can access those two indexes. The same conventional applies to index 3.
We can access the elements in those tuples as well. We can continue the process,

3:50
we can even access deeper levels of the tree by adding another square bracket. We can access different characters in the string or various elements in the second tuple contained in the first.



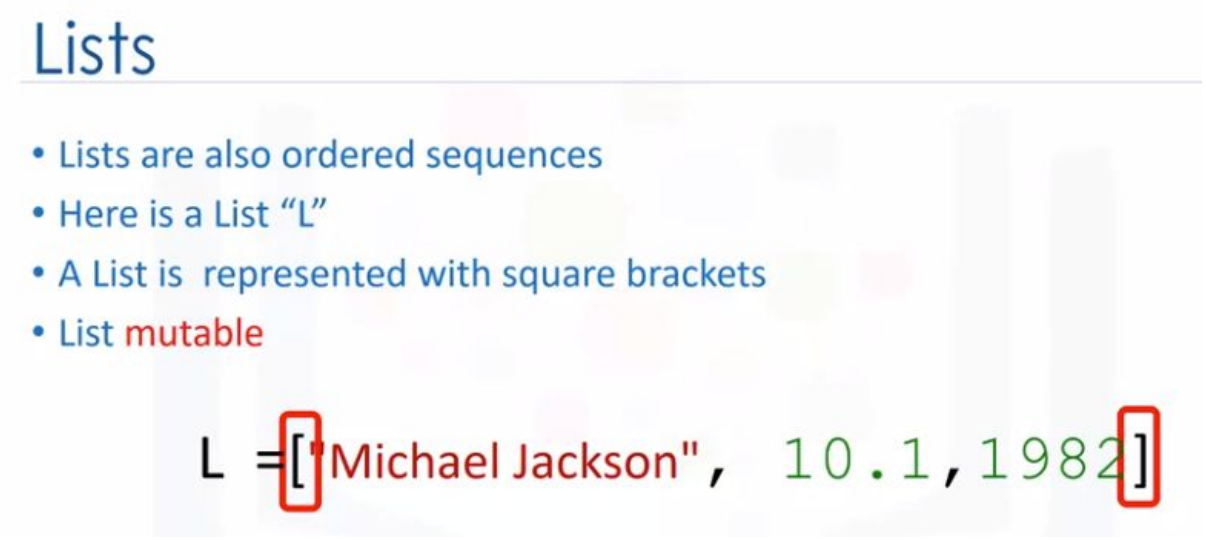
NT=(1,2,("pop","rock"),(3,4),("disco",(1,2)))

NT[2][1][1] = ‘o’
NT[0] = 1
NT[4] = ('disco', (1, 2))
NT[4][1][1] = 2

LISTS

Lists are also a popular data structure in Python. Lists are also an ordered sequence.

Here is a list L. **A list is represented with square brackets.** In many respects, lists are like tuples. One key difference is they are mutable.



Referencing a list

B=[1,2,[3,'a'],[4,'b']]

B[0] = 1
B[1] = 2
B[2] = [3, 'a']
B[3] = [4, 'b']
B[3][1] = ‘’b’

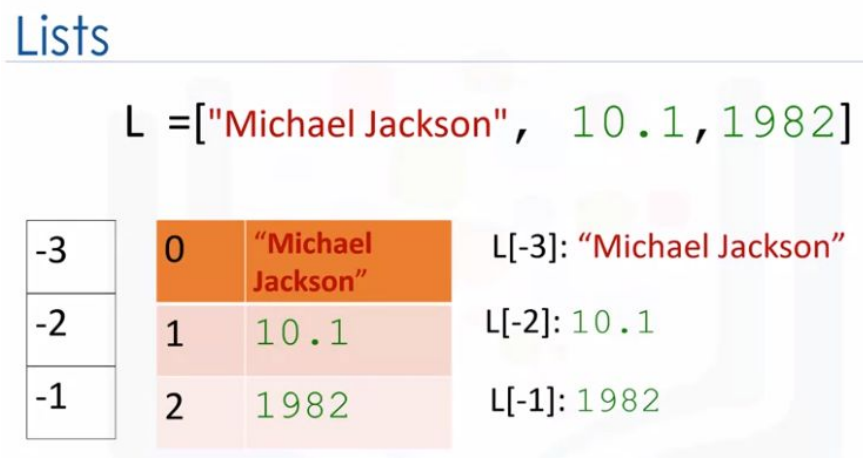
Lists can contain strings, floats, integers. We can nest other lists. We also nest tuples and other data structures.

