This module teaches the basics of Python and begins by exploring some of the different data types such as integers, real numbers, and strings. Continue with the module and learn how to use expressions in mathematical operations, store values in variables, and the many different ways to manipulate strings.

**Learning Objectives**

* Demonstrate an understanding of types in Python by converting or casting data types such as strings, floats, and integers.
* Interpret variables and solve expressions by applying mathematical operations.
* Describe how to manipulate strings by using a variety of methods and operations.
* Build a program in JupyterLab to demonstrate your knowledge of types, expressions, and variables.
* Work with, manipulate, and perform operations on strings in Python

# About this course

This course was designed to provide the building blocks for Python programming and data collection for those choosing a career in Data Science, Data Engineering, AI or Application Development.

Initially conceived as a foundation course for Data Science and AI it has been refreshed several times to keep pace with emerging career options. Additional content has been added which is applicable to Data Science, Data Engineering, AI or Application Development.

After completing this course you will have learned foundational skills in Python programming which you can then go on to apply in the Python Project course for your chosen career.  The Python Project courses involve real world scenarios where you are in charge of a final project as a Data Scientist, a Data Engineer, or in AI and Application Development. By finishing this course and your follow-on Python Project, you will gain the basic skills to continue the steps on your chosen career path.

**Note: This course is a pre-requisite for the Python Project courses and should be completed in full before attempting the appropriate Python Project course.**

# Types

0:00

A type is how Python represents different types of data.

In this video, we will discuss some widely used types in Python.

You can have different types in Python.

They can be integers like 11, real numbers like 21.213, they can even be words.

Integers, real numbers, and words can be expressed as different data types.

The following chart summarizes three data types for the last examples.

The first column indicates the expression.

The second column indicates the data type.

We can see the actual data type in Python by using the type command.

We can have int, which stands for an integer and float that stands for

float, essentially a real number.

The type string is a sequence of characters.

Here are some integers.

Integers can be negative or positive.

It should be noted that there is a finite range of integers but it is quite large.

Floats are real numbers.

They include the integers but also numbers in between the integers.

Consider the numbers between 0 and 1.

We can select numbers in between them.

These numbers are floats.

Similarly, consider the numbers between 0.5 and 0.6.

We can select numbers in between them.

These are floats as well.

We can continue the process zooming in for different numbers.

Of course there is a limit but it is quite small.

You can change the type of the expression in Python, this is called typecasting.

You can convert an int to a float.

For example, you can convert or cast the integer 2 to a float 2.0.

Nothing really changes, if you cast a float to an integer, you must be careful.

For example, if you cast the float 1.1 to 1, you will lose some information.

If a string contains an integer value, you can convert it to int.

If we convert a string that contains a non-integer value, we get an error.

Check out more examples in the lab.

You can convert an int to a string or a float to a string.

Boolean is another important type in Python.

A Boolean can take on two values.

The first value is True, just remember we use an uppercase T.

Boolean values can also be False with an uppercase F.

Using the type command on a Boolean value, we obtain the term bool.

This is short for Boolean, if we cast a Boolean True to an integer or

float, we will get a 1.

If we cast a Boolean False to an integer or float, we get a 0.

If you cast a 1 to a Boolean, you get a True.

Similarly, if you cast a 0 to a Boolean, you get a False.

Check the labs for more examples or

check Python.org for other kinds of types in Python.

(Music)

# Expressions and Variables

0:00

In this video, we will cover expressions and variables.

Expressions describe a type of operation the computers perform.

Expressions are operations the python performs. For example,

basic arithmetic operations like adding multiple numbers.

The result in this case is 160.

We call the numbers operands, and the math symbols in this case,

addition, are called operators.

We can perform operations such as traction using the subtraction sign.

In this case, the result is a negative number.

We can perform multiplication operations using the asterisk. The result is 25.

In this case, the operands are given by negative and asterisk.

We can also perform division with the forward slash-

25 / 5 is 5.0;

25 / 6 is approximately 4.167.

In Python 3, the version we will be using in this course, both will result in a float.

We can use the double slash for integer division, where the result is rounded.

Be aware, in some cases the results are not the same as regular division.

Python follows mathematical conventions when performing mathematical expressions.

The following operations are in a different order.

In both cases, Python performs multiplication,

then addition, to obtain the final result.

There are a lot more operations you can do with Python, check the labs for

more examples.

We will also be covering more complex operations throughout he course.

The expressions in the parentheses are performed first.

We then multiply the result by 60.

The result is 1,920.

Now, let's look at variables.

We can use variables to store values. In this case, we assign a value of 1 to

the variable my\_variable using the assignment operator, i.e, the equal sign.

We can then use the value somewhere else in the code

by typing the exact name of the variable.

We will use a colon to denote the value of the variable.

We can assign a new value to my\_variable using the assignment operator.

We assign a value of 10. The variable now has a value of 10.

The old value of the variable is not important.

We can store the results of expressions. For example, we add several values and

assign the result to x. X now stores the result.

We can also perform operations on x and save the result to a new variable-y.

Y now has a value of 2.666.

We can also perform operations on x and assign the value x.

The variable x now has a value: 2.666.

As before, the old value of x is not important.

We can use the type command in variables as well.

It's good practice to use meaningful variable names; so,

you don't have to keep track of what the variable is doing.

Let say, we would like to convert the number of minutes

in the highlighted examples to number of hours in the following music data-set.

We call the variable, that contains the total number of minutes, total\_min.

It's common to use the underscore to represent the start of a new word.

You could also use a capital letter.

We call the variable that contains the total number of hours, total\_hour.

We can obtain the total number of hours by dividing total\_min by 60.

The result is approximately 2.367 hours.

If we modify the value of the first variable,

the value of the variable will change.

The final result values change accordingly, but

we do not have to modify the rest of the code.

(Music)

# String Operations

0:00

In Python, a string is a sequence of characters.

A string is contained within two quotes.

You could also use single quotes.

A string can be spaces or digits.

A string can also be special characters.

We can bind or assign a string to another variable.

It is helpful to think of a string as an ordered sequence.

Each element in the sequence can be accessed using

an index represented by the array of numbers.

The first index can be accessed as follows:

We can access index six.

Moreover, we can access the 13th index.

We can also use negative indexing with strings.

The last element is given by the index negative one.

The first element can be obtained by index negative 15 and so on.

We can bind a string to another variable.

It is helpful to think of string as a list or tuple.

We can treat the string as a sequence and perform sequence operations.

We can also input a stride value as follows:

The two indicates we'd select every second variable.

We can also incorporate slicing.

In this case, we return every second value up to index four.

We can use the len command to obtain the length of the string.

As there are 15 elements, the result is 15.

We can concatenate or combine strings.

We use the addition symbols.

The result is a new string that is a combination of both.

We can replicate values of a string.

We simply multiply the string by the number of times we would like to replicate it-

in this case, three.

The result is a new string.

The new string consists of three copies of the original string.

This means you cannot change the value of the string, but you can create a new string.

For example, you can create a new string by setting it to

the original variable and concatenate it with a new string.

The result is a new string that changes from

Michael Jackson to Michael Jackson is the best.

Strings are immutable.

Back slashes represent the beginning of escape sequences.

Escape sequences represent strings that may be difficult to input.

For example, backslashes "n" represent a new line.

The output is given by a new line after the backslashes "n" is encountered.

Similarly, backslash "t" represents a tab.

The output is given by a tab where the backslash, "t" is.

If you want to place a backslash in your string,

use a double backslash.

The result is a backslash after the escape sequence.

We can also place an "r" in front of the string.

Now, let's take a look at string methods.

Strings are sequences and as such,

have apply methods that work on lists and tuples.

Strings also have a second set of methods that just work on strings.

When we apply a method to the string A,

we get a new string B that is different from A.

Let's do some examples.

Let's try with the method "Upper".

This method converts lowercase characters to uppercase characters.

In this example, we set the variable A to the following value.

We apply the method "Upper", and set it equal to B.

The value for B is similar to A, but all the characters are uppercase.

The method replaces a segment of the string- i.e.

a substring with a new string.

We input the part of the string we would like to change.

The second argument is what we would like to exchange the segment with.

The result is a new string with a segment changed.

The method find, find substrings.

The argument is the substring you would like to find.

The output is the first index of the sequence.

We can find the substring Jack.

If the substring is not in the string,

the output is negative one.

Check the labs for more examples.

(Music)

# String Operations

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-applied-data-science-ai/discussions/weeks/1)

0:00

In Python, a string is a sequence of characters.

A string is contained within two quotes.

You could also use single quotes.

A string can be spaces or digits.

A string can also be special characters.

We can bind or assign a string to another variable.

It is helpful to think of a string as an ordered sequence.

Each element in the sequence can be accessed using

an index represented by the array of numbers.

The first index can be accessed as follows:

We can access index six.

