Complex Data Types in Python: Working with Dictionaries & Sets in Python

This 9-video course helps learners explore dictionary data type in Python. Dictionaries are associative containers used to store key-value pairs. Given a key, finding the associated value is optimized by Python to be extremely efficient. First, receive an introduction to dictionaries in Jupyter Notebook in Python. You will learn how to create and initialize dictionaries, then learn about nesting complex data types within dictionaries. Continuing with the study of Python dictionaries, you will explore what functions and methods can be invoked on these dictionaries, such as modifying and updating dictionaries using dictionary methods. Next, you will be introduced to sets, another commonly used complex data type that Python supports. You will then create and initialize sets. This leads on to performing set operations such as union, intersection difference, and other set operations. You will also examine nested lists, and work with nested types within other complex data types. In the final tutorial, you will learn how to convert lists to dictionaries and vice versa. The concluding exercise entails recalling features of dictionaries and sets.

Table of Contents

Course Overview

[Video description begins] *Topic title: Course Overview* [Video description ends]

Hello there, welcome to this course working with dictionaries and sets in Python.

[Video description begins] *Your host for this session is Janani Ravi. She is a software engineer and big data expert.* [Video description ends]

My name is Janani Ravi, and I'll be your instructor for this course. A little about myself first. I am a co-founder at Loonycorn, a studio for high-quality video content. Before founding Loonycorn, I worked at various companies, including Google for seven years in New York and Singapore, and Microsoft for three years in Redmond. And before that, I attended grad school at Stanford. In this course, you will continue your exploration of fundamental building blocks of Python, now moving on specifically to working with dictionaries.

This course assumes that you have an interactive Python development environment such as Jupyter installed, and are familiar with defining variables and invoking simple built-in functions. You'll also need to be familiar with working with lists, and with operations such as slicing. In this course, you'll work with the dictionary, another extremely important data type in Python. You'll see how dictionaries are associative containers used to store key-value pairs. Given a key, finding the associated value is optimized by Python to be extremely efficient.

You'll see how all keys in a dictionary need to be unique, while values can contain duplicates. Then you'll move on to sets, which are modeled on the concept of sets in mathematics. Sets share similarities with both dictionaries

and lists. Like lists, sets are single-dimensional containers. Like the keys in dictionaries, duplicates are not allowed in sets.

By the end of this course, you will be able to create dictionaries and sets, insert data into them, and access and manipulate the data. You will know how to find the key that corresponds to a particular value, how to check whether a particular key is present or not, and how to go from dictionaries to list of tuples, and vice versa.

Introducing Dictionaries

[Video description begins] *Topic title: Introducing Dictionaries. Your host for this session is Janani Ravi.* [Video description ends]

An important complex data type, which you'll often find use for in Python is the dictionary. And, the dictionary is what we'll study here in this video.

[Video description begins] *A Jupyter Notebook named PythonDictionary-01 displays in the web browser. On the top-left side of the interface, various tabs, such as File, Edit, View, and so on, appear. Below these tabs, various buttons are available. Under these buttons, various input boxes display. Further, a Logout button is available at the top-right of this interface.* [Video description ends]

Let's first instantiate an empty dictionary.

[Video description begins] *She enters the following code in the first input box: empty_dict = {}* [Video description ends]

Observe that a dictionary is basically just curly brackets. So, anytime you specify a dictionary, you have to use the curly braces in Python.

[Video description begins] *Now, she enters the following code in the second input box: empty_dict. Next, she executes the entered code. The following output displays: {}* [Video description ends]

A dictionary in Python can be best defined as a mapping of key-value pairs. And, a dictionary allows you to look up a particular value once you specify the key. Let's see how this is set up. Here, we're going to instantiate a dictionary of bike_owners.

[Video description begins] *She enters the following code in the third input box: bike_owners = {"James" : "Ducati Monster 1200", "Jacob" : "Ducati Scrambler 1100", "William" : "BMW S 1000 RR", "Aiden" : "Harley Davidson"} bike_owners* [Video description ends]

Notice that all of the key-value pairs are specified within curly brackets. That is important, that is what indicates to Python that this is a dictionary. And, all of the keys in this dictionary are the names of owners of motorcycles. James, Jacob, William, and Aiden. And, all of the values in this dictionary are names of bikes.

The Ducati Monster 1200, the Harley Davidson, the BMW, and so on. A record in a dictionary is a key-value mapping. And individual records are separated from one another using commas. So, one record is the key James, and the value, the Ducati Monster 1200 that James owns. And, notice keys and values are separated using the colon.

And, a record that is a key-value mapping, that is the record for the key James, is separated from the record for the key Jacob using a comma. This is a dictionary, and let's take a look at the resulting dictionary that was created.

[Video description begins] *She executes the entered code. The following output displays: {'James' : 'Ducati Monster 1200', 'Jacob' : 'Ducati Scrambler 1100', 'William' : 'BMW S 1000 RR', 'Aiden' : 'Harley Davidson'}* [Video description ends]

Now, the cool thing about a dictionary is that you can specify a key within square brackets and get the corresponding value. So, if you specify bike_owners, square bracket, James, you get the Ducati Monster 1200, which is the bike that James owns.

[Video description begins] *Next, she enters the following code in the fourth input box: bike_owners["James"]. Then, she executes this code. The following output displays: 'Ducati Monster 1200'* [Video description ends]

If you want to find out what bike Aiden owns, you'll specify Aiden within square brackets, and the dictionary will return the Harley Davidson.

[Video description begins] *Next, she enters the following code in the fifth input box: bike_owners["Aiden"]. Then, she executes this code. The following output displays: 'Harley Davidson'. Also, a screenshot of the code entered in the input box 3 displays on the right side of the screen. In this code, the following code segment is highlighted: "Aiden" : "Harley Davidson"* [Video description ends]

It's important to remember that dictionaries allow you to look up a particular value using the corresponding keys.

[Video description begins] *She enters the following code in the sixth input box: bike_owners["BMW S 1000 R"]. Also, a screenshot of the code entered in the input box 3 displays on the right side of the screen. In this code, the following code segment is highlighted: "William" : "BMW S 1000 RR",* [Video description ends]

If you try to specify a value within your dictionary, this is not valid. There is a KeyError that Python throws, indicating that the BMW S 1000 R is not a valid key within that dictionary. Even though the BMW is a part of this dictionary, it is a value. It is not recognized as a key, and that's why you see this error.

[Video description begins] *She executes the entered code and the following Key Error displays on the screen: 'BMW S 1000 R'* [Video description ends]

The most common data type used as keys in a dictionary are strings. However, you can specify integer values as your keys as well.

[Video description begins] *Next, she enters the following code in the seventh input box: int_dict = {1 : 45, 2 : 55, 3 : 65} int_dict* [Video description ends]

Here is a dictionary where all of the keys are integer values. 1, 2, and 3 are the dictionary keys. It so happens that the values corresponding to the keys of this dictionary also happened to be integers.

[Video description begins] *She executes the code entered in the seventh input box. The following output displays: {1 : 45, 2 : 55, 3 : 65}* [Video description ends]

When the keys of a dictionary are integers, you need to specify these integers within square brackets to look up corresponding values.

[Video description begins] *She enters the following code in the eighth input box: int_dict[2]* [Video description ends]

So, the value corresponding to the integer 2 is 55.

[Video description begins] *She executes the code entered in the eighth input box. The following output displays: 55* [Video description ends]

Now, it's important to remember that the keys in a dictionary have to be unique. That's the only way you can use a key to look up a particular value.

[Video description begins] *She enters the following code in the ninth input box: int_dict = {1 : 45, 2 : 55, 3 : 85, 3 : 75} int_dict* [Video description ends]

Observe this dictionary that we've created here. You can see that the keys are 1, 2, 3, and 3. So, the key 3 appears twice. The key 3 is not unique. Now, if you instantiate this in dictionary, you can see in the resulting dictionary, the key 3 appears exactly once, and it's associated with the value 75.

[Video description begins] *She executes the entered code. The following output displays: {1 : 45, 2 : 55, 3 : 75}* [Video description ends]

This means that Python has simply taken the last value assigned to that key. The first value that we assigned to the key 3 was 85. That has been overwritten by the last value assigned to that key, which is the number 75. Let's now set up another dictionary containing some student_details.

[Video description begins] *She enters the following code in the tenth input box: student_details = {"Henry" : 1995, "Samuel" : 1999, "Ingrid" : 2005 } student_details* [Video description ends]

Here the keys of this dictionary are strings and the values are numbers representing the year in which these students were born.

[Video description begins] *She executes the entered code. The following output displays on the screen: {'Henry' : 1995, 'Samuel' : 1999, 'Ingrid' : 2005 }* [Video description ends]

Just as list and tuples, there are functions that you can invoke on dictionaries as well. The dot keys function in a dictionary will give you a list of all of the keys present in that dictionary.

[Video description begins] *Next, she enters the code: student_details.keys() in the eleventh input box and then executes it. The following output displays on the screen: dict_keys (['Henry' , 'Samuel' , 'Ingrid'])* [Video description ends]

These are the names of our students, Henry, Samuel, and Ingrid. You can use the Python in keyword to check whether a particular key is present in the dictionary. Here, we want to know whether the name Samuel is present in the student_details dictionary. We check within keys, and Python says true because Samuel is indeed a key.

[Video description begins] *Next, she enters the code: "Samuel" in student_details.keys() in the twelfth input box and then executes it. The following output displays on the screen: True* [Video description ends]

This way, we can check for other student names as well. Is Andrew within the student_details keys? No, that's why Python returns false here.