The same indexing conventions apply for nesting. Like tuples, each element of a list can be accessed via an index.

The following table represents the relationship between the index and the elements in the list. The first element can be accessed by the name of the list followed by a square bracket with the index number, in this case, 0.

We can access the second element as follows.

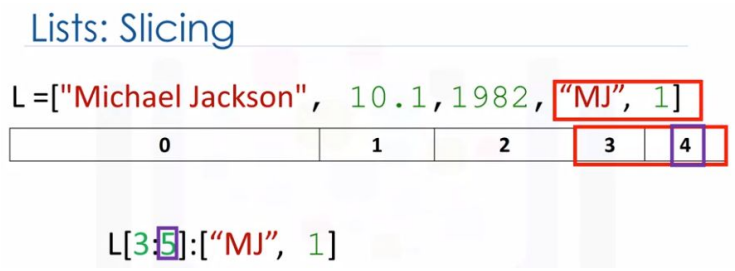
We can also access the last element. In Python, we can use a negative index. The relationship is as follows. The corresponding indexes are as follows.



Slicing Lists

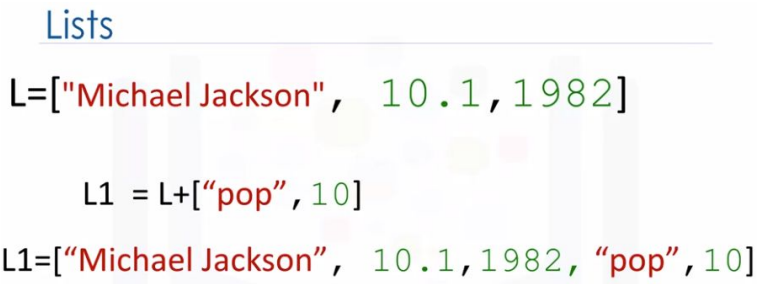
We can also perform slicing in lists. For example, if we want the last two elements in this list, we use the following command.

Notice how the last index is one larger than the length of the list.



The index conventions for list and tuples are identical. Check the labs for more examples.

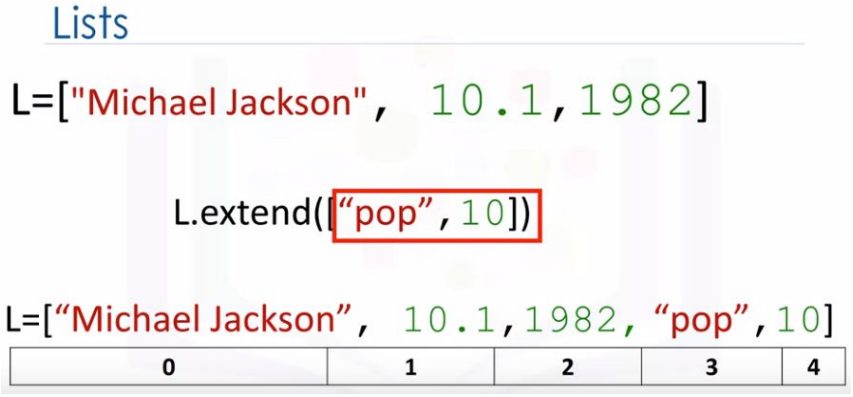
We can **concatenate** or combine list by adding them. The result is the following. The new list has the following indices.



[1,2,3]+[1,1,1] = [1, 2, 3, 1, 1, 1]

Extend Method:

Lists are mutable, therefore, we can change them. For example, we apply the method extends by adding a dot followed by the name of the method, then parentheses.