Moreover, we can access the 13th index.

We can also use negative indexing with strings.

The last element is given by the index negative one.

The first element can be obtained by index negative 15 and so on.

We can bind a string to another variable.

It is helpful to think of string as a list or tuple.

We can treat the string as a sequence and perform sequence operations.

We can also input a stride value as follows:

The two indicates we'd select every second variable.

We can also incorporate slicing.

In this case, we return every second value up to index four.

We can use the len command to obtain the length of the string.

As there are 15 elements, the result is 15.

We can concatenate or combine strings.

We use the addition symbols.

The result is a new string that is a combination of both.

We can replicate values of a string.

We simply multiply the string by the number of times we would like to replicate it-

in this case, three.

The result is a new string.

The new string consists of three copies of the original string.

This means you cannot change the value of the string, but you can create a new string.

For example, you can create a new string by setting it to

the original variable and concatenate it with a new string.

The result is a new string that changes from

Michael Jackson to Michael Jackson is the best.

Strings are immutable.

Back slashes represent the beginning of escape sequences.

Escape sequences represent strings that may be difficult to input.

For example, backslashes "n" represent a new line.

The output is given by a new line after the backslashes "n" is encountered.

Similarly, backslash "t" represents a tab.

The output is given by a tab where the backslash, "t" is.

If you want to place a backslash in your string,

use a double backslash.

The result is a backslash after the escape sequence.

We can also place an "r" in front of the string.

Now, let's take a look at string methods.

Strings are sequences and as such,

have apply methods that work on lists and tuples.

Strings also have a second set of methods that just work on strings.

When we apply a method to the string A,

we get a new string B that is different from A.

Let's do some examples.

Let's try with the method "Upper".

This method converts lowercase characters to uppercase characters.

In this example, we set the variable A to the following value.

We apply the method "Upper", and set it equal to B.

The value for B is similar to A, but all the characters are uppercase.

The method replaces a segment of the string- i.e.

a substring with a new string.

We input the part of the string we would like to change.

The second argument is what we would like to exchange the segment with.

The result is a new string with a segment changed.

The method find, find substrings.

The argument is the substring you would like to find.

The output is the first index of the sequence.

We can find the substring Jack.

If the substring is not in the string,

the output is negative one.

Check the labs for more examples.

(Music)

0:00 In Python, a string is a sequence of characters. A string is contained within two quotes. You could also use single quotes. A string can be spaces or digits. A string can also be special characters. We can bind or assign a string to another variable. It is helpful to think of a string as an ordered sequence. Each element in the sequence can be accessed using an index represented by the array of numbers. The first index can be accessed as follows: We can access index six. Moreover, we can access the 13th index. We can also use negative indexing with strings. The last element is given by the index negative one. The first element can be obtained by index negative 15 and so on. We can bind a string to another variable. It is helpful to think of string as a list or tuple. We can treat the string as a sequence and perform sequence operations. We can also input a stride value as follows: The two indicates we'd select every second variable. We can also incorporate slicing. In this case, we return every second value up to index four. We can use the len command to obtain the length of the string. As there are 15 elements, the result is 15. We can concatenate or combine strings. We use the addition symbols. The result is a new string that is a combination of both. We can replicate values of a string. We simply multiply the string by the number of times we would like to replicate it- in this case, three. The result is a new string. The new string consists of three copies of the original string. This means you cannot change the value of the string, but you can create a new string. For example, you can create a new string by setting it to the original variable and concatenate it with a new string. The result is a new string that changes from Michael Jackson to Michael Jackson is the best. Strings are immutable. Back slashes represent the beginning of escape sequences. Escape sequences represent strings that may be difficult to input. For example, backslashes "n" represent a new line. The output is given by a new line after the backslashes "n" is encountered. Similarly, backslash "t" represents a tab. The output is given by a tab where the backslash, "t" is. If you want to place a backslash in your string, use a double backslash. The result is a backslash after the escape sequence. We can also place an "r" in front of the string. Now, let's take a look at string methods. Strings are sequences and as such, have apply methods that work on lists and tuples. Strings also have a second set of methods that just work on strings. When we apply a method to the string A, we get a new string B that is different from A. Let's do some examples. Let's try with the method "Upper". This method converts lowercase characters to uppercase characters. In this example, we set the variable A to the following value. We apply the method "Upper", and set it equal to B. The value for B is similar to A, but all the characters are uppercase. The method replaces a segment of the string- i.e. a substring with a new string. We input the part of the string we would like to change. The second argument is what we would like to exchange the segment with. The result is a new string with a segment changed. The method find, find substrings. The argument is the substring you would like to find. The output is the first index of the sequence. We can find the substring Jack. If the substring is not in the string, the output is negative one. Check the labs for more examples. (Music): Added to Selection. Press [CTRL + S] to save as a note

# List and Tuples

0:00

In this video we will cover lists and tuples.

These are called compound data types and are one of the key types of data

structures in Python. Tuples.

Tuples are an ordered sequence. Here is a tuple

ratings. Tuples are expressed as comma separated elements within parentheses.

These are values inside the parentheses. In Python, there are different types:

strings, integer, float. They can all be contained

in a tuple but the type of the variable is tuple.

Each element of a tuple can be accessed via an index.

The following table represents the relationship between the index and the

elements in the tuple. The first element can be accessed by the

name of the tuple followed by a square bracket with the index number,

in this case zero. We can access the second element as follows.

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

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

We can concatenate or combine tuples by adding them.

The result is the following with the following index.

If we would like multiple elements from a tuple, we could also slice

tuples. For example, if we want the first three elements

we use the following command. The last index is one larger than the

index you want; similarly if we want the last two elements,

we use the following command. Notice, how the last index is one larger than the

length of the tuple. We can use the len command to obtain the

length of a tuple. As there are five elements, the result is 5.

Tuples are immutable which means we can't change them.

To see why this is important, let's see what happens when we set the variable

ratings 1 to ratings. Let's use the image to

provide a simplified explanation of what's going on.

Each variable does not contain a tuple, but references the same immutable tuple

object. See the objects and classes module for

more about objects. Let's say, we want to change the

element at index 2. Because tuples are immutable we can't,

therefore ratings 1 will not be affected by a change in rating

because the tuple is immutable, i.e we can't change it.

We can assign a different tuple to the ratings variable.

The variable ratings now references another tuple.

As a consequence of immutability, if we would like to manipulate a tuple

we must create a new tuple instead. For example,

if we would like to sort a tuple we use the function sorted.

The input is the original tuple, the output is a new sorted

list. For more on functions, see our video on functions.

A tuple can contain other tuples as well as other complex data types.

This is called nesting. We can access these elements using the standard

indexing methods. If we select an index with a tuple, the

same index convention applies. As such,

we can then access values in the tuple. For example, we could access the second

element. We can apply this indexing directly to

the tuple variable NT. It is helpful to visualize this as a

tree. We can visualize this nesting as a tree.

The tuple has the following indexes. If we consider indexes with other tuples,

we see the tuple at index 2 contains a tuple with two

elements. We can access those two indexes. The same convention applies to index

3. We can access the elements in those

tuples as well. We can continue the process. We can even

access deeper levels of the tree by adding another square bracket.

We can access different characters in the string or various elements in the

second tuple contained in the first. Lists are also a popular

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

Here is a list, "L." A list is represented with square brackets.

In many respect, lists are like tuples. One key difference is they are mutable.

Lists can contain strings, floats, integers.

We can nest other lists. We also nest tuples and other data structures.

The same indexing conventions apply for nesting

Like tuples, each element of a list can be accessed via an index.

The following table represents the relationship between the index and the

elements in the list. The first element can be accessed by the name of the list

followed by a square bracket with the index number,

in this case zero. We can access the second element as follows.

We can also access the last element. In Python, we can use a negative index;

the relationship is as follows. The corresponding indexes

are as follows. We can also perform slicing in lists. For example, if we want

the last two elements in this list we use the following command.

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

The index conventions for lists and tuples are identical.

Check the labs for more examples. We can concatenate or combine lists by adding

them. The result is the following. The new list

has the following indices. Lists are mutable, therefore we

can change them. For example, we apply the method extends

by adding a dot followed by the name of the method then

parentheses. The argument inside the parentheses is a

new list that we are going to concatenate

to the original list. In this case, instead of creating a new list,

"L1," the original list, "L," is modified by adding two new elements.

To learn more about methods check out our video on objects and classes.

Another similar method is append. If we apply append instead of extended,

we add one element to the list. If we look at the index

there is only one more element. Index 3 contains the list we appended.

Every time we apply a method, the list changes.

If we apply extend, we add two new elements to the list.

The list L is modified by adding two new elements.

If we append the string A, we further change the list,

adding the string A. As lists are mutable we can change them.

For example, we can change the first element as follows.