[Video description begins] *Next, she enters the code: "Andrew" in student_details.keys() in the thirteenth input box and then executes it. The following output displays on the screen: False* [Video description ends]

Let's check for Henry, and Python correctly returns true.

[Video description begins] *Similarly, she enters the "Henry" in student_details.keys() code in the fourteenth input box and then executes it. The following output displays on the screen: True* [Video description ends]

There is another helpful function that Python dictionaries have, and that is the values function.

[Video description begins] *She enters the following code in the fifteenth input box: student_details.values()* [Video description ends]

And, this function allows you to access all of the values present within the dictionary. Here are all of the years in which these students were born.

[Video description begins] *She executes the entered code. The following output displays on the screen: dict_values([1995, 1999, 2005]). Also, a screenshot of the code entered in the input box 10 displays on the right*

side of the screen. In this code, the following code segments are highlighted: 1995, 1999, and 2005. [Video description ends]

It's not necessary in Python that all of the keys and values for a dictionary be of the same type. We can have a mix dictionary. Notice here that the keys are a Boolean value and strings, and the values are a string, a list, and a Boolean value.

[Video description begins] *She enters the following code in the sixteenth input box: mixed_dict = {False: "Daniel", "Aria" : [1,2,3], "Jacob" : True } mixed_dict* [Video description ends]

Now, this is a dictionary that may be hard to use, but it's perfectly valid. And Python has accepted it and created this dictionary for you.

[Video description begins] *She executes the entered code. The following output displays on the screen: {False: 'Daniel', 'Aria': [1, 2, 3], 'Jacob': True}* [Video description ends]

Let's look up the value associated with the key Aria, that is the list 1, 2, 3. Let's perform another look up here.

[Video description begins] *She enters the mixed_dict["Aria"] code in the seventeenth input box. Next, she executes this code. The following output displays on the screen: [1, 2, 3]* [Video description ends]

We'll look up the value associated with the Boolean false, which is the name Daniel.

[Video description begins] *She enters the code, mixed_dict[False], in the eighteenth input box. Next, she executes this code. The following output displays on the screen: 'Daniel'* [Video description ends]

All of these work perfectly well. Dictionaries in Python are often used to hold different kinds of information about an entity. In this particular dictionary that we've created, the entity is a bike.

And, the details of the bike are the name of the bike_owner, the model, the price, and the displacement of its engine. The keys indicate the properties of the entry, and the values are the values for those properties. Here are the bike_details.

[Video description begins] *She enters the following code in the nineteenth input box: bike_details = {"bike_owner" : "James Smith", "bike_model" : "Ducati Monster 1200", "bike_price" : 28140, "engine_displacement" : 1197} bike_details. Then, she executes this code. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 28140, 'engine_displacement' : 1197}* [Video description ends]

Once we've created a dictionary, it's possible to set up new key-value mappings in that dictionary. You simply specify the square brackets, and the name of the key, and assign a value.

[Video description begins] *Next, she enters the code, bike_details['num_cylinders'] = 2, in the twentieth input box.* [Video description ends]

If you take a look at the bike_details dictionary now, you'll find that it's key-value mappings have been updated to include the key-value, num_cylinders and 2.

[Video description begins] *In the twenty-first input box, she enters the code, bike_details. Then she executes the entered code. The following output displays: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 28140, 'engine_displacement' : 1197, 'num_cylinders' : 2}* [Video description ends]

In addition to adding new key-value pairs to a dictionary, you can also update the value associated with a particular key. Here, I want to update the price of this bike to $29,140. And, if you take a look at the bike_price, you'll find that the corresponding price has been updated.

[Video description begins] *She enters the following lines of code in the twenty-second and twenty-third input box: bike_details['bike_price'] = 29140 bike_details['bike_price']. Then, she executes the entered code. The following output displays on the screen: 29140. Also, the code entered in the input box 21 displays on the right side of the screen along with its output. In the output code, the bike price, 28140, which was initially entered by the host is highlighted.* [Video description ends]

In addition to using square brackets, you can also use the .get function on a dictionary to look up the value associated with the key. Call .get, pass in a key, and you'll get the value in return.

[Video description begins] *She enters the code, bike_details.get('bike_price'), in the twenty-fourth input box and then executes it. The following output displays on the screen: 29140* [Video description ends]

Within the bike_details dictionary, the price of our bike has been updated to reflect the new value that we assigned. You can take a look at bike_details, and find the update in here as well.

[Video description begins] *She enters the code, bike_details, in the twenty-fifth input box and then executes it. The following output is generated: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 29140, 'engine_displacement' : 1197, 'num_cylinders' : 2}. Also, a screenshot of the code entered in the input box 21 displays on the right side of the screen along with its output. In the output code, the following code segment is highlighted: 'bike_price' : 28140* [Video description ends]

Dictionaries in Python, unlike tuples, are mutable. If you find that a particular key-value pair no longer makes sense within a dictionary, you can call the del command or the del keyword, and pass in the key that you want to delete.

Here, I want to get rid of the engine_displacement key. And, if you take a look at the bike_details, you'll find that the engine_displacement key and the corresponding value has disappeared from this dictionary.

[Video description begins] *She enters the following lines of code in the twenty-sixth and twenty-seventh input box: del bike_details["engine_displacement"] bike_details. Then, she executes this code. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 29140, 'num_cylinders' : 2}. Also, a screenshot of the code entered in the input box 25 displays on the right side of the screen along with its output. In the output code, the following code segment is highlighted: 'engine_displacement' : 1197,* [Video description ends]

Nesting Complex Data Types Within Dictionaries

[Video description begins] *Topic title: Nesting Complex Data Types Within Dictionaries. Your host for this session is Janani Ravi.* [Video description ends]

Let's continue working with Python dictionaries, and see how dictionaries can hold within it other complex data types, such as tuples, lists, or strings.

[Video description begins] *A Jupyter Notebook named PythonDictionary-02 displays in the web browser.* [Video description ends]

I'll first create a completely empty dictionary called fruit_qty. Remember, the empty dictionary is simply specified by an empty pair of curly braces.

[Video description begins] *She enters the code, fruit_qty = {} fruit_qty, in the first input box and then executes it. The following output displays on the screen: {}* [Video description ends]

Once this empty dictionary has been created, you can add key-value pairs to this dictionary using the assignment operators. Here, all of the keys are names of fruits such as Banana, Apple, and Apricot. And we've used the

assignment statement to assign the quantity of fruit available. If you now take a look at the fruit_qty dictionary, it's no longer empty. The key Banana is associated with the value 50, Apple with 78, and Apricot with 40.

[Video description begins] *First, she enters the following code in the second input box: fruit_qty['Banana'] = 50 fruit_qty['Apple'] = 78 fruit_qty['Apricot'] = 40. Then, she enters the code, fruit_qty, in the third input box. Now, she executes the entered code. The following output displays on the screen: {'Banana' : 50, 'Apple' : 78, 'Apricot': 40}* [Video description ends]

You can use the square bracket notation to update and set up new key-value pairs. I've added Orange and Avocado to the fruit_qty dictionary, and these fruits are now also available within fruit_qty. Orange and Avocado are new keys, which are present along with their corresponding values.

[Video description begins] *She enters the following code in the fourth input box: fruit_qty['Orange']= 70 fruit_qty['Avocado']= 30 fruit_qty. Then, she executes the entered code. The following output displays on the screen: {'Banana' : 50, 'Apple' : 78, 'Apricot': 40, 'Orange' : 70, 'Avocado': 30 }* [Video description ends]

As we've seen earlier, you can now use the square bracket notation, and within the square bracket, specify the exact form of the key whose value you want to look up within the dictionary. What is the quantity associated with the key Apricot? Well, it's 40, and that's what Python gives us.

[Video description begins] *She enters the code, fruit_qty = ['Apricot'], in the fifth input box, and then executes it. The following output displays on the screen: 40* [Video description ends]

If you're using string keys within your dictionary, remember that they are case sensitive, so you have to specify the key in its exact form.

[Video description begins] *She enters the following code in the sixth input box: fruit_qty = ['Watermelon']* [Video description ends]

If you try to look up a value for a key that does not exist within your dictionary, Python will give you a KeyError. Python looked through your fruit_qty dictionary, could not find a key Watermelon, which is why it threw an error here.

[Video description begins] *She executes the entered code. The following KeyError displays on the screen: 'Watermelon'* [Video description ends]

You can now update this dictionary to include the key Watermelon, and you have 10 Watermelons within your fruit_qty. You can see that the fruit_qty dictionary now includes Watermelon. If you now try to look up the key Watermelon, you'll get its corresponding value, 10.

[Video description begins] *She enters the code, fruit_qty = ['Watermelon'] = 10, in the seventh input box, and then executes it. The following output displays on the screen: {'Banana' : 50, 'Apple' : 78, 'Apricot': 40, 'Orange' : 70, 'Avocado': 30, 'Watermelon' : 10 }* [Video description ends]

[Video description begins] *She enters the code, fruit_qty = ['Watermelon'], in the eighth input box, and then executes it. The following output displays on the screen: 10* [Video description ends]

In this particular dictionary that we worked with, the keys were strings, and the values were integers. It's also possible to have values be complex data types. Here, I'm going to have a dictionary called fruit_qty_consumed, and the keys are strings which are fruits, Banana, Apple, Apricot, Orange, and Avocado and the values are lists. For each of these fruits, we've associated a list of values, and this is perfectly legit and allowed in Python.

[Video description begins] *She enters the following code in the ninth input box: fruit_qty_consumed = {'Banana' : [50, 60, 40, 55], 'Apple' : [78, 86, 80, 60], 'Apricot' : [40, 70, 30, 55], 'Orange' : [70, 80, 60, 65], 'Avocado' : [30, 50, 40, 45]}* [Video description ends]

If you take a look at the fruit_qty_consumed, you'll see that all of the keys are values, which are the quantities of fruits that were sold on different days of the week sale.