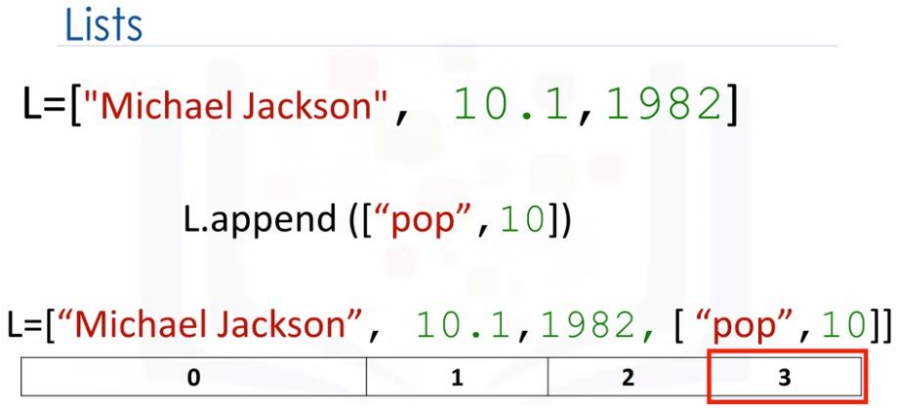


The argument inside the parentheses is a new list that we are going to **concatenate to the original list.**

In this case, instead of creating a new list L1, the original list L is modified by adding two new elements. To learn more about methods, check out our video on objects and classes.

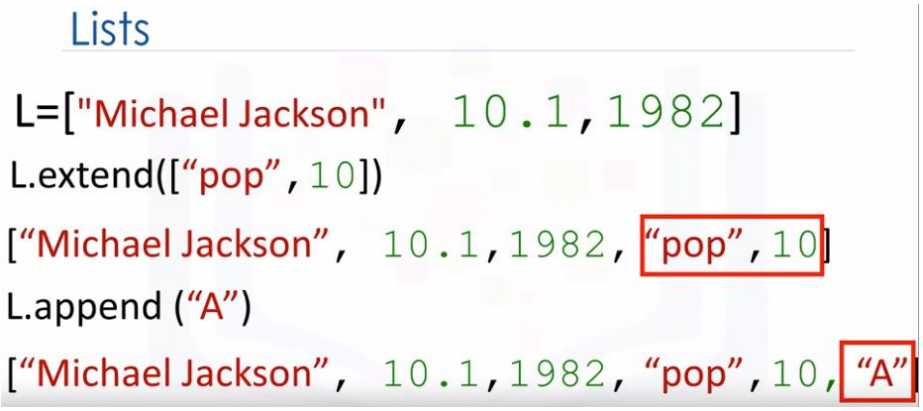
Append Method:
Another similar method is **append**. If we apply append instead of extended, we **add one element to the list.**

If we look at the index, there is only one more element. Index 3 includes the list that we appended.



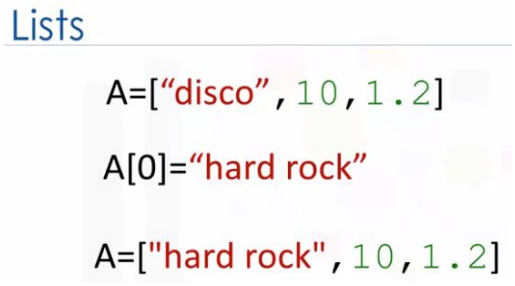
What is the length of the list A = [1] after the following operation: A.append([2,3,4,5])
A=[1]
A.append([2,3,4,5])
A = [1, [2, 3, 4, 5]]
len(A) = 2

6:19
Every time we apply a method, the list changes. If we apply extend, we add two new elements to the list. The list L is modified by adding two new elements. If we append the string A, we further change the list, adding the string A.



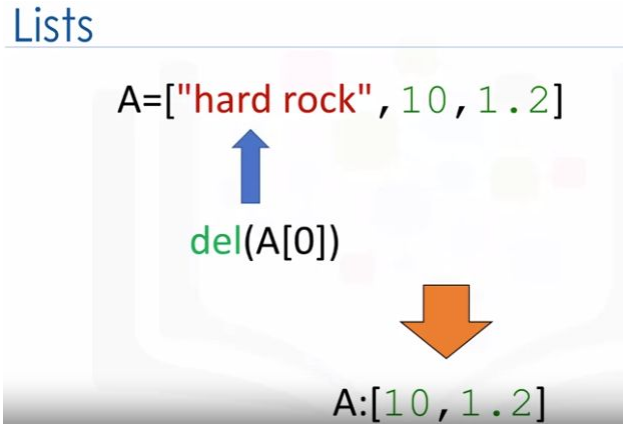
6:37
As lists are mutable, we can change them. For example, we can change the first element as follows.

The list now becomes hard rock, 10, 1.2.