The list now becomes hard rock 10 1.2. We can delete an element of a list

using the del command. We simply indicate the list item we

would like to remove as an argument. For example, if we would like to remove

the first element the result becomes 10 1.2.

We can delete the second element. This operation removes the second element off

the list. We can convert a string to a list using

split. For example, the method split converts

every group of characters separated by a space

into an element of a list. We can use the split function to separate strings on a

specific character known, as a delimiter. We simply pass the

delimiter we would like to split on as an argument, in this case a comma.

The result is a list. Each element corresponds to a set of characters that

have been separated by a comma. When we set one variable B equal to A,

both A and B are referencing the same list.

Multiple names referring to the same object is known as aliasing.

We know from the list slide that the first element in B is set as hard rock.

If we change the first element in A to banana,

we get a side effect, the value of B will change as a consequence.

A and B are referencing the same list, therefore if we change A,

list B also changes. If we check the first element of B

after changing list A, we get banana instead of hard rock.

You can clone list A by using the following syntax.

Variable A references one list. Variable B

references a new copy or clone of the original list.

Now if you change A, B will not change. We can get more info on lists, tuples, and

many other objects in Python using the help command. Simply pass in the list, tuple, or any other Python object.

See the labs for more things, you can do with lists.

(Music)

# Dictionaries

0:00

Let's cover Dictionaries in Python.

Dictionaries are a type of collection in Python.

If you recall, a list is integer indexes.

These are like addresses.

A list also has elements.

A dictionary has keys and values.

The key is analogous to the index.

They are like addresses, but they don't have to be integers.

They are usually characters.

The values are similar to the element in a list and contain information.

To create a dictionary,

we use curly brackets.

The keys are the first elements.

They must be immutable and unique.

Each key is followed by a value separated by a colon.

The values can be immutable, mutable, and duplicates.

Each key and value pair is separated by a comma.

Consider the following example of a dictionary.

The album title is the key, and the value is the released data.

We can use yellow to highlight the keys and leave the values in white.

It is helpful to use the table to visualize

a dictionary where the first column represents the keys,

and the second column represents the values.

We can add a few more examples to the dictionary.

We can also assign the dictionary to a variable.

The key is used to look at the value.

We use square brackets.

The argument is the key.

This outputs the value.

Using the key of "Back in Black,"

this returns the value of 1980.

The key, "The Dark Side Of The Moon,"

gives us the value of 1973.

Using the key,"The bodyguard,"

gives us the value 1992 and so on.

We can add a new entry to the dictionary as follows.

This will add the value 2007 with a new key called "Graduation."

We can delete an entry as follows.

This gets rid of the key

"Thriller" and it's value.

We can verify if an element is in the dictionary using the "in" command as follows:

The command checks the keys.

If they are in the dictionary,

they return a true.

If we try the same command with a key that is not in the dictionary,

we get a false.

In order to see all the keys in the dictionary,

we can use the method keys to get the keys.

The output is a list-like object with all the keys.

In the same way, we can obtain the values using the method values.

Check out the labs for more examples and info. on dictionaries.

(Music)

# Sets

0:00

Let's cover sets.

They are also a type of collection.

Sets are a type of collection.

This means that like lists and tuples,

you can input different Python types.

Unlike lists and tuples, they are unordered.

This means sets do not record element position.

Sets only have unique elements.

This means there is only one of a particular element in a set.

To define a set, you use curly brackets.

You place the elements of a set within the curly brackets.

You notice there are duplicate items.

When the actual set is created,

duplicate items will not be present.

You can convert a list to a set by using the function set,

this is called type casting.

You simply use the list as the input to the function set.

The result will be a list converted to a set.

Let's go over an example.

We start off with a list.

We input the list to the function set.

The function set returns a set.

Notice how there are no duplicate elements.

Let's go over set operations.

These could be used to change the set.

Consider the set A.

Let's represent this set with a circle.

If you are familiar with sets,

this could be part of a venn diagram.

A venn diagram is a tool that uses shapes usually to represent sets.

We can add an item to a set using the add-method.

We just put the set name followed by a dot,

then the add-method.

The argument is the new element of the set we would like to add,

in this case, NSYNC.

The set A now has in NSYNC as an item.

If we add the same item twice,

nothing will happen as there can be no duplicates in a set.

Let's say we would like to remove NSYNC from set A.

We can also remove an item from a set using the remove-method.

We just put the set name followed by a dot,

then the remove-method.

The argument is the element of the set we would like to remove,

in this case, NSYNC.

After the remove-method is applied to the set,

set A does not contain the item NSYNC.

You can use this method for any item in the set.

We can verify if an element is in the set using the in command as follows.

The command checks that the item,

in this case AC/DC, is in the set.

If the item is in the set, it returns true.

If we look for an item that is not in the set,

in this case for the item Who,

adds the item is not in the set,

we will get a false.

These are types of mathematical set operations.

There are other operations we can do.

There are lots of useful mathematical operations we can do between sets.

Let's define the set album set one.

We can represent it using a red circle or venn diagram.

Similarly, we can define the set album set two.

We can also represent it using a blue circle or venn diagram.

The intersection of two sets is

a new set containing elements which are in both of those sets.

It's helpful to use venn diagrams.

The two circles that represent the sets combine,

the overlap, represents the new set.

As the overlap is comprised with the red circle and blue circle,

we define the intersection in terms of and.

In Python, we use an ampersand to find the intersection of the two sets.

If we overlay the values of the set over

the circle placing the common elements in the overlapping area,

we see the correspondence.

After applying the intersection operation,

all the items that are not in both sets disappear.

In Python, we simply just place the ampersand between the two sets.

We see that both AC/DC and Back in Black are in both sets.

The result is a new set album: set three

containing all the elements in both albums set one and album set two.

The union of two sets is the new set of

elements which contain all the items in both sets.

We can find the union of the sets album set one and album set two as follows.

The result is a new set that has all the elements of album set one and album set two.

This new set is represented in green.

Consider the new album set-album set three.

The set contains the elements AC/DC and Back in Black.

We can represent this with a Venn diagram,

as all the elements and album set three are in album set one.

The circle representing album set one

encapsulates the circle representing album set three.

We can check if a set is a subset using the issubset method.

As album set three is a subset of the album set one,

the result is true.

There is a lot more you can do with sets.

Check out the lab for more examples.

(Music)

# Conditions and Branching

0:00

In this video, you will learn about conditions and branching.

Comparison operations compares some value or operand.

Then based on some condition,

they produce a Boolean.

Let's say we assign a value of a to six.

We can use the equality operator denoted with

two equal signs to determine if two values are equal.

In this case, if seven is equal to six.

In this case, as six is not equal to seven,

the result is false.

If we performed an equality test for the value six,

the two values would be equal.

As a result, we would get a true.

Consider the following equality comparison operator:

If the value of the left operand, in this case,

the variable i is greater than the value of the right operand,

in this case five,

the condition becomes true or else we get a false.

Let's display some values for i on the left.

Let's see the value is greater than five in green and the rest in red.

If we set i equal to six,

we see that six is larger than five and as a result, we get a true.

We can also apply the same operations to floats.

If we modify the operator as follows,

if the left operand i is greater than or equal to the value of the right operand,

in this case five,

then the condition becomes true.

In this case, we include the value of

five in the number line and the color changes to green accordingly.

If we set the value of i equal to five,

the operand will produce a true.

If we set the value of i to two,

we would get a false because two is less than five.

We can change the inequality if the value of the left operand, in this case,

i is less than the value of the right operand,

in this case, six.

Then condition becomes true.

Again, we can represent this with a colored number line.

The areas where the inequality is true are marked in

green and red where the inequality is false.

If the value for i is set to two,

the result is a true.

As two is less than six.

The inequality test uses an explanation mark preceding the equal sign.

If two operands are not equal,

then the condition becomes true.

We can use a number line.

When the condition is true,

the corresponding numbers are marked in green and red for where the condition is false.

If we set i equal to two,

the operator is true as two is not equal to six.

We compare strings as well.

Comparing ACDC and Michael Jackson using the equality test,

we get a false, as the strings are not the same.

Using the inequality test,

we get a true, as the strings are different.

See the Lapps for more examples.

Branching allows us to run different statements for a different input.

It's helpful to think of an if statement as a locked room.

If this statement is true,

you can enter the room and your program can run some predefined task.

If the statement is false,

your program will skip the task.

For example, consider the blue rectangle representing an ACDC concert.

If the individual is 18 or older,

they can enter the ACDC concert.

If they are under the age of 18,

they cannot enter the concert.

Individual proceeds to the concert their age is 17,

therefore, they are not granted access to the concert and they must move on.

If the individual is 19,

the condition is true.

They can enter the concert then they can move on.

This is the syntax of the if statement from our previous example.

We have the if statement.

We have the expression that can be true or false.

The brackets are not necessary. We have a colon.