[Video description begins] *She enters the code, fruit_qty_consumed, in the tenth input box, and then executes it. The following output displays on the screen: {'Banana' : [50, 60, 40, 55], 'Apple' : [78, 86, 80, 60], 'Apricot' : [40, 70, 30, 55], 'Orange' : [70, 80, 60, 65], 'Avocado' : [30, 50, 40, 45]}* [Video description ends]

If you want to find the value associated with a particular key, you'll use square brackets and specify the key that you want to look up. Python returns the list of oranges consumed on, say, different week days, we don't know what the week days are, this is just a list.

[Video description begins] *She enters the code, fruit_qty_consumed['Orange'], in the eleventh input box, and then executes it. The following output displays on the screen: [70, 80, 60, 65]* [Video description ends]

Remember that lists can also be accessed using index positions specified within square brackets. So you can access fruit, Orange. And within that list, you want to access the quantity consumed at index position 3, and Python will give you 65. The value at index position 3 in the list associated with the Orange key.

[Video description begins] *She enters the code, fruit_qty_consumed['Orange'] [3], in the twelfth input box, and then executes it. The following output displays on the screen: 65* [Video description ends]

I'll now access the list associated with the key, Avocado. And this will give me the list 30, 50, 40, and 45. Now, if I want to update a particular element in this list, I can do so using two square brackets.

[Video description begins] *Now, she enters the code, fruit_qty_consumed['Avocado'], in the thirteenth input box. Then, she executes the entered code. The following output displays on the screen: [30, 50, 40, 45]* [Video description ends]

The first set of square brackets has the key Avocado, that will help me look up the list associated with the Avocado key. And the second set of square bracket specifies the index position of the element that I want to update. I'm going to update this value to be 50. And now, if you look up the list associated with Avocado, you'll find that the last element at index position 3 in this resulting list is 50. This is the updated value that we had specified.

[Video description begins] *Next, she enters the code, fruit_qty_consumed['Avocado'][3] = 50, in the fourteenth input box and the code, fruit_qty_consumed['Avocado'] in the fifteenth input box. Then, she executes the entered code. The following output displays on the screen: [30, 50, 40, 50]. Also, the code entered in the input box 10 displays on the right side of the screen along with its output. In the output, the last value, 45, at the index position 3 of the fruit, Avocado, is highlighted* [Video description ends]

The values of your dictionaries can be dictionaries as well. So dictionaries can have any kind of complex data types in its values.

[Video description begins] *Next, she enters the following code in the sixteenth input box: fruit_qty_consumed = { 'Banana' : {'Fri' : 70, 'Sat' : 80, 'Sun' : 95}, 'Apple' : {'Fri' : 90, 'Sat' : 95, 'Sun' : 110}, 'Apricot' : {'Fri' : 70, 'Sat' : 85, 'Sun' : 95}, 'Orange' : {'Fri' : 80, 'Sat' : 90, 'Sun' : 100}, 'Avocado' : {'Fri' : 60, 'Sat' : 70, 'Sun' : 80} }* [Video description ends]

Here is an updated version of the fruit_qty_consumed dictionary. Here the keys are names of fruits, and the values are dictionaries where the keys within this nested dictionary are the days on which these fruits were consumed. Let's take a look at the fruit, Banana. On Friday, there were 70 bananas consumed, on Saturday, 80, and on Sunday, 95. The fruit_qty_consumed here is an example of a dictionary nested within an outer dictionary.

[Video description begins] *She enters the code, fruit_qty_consumed, in the seventeenth input box, and then executes it. The following output displays on the screen: { 'Banana' : {'Fri' : 70, 'Sat' : 80, 'Sun' : 95}, 'Apple' :*

{'Fri' : 90, 'Sat' : 95, 'Sun' : 110}, 'Apricot' : {'Fri' : 70, 'Sat' : 85, 'Sun' : 95}, 'Orange' : {'Fri' : 80, 'Sat' : 90, 'Sun' : 100}, 'Avocado' : {'Fri' : 60, 'Sat' : 70, 'Sun' : 80} } [Video description ends]

If you look up the fruit_qty_consumed dictionary with one set of square brackets, this will give you the dictionary of how many apples were consumed on the different days of the week.

[Video description begins] *She enters the code, fruit_qty_consumed['Apple'], in the eighteenth input box, and then executes it. The following output displays on the screen: {'Fri' : 90, 'Sat' : 95, 'Sun' : 110}* [Video description ends]

You can use two square brackets to check up how many Apricot consumed only on Sunday, and you get the result 95.

[Video description begins] *Next, she enters the code, fruit_qty_consumed['Apricot']['Sun'] in the nineteenth input box, and then executes it. The following output displays on the screen: 95. Also, the code entered in the input box 17 displays on the right side of the screen along with its output. In the output, the value, 95, in the code segment, 'Apricot' : {'Fri' : 70, 'Sat' : 85, 'Sun' : 95},, is highlighted.* [Video description ends]

The first pair of square brackets looks up the outer dictionary. That is we use Apricot to get the dictionary associated with Apricot. And then we use the second pair of square brackets to get the value associated with Sunday in the nested dictionary. Let's take another example of a nested look up. Here we have the Apricot fruit present within the outer dictionary, but we have no fruit_qty_consumed for Monday, and Python rightfully throws an error. The key Monday wasn't found within the inner nested dictionary.

[Video description begins] *Next, she enters the code, fruit_qty_consumed['Apricot']['Mon'] in the twentieth input box, and then executes it. The following KeyError displays on the screen: 'Mon'. Also, the code entered in the input box 17 displays on the right side of the screen along with its output. In the output, the following code segment for the fruit Apricot is highlighted : {'Fri' : 70, 'Sat' : 85, 'Sun' : 95},* [Video description ends]

Invoking Functions on Dictionaries

[Video description begins] *Topic title: Invoking Functions on Dictionaries. Your host for this session is Janani Ravi.* [Video description ends]

Let's continue our study of Python dictionaries by seeing what functions and methods can be invoked on these dictionaries.

[Video description begins] *A Jupyter Notebook named FunctionsAndMethodsInDictionaries displays in the web browser.* [Video description ends]

I'm going to instantiate the dictionary that we worked with before, the bike_details dictionary containing the owner, model, price, and engine_displacement for a bike.

[Video description begins] *She enters the following code in the first input box: bike_details = {"bike_owner" : "James Smith", "bike_model" : "Ducati Monster 1200", "bike_price" : 28140, "engine_displacement" : 1197} bike_details. Then, she executes this code. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 28140, 'engine_displacement' : 1197}* [Video description ends]

Dictionaries can be passed as input arguments to the built-in len function, and this will give you the number of records or key-value pairs in a dictionary.

[Video description begins] *She enters the code, len(bike_details), in the second input box, and then executes it. The following output displays on the screen: 4* [Video description ends]

The bike_details dictionary has four key-value pairs, which is why its length is four. We've already seen the keys function that you can invoke on a dictionary to access all of the keys present.

[Video description begins] *She enters the code, len(bike_details.keys()), in the third input box, and then executes it. The following output displays on the screen: 4* [Video description ends]

You can see the length of the keys in this dictionary is four, and the length of the values in the bike_details dictionary is also four. The built-in function sorted sorts the values present in a collection such as a list or a tuple and return sorted values.

[Video description begins] *Next, she enters the following code in the fifth input box: sorted(bike_details)* [Video description ends]

When you pass in a dictionary, it'll return a sorted list of the keys in the dictionary.

[Video description begins] *She executes the entered code. The following output displays on the screen: ['bike_model', 'bike_owner', 'bike_price', 'engine_displacement']. Also, the code written in the input box 1 displays on the right side of the screen.* [Video description ends]

The sorted function has accessed all of the keys in bike_details, and return the keys in the sorted lexicographical order. The built-in function sorted, takes in an additional input argument as well if you want to sort your collection in the reverse order. Pass in reverse=True, and the keys in your bike_details dictionary will be sorted in the reverse order, reverse lexicographical order. engine_displacement appears first.

[Video description begins] *She enters the following code in the sixth input box: sorted(bike_details, reverse=True). Next, she executes this code. The following output is generated: ['engine_displacement', 'bike_price', 'bike_owner', 'bike_model'].* [Video description ends]

If you want to sort the values in a particular dictionary, access the dictionary values by calling the .values function and pass this as an input argument to the sorted function. Now you'll see that this is an error.

[Video description begins] *She enters the following code in the seventh input box: (bike_details.values())* [Video description ends]

This is because the values of your dictionary are made up of both numbers as well as strings. And the sorted function doesn't know how to compare a string with a number. What does it mean for a string to be less than a number?

[Video description begins] *She executes the entered code. The following TypeError displays on the screen: '<' not supported between instances of 'int' and 'str'* [Video description ends]

There is no answer, which is why sorted returns a TypeError. Instead of accessing keys and values individually in a dictionary, it's possible to access key-value pairs in the form of tuples by calling the .items function on a dictionary. .items will return key-value pairs where every key-value pair is in the form of a tuple. Observe that items returns a list of key-value pairs in the form of tuples, bike_owner and James Smith is a tuple. bike_model and the Ducati Monster 1200 is a tuple.

