**Lists and Tuples**

These are called compound data types and are one of the key types of data structures in Python. 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. 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 zero. We can access the second element as follows. We can also access the last element. In Python, we can use negative index. 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. 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; 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. We can use the len command to obtain the length of a tuple. As there are five elements, the result is 5. Tuples are immutable which means we can't change them. To see why this is important, let's see what happens when we set the variable ratings 1 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. 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. 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 list. For more on functions, see our video on functions. 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 a 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. It is helpful to visualize this as a tree. We can visualize this nesting as a tree. The tuple has the following indexes. 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 convention applies to index 3. We can access the elements in those tuples as well. We can continue the process. 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. 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 respect, lists are like tuples. One key difference is they are mutable. 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 zero. 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. 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 lists and tuples are identical. Check the labs for more examples. We can concatenate or combine lists by adding them. The result is the following. The new list has the following indices. 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. 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 contains the list we appended. 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. 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. 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 off the list. We can convert a string to a list using split. For example, the method split converts every group of characters separated by a space into an element of a list. 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 will change as a consequence. A and B are referencing the same list, therefore if we change A, list B also changes. If we check the first element of B after changing list A, we get banana instead of hard rock. 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 lists, tuples, and many other objects in Python using the help command. Simply pass in the list, tuple, or any other Python object.

**Dictionaries**

Dictionaries are a type of collection in Python. If you recall, a list is integer indexes. These are like addresses. A list also has elements. A dictionary has keys and values. The key is analogous to the index. They are like addresses, but they don't have to be integers. They are usually characters. The values are similar to the element in a list and contain information. To create a dictionary, we use curly brackets. The keys are the first elements. They must be immutable and unique. Each key is followed by a value separated by a colon. The values can be immutable, mutable, and duplicates. Each key and value pair is separated by a comma. Consider the following example of a dictionary. The album title is the key, and the value is the released data. We can use yellow to highlight the keys and leave the values in white. It is helpful to use the table to visualize a dictionary where the first column represents the keys, and the second column represents the values. We can add a few more examples to the dictionary. We can also assign the dictionary to a variable. The key is used to look at the value. We use square brackets. The argument is the key. This outputs the value. Using the key of "Back in Black," this returns the value of 1980. The key, "The Dark Side Of The Moon," gives us the value of 1973. Using the key,"The bodyguard," gives us the value 1992 and so on. We can add a new entry to the dictionary as follows. This will add the value 2007 with a new key called "Graduation." We can delete an entry as follows. This gets rid of the key "Thriller" and it's value. We can verify if an element is in the dictionary using the "in" command as follows: The command checks the keys. If they are in the dictionary, they return a true. If we try the same command with a key that is not in the dictionary, we get a false. In order to see all the keys in the dictionary, we can use the method keys to get the keys. The output is a list-like object with all the keys. In the same way, we can obtain the values using the method values.

**Sets**

Sets are also a type of collection. Sets are a type of collection. This means that like lists and tuples, you can input different Python types. Unlike lists and tuples, they are unordered. This means sets do not record element position. Sets only have unique elements. This means there is only one of a particular element in a set. To define a set, you use curly brackets. You place the elements of a set within the curly brackets. You notice there are duplicate items. When the actual set is created, duplicate items will not be present. You can convert a list to a set by using the function set, this is called type casting. You simply use the list as the input to the function set. The result will be a list converted to a set. Let's go over an example. We start off with a list. We input the list to the function set. The function set returns a set. Notice how there are no duplicate elements. Let's go over set operations. These could be used to change the set. Consider the set A. Let's represent this set with a circle. If you are familiar with sets, this could be part of a venn diagram. A venn diagram is a tool that uses shapes usually to represent sets. We can add an item to a set using the add-method. We just put the set name followed by a dot, then the add-method. The argument is the new element of the set we would like to add, in this case, NSYNC. The set A now has in NSYNC as an item. If we add the same item twice, nothing will happen as there can be no duplicates in a set. Let's say we would like to remove NSYNC from set A. We can also remove an item from a set using the remove-method. We just put the set name followed by a dot, then the remove-method. The argument is the element of the set we would like to remove, in this case, NSYNC. After the remove-method is applied to the set, set A does not contain the item NSYNC. You can use this method for any item in the set. We can verify if an element is in the set using the in command as follows. The command checks that the item, in this case AC/DC, is in the set. If the item is in the set, it returns true. If we look for an item that is not in the set, in this case for the item Who, adds the item is not in the set, we will get a false. These are types of mathematical set operations. There are other operations we can do. There are lots of useful mathematical operations we can do between sets. Let's define the set album set one. We can represent it using a red circle or venn diagram. Similarly, we can define the set album set two. We can also represent it using a blue circle or venn diagram. The intersection of two sets is a new set containing elements which are in both of those sets. It's helpful to use venn diagrams. The two circles that represent the sets combine, the overlap, represents the new set. As the overlap is comprised with the red circle and blue circle, we define the intersection in terms of and. In Python, we use an ampersand to find the intersection of the two sets. If we overlay the values of the set over the circle placing the common elements in the overlapping area, we see the correspondence. After applying the intersection operation, all the items that are not in both sets disappear. In Python, we simply just place the ampersand between the two sets. We see that both AC/DC and Back in Black are in both sets. The result is a new set album: set three containing all the elements in both albums set one and album set two. The union of two sets is the new set of elements which contain all the items in both sets. We can find the union of the sets album set one and album set two as follows. The result is a new set that has all the elements of album set one and album set two. This new set is represented in green. Consider the new album set-album set three. The set contains the elements AC/DC and Back in Black. We can represent this with a Venn diagram, as all the elements and album set three are in album set one. The circle representing album set one encapsulates the circle representing album set three. We can check if a set is a subset using the issubset method. As album set three is a subset of the album set one, the result is true.