Within an indent, we have the expression that is run if the condition is true.

The statements after the if statement will run

regardless if the condition is true or false.

For the case where the age is 17,

we set the value of the variable age to 17.

We check the if statement,

the statement is false.

Therefore the program will not execute the statement to print, "you will enter".

In this case, it will just print "move on".

For the case where the age is 19,

we set the value of the variable age to 19.

We check the if statement.

The statement is true.

Therefore, the program will execute the statement to print "you will enter".

Then it will just print "move on".

The else statement will run a different block of code if the same condition is false.

Let's use the ACDC concert analogy again.

If the user is 17,

they cannot go to the ACDC concert but they

can go to the Meat Loaf concert represented by the purple square.

If the individual is 19,

the condition is true,

they can enter the ACDC concert then they can move on as before.

The syntax of the else statement is similar.

We simply append the statement else.

We then add the expression we would like to execute with an indent.

For the case where the age is 17,

we set the value of the variable age to 17.

We check the if statement,

the statement is false.

Therefore, we progress to the else statement.

We run the statement in the indent.

This corresponds to the individual attending the Meat Loaf concert.

The program will then continue running.

For the case where the age is 19,

we set the value of the variable age to 19.

We check the if statement,

the statement is true.

Therefore, the program will execute the statement to print "you will enter".

The program skips the expressions in

the else statement and continues to run the rest of the expressions.

The elif statement, short for else if,

allows us to check additional conditions if the preceding condition is false.

If the condition is true,

the alternate expressions will be run.

Consider the concert example,

if the individual is 18,

they will go to the Pink Floyd concert instead of

attending the ACDC or Meat Loaf concerts.

The person of 18 years of age enters the area as they are not over 19 years of age.

They cannot see ACDC but as their 18 years,

they attend Pink Floyd.

After seeing Pink Floyd, they move on.

The syntax of the elif statement is similar.

We simply add the statement elif with the condition.

We, then add the expression we would like to

execute if the statement is true with an indent.

Let's illustrate the code on the left.

An 18 year old enters.

They are not older than 18 years of age.

Therefore, the condition is false.

So the condition of the elif statement is checked.

The condition is true.

So then we would print "go see Pink Floyd".

Then we would move on as before.

If the variable age was 17,

the statement "go see Meat Loaf" would print.

Similarly, if the age was greater than 18,

the statement "you can enter" would print.

Check the Lapps for more examples.

Now let's take a look at logic operators.

Logic operations take Boolean values and produce different Boolean values.

The first operation is the not operator.

If the input is true,

the result is a false.

Similarly, if the input is false,

the result is a true.

Let A and B represent Boolean variables.

The OR operator takes in the two values and produces a new Boolean value.

We can use this table to represent the different values.

The first column represents the possible values of A.

The second column represents the possible values of B.

The final column represents the result of applying the OR operation.

We see the OR operator only produces a false if all the Boolean values are false.

The following lines of code will print out: "This album was made in the 70s' or 90's",

if the variable album year does not fall in the 80s.

Let's see what happens when we set the album year to 1990.

The colored number line is green when the condition is

true and red when the condition is false.

In this case, the condition is false.

Examining the second condition,

we see that 1990 is greater than 1989.

So the condition is true.

We can verify by examining the corresponding second number line.

In the final number line,

the green region indicates, where the area is true.

This region corresponds to where at least one statement is true.

We see that 1990 falls in the area.

Therefore, we execute the statement.

Let A and B represent Boolean variables.

The AND operator takes in the two values and produces a new Boolean value.

We can use this table to represent the different values.

The first column represents the possible values of A.

The second column represents the possible values of B.

The final column represents the result of applying the AND operation.

We see the OR operator only produces a true if all the Boolean values are true.

The following lines of code will print out "This album was made in the

80's" if the variable album year is between 1980 and 1989.

Let's see what happens when we set the album year to 1983.

As before, we can use the colored number line to examine where the condition is true.

In this case, 1983 is larger than 1980,

so the condition is true.

Examining the second condition,

we see that 1990 is greater than 1983.

So, this condition is also true.

We can verify by examining the corresponding second number line.

In the final number line,

the green region indicates where the area is true.

Similarly, this region corresponds to where both statements are true.

We see that 1983 falls in the area.

Therefore, we execute the statement.

Branching allows us to run different statements for different inputs.

(Music)

# Loops

0:00

In this video we will cover Loops in particular for loops and while loops.

We will use many visual examples in this video.

See the labs for examples with data.

Before we talk about loops,

let's go over the range function.

The range function outputs and ordered sequence as a list I.

If the input is a positive integer,

the output is a sequence.

The sequence contains the same number of elements as the input but starts at zero.

For example, if the input is three the output is the sequence zero, one, two.

If the range function has two inputs where

the first input is smaller than the second input,

the output is a sequence that starts at the first input.

Then the sequence iterates up to but not including the second number.

For the input 10 and 15 we get the following sequence.

See the labs for more capabilities of the range function.

Please note, if you use Python three,

the range function will not generate a list explicitly like in Python two.

In this section, we will cover for loops.

We will focus on lists, but many of the procedures can be used on tuples.

Loops perform a task over and over.

Consider the group of colored squares.

Let's say we would like to replace each colored square with a white square.

Let's give each square a number to make things a little

easier and refer to all the group of squares as squares.

If we wanted to tell someone to replace squares zero with

a white square, we would say equals replace square

zero with a white square or we can say four squares

zero in squares square zero equals white square.

Similarly, for the next square we can say for square one in squares,

square one equals white square.

For the next square we can say for square two in squares,

square two equals white square.

We repeat the process for each square.

The only thing that changes is the index of the square we are referring to.

If we're going to perform a similar task in Python we cannot use actual squares.

So let's use a list to represent the boxes.

Each element in the list is a string representing the color.

We want to change the name of the color in each element to white.

Each element in the list has the following index.

This is a syntax to perform a loop in Python.

Notice the indent, the range function generates a list.

The code will simply repeat everything in the indent five times.

If you were to change the value to six it would do it 6 times.

However, the value of I is incremented by one each time.

In this segment we change the I element of the list to the string white.

The value of I is set to zero.

Each iteration of the loop starts at the beginning of the indent.

We then run everything in the indent.

The first element in the list is set to white.

We then go to the start of the indent,

we progress down each line.

When we reach the line to change the value of the list,

we set the value of index one to white.

The value of I increases by one.

We repeat the process for index two.

The process continues for the next index,

until we've reached the final element.

We can also iterate through a list or tuple directly in python,

we do not even need to use indices.

Here is the list squares.

Each iteration of the list we pass

one element of the list squares to the variable square.

Lets display the value of the variable square on this section.

For the first iteration,

the value of square is red,

we then start the second iteration.

For the second iteration,

the value of square is yellow.

We then start the third iteration. For the final iteration, the value of square is green,

a useful function for iterating data is enumerate.

It can be used to obtain the index and the element in the list.

Let's use the box analogy with the numbers representing the index of each square.

This is the syntax to iterate through a list and provide the index of each element.

We use the list squares and use the names of the colors to represent the colored squares.

The argument of the function enumerate is the list.

In this case squares the variable I is

the index and the variable square is the corresponding element in the list.

Let's use the left part of the screen to

display the different values of the variable square

and I for the various iterations of the loop. For the first iteration,

the value of the variable is red corresponding to

the zeroth index, and the value for I is zero for the second iteration.

The value of the variable square is yellow, and

the value of I corresponds to its index i.e.

1. We repeat the process for the last index.

While loops are similar to for loops but instead of executing

a statement a set number of times a while loop will only run if a condition is met.

Let's say we would like to copy

all the orange squares from the list squares to the list New squares.

But we would like to stop if we encounter a non-orange square.

We don't know the value of the squares beforehand.

We would simply continue the process while the square is

orange or see if the square equals orange.

If not, we would stop. For the first example,

we would check if the square was orange.

It satisfies the conditions so we would copy the square.

We repeat the process for the second square. The condition is met.

So we copy the square. In the next iteration, we encounter a purple square.

The condition is not met.

So we stop the process.

This is essentially what a while loop does.

Let's use the figure on the left to represent the code. We will

use a list with the names of the color to represent the different squares.

We create an empty list of new squares.

In reality the list is of indeterminate size.

We start the index at zero the while statement will repeatedly

execute the statements within the indent until the condition inside the bracket is false.

We append the value of the first element of the list squares to the list new squares.

We increase the value of I by one.

We append the value of the second element of the list squares to the list new squares.

We increment the value of I.

Now the value in the array squares is purple;

therefore, the condition for the while statement is false and we exit the loop.

Check out the labs for more examples of loop many with real data.

(Music)

# Functions

0:00

In this video we will cover functions. You will learn how to use some of Python’s built-in

functions as well as how to build your own functions.