[Video description begins] *She enters the code, bike_details.items(), in the ninth input box, and then executes it. The following output displays on the screen: dict_items([('bike_owner', 'James Smith'), ('bike_model', 'Ducati Monster 1200'), ('bike_price', 28140), ('engine_displacement', 1197)])* [Video description ends]

Python dictionaries also have a copy function. When you invoke the copy function on a dictionary, it will return a copy of the original dictionary which we stored in the copy_bike_details variable. copy_bike_details contains the same key-value pairs that are present in the original bike_details dictionary. Now, there are several nuances to copying with complex data types, which we'll address a little later on.

[Video description begins] *First, she enters the code, copy_bike_details = bike_details.copy(), in the tenth input box, and then she enters the code, copy_bike_details, in the eleventh input box. Then, she executes the entered code. The following output displays on the screen: {'bike_owner': 'James Smith', 'bike_model': 'Ducati Monster 1200', 'bike_price': 28140, 'engine_displacement': 1197}. Also, the code written in the input box 1 displays on the right side of the screen.* [Video description ends]

Python dictionaries also support a pop function. Invoking the pop function on a dictionary will remove the key that we have specified in the dictionary and return the value. engine_displacement key that we passed in, that record will be removed from the copy_ bike_details dictionary, and the pop function will simply return the value, 1197, here corresponding to the engine_displacement key. This returned value can then be stored in a variable if we want to.

[Video description begins] *She enters the code, copy_bike_details.pop('engine_displacement'), in the twelfth input box, and then executes it. The following output displays on the screen: 1197* [Video description ends]

If you take a look at copy_bike_details, you will find that the engine_displacement key-value pair has disappeared. It has been popped from this dictionary.

[Video description begins] *She enters the code, copy_bike_details, in the thirteenth input box, and then executes it. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 28140}. Also, the code written in the input box 11 along with its output displays on the right side of the screen. Further, in the output, the following code segment is highlighted: 'engine_displacement': 1197}* [Video description ends]

If you take a look at the original dictionary from which we made a copy, you'll find that the engine_displacement key-value pair is still present within bike_details. It has been popped, only from copy_bike_details.

[Video description begins] *She enters the code, bike_details, in the fourteenth input box, and then executes it. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 28140, 'engine_displacement' : 1197}* [Video description ends]

Changing the contents of a copy of a dictionary created using the copy function did not affect the original dictionary. This is something you need to remember. Now, let's go ahead and pop the engine_displacement key from the copy_bike_details dictionary once again. And this will be an error because this key is no longer present within copy_bike_details.

[Video description begins] *She enters the code, copy_bike_details.pop('engine_displacement'), in the fifteenth input box, and then executes it. The following KeyError displays on the screen: 'engine_displacement'* [Video description ends]

Dictionaries in Python also support a popitem function, which arbitrarily removes any key-value pair from a dictionary. So at random from copy_bike_details, the popitem function has picked the bike_price, and its corresponding value to pop or remove from the dictionary.

[Video description begins] *She enters the code, copy_bike_details.popitem(), in the sixteenth input box, and then executes it. The following output displays on the screen: ('bike_price', 28140) Also, the code written in the input box 13 displays on the right side of the screen along with its output. In the output, the code segment, 'bike_price' : 28140}, is highlighted.* [Video description ends]

If you take a look at copy_bike_details now, you'll find that the key bike_price and its corresponding value is no longer present within this dictionary. The entire item, that is the key-value pair, has been popped.

[Video description begins] *She enters the code, copy_bike_details, in the seventeenth input box, and then executes it. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati*

Monster 1200'}. Also, the code written in the input box 13 displays on the right side of the screen along with its output. In the output, the code segment, 'bike_price' : 28140}, is highlighted. [Video description ends]

Now it's possible that when you execute this code, that another key-value pair has been picked at random. It may not necessarily be the bike_price key. Dictionaries also support an update function. And in order to see how that works, let's set up two dictionaries with the names of individuals and their corresponding ages.

[Video description begins] *Now, she enters the following code in the eighteenth input box: dict_age = {"Ethan": 54, "Sofia": 50} new_dict_age = {"Ethan": 55, "Harper": 58}* [Video description ends]

We have the dict_age and the new_dict_age. Notice that there is one key in common between both of these dictionaries. The key for the person, Ethan. We'll now call the update function on the dict_age dictionary, .update, and we pass in another dictionary to the update function as its input argument. And, if you take a look at dict_ age, you'll find that the keys of this dictionary have been updated with the values specified in the new_dict_age.

[Video description begins] *She enters the following code in the nineteenth input box: dict_age.update(new_dict_age) dict_age. Then, she executes the entered code. The following output displays on the screen: {'Ethan' : 55, 'Sofia' : 50, 'Harper' : 58}* [Video description ends]

Ethan was originally 54 years old in dict_age. Now, Ethan's age has been updated to be 55. This is the value associated with Ethan in the new_dict_age dictionary that we use to update the original dictionary. We can also see that the key for Harper, which was not originally present in dict_age, is now part of the dict_age dictionary. Let's take a look at two of the dictionaries that we've been working with copy_bike_details now contain two keys, bike_owner and bike_model and their corresponding values.

[Video description begins] *She enters the code, copy_bike_details, in the twentieth input box, and then executes it. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200'}.* [Video description ends]

The original bike_details dictionary contains four keys and corresponding values.

[Video description begins] *She enters the code, bike_details, in the twenty-first input box, and then executes it. The following output displays on the screen: {'bike_owner' : 'James Smith', 'bike_model' : 'Ducati Monster 1200', 'bike_price' : 28140, 'engine_displacement' : 1197}* [Video description ends]

If you want to completely empty a dictionary of all of the key-value pairs that it holds, you can call the clear function.

[Video description begins] *She enters the code, bike_details.clear(), in the twenty-second input box.* [Video description ends]

I'll call bike_details.clear, and if you take a look at bike_details, you'll find that it's now a completely empty dictionary.

[Video description begins] *Next, she enters the code, bike_details in the twenty-third input box. Then, she executes the entered code. The following output displays on the screen: {}* [Video description ends]

Instead of just clearing the elements in a dictionary, if you no longer want a reference to that dictionary, you can use the del command to free up all the memory associated with that dictionary.

[Video description begins] *She enters the del copy_bike_details code in the twenty-fourth input box.* [Video description ends]

Once you use the del command on copy_bike_details, you can no longer access that variable.

[Video description begins] *Next, she enters the copy_bike_details code in the twenty-fifth input box and then executes it. The following NameError displays on the screen: name 'copy_bike_details' is not defined* [Video description ends]

Introducing Sets

[Video description begins] *Topic title: Introducing Sets. Your host for this session is Janani Ravi.* [Video description ends]

Another commonly used complex data type that python supports is the set. A set is a collection of elements where the elements are unique. A set can contain no duplicates, and a set has no intrinsic order.

[Video description begins] *A Jupyter Notebook named PythonSet-01 displays in the web browser.* [Video description ends]

Here is how we initialize a set in Python, set_string is a set.

[Video description begins] *She enters the following code in the first input box: set_string = {"Emma", "Olivia", "Ava", "Mia"} set_string* [Video description ends]

Observe that we use the curly braces, as we did with dictionaries. But the elements of a set are not key-value pairs, they are just simple elements, Emma, Olivia, Ava, and Mia. This is a set of strings.

[Video description begins] *She executes the entered code. The following output displays on the screen: {'Ava','Emma', 'Mia', 'Olivia'}* [Video description ends]

You can pass in this set to the print function, and it'll print out this set within curly brackets.

[Video description begins] *She enters the code, print(set_string), in the second input box, and then executes it. The following output displays on the screen: {'Mia','Ava', 'Olivia', 'Emma'}* [Video description ends]

Observe that the order in which the elements appear within a set changes when you print it out or display it on screen, which means there is no intrinsic ordering of the elements. In the case of a list, the elements will appear in the order in which you assign them to the list. You can create an empty set in Python by simply calling the set function. An empty set will be displayed just as the word set with brackets. This is so that sets can be distinguished from dictionaries, which also use curly braces.

[Video description begins] *She enters the following code in the third input box: empty_set = set() empty_set. Then, she executes the entered code. The following output displays on the screen: set()* [Video description ends]

Any data type that is of type set belongs to the class set, we've spoken of classes before. The only thing that you need to know at this point in time is that classes are templates, which define the functions that you can call on a set and the properties that the set has.

[Video description begins] *She enters the following code in the fourth input box: print(type(empty_set)). Next, the following code displays under the code written in the fourth input box: <class 'set'>* [Video description ends]

A set in Python can have different kinds of elements. Here is a mixed set with a string, an integer, a floating point value, and a Boolean value.

[Video description begins] *She enters the following code in the fifth input box: mixed_set = {"Emma", 5, 1.5, True}* [Video description ends]

As long as the elements within a set are unique, you can have different kinds of elements within the same set.

[Video description begins] *She enters the code, mixed_set, in the sixth input box, and then executes it. The following output displays on the screen: {1.5, 5, 'Emma', True}* [Video description ends]

It's also possible to have a tuple be an element in the set. Here we have all of the primitive data types. In addition, the last element in this set is the tuple containing the fields, 1, 2, and 3.

[Video description begins] *Now, she enters the following code in the seventh input box: mixed_set1 = {"Emma", 5, 1.5, True, (1, 2, 3)} mixed_set1 . Next, she executes this code. The following output displays on the screen: {(1, 2, 3), 1.5, 5, 'Emma', True}* [Video description ends]

Here is something that might be strange to you when you first encounter it. You cannot have a list be part of a set. Here, we have a list, 4, 5, and 6 within the outer set, and you'll find that this results in an error.