Del command

We can delete an element of a list using the Del command. We simply indicate the list item we would like to remove as an argument. For example, if we would like to remove the first element, the result becomes 10, 1.2.



We can delete the second element. This operation removes the second element of the list.

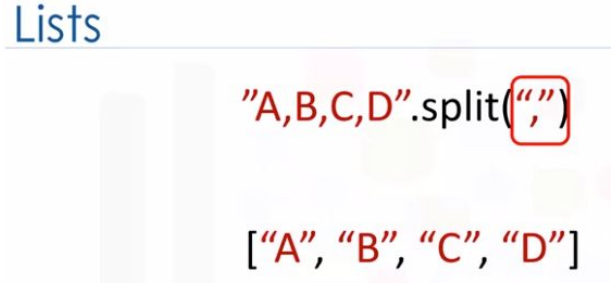
Split command

We can convert a string to a list using split. For example, the method split converts every group of characters separated by space into an element of a list.

What is the result of the following: "Hello Mike".split()

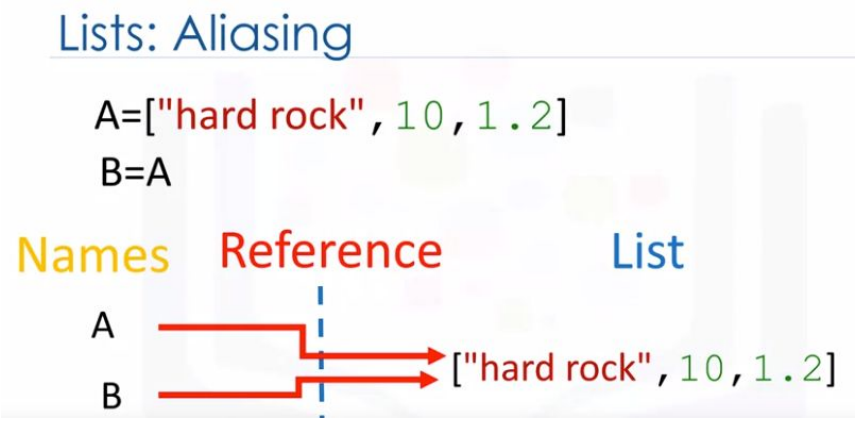
= ["Hello","Mike"]

We can use the split function to separate strings on a specific character known as a delimiter.



We simply pass the delimiter we would like to split on as an argument, in this case, a comma. The result is a list, each element corresponds to a set of characters that have been separated by a comma.

When we set one variable B equal to A, both A and B are referencing the same list.
Multiple names referring to the same object is known as aliasing.

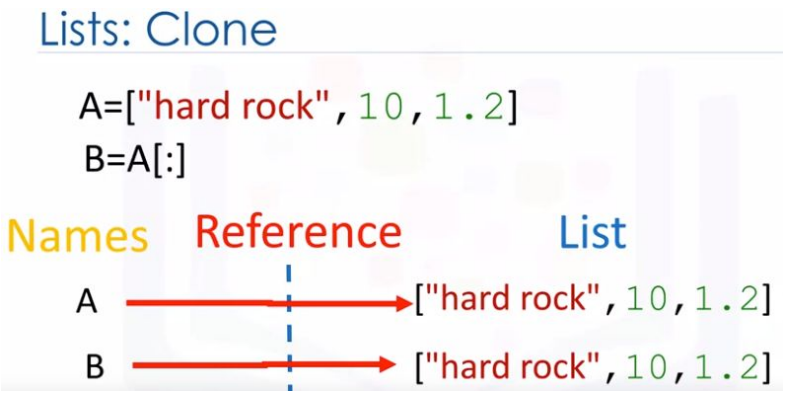


We know from the list slide that the first element in B is set as hard rock. If we change the first element in A to banana, we get a side effect. The value of B would change as a consequence.

8:03
A and B are referencing the same list, therefore, if we change A, list B also changes.

8:10
If we check the first element of B after changing list A, we get banana instead of hard rock.

8:17
You can clone list A by using the following syntax, variable A references one list. Variable B references a new copy or clone of the original list.
Now, if you change A, B will not change.



We can get more info on list, tuples, and many other objects in Python using the help command. Simply pass in the list, tuple, or any other Python object. See the labs for more things you can do with lists.