Functions take some input then produce some output or change.

The function is just a piece of code you can reuse.

You can implement your own function, but in many cases, you use other people’s functions.

In this case, you just have to know how the function works and in some cases how to import

the functions. Let the orange and yellow squares represent

similar blocks of code. We can run the code using some input and get

an output. If we define a function to do the task we

just have to call the function. Let the small squares represent the lines

of code used to call the function. We can replace these long lines of code by

just calling the function a few times. Now we can just call the function; our code

is much shorter. The code performs the same task.

You can think of the process like this: when we call the function f1, we pass an input

to the function. These values are passed to all those lines of code you wrote.

This returns a value; you can use the value. For example, you can input this value to a

new function f2. When we call this new function f2, the value

is passed to another set of lines of code. The function returns a value.

The process is repeated passing the values to the function you call.

You can save these functions and reuse them, or use other people’s functions.

Python has many built-in functions; you don't have to know how those functions work internally,

but simply what task those functions perform. The function len takes in an input of type

sequence, such as a string or list, or type collection, such as a dictionary or set, and

returns the length of that sequence or collection. Consider the following list.

The len function takes this list as an argument, and we assign the result to the variable L.

The function determines there are 8 items in the list, then returns the length of the

list, in this case, 8. The function sum takes in an iterable like

a tuple or list and returns the total of all the elements.

Consider the following list. We pass the list into the sum function and

assign the result to the variable S. The function determines the total of all the

elements, then returns it, in this case, the value is 70.

There are two ways to sort a list. The first is using the function sorted.

We can also use the list method sort. Methods are similar to functions.

Let's use this as an example to illustrate the difference.

The function sorted Returns a new sorted list or tuple.

Consider the list album ratings. We can apply the function sorted to the list

album ratings and get a new list sorted album rating.

The result is a new sorted list. If we look at the list album ratings, nothing

has changed. Generally, functions take an input, in this

case, a list. They produce a new output, in this instance, a sorted list.

If we use the method sort, the list album ratings will change and no new list will be

created. Let's use this diagram to help illustrate

the process. In this case, the rectangle represents the

list album ratings. When we apply the method sort to the list,

the list album rating changes. Unlike the previous case, we see that the

list album rating has changed. In this case, no new list is created.

Now that we have gone over how to use functions in Python, let’s see how to build our own

functions. We will now get you started on building your

own functions in python. This is an example of a function in python

that returns its input value + 1. To define a function, we start with the keyword

def. The name of the function should be descriptive

of what it does. We have the function formal parameter "A"

in parentheses. Followed by a colon.

We have a code block with an indent, for this case, we add 1 to "A" and assign it to B.

We return or output the value for b. After we define the function, we can call

it. The function will add 1 to 5 and return a

6. We can call the function again; this time

assign it to the variable "c" The value for 'c' is 11.

Let's explore this further. Let's go over an example when you call a function.

It should be noted that this is a simplified model of Python, and Python does not work

like this under the hood. We call the function giving it an input, 5.

It helps to think of the value of 5 as being passed to the function.

Now the sequences of commands are run, the value of "A" is 5.

"B" would be assigned a value of 6. We then return the value of b, in this case,

as b was assigned a value of 6, the function returns a 6.

If we call the function again, the process starts from scratch; we pass in an 8.

The subsequent operations are performed. Everything that happened in the last call

will happen again with a different value of "A"

The function returns a value, in this case, 9.

Again, this is just a helpful analogy. Let’s try and make this function more complex.

It's customary to document the function on the first few lines; this tells anyone who

uses the function what it does. This documentation is surrounded in triple

quotes. You can use the help command on the function

to display the documentation as follows. This will printout the function name and the

documentation. We will not include the documentation in the

rest of the examples. A function can have multiple parameters.

The function mult multiplies two numbers; in other words, it finds their product.

If we pass the integers 2 and 3, the result is a new integer.

If we pass the integer 10 and the float 3.14, the result is a float 31.4.

If we pass in the integer two and the string “Michael Jackson,” the string Michael

Jackson is repeated two times. This is because the multiplication symbol

can also mean repeat a sequence. If you accidentally multiply an integer and

a String instead of two integers, you won’t get an error.

Instead, you will get a String, and your program will progress, potentially failing later because

you have a String where you expected an integer. This property will make coding simpler, but

you must test your code more thoroughly. In many cases a function does not have a return

statement. In these cases, Python will return the special

“None” object. Practically speaking, if your function has

no return statement, you can treat it as if the function returns nothing at all.

The function MJ simply prints the name 'Michael Jackson’.

We call the function. The function prints “Michael Jackson.”

Let's define the function “No work” that performs no task.

Python doesn’t allow a function to have an empty body, so we can use the keyword pass,

which doesn’t do anything, but satisfies the requirement of a non-empty body.

If we call the function and print it out, the function returns a None.

In the background, if the return statement is not called, Python will automatically return

a None. It is helpful to view the function No Work

with the following return statement. Usually, functions perform more than one task.

This function prints a statement then returns a value.

Let's use this table to represent the different values as the function is called.

We call the function with an input of 2. We find the value of b.

The function prints the statement with the values of a and b.

Finally, the function returns the value of b, in this case, 3.

We can use loops in functions. This function prints out the values and indexes

of a loop or tuple. We call the function with the list album ratings

as an input. Let's display the list on the right with its

corresponding index. Stuff is used as an input to the function

enumerate. This operation will pass the index to i and

the value in the list to “s”. The function would begin to iterate through

the loop. The function will print the first index and

the first value in the list. We continue iterating through the loop.

The values of i and s are updated. The print statement is reached.

Similarly, the next values of the list and index are printed.

The process is repeated. The values of i and s are updated.

We continue iterating until the final values in the list are printed out.

Variadic parameters allow us to input a variable number of elements.

Consider the following function; the function has an asterisk on the parameter names.

When we call the function, three parameters are packed into the tuple names.

We then iterate through the loop; the values are printed out accordingly.

If we call the same function with only two parameters as inputs, the variable names only

contain two elements. The result is only two values are printed

out. The scope of a variable is the part of the

program where that variable is accessible. Variables that are defined outside of any

function are said to be within the global scope, meaning they can be accessed anywhere

after they are defined. Here we have a function that adds the string

DC to the parameter x. When we reach the part where the value of

x is set to AC, this is within the global scope, meaning x is accessible anywhere after

it is defined. A variable defined in the global scope is

called a global variable. When we call the function, we enter a new

scope or the scope of AddDC. We pass as an argument to the AddDC function,

in this case, AC. Within the scope of the function, the value

of x is set to ACDC. The function returns the value and is assigned

to z. Within the global scope, the value z is set

to ACDC After the value is returned, the scope of

the function is deleted. Local variables only exist within the scope

of a function. Consider the function thriller; the local

variable Date is set to 1982. When we call the function, we create a new

scope. Within that scope of the function, the value

of the date is set to 1982. The value of date does not exist within the

global scope. Variables inside the global scope can have

the same name as variables in the local scope with no conflict.

Consider the function thriller; the local variable Date is set to 1982.

The global variable date is set to 2017. When we call the function, we create a new

scope. Within that scope, the value of the date is

set to 1982. If we call the function, it returns the value

of Date in the local scope, in this case, 1982.

(click6) When we print in the global scope, we use the global variable value.

The global value of the variable is 2017. Therefore, the value is set to 2017.

If a variable is not defined within a function, Python will check the global scope.

Consider the function "AC-DC“. The function has the variable rating, with no value assigned.

If we define the variable rating in the global scope, then call the function, Python will

see there is no value for the variable Rating. As a result, python will leave the scope and

check if the variable Ratings exists in the global scope. It will use this value of Ratings

in the global scope within the scope of "AC-DC“. In the function, will print out a 9.

The value of z in the global scope will be 10, as we added one.

The value of rating will be unchanged within the global scope.

Consider the function Pink Floyd. If we define the variable Claimed Sales with

the keyword global, the variable will be a global variable.

We call the function Pink Floyd. The variable claimed sales is set to the string

“45 million” in the global scope. When we print the variable, we get a value

of “45 million.” There is a lot more you can do with functions.

Check out the lab for more examples.

(Music)

# Exception Handling

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-applied-data-science-ai/discussions/weeks/3)

0:07

Hello.

Welcome to Exception Handling.

After watching this video, you will be able to:

Explain Exception Handling,

Demonstrate the use of exception handling, and

Understand the basics of exception handling.

Have you ever mistakenly entered a number when you were supposed to enter text in an

input field?

Most of us have either in error or when testing out a program, but do you know why it gave

an error message instead of completing and terminating the program?

In order for the error message to appear an event was triggered in the background.

This event was activated because the program tried to perform a computation on the name

entry and realized the entry contained numbers and not letters.

By encasing this code in an exception handler the program knew how to deal with this type

of error and was able to output the error message to continue along with the program.