[Video description begins] *She enters the code, set_list = {1, 2, 3, [4, 5, 6]}, in the eighth input box, and then executes it. The following TypeError displays on the screen: unhashable type: 'list'* [Video description ends]

This is because the elements in a set in Python can only be immutable values, a tuple, if you remember, is immutable. A list is not, a list can be updated and changed. With just immutable elements, a set can ensure that all of the values that it stores within it are unique. If a value can be updated, the set cannot ensure uniqueness which is why you can only hold immutable values within a set. So if you try creating a set where one of the elements within the set is a dictionary, you will find that this is an error as well. A dictionary is also mutable and can be changed once created.

[Video description begins] *She enters the following code in the ninth input box: set_dict = {1,2,3,{4: "Emma"}}. Next, she executes this code. The following TypeError displays on the screen: unhashable type: 'dict'* [Video description ends]

The only complex data type that we've studied so far that is immutable is the tuple, so you can have a tuple within a set.

[Video description begins] *She enters the following code in the tenth input box: set_tuple = {1, 2, 3, (4, 5, 6)} set_tuple. Next, she executes the entered code. The following output displays on the screen: {(4, 5, 6), 1, 2, 3}* [Video description ends]

Let's set up a set of students. Here are the names of all of the students, observe that the name Emma and Ava appears twice.

[Video description begins] *She enters the following code in the eleventh input box: student_set = {"Emma", "Olivia", "Ava", "Mia", "Emma", "Ava"} student_set* [Video description ends]

If you take a look at the student_set that Python has created, you will find that duplicates have been eliminated. Emma and Ava are present exactly once in the set of students. Here is another set that we've initialized and which contains duplicates, we can see that the numbers 3 and 4 are present twice.

[Video description begins] *She executes the entered code. The following output displays on the screen: {'Ava','Emma', 'Mia', 'Olivia'}* [Video description ends]

[Video description begins] *Next, she enters the following code in the twelfth input box: set_int = {1, 3, 4, 5, 6, 2, 5, 3, 4} set_int* [Video description ends]

If you take a look at the resulting set, the duplicates have been eliminated, and all of the elements are present exactly once.

[Video description begins] *She executes the entered code. The following output displays on the screen: {1 , 2, 3, 4, 5 , 6}* [Video description ends]

We'll now perform a few operations with the student_set that we had set up earlier.

[Video description begins] *She enters the code, student_set, in the thirteenth input box, and then executes it. The following output displays on the screen: {'Ava','Emma', 'Mia', 'Olivia'}* [Video description ends]

A set contains elements in no particular order. So, if you try to access a particular element within a set using an index position, that is an error.

[Video description begins] *She enters the code, student_set[1], in the fourteenth input box, and then executes it. The following TypeError displays on the screen: 'set' object does not support indexing* [Video description ends]

You can only index into complex data types that have an intrinsic ordering, and those data types are list and tuples. Sets have no order, it doesn't support indexing. I've set up a brand new set here called set_1, it contains characters as well as integers, and I've specified a few duplicates during initialization. You take a look at the set that Python created, all of the duplicates have been eliminated.

[Video description begins] *Next, she enters the following code in the fifteenth input box: set_1 = {1, 3, 4, 8, 'a', 6, 2, 10, 3, 4}set_1 . Next, she executes the entered code. The following output displays on the screen: {1, 10, 2, 3, 4, 6, 8, 'a'}* [Video description ends]

The elements 3 and 4 are present exactly once in the set. Now let's go ahead and add a new value to a set by calling set_1.add. Sets can be mutated by invoking functions on sets.

[Video description begins] *Now, in the sixteenth input box, she enters the following code: set_1.add(9) set_1* [Video description ends]

The resulting set now includes the element 9. I'll invoke the add function on the set once again and add in the element 9.5.

[Video description begins] *She executes the entered code. The following output displays on the screen: {1, 10, 2, 3, 4, 6, 8, 9,'a'}* [Video description ends]

9.5 is now a part of this set. Now let's try and add an element to a set, and this element already exists within the set the element 9.

[Video description begins] *Next, she enters the following code in the seventeenth input box: set_1.add(9.5) set_1. Then, she executes the entered code. The following output displays on the screen: {1, 10, 2, 3, 4, 6, 8, 9, 9.5,'a'}* [Video description ends]

If you take a look at set_1, you will find that there is exactly one instance of the number 9 in that set. Python took care of eliminating the duplicate under the hood.

[Video description begins] *In the eighteenth input box, she enters the following code: set_1.add(9) set_1. Then, she executes the entered code. The following output displays on the screen: {1, 10, 2, 3, 4, 6, 8, 9, 9.5,'a'}. Also, the code written in the input box 17 displays on the right side of the screen along with its output. In the output, the value, 9, is highlighted.* [Video description ends]

You can invoke built-in functions on a set as well. Here is a set of teacher names. There are seven different teachers here.

[Video description begins] *In the nineteenth input box, she enters the following code: teachers_set = {"Emma", "Olivia", "Ava", "Mia", "Avery", "Jacob", "Daniel"} teachers_set She executes the entered code. The following output displays on the screen: {'Ava', 'Avery', 'Daniel', 'Emma', 'Jacob', 'Mia', 'Olivia'}* [Video description ends]

Let's invoke the len built-in function on the teachers_set. And you'll see that there are seven elements in that set.

[Video description begins] *She enters the code, len(teachers_set), in the twentieth input box, and then executes it. The following output displays on the screen: 7* [Video description ends]

You can try other built-in function such as max as well. The max function may not make that much sense in a set of strings. It'll simply return the string that is the last in the lexicographical order, that is the string, Olivia.

[Video description begins] *She enters the code, max(teachers_set), in the twenty-first input box, and then executes it. The following output displays on the screen: 'Olivia'* [Video description ends]

The min function will return the string that is first in the lexicographical order, Ava. If you want to get rid of an element in a particular set, one way to do this is to invoke the discard function on the set.

[Video description begins] *She enters the code, min(teachers_set), in the twenty-second input box, and then executes it. The following output displays on the screen: 'Ava'* [Video description ends]

Here we've discarded the name, Emma. The resulting set has just six elements, Emma has been removed.

[Video description begins] *She enters the following code in the twenty-third input box: teachers_set.discard('Emma') teachers_set. The following output displays on the screen: {'Ava', 'Avery', 'Daniel', 'Jacob', 'Mia', 'Olivia'} Also, the code written in the input box 19 displays on the right side of the screen along with its output. In the output, the name, Emma, is highlighted.* [Video description ends]

Another way to get rid of an element within a set is to invoke the remove function on the set, you pass in the element. Remember, the strings have to be an exact match. The teachers_set now no longer contains the element Mia.

[Video description begins] *She enters the following code in the twenty-fourth input box: teachers_set.remove('Mia') teachers_set. The following output displays on the screen: {'Ava', 'Avery', 'Daniel', 'Jacob', 'Olivia'} Also, the code written in the input box 19 displays on the right side of the screen along with its output. In the output, the name, Mia, is highlighted.* [Video description ends]

What's the difference between discard and remove? Well, that's what we'll see right now. I'll call teachers_set.discard and pass in Mia. Remember, Mia is an element that has already been removed from this set. The discard function simply does nothing.

[Video description begins] *Next, she enters the code, teachers_set.discard('Mia'), in the twenty-fifth input box and then executes it. No output is generated.* [Video description ends]

If you try to use the remove function by passing in an element that no longer exists in that particular set, the remove function will throw an error.

[Video description begins] *Now, she enters the code, teachers_set.remove('Mia'), in the twenty-sixth input box and then executes it. The following KeyError displays on the screen: 'Mia'* [Video description ends]

So there's no difference between how discard and remove works for elements that are indeed present in the set. But if the element that you want to remove from a set does not exist, discard will do nothing, remove will throw an error. If you want to get rid of all the elements in a set, simply call the clear function, the teachers_set will now be completely empty. There are no elements present within it.

[Video description begins] *First, she enters the code, teachers_set.clear(), in the twenty-seventh input box. Then, she enters the code, teachers_set, in the twenty-eighth input box. Then, she executes the entered code. The following output displays on the screen: set()* [Video description ends]

Performing Set Operations

[Video description begins] *Topic title: Performing Set Operations. Your host for this session is Janani Ravi.* [Video description ends]

Now if you remember sets from back in high school, you may also remember operations that can be performed on sets.

[Video description begins] *A Jupyter Notebook named PythonSet-02 displays in the web browser.* [Video description ends]

Python supports all of these operations, as we shall see in this video. Let's set up three sets of numbers, number1, number2, and number3.

[Video description begins] *She enters the following code in the first input box: number1 = {1, 2, 3, 4, 5} number2 = {4, 5, 6, 7, 8} number3 = {7, 8, 9, 10, 11}* [Video description ends]

If you want to combine two sets together, you can simply invoke the union function on a set. number1.union(number2) will combine the elements of these two sets. Remember, duplicate elements will be removed, and the result here is all of the numbers from 1 through 8.

[Video description begins] *She enters the following code in the second input box: number1.union(number2). Then, she executes the entered code. The following output displays on the screen: {1, 2, 3, 4, 5, 6, 7, 8}* [Video description ends]

The numbers 4 and 5 are duplicates in the set number1 and number2. The duplicates have been eliminated. And exactly one element 4, and one element 5 is present in the result. The union function on a set in Python can take in multiple sets as its input argument.

Here number1.union(number2, number3) will combine the elements of all three sets and store the result in number_union. You can see that number_union contains the numbers 1 all the way through 11, with all of the duplicates in the union having been removed.

[Video description begins] *In the third input box, she enters the following code: number_union = number1.union(number2, number3) number_union* [Video description ends]

Let's now set up three sets of strings.

[Video description begins] *She executes the entered code. The following output displays on the screen: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}* [Video description ends]

These are names of students, set_1, set_2, and set_3.

[Video description begins] *She enters the following code in the fourth input box: student_set_1 = {"Ava", "Emma", "James"} student_set_2 = {"James", "Mia", "Olivia"} student_set_3 = {"Olivia", "Sarah", "Sofia"}* [Video description ends]

Whatever kind of set you create, Python supports union operations for all of these sets, student_set_1.union, set_2, set_3 will give us a set of all of the student names.

[Video description begins] *Next, she enters the following code in the fifth input box: student_set_1.union(student_set_2, student_set_3). Then, she executes the entered code. The following output displays on the screen: {'Ava', 'Emma', 'James', 'Mia', 'Olivia', 'Sarah', 'Sofia'}* [Video description ends]

Note that the names James and Olivia, which are present more than once across the sets, are present exactly once in the union result. Observe that the union operation creates a new set with the union of all of the sets that were included. The values within the original sets remain unchanged. Just like the union operation, Python supports other set operations as well.

[Video description begins] *Now, she enters the following code in the sixth input box: print (number1) print (number2) print (number3)* [Video description ends]

Let's take a look at number1, number2, number3.

[Video description begins] *Next, the following code displays on the screen below the sixth input box: {1, 2, 3, 4, 5}{4, 5, 6, 7, 8}{7, 8, 9, 10, 11}* [Video description ends]

These are three sets of numbers that we had initialized earlier. And let's take a look at the intersection operation. If you remember your math, intersection finds the common elements between two sets. The only elements in common between number1 and number2 are 4 and 5, and that is what is present in the intersection set.

[Video description begins] *Next, she enters the code, number2.intersection(number1), in the seventh input box. Then, she executes this code. The following output displays on the screen: {4, 5}. Also, the following code displays on the right-hand side of the screen wherein the numbers, 4 and 5, are highlighted: number1 = {1, 2, 3, 4, 5} number2 ={4, 5, 6, 7, 8}* [Video description ends]

We'll continue to use the number set, number1, 2, and 3 in order to perform set operations, and this time we look at the difference operation.

[Video description begins] *She again enters the following code in the eighth input box: print (number1) print (number2) print (number3). Upon doing so, the following code appears below the code entered in the eighth input box: {1, 2, 3, 4, 5}{4, 5, 6, 7, 8}{7, 8, 9, 10, 11}* [Video description ends]

The difference between two sets are all of the elements that are present in number1 after having taken out any elements that are common with the set, number2.

[Video description begins] *Now, she enters the following code in the ninth input box: number1.difference(number2)* [Video description ends]

The result here contains the elements 1, 2, and 3. The elements in common between the sets number1 and number2, that is the elements 4 and 5, have been removed.

[Video description begins] *She executes the code entered in the ninth input box. The following output displays on the screen: {1, 2, 3}* [Video description ends]

You can perform any of the set operations, union, intersection, difference, and assign the result to a variable here. This set is the variable that contains the difference between the student_set_1 and student_set_2.

[Video description begins] *She enters the following code in the tenth input box: diff_set = student_set_1.difference(student_set_2) diff_set. Next, she executes this code. The following output displays on the screen: {'Ava', 'Emma'}. Also, the following code displays on the right-hand side of the screen: student_set_1 = {"Ava", "Emma", "James"} student_set_2 = {"James", "Mia", "Olivia"}* [Video description ends]

The students Ava and Emma are present in set_1, not present in set_2, which is why they are present in the difference set. So far all of the set operations that we've performed have not affected the original sets. You can take a look at number1 and number2. We performed a number of operations, such as union and intersection, between these two sets, the original sets remain unchanged.

[Video description begins] *She enters the following code in the eleventh input box: print (number1) print (number2). Next, the following code displays on the screen below the eleventh input box: {1, 2, 3, 4, 5}{4, 5, 6, 7, 8}* [Video description ends]

Here is another method that you can invoke on a set, the intersection_update.

[Video description begins] *She enters the following code in the twelfth input box: number1.intersection_update(number2) number1* [Video description ends]

This will perform the intersection operation between number1 and number2, and the updated results will be stored in the set number1. The set number1 now holds the intersection between the sets number1 and number2, which are the elements 4 and 5.

[Video description begins] *Next, she executes the entered code. The following output displays on the screen: {4, 5}* [Video description ends]

The elements in the set number2 remain unchanged, it still contains all of the original elements, only the set number1 has been updated.

[Video description begins] *She enters the code, number2, in the thirteenth input box, and then executes it. The following output displays on the screen: {4, 5, 6, 7, 8}* [Video description ends]

This is because we invoke the intersection_update command on the set number1. Let's go back and perform this intersection_update with the student sets that we were working with. Here are the three sets of students.

[Video description begins] *She enters the code, student_set_1, student_set_2, student_set_3, in the fourteenth input box. Then, she executes this code. The following output displays on the screen: ({'Ava', 'Emma', 'James '} {'James', 'Mia', 'Olivia '}{'Olivia', 'Sarah', 'Sofia'})* [Video description ends]

And we'll now invoke the intersection_update command on student_set_1, student_set_1.intersection_update(student_set_2) will update student_set_1 to hold the intersection between these two sets, which is just the student James.

[Video description begins] *She enters the following code in the fifteenth input box: student_set_1.intersection_update(student_set_2) student_set_1. Then, she executes this code. The following output displays on the screen: {'James'}* [Video description ends]

The contents of student_set_2 remains unchanged, it still contains the same three elements it did before.

[Video description begins] *She enters the code, student_set_2, in the sixteenth input box, and then she executes it. The following output displays on the screen: {'James', 'Mia', 'Olivia '}* [Video description ends]

All of the set operations contain an update equivalent. Just like the intersection_update, you can call difference_update as well. And the difference between student_set_1 and student_set_2 will be stored in student_set_1.

[Video description begins] *First, she enters the following code in the seventeenth input box: student_set_1 = {"Ava", "Emma", "James"} student_set_2 = {"James", "Mia", "Olivia"} student_set_3 = {"Olivia", "Sarah", "Sofia"} . Then, she enters the following code in the eighteenth input box: student_set_1.difference_update(student_set_2) student_set_1. Then, she executes the entered code. The following output displays on the screen: {'Ava', 'Emma'}* [Video description ends]

The elements Ava and Emma are present in student_set_1 but not in student_set_2, and the student_set_1 has been accordingly updated. Let's quickly look at some other set operations. The isdisjoint operation tells you whether two sets are disjoints, whether they contain elements in common or not.

[Video description begins] *She enters the following code in the nineteenth input box: {1, 2, 3}.isdisjoint({4, 5, 6})* [Video description ends]

Observe how we instantiate the set with elements 1, 2, 3 with curly brackets, and call the isdisjoint function on it directly. isdisjoint is True here because there are no elements in common between the two sets, (1, 2, 3) and (4, 5, 6).

[Video description begins] *Next, she executes the entered code. The following output displays on the screen: True* [Video description ends]

Let's try this once again and call isdisjoint on two sets, (1, 2, 3) and (1, 5, 6). The element 1 isn't common between these two sets, which is why isdisjoint is False.

[Video description begins] *Now, she enters the code, {1, 2, 3}.isdisjoint({1, 5, 6}), in the twentieth input box, and then executes it. The following output displays on the screen: False*. [Video description ends]

What about the sets number1 and number2, are they disjoint? Well Python says False because they have elements in common.

[Video description begins] *She enters the following code in the twenty-first input box: number1.isdisjoint(number2). Then, she executes this code. The following output displays on the screen: False. Also, the code of the first input box displays on the right-hand side of the screen, which is number1 = {1, 2, 3, 4, 5} number2 = {4, 5, 6, 7, 8} number3 = {7, 8, 9, 10, 11}* . [Video description ends]

Just like isdisjoint, there are other set functions as well such as issubset. Is (10, 20) a subset of (10, 20, 30)? Well yes, which is why Python returns True.

[Video description begins] *Now, she enters the code, {10, 20}.issubset({10, 20, 30}), in the twenty-second input box, and then executes it. The following output displays on the screen: True*. [Video description ends]

Let's try this once again. This time, we will call the issuperset function. Is (10, 20, 30) a superset of (10, 20)? Yes, and Python returns True.

[Video description begins] *Now, she enters the code, {10, 20, 30}.issuperset({10, 20}), in the twenty-third input box, and then executes it. The following output displays on the screen: True*. [Video description ends]

Working with Nested Lists

[Video description begins] *Topic title: Working with Nested Lists. Your host for this session is Janani Ravi*. [Video description ends]

We've seen that Python list can contain elements, which can be of any data type.

[Video description begins] *A Jupyter Notebook named NestedListAndConversionOfList displays in the web browser.* [Video description ends]

Floating points, integers, strings. Python lists can also contain other lists, and these are nested lists or matrices. Let's see an example here, the car_matrix here is a list of lists.

[Video description begins] *She enters the following code in the first input box: car_matrix = [['Hennessey Venom GT', 1244], ['SSC Ultimate Aero', 1287], ['Zenvo ST1', 1100]] car_matrix* [Video description ends]

Notice that the inner elements in the outer list are list by themselves. The first element in the outer list is the list with the Hennessey Venom GT and 1244. The numbers here represent the horsepower for that particular car. The second list in the outer list is the list which has the SSC Ultimate Aero and 1287.

And the third list within the outer list is the Zenvo ST1 and 1100. Now if you display this car_matrix, this is a list of lists. You know it's one list nested within another because the list starts with two square brackets, indicating there is a nesting involved and ends with two square brackets as well.