This is one of many errors that can happen when asking for user input, so let’s see

how exception handling works.

Let’s first explore the try…except statement.

This type of statement will first attempt to execute the code in the “try” block,

but if an error occurs it will kick out and begin searching for the exception that matches

the error.

Once it finds the correct exception to handle the error it will then execute that line of

code.

For example, let’s say you are writing a program that will open and write a file.

After starting the program an error occurred as the data was not able to be read.

Because of this error the program skipped over the code lines under the “try” statement

and went directly to the exception line.

Since this error fell within the IOError guidelines it printed “Unable to open or read the data

in the file.” to our console.

When writing simple programs we can sometimes get away with only one except statement, but

what happens if another error occurs that is not caught by the IOError?

If that happened we would need to add another except statement.

For this except statement you will notice that the type of error to catch is not specified.

While this may seem a logical step so the program will catch all errors and not terminate

this is not a best practice.

For example, let’s say our small program was just one section of a much larger program

that was over a thousand lines of code.

Our task was to debug the program as it kept throwing an error causing a disruption for

our users.

When investigating the program you found this error kept appearing.

Because this error had no details you ended up spending hours trying to pinpoint and fix

the error.

So far in our program we have defined that an error message should print out if an error

occurs, but we don’t receive any messages that the program executed properly.

This is where we can now add an else statement to give us that notification.

By adding this else statement it will provide us a notification to the console that “The

file was written successfully”.

Now that we have defined what will happen if our program executes properly, or if an

error occurs there is one last statement to add.

For this example since we are opening a file the last thing we need to do is close the

file.

By adding a finally statement it will tell the program to close the file no matter the

end result and print “File is now closed” to our console.

In this video, you learned:

How to write a try…except statement,

Why is it important to always define errors when creating exceptions, and

How to add an else and finally statement.

# Objects and Classes

0:00

In this module, we're going to talk about objects and classes.

Python has many different kinds of data types: integers,

floats, strings, lists, dictionaries, booleans.

In Python, each is an object.

Every object has the following: a type,

internal representation, a set of functions called methods to interact with the data.

An object is an instance of a particular type.

For example, we have two types,

type one and type two.

We can have several objects of type one as shown in yellow.

Each object is an instance of type one.

We also have several objects of type two shown in green.

Each object is an instance of type two.

Let's do several less abstract examples.

Every time we create an integer,

we are creating an instance of type integer,

or we are creating an integer object.

In this case, we are creating five instances of type integer or five integer objects.

Similarly, every time we create a list,

we are creating an instance of type list,

or we are creating a list object.

In this case, we are creating five instances of type list or five list objects.

We could find out the type of an object by using the type command.

In this case, we have an object of type list,

we have an object of type integer,

we have an object of type string.

Finally, we have an object of type dictionary.

A class or type's methods are functions

that every instance of that class or type provides.

It's how you interact with the object.

We have been using methods all this time,

for example, on lists.

Sorting is an example of a method that interacts with the data in the object.

Consider the list ratings,

the data is a series of numbers contained within the list.

The method sort will change the data within the object.

We call the method by adding a period at the end of the object's name,

and the method's name we would like to call with parentheses.

We have the rating's list represented in orange.

The data contained in the list is a sequence of numbers.

We call the sort method,

this changes the data contained in the object.

You can say it changes the state of the object.

We can call the reverse method on the list,

changing the list again.

We call the method, reversing the order of the sequence within the object.

In many cases, you don't have to know the inner workings of the class and its methods,

you just have to know how to use them.

Next, we will cover how to construct your own classes.

You can create your own type or class in Python.

In this section, you'll create a class.

The class has data attributes.

The class has methods.

We then create instances or instances of that class or objects.

The class data attributes define the class.

Let's create two classes.

The first class will be a circle,

the second will be a rectangle.

Let's think about what constitutes a circle.

Examining this image, all we need is a radius to define a circle,

and let's add color to make it easier to

distinguish between different instances of the class later.

Therefore, our class data attributes are radius and color.

Similarly, examining the image in order to define a rectangle,

we need the height and width.

We will also add color to distinguish between instances later.

Therefore, the data attributes are color, height, and width.

To create the class circle,

you will need to include the class definition.

This tells Python you're creating your own class,

the name of the class.

For this course in parentheses,

you will always place the term object,

this is the parent of the class.

For the class rectangle,

we changed the name of the class,

but the rest is kept the same.

Classes are outlines we have to set the attributes to create objects.

We can create an object that is an instance of type circle.

The color data attribute is red,

and the data attribute radius is four.

We could also create a second object that is an instance of type circle.

In this case, the color data attribute is green,

and the data attribute radius is two.

We can also create an object that is an instance of type rectangle.

The color data attribute is blue,

and the data attribute of height and width is two.

The second object is also an instance of type rectangle.

In this case, the color data attribute is yellow,

and the height is one,

and the width is three.

We now have different objects of class circle or type circle.

We also have different objects of class rectangle or type rectangle.

Let us continue building the circle class in Python.

We define our class.

We then initialize each instance of the class with data attributes,

radius, and color using the class constructor.

The function init is a constructor.

It's a special function that tells Python you are making a new class.

There are other special functions in Python to make more complex classes.

The radius and color parameters are used to

initialize the radius and color data attributes of the class instance.

The self parameter refers to the newly created instance of the class.

The parameters, radius, and color can be used in the constructors body to

access the values passed to the class constructor when the class is constructed.

We could set the value of

the radius and color data attributes to the values passed to the constructor method.

Similarly, we can define the class rectangle in Python.

The name of the class is different.

This time, the class data attributes are color, height, and width.

After we've created the class,

in order to create an object of class circle,

we introduce a variable.

This will be the name of the object.

We create the object by using the object constructor.

The object constructor consists of the name of the class as well as the parameters.

These are the data attributes.

When we create a circle object,

we call the code like a function.

The arguments passed to the circle constructor are used to

initialize the data attributes of the newly created circle instance.

It is helpful to think of self as a box that

contains all the data attributes of the object.

Typing the object's name followed by a dot and the data attribute

name gives us the data attribute value, for example, radius.

In this case, the radius is 10.

We can do the same for color.

We can see the relationship between the self parameter and the object.

In Python, we can also set or change the data attribute directly.

Typing the object's name followed by a dot and the data attribute name,

and set it equal to the corresponding value.

We can verify that the color data attribute has changed.

Usually, in order to change the data in an object,

we define methods in the class.

Let's discuss methods.

We have seen how data attributes consist of the data defining the objects.

Methods are functions that interact and change the data attributes,

changing or using the data attributes of the object.

Let's say we would like to change the size of a circle.

This involves changing the radius attribute.

We add a method, add radius to the class circle.

The method has a function that requires the self as well as other parameters.

In this case, we are going to add a value to the radius,

We denote that value as r. We are going to add r to the data attribute radius.

Let's see how this part of the code works when we create

an object and call the add\_radius method.

As before, we create an object with the object constructor.

We pass two arguments to the constructor.

The radius is set to two and the color is set to red.

In the constructor's body,

the data attributes are set.

We can use the box analogy to see the current state of the object.

We call the method by adding a dot followed by the method, name, and parentheses.

In this case, the argument of the function is the amount we would like to add.

We do not need to worry about the self parameter when calling the method.

Just like with the constructor,

Python will take care of that for us.

In many cases, there may not be

any parameters other than self specified in the method's definition.

So we don't pass any arguments when calling the function.

Internally, the method is called with a value of eight,

and the proper self object.

The method assigns a new value to self radius.

This changes the object,

in particular, the radius data attribute.

When we call the add\_radius method,

this changes the object by changing the value of the radius data attribute.

We can add default values to the parameters of a class as constructor.

In the labs, we also create the method called drawCircle.

See the lab for the implementation of drawCircle.

In the labs, we can create a new object of type circle using the constructor.

The color will be red and the radius will be three.

We can access the data attribute radius.

We can access the attribute color.

Finally, we can use the method drawCircle to draw the circle.

Similarly, we can create a new object of type circle.

We can access the data attribute of radius.

We can access the data attribute color.

We can use the method drawCircle to draw the circle.

In summary, we have created an object of

class circle called RedCircle with a radius attribute of three,

and a color attribute of red.

We also created an object of class circle called BlueCircle,

with a radius attribute of 10 and a color attribute of blue.

In the lab, we have a similar class for rectangle.

We can create a new object of type rectangle using the constructor.

We can access a data attribute of height.

We can also access the data attribute of width.

We could do the same for the data attribute of color.

We can use the method drawRectangle to draw the rectangle.

So we have a class,

an object that is a realization or instantiation of that class.

For example, we can create two objects of class Circle,

or two objects of class Rectangle.

The dir function is useful for obtaining the list

of data attributes and methods associated with a class.

The object you're interested in is passed as an argument.

The return value is a list of the objects data attributes.

The attribute surrounded by underscores are for internal use,

and you shouldn't have to worry about them.

The regular looking attributes are the ones you should concern yourself with.

These are the objects,

methods, and data attributes.

There is a lot more you can do with objects in Python.

Check Python.org for more info.

(Music)

# Reading Files with Open

0:00

In this section, we will use Python's built-in open function to create a file object,

and obtain the data from a "txt" file.

We will use Python's open function to get a file object.

We can apply a method to that object to read data from the file.

We can open the file, Example1.txt, as follows.

We use the open function.

The first argument is the file path.

This is made up of the file name,

and the file directory.

The second parameter is the mode.

Common values used include 'r' for reading,

'w' for writing, and 'a' for appending.

We will use 'r' for reading.

Finally, we have the file object.

We can now use the file object to obtain information about the file.

We can use the data attribute name to get the name of the file.

The result is a string that contains the name of the file.

We can see what mode the object is in using the data attribute mode,

and 'r' is shown representing read.

You should always close the file object using the method close.

This may get tedious sometimes,

so let's use the "with" statement.

Using a "with" statement to open a file is

better practice because it automatically closes the file.

The code will run everything in the indent block,

then closes the file.

This code reads the file, Example1.txt.

We can use the file object, "File1."

The code will perform all operations in

the indent block then close the file at the end of the indent.

The method "read" stores the values of the file in the variable "file\_stuff" as a string.

You can print the file content.

You can check if the file content is closed,

but you cannot read from it outside the indent.

But you can print the file content outside the indent as well.

We can print the file content.

We will see the following.

When we examine the raw string,

we will see the " ."

This is so Python knows to start a new line.

We can output every line as an element in a list using the method "readlines."

The first line corresponds to the first element in the list.

The second line corresponds to the second element in the list, and so on.

We can use the method "readline" to read the first line of the file.

If we run this command,

it will store the first line in the variable "file\_stuff" then print the first line.

We can use the method "readline" twice.

The first time it's called,

it will save the first line in the variable "file\_stuff," and then print the first line.

The second time it's called,

it will save the second line in the variable

"file\_stuff," and then print the second line.

We can use a loop to print out each line individually as follows.

Let's represent every character in a string as a grid.

We can specify the number of characters we would like to read from

the string as an argument to the method "readlines."

When we use a four as an argument in the method

"readlines," we print out the first four characters in the file.

Each time we call the method,

we will progress through the text.

If we call a method with the arguments 16,

the first 16 characters are printed out, and then the new line.

If we call the method a second time,

the next five characters are printed out.

Finally, if we call the method the last time with the argument nine,

the last nine characters are printed out.

Check out the labs for more examples of methods and other file types.

(Music)

# Writing Files with Open

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-applied-data-science-ai/discussions/weeks/4)

0:00

We can also write to files using the open function.

We will use Python's open function to get a file object to create a text file.

We can apply method write to write data to that file.

As a result, text will be written to the file.

We can create the file Example2.txt as follows.

We use the open function.

The first argument is the file path.

This is made up of the file name.

If you have that file in your directory,

it will be overwritten, and the file directory.

We set the mode parameter to W for writing.

Finally, we have the file object.

As before we use the with statement.

The code will run everything in the indent block then close the file.

We create the file object, File1.

We use the open function.

This creates a file Example2.txt in your directory.

We use the method write,

to write data into the file.

The argument is the text we would like input into the file.

If we use the write method successively,

each time it is called, it will write to the file.

The first time it is called, we will write,

"This is line A " to represent a new line.

The second time we call the method, it will write,

"this is line B " then it will close the file.

We can write each element in a list to a file.

As before, we use a with command and the open function to create a file.

The list, Lines, has three elements consisting of text.

We use a for loop to read each element of

the first lines and pass it to the variable line.

The first iteration of the loop writes

the first element of the list to the file Example2.

The second iteration writes the second element of the list and so on.

At the end of the loop, the file will be closed.

We can set the mode to appended using a lowercase a.

This will not create a new file but just use the existing file.

If we call the method write,

it will just write to the existing file,

then add "This is line C" then close the file.

We can copy one file to a new file as follows.

First, we read the file Example1 and interact with it via the file object, read file.

Then we create a new file Example3 and

use the file object write file to interact with it.

The for loop takes a line from the file object, read file,

and stores it in the file Example3 using the file object, write file.

The first iteration copies the first line.

The second iteration copies the second line,

till the end of the file is reached.

Then both files are closed.

Check out the labs for more examples.

(Music)

# Loading Data with Pandas

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-applied-data-science-ai/discussions/weeks/4)

0:00

Dependencies or libraries are pre-written code to help solve problems.

In this video, we will introduce Pandas,

a popular library for data analysis.

We can import the library or a dependency like Pandas using the following command.

We start with the import command followed by the name of the library.

We now have access to a large number of pre-built classes and functions.

This assumes the library is installed.

In our lab environment,

all the necessary libraries are installed.

Let's say we would like to load a CSV file using the Pandas built-in function, read csv.

A CSV is a typical file type used to store data.

We simply typed the word Pandas,

then a dot, and the name of the function with all the inputs.

Typing Pandas all the time may get tedious.

We can use the as statement to shorten the name of the library.

In this case, we use the standard abbreviation, pd.

Now we type pd, and a dot,

followed by the name of the function we would like to use.

In this case, read\_csv.

We are not limited to the abbreviation pd.

In this case, we use the term banana.

We will stick with pd for the rest of this video.

Let's examine this code more in-depth.

One way Pandas allows you to work with data is with the data frame.

Let's go over the process to go from a CSV file to a data frame.

This variable stores the path of the CSV.

It is used as an argument to the read\_csv function.

The result is stored to the variable df.

This is short for data frame.

Now that we have the data in a data frame, we can work with it.

We can use the method head to examine the first five rows of a data frame.

The process for loading an Excel file is similar.

We use the path of the Excel file.

The function reads Excel.

The result is a data frame.

A data frame is comprised of rows and columns.

We can create a data frame out of a dictionary.

The keys correspond to the column labels.

The values or lists corresponding to the rows.

We then cast the dictionary to a data frame using the function data frame.

We can see the direct correspondence between the table.

The keys correspond to the table headers.

The values are lists corresponding to the rows.

We can create a new data frame consisting of one column.

We just put the data frame name, in this case, df,

and the name of the column header enclosed in double brackets.

The result is a new data frame comprised of the original column.

You can do the same thing for multiple columns.

We just put the data frame name, in this case, df,

and the name of the multiple column headers enclosed in double brackets.

The result is a new data frame comprised of the specified columns.

(Music)

# Pandas: Working with and Saving Data

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-applied-data-science-ai/discussions/weeks/4)

0:00

When we have a data frame we can work with

the data and save the results in other formats.

Consider the stack of 13 blocks of different colors.

We can see there are three unique colors.

Let's say you would like to find out

how many unique elements are in a column of a data frame.

This may be much more difficult because instead of 13 elements,

you may have millions.

Pandas has the method unique to

determine the unique elements in a column of a data frame.

Lets say we would like to determine the unique year of the albums in the data set.

We enter the name of the data frame,

then enter the name of the column released within brackets.

Then we apply the method unique.

The result is all of the unique elements in the column released.

Let's say we would like to create a new database

consisting of songs from the 1980s and after.

We can look at the column released for songs made after 1979,

then select the corresponding columns.

We can accomplish this within one line of code in Pandas.

But let's break up the steps.

We can use the inequality operators for the entire data frame in Pandas.

The result is a series of Boolean values.

For our case, we simply specify the column

released and the inequality for the albums after 1979.

The result is a series of Boolean values.

The result is true when the condition is true and false otherwise.

We can select the specified columns in one line.

We simply use the data frames names and square brackets we placed

the previously mentioned inequality and assign it to the variable df1.

We now have a new data frame,

where each album was released after 1979.

We can save the new data frame using the method to\_csv.

The argument is the name of the csv file.

Make sure you include a.csv extension.

There are other functions to save the data frame in other formats.

(Music)

# One Dimensional Numpy

0:00

In this video we will be covering numpy in 1D,

in particular ND arrays.

Numpy is a library for scientific computing.

It has many useful functions.

There are many other advantages like speed and memory.

Numpy is also the basis for pandas.

So check out our pandas video.

In this video we will be covering

the basics and array creation,

indexing and slicing,

basic operations, universal functions.

Let's go over how to create a numpy array.

A Python list is a container that

allows you to store and access data.

Each element is associated with an index.

We can access each element using

a square bracket as follows.

A numpy array or ND array is similar to a list.

It's usually fixed in size

and each element is of the same type,

in this case integers.

We can cast a list to

a numpy array by first importing numpy.

We then cast the list as follows;

we can access the data via an index.

As with the list, we can access

each element with an integer and a square bracket.

The value of a is stored as follows.

If we check the type of the array we get, numpy.ndarray.

As numpy arrays contain data of the same type,

we can use the attribute

dtype to obtain the data type of the array's elements.

In this case a 64-bit integer.

Let's review some basic array attributes

using the array a.

The attribute size is

the number of elements in the array.

As there are five elements the result is five.

The next two attributes will make

more sense when we get to higher dimensions,

but let's review them.

The attribute ndim represents

the number of array dimensions or the rank of the array,

in this case one.

The attribute shape is a tuple of

integers indicating the size of

the array in each dimension.

We can create a numpy array with real numbers.

When we check the type of the array,

we get numpy.ndarray.

If we examine the attribute D type,

we see float64 as the elements are not integers.

There were many other attributes, check out numpy.org.

Let's review some indexing and slicing methods.

We can change the first element of

the array to 100 as follows.

The array's first value is now 100.

We can change the fifth element of the array as follows.

The fifth element is now zero.

Like lists and tuples we can slice a NumPy array.

The elements of the array correspond

to the following index.

We can select the elements from one to three and assign

it to a new numpy array d as follows.

The elements in d correspond to the index.

Like lists, we do not count

the element corresponding to the last index.

We can assign the corresponding indices

to new values as follows.

The array c now has new values.

See the labs or numpy.org for

more examples of what you can do with numpy.

Numpy makes it easier to do

many operations that are

commonly performed in data science.

The same operations are

usually computationally faster and

require less memory in numpy compared to regular Python.

Let's review some of these operations

on one-dimensional arrays.

We will look at many of the operations in the context of

Euclidian vectors to make things more interesting.

Vector addition is a widely used operation

in data science.

Consider the vector u with two elements,

the elements are distinguished by the different colors.

Similarly, consider the vector v with two components.

In vector addition, we create

a new vector in this case z.

The first component of z

is the addition of the first component

of vectors u and v. Similarly,

the second component is

the sum of the second components of

u and v. This new vector z is now

a linear combination of the vector u and

v. Representing vector addition

with line segment or arrows is helpful.

The first vector is represented in red.

The vector will point in

the direction of the two components.

The first component of the vector is one.

As a result the arrow is offset

one unit from the origin in the horizontal direction.

The second component is zero,

we represent this component in the vertical direction.

As this component is zero,

the vector does not point in the vertical direction.

We represent the second vector in blue.

The first component is zero,

therefore the arrow does not

point to the horizontal direction.

The second component is one.

As a result the vector points in

the vertical direction one unit.

When we add the vector u and v,

we get the new vector z.

We add the first component,

this corresponds to the horizontal direction.

We also add the second component.

It's helpful to use the tip to

tail method when adding vectors,

placing the tail of the vector v on the tip of vector u.

The new vector z is constructed by connecting

the base of the first vector u with the tail of the

second v. The following three lines of code

we'll add the two lists and place

the result in the list z.

We can also perform

vector addition with one line of NumPy code.

It would require multiple lines to perform

vector subtraction on two lists

as shown on the right side of the screen.

In addition, the numpy code will run much faster.

This is important if you have lots of data.

We can also perform vector subtraction by changing

the addition sign to a subtraction sign.

It would require multiple lines

perform vector subtraction

on two lists as shown on the right side of the screen.

Vector multiplication with a scalar is

another commonly performed operation.

Consider the vector y,

each component is specified by a different color.

We simply multiply the vector by

a scalar value in this case two.

Each component of the vector is multiplied by two,

in this case each component is doubled.

We can use the line segment or

arrows to visualize what's going on.

The original vector y is in purple.

After multiplying it by a scalar value of two,

the vector is stretched out by two units as shown in red.

The new vector is twice as long in each direction.

Vector multiplication with a scalar only

requires one line of code using numpy.

It would require multiple lines

to perform the same task as

shown with Python lists

as shown on the right side of the screen.

In addition, the operation would also be much slower.

Hadamard product is

another widely used operation in data science.

Consider the following two vectors,

u and v. The Hadamard product

of u and v is a new vector z.

The first component of z is the product of

the first element of u and v. Similarly,

the second component is

the product of the second element of

u and v. The resultant vector consists of

the entry wise product of u and v. We can

also perform hadamard product

with one line of code in numpy.

It would require multiple lines to perform

hadamard product on two lists

as shown on the right side of the screen.

The dot product is

another widely used operation in data science.

Consider the vector u and v,

the dot product is a single number given by

the following term and

represents how similar two vectors are.

We multiply the first component from v and u,

we then multiply the second component

and add the result together.

The result is a number that represents

how similar the two vectors are.

We can also perform dot product using the numpy function

dot and assign it with the variable result as follows.

Consider the array u,

the array contains the following elements.

If we add a scalar value to the array,

numpy will add that value to each element.

This property is known as broadcasting.

A universal function is a function that

operates on ND arrays.

We can apply a universal function to a numpy array.

Consider the arrays a,

we can calculate the mean or average value of

all the elements in a using the method mean.

This corresponds to the average of all the elements.

In this case the result is zero.

There are many other functions.

For example, consider the numpy arrays b.

We can find the maximum value using the method five.

We see the largest value is five,

therefore the method max returns a five.

We can use numpy to create functions that

map numpy arrays to new numpy arrays.

Let's implement some code on the left side of the screen

and use the right side of

the screen to demonstrate what's going on.

We can access the value of pie in numpy as follows.

We can create the following numpy array in radians.

This array corresponds to the following vector.

We can apply the function sin to the array

x and assign the values to the array y.

This applies the sin function

to each element in the array,

this corresponds to applying

the sine function to each component of the vector.

The result is a new array y,

where each value corresponds to

a sine function being applied to

each element in the array x.

A useful function for plotting

mathematical functions is line space.

Line space returns evenly

spaced numbers over specified interval.

We specify the starting point of the sequence,

the ending point of the sequence.

The parameter num indicates

the number of samples to generate,

in this case five.

The space between samples is one.

If we change the parameter num to nine,

we get nine evenly spaced numbers

over the integral from negative two to two.

The result is the difference

between subsequent samples is

0.5 as opposed to one as before.

We can use the function line space to generate 100

evenly spaced samples from the interval zero to two pie.

We can use the numpy function sin to

map the array x to a new array y.

We can import the library pyplot as

plt to help us plot the function.

As we are using a Jupiter notebook,

we use the command matplotlib inline to display the plot.

The following command plots a graph.

The first input corresponds to

the values for the horizontal or x-axis.

The second input corresponds to

the values for the vertical or y-axis.

There's a lot more you can do with numpy.

Check out the labs and numpy.org for more.

Thanks for watching this video.

(Music)

# Two Dimensional Numpy

0:00

We can create numpy arrays with more than one dimension.

This section will focus only on 2D arrays but

you can use numpy to build

arrays of much higher dimensions.

In this video, we will cover

the basics and array creation in 2D,

indexing and slicing in 2D,

and basic operations in 2D.

Consider the list a,

the list contains three nested lists each of equal size.

Each list is color-coded for simplicity.

We can cast the list to a numpy array as follows.

It is helpful to visualize the numpy array as

a rectangular array each

nested lists corresponds to

a different row of the matrix.

We can use the attribute ndim to obtain

the number of axes or dimensions referred to as the rank.

The term rank does not refer to the number of

linearly independent columns like a matrix.

It's useful to think of

ndim as the number of nested lists.

The first list represents the first dimension.

This list contains another set of lists.

This represents the second dimension or axis.

The number of lists the list contains does not

have to do with the dimension but the shape of the list.

As with a 1D array,

the attribute shape returns a tuple.

It's helpful to use

the rectangular representation as well.

The first element in the tuple

corresponds to the number of nested lists contained

in the original list or the number of

rows in the rectangular representation,

in this case three.

The second element corresponds to the size of each of

the nested list or the number of

columns in the rectangular array zero.

The convention is to label this axis

zero and this axis one as follows.

We can also use the attribute size

to get the size of the array.

We see there are three rows and three columns.

Multiplying the number of columns and rows together,

we get the total number of elements,

in this case nine.

Check out the labs for arrays of

different shapes and other attributes.

We can use rectangular brackets to

access the different elements of the array.

The following image demonstrates the relationship between

the indexing conventions for

the lists like representation.

The index in the first bracket corresponds to

the different nested lists each a different color.

The second bracket corresponds to the index

of a particular element within the nested list.

Using the rectangular representation,

the first index corresponds to the row index.

The second index corresponds to the column index.

We could also use a single bracket

to access the elements as follows.

Consider the following syntax.

This index corresponds to the second row,

and this index the third column,

the value is 23.

Consider this example, this index corresponds to

the first row and

the second index corresponds to the first column,

and a value of 11.

We can also use slicing in numpy arrays.

The first index corresponds