[Video description begins] *She executes the code entered in the first input box. The following output displays on the screen: [['Hennessey Venom GT', 1244], ['SSC Ultimate Aero', 1287], ['Zenvo ST1', 1100]]* [Video

description ends]

Now let's see how some of the built-in functions work, for example, the len function. When you pass in the car_matrix, list will only give you the len of the outer list. There are three elements in the outer list, the return value from the len function is 3.

[Video description begins] *She enters the code, len(car_matrix), in the second input box, and then executes it. The following output displays on the screen: 3* [Video description ends]

This is an example of a list of lists where the nesting is up to two levels. But in Python, you can have list nested up to any number of levels. And when you call the len function on the outermost list, you get the length only of the outermost list. When you index into the car_matrix with just a single pair of square brackets, here we are trying to access the element at position 1.

But of what list? Well, the outermost list. The first pair of square brackets always applies to the outermost list. At index 1 in the outermost list, we have an inner list which contains the values, SSC Ultimate Aero and 1287. The len function will calculate the length of this inner list at index 1 and the len of this inner list is 2.

[Video description begins] *She enters the code, len(car_matrix[1]), in the third input box, and then executes it. The following output displays on the screen: 2* [Video description ends]

Once again, we index into the car_matrix with just a single pair of square brackets. That means we're accessing the element at index 2 in the outer list, which is a list by itself, and the length of that list is 2 again.

[Video description begins] *Now, she enters the code, len(car_matrix[2]), in the fourth input box, and then executes it. The following output displays on the screen: 2* [Video description ends]

If you want to access a specific element in the inner list, you need to use two pairs of square brackets. The index within the first pair of square brackets, that is index 1, refers to the index of the element within the outer list.

And the index 0 within the second pair of square brackets refers to the element within the inner list. You can see that the result here is the SSC Ultimate Aero, the first element at position 0 within the outer list, which is at position 1.

[Video description begins] *In the fifth input box, she enters the code, car_matrix[1][0], and then executes it. The following output displays on the screen: 'SSC Ultimate Aero'* [Video description ends]

So if you have a list of lists, two square brackets is what you need to access a particular element within the inner list. Here we access the list at index 1 in the outer list, and we access the value at index 1 in the inner list, which is 1287.

[Video description begins] *Next, she enters the following code in the sixth input box: car_matrix[1][1]. Next, she executes this code. The following output displays on the screen: 1287. Also, a screenshot of the output generated from the code entered in the 1st input box displays. The following code is highlighted:['SSC Ultimate Aero', 1287],* [Video description ends]

If you're learning Python in order to work with data science, you should be comfortable with nested lists or matrices. So try a couple of other examples to see how indexing works. The index in the first pair of square brackets, 2, refers to the last element in the outer list. And 0 then refers to the first element in the inner list, giving us the Zenvo ST1.

[Video description begins] *Next, she enters the following code in the seventh input box: car_matrix[2][0]. Then, she executes this code. The following output displays on the screen: 'Zenvo ST1'. Also, the following code segment is highlighted in a screenshot of the output, which is displayed on the right-hand side of the screen: ['Zenvo ST1', 1100]]* [Video description ends]

If any of the indices you specify within square brackets, whether it's for the outer list or for the inner list is out of range, Python will throw an error. Here notice that the list index 2 for the inner list is out of the range for our inner list, which has just two elements at index positions 0 and 1.

[Video description begins] *Next, she enters the following code in the eighth input box: car_matrix[1][2]. Then, she executes this code. The following IndexError displays on the screen: list index out of range. Also, the following code segment is highlighted in a screenshot of the output, which is displayed on the right-hand side of the screen: ['SSC Ultimate Aero', 1287],* [Video description ends]

Slicing operations are possible with nested lists as well. So you can specify a slicing operation on only one of the nested list. For example, here we access the list at index 2 in the outer list, and then perform a slicing operation on the inner list [0:2].

[Video description begins] *She enters the code, car_matrix[2][0:2], in the tenth input box. Also, the following code segment is highlighted in a screenshot of the output, which is displayed on the right-hand side of the screen: ['Zenvo ST1', 1100]]* [Video description ends]

Index 2 at the outer list contains the Zenvo ST1 and 1100, [0:2] gives us the entire inner list.

[Video description begins] *She executes the code entered in the tenth input box. The following output displays on the screen: ['Zenvo ST1', 1100]* [Video description ends]

Here is another example, where we perform a slicing operation only on the inner list. The outer list, we access the element at index 1. And this gives us the SSC Ultimate Aero and 1287 because we sliced the entire inner list.

[Video description begins] *She enters the code, car_matrix[1][0:2], in the eleventh input box. Next, she executes the entered code. The following output displays on the screen: ['SSC Ultimate Aero', 1287]. Also, the following code segment is highlighted in a screenshot of the output, which is displayed on the right-hand side of the screen: ['SSC Ultimate Aero', 1287],* [Video description ends]

Performing List Conversions

[Video description begins] *Topic title: Performing List Conversions. Your host for this session is Janani Ravi.* [Video description ends]

We'll now move on and work with a slightly different list example. Here is my_list, which is initialized to a mixed list. It contains strings as well as integer values. Four names and four ages corresponding to the names.

[Video description begins] *A Jupyter Notebook named NestedListAndConversionOfList displays in the web browser. Currently, the host has already entered the following code in the twelfth input box: my_list = ['Leo', 18, 'Aaron', 25, 'Easton', 34, 'Jordan', 30] my_list. Then, she executes this code. The following output displays on the screen: ['Leo', 18, 'Aaron', 25, 'Easton', 34, 'Jordan', 30]* [Video description ends]

Once you have a Python list, it's possible to create a tuple from this list by invoking the tuple function, and passing in the list as an input argument.

[Video description begins] *Next, she enters the code, tuple (my_list), in the thirteenth input box.* [Video description ends]

A tuple in Python is an ordered collection, exactly like a list is. The difference is that the tuple is immutable. Once created, the elements of a tuple cannot be changed. Here is our tuple from our my_list and comprising of names and ages.

[Video description begins] *She executes the code entered in the thirteenth input box. The following output displays on the screen: ('Leo', 18, 'Aaron', 25, 'Easton', 34, 'Jordan', 30)* [Video description ends]

Observe that the original list here is a one-dimensional list. There was no nesting in the original list, which is why we got a single tuple with all of the original values from the list. Let's create a list with the same information, names and ages, but this time we'll have a nested list. Observe that my_list1 is a list of lists, and every list within the outer list comprises of a name and an age.

[Video description begins] *Now, she enters the following code in the fourteenth input box: my_list1 = [['Leo', 18], ['Aaron', 25], ['Easton', 34], ['Jordan', 30]]my_list1*. [Video description ends]

The result is a nested list, with every inner list is a list in itself with name and age as its elements.

[Video description begins] *She executes the code entered in the fourteenth input box. The following output displays on the screen: [['Leo', 18], ['Aaron', 25], ['Easton', 34], ['Jordan', 30]]* [Video description ends]

You can convert this nested list to a tuple as well, invoke the tuple function, and pass in my_list1. Observe the fields of the resulting tuple. The outer list has been converted to a tuple, as you can see that the square brackets in the result have been replaced by the ordinary brackets that denote a tuple.

[Video description begins] *She enters the code, tuple (my_list1), in the fifteenth input box, and then executes it. The following output displays on the screen: (['Leo', 18], ['Aaron', 25], ['Easton', 34], ['Jordan', 30])* [Video description ends]

Every field within this outer tuple is a list, so you can have a tuple of lists as well. You can see that lists can be converted to tuples in a very straightforward and simple manner. Let's study some more conversions, and for that we'll create another nested list. We'll call this the students_list. Once again, this is a list of lists, and every inner list has four elements, a number, name, number, name.

[Video description begins] *She enters the following code in the sixteenth input box: students_list = [[1, 'Caleb', 2, 'Leo'], [3, 'Easton', 4, 'Jordan']]* [Video description ends]

A list of lists can also be thought of as lists in two dimensions, the outer dimension and the inner dimension, which is also a list.

[Video description begins] *Next, she enters the code, students_list, in the seventeenth input box. Then, she executes the entered code. The following output displays on the screen: [[1, 'Caleb', 2, 'Leo'], [3, 'Easton', 4, 'Jordan']]* [Video description ends]

Now let's see if we can convert this students_list to a dictionary by invoking the dict function, and passing in the students_list. Well, Python doesn't know how to interpret that list as a dictionary. Remember, a dictionary is a mapping of key-value pairs. So Python simply throws an error.

[Video description begins] *In the eighteenth input box, she enters the code, dict(students_list). Then, she executes the entered code. The following ValueError displays on the screen: dictionary update sequence element #0 has length 4; 2 is required* [Video description ends]

If you look at the error, Python actually tells you something meaningful, dictionary update sequence element #0 has length 4; 2 is required. Python tries to take every inner list and tries to convert it to a key-value pair mapping. But when there are four elements in the inner list, Python doesn't know what to do.

That's what this error is trying to say. Let's redefine this nested list to be a little different, here is the students_list_2. Observe that the inner list here has just two elements, a number and a name. If you see the students_list_2, each inner list has just two elements.

[Video description begins] *In the nineteenth input box, she enters the following code: students_list_2 =[[1, 'Caleb'], [2, 'Leo'], [3, 'Easton'], [4, 'Jordan']] students_list_2. Next, she executes the entered code. The following output displays on the screen: [[1, 'Caleb'], [2, 'Leo'], [3, 'Easton'], [4, 'Jordan']]* [Video description ends]

Now if you try and convert this list to a dictionary, Python knows what to do. Let's invoke the dict function and pass in students_list_2, and Python will successfully convert this to a dictionary. Observe the dictionary, enclosed in curly braces, all of the keys are numbers, and all of the values are names.

[Video description begins] *Next, she enters the code, dict(students_list_2), in the twentieth input box. Then, she executes the entered code. The following output displays on the screen: {1:'Caleb', 2: 'Leo', 3: 'Easton', 4: 'Jordan'}* [Video description ends]

In every inner list, the first element, which is the number has been made the key for that list. And the value is the second element, which is the name. Let's try this again with a third nested list, students_list_3. Observe that this is a list of lists, but the last element in the outer list is an inner list with three elements. All of the other inner lists have just two elements.

[Video description begins] *Now, she enters the following code in the twenty-first input box: students_list_3 =[[1, 'Caleb'], [2, 'Leo'], [3, 'Easton'], [4, 'Jordan', 'Adam']]* [Video description ends]

Well, if you try to convert this nested list to a dictionary, Python, once again, doesn't know what to do with the last element in the outer list.

[Video description begins] *Next, she enters the code, dict(students_list_3), in the twenty-second input box. Then, she executes the entered code. The following ValueError displays on the screen: dictionary update sequence element #3 has length 3; 2 is required* [Video description ends]

If every inner list had just two values, then it knows how to convert this to a dictionary. But that last element 4, Jordan, Adam, that inner list with three elements is what throws Python. Let's now tweak this nested list a little bit. Once again we'll have a nested list, but this list has been nested up to three levels.

[Video description begins] *In the twenty-third input box, she enters the following code: students_list_4 =[[1, 'Caleb'], [2, 'Leo'], [3, 'Easton'], [4, ['Jordan', Adam']]]* [Video description ends]

Observe the last inner list, the names Jordan and Adam are not separate elements within that list, instead they're enclosed in another list. This is the third level of nesting. So we have the outermost list, the middle list, and the innermost list. The innermost list just contains the elements, Jordan and Adam.

Now this middle list, all the elements of the middle list have just two items. Now if we try and convert this list to a dictionary, everything will work fine, and Python knows how to interpret this.

[Video description begins] *Now, she enters the code, dict(students_list_4), in the twenty-fourth input box. Then, she executes the entered code. The following output displays on the screen: {1:'Caleb', 2: 'Leo', 3: 'Easton', 4: ['Jordan', 'Adam']}* [Video description ends]

All of the keys are numbers, that is the first elements in the middle list. And all of the values are either names or a list. This is a slightly complicated example. But if you stare up the screen for a little bit, understand how the original list was composed, and how Python converted to a dictionary, it'll make sense.

Let's go back to the car_matrix that we worked with earlier. Remember, this was a list of lists. The inner list here contains the name of a car and a number. That number represents the car's horsepower.

[Video description begins] *She enters the code, car_matrix, in the twenty-fifth input box. The following output displays on the screen: [['Hennessey Venom GT', 1244], ['SSC Ultimate Aero', 1287], ['Zenvo ST1', 1100]]* [Video description ends]

Now, converting this list to a dictionary is very straightforward. So let's call the dict function, pass in the car_matrix, and store the resulting dictionary in the car_matrix_dict variable.

[Video description begins] *Next, she enters the following code in the twenty-sixth input box: car_matrix_dict = dict(car_matrix). Then, she enters the code, car_matrix_dict, in the twenty-seventh input box.* [Video description ends]

The keys of the car_matrix_dict are the names of the cars, and the values are the horsepowers for these cars. Now that we've converted this list to a dictionary, let's modify the original list by calling the pop operation on the car_matrix.

[Video description begins] *Then, she executes the entered code. The following output displays: {'Hennessey Venom GT': 1244,'SSC Ultimate Aero': 1287, 'Zenvo ST1': 1100}* [Video description ends]

[Video description begins] *She enters the code, car_matrix.pop(), in the twenty-eighth input box, and then executes it. The following output displays on the screen: ['Zenvo ST1', 1100]. Also, the code entered in the input box 25 displays on the right side of the screen along with its output. Further, in the output the following code segment is highlighted: ['Zenvo ST1', 1100]]* [Video description ends]

Pop operation will operate on the outer list, and pop out the last element in the outer list, which is the Zenvo ST1, and it's horsepower, 1100. If you take a look at the car_matrix now, you'll find that the list has been updated.

[Video description begins] *Next, she enters the code, car_matrix, in the twenty-ninth input box, and then executes it. The following output displays on the screen: [['Hennessey Venom GT', 1244], ['SSC Ultimate Aero', 1287]]. Also, the code entered in the input box 25 displays on the right side of the screen along with its output.* [Video description ends]

The last element in the outer list is no longer present. Let's modify the car_matrix list, and clear all of its values. Now the car_matrix will be a completely empty list. All of its elements have been removed.

[Video description begins] *First, she enters the code, car_matrix.clear(), in the thirtieth input box. Then, she enters the code, car_matrix, in the thirty-first input box. Finally, she executes the entered code and the following output displays on the screen: []* [Video description ends]

But if you take a look at the car_matrix_dict, which we created from the original car_matrix list, that remains unchanged.

[Video description begins] *She enters the code, car_matrix_dict, in the thirty-second input box. Then, she executes the entered code. The following output displays: {'Hennessey Venom GT': 1244,'SSC Ultimate Aero': 1287, 'Zenvo ST1': 1100}* [Video description ends]

The dictionary no longer references the original list. So the changes to the original list are not reflected in the dictionary we created from that list. Now that we have this dictionary, let's try recreating a list from this dictionary.

[Video description begins] *In the thirty-third input box, she enters the following code: car_names = list(car_matrix_dict)* [Video description ends]

And you'll find that you don't exactly get what you expect. You take a look at car_names, you'll find that the list has been created only using the keys of the dictionary, not the values.

[Video description begins] *Next, she enters the code, car_names, in the thirty-fourth input box. Then, she executes the entered code. The following output displays on the screen: ['Hennessey Venom GT','SSC Ultimate Aero', 'Zenvo ST1']. Also, the code entered in the input box 32 displays on the bottom-right side of the screen along with its output.* [Video description ends]

If you want to create a list of values, you have to explicitly invoke the values function of the dictionary, and pass the result into the list function.

[Video description begins] *First, she enters the following code in the thirty-fifth input box: car_hp = list(car_matrix_dict.values()). Then, she enters the following code in the thirty-sixth input box: car_hp. Finally, she executes the entered code. The following output displays on the screen: [1244, 1287, 1100] . Also, the code entered in the input box 32 displays on the bottom-right side of the screen along with its output*. [Video description ends]

car_hp, now, will contain a list of horsepower values.

Exercise: Dictionaries and Sets

[Video description begins] *Topic title: Exercise: Dictionaries and Sets. Your host for this session is Janani Ravi*. [Video description ends]

Here are some questions for you to think about in this exercise. You'll first try and list some features of dictionaries in Python. You'll then move on to describing the basic features of sets in Python, another complex data type.

And then compare and contrast, list some similarities and differences between the complex data types that you've understood, lists, dictionaries, and sets in Python. I'd suggest that you pause the video here, think about these questions for a while. Maybe, write down your answers before moving on to look at some sample answers.

[Video description begins] *Solution* [Video description ends]

Dictionaries in Python are a very commonly used and an extremely useful complex data type. Dictionaries in Python store information in the form of an unordered collection of key-value pairs. Every value has a corresponding key, and that value can be look up using that key. So when you construct a dictionary, you'll set up key-value pairs.

And there is no inherent ordering amongst the keys or the values. Each key in a dictionary is unique and values can be duplicates. Keys have to be unique because keys are what you use to look up corresponding values in a dictionary. Values are looked up by keys, so you can't go the other way round. So you can't specify a value and find the corresponding key. The mapping only goes in one direction.

Values can be either simple or complex data types. So you can have nested lists within dictionaries. You can have dictionaries within dictionaries. Tuples within dictionaries and so on. And, finally, dictionaries are mutable. Once created, you can add key-value pairs to a dictionary. You can change values corresponding to a key. You can delete key-value pairs from a dictionary as well.

Let's now move on to discussing the important features of sets in Python. Sets in Python are also an unordered collection. But they are made up of completely unique elements. All elements in a set have to be unique, duplicates are not allowed. If you add a duplicate values to a set, the set will automatically keep only one copy of the duplicate. So, for example, if you try to add the number ten multiple times, there'll be exactly one element ten within your set.

Now, sets in Python cannot contain other mutable complex data types. It cannot contain lists or dictionaries. In addition to the common operations of adding elements, removing elements from sets, sets also support other interesting operations. Such as union, intersection, difference, and so on. All operations that you might expect to perform with sets are available in the form of functions.

Now that we've understood the complex data types, let's move on to understanding similarities and differences between lists, dictionaries, and sets. Lists, as we spoke about, are ordered collections. Both dictionaries and sets are unordered. There is no intrinsic ordering of elements within dictionaries and sets. There is no constraint on how many times the same element can be added to a list. So you can have the element 10 present ten times if you want to.

Lists can contain duplicates, sets cannot contain duplicates, dictionaries cannot have duplicate keys. The non-duplicate constraint applies only to keys in a dictionary, not to its values. Lists can contain nested complex data types. So you can have a list of dictionaries, a list of sets, a list of tuples, and so on.

And values in dictionaries can also be complex data types. Values in dictionaries can be a nested dictionary, a nested list, a nested set, a nested tuple, anything. Sets, on the other hand, can contain only immutable tuples. It cannot contain other complex data types, such as lists or dictionaries because lists and dictionaries are mutable.

Here is one last point of comparison between lists, dictionaries, and sets. List elements can be looked up by index values starting at zero. There is a no corresponding look up for set elements. Sets don't have an intrinsic order. Dictionary values, on the other hand, can be looked up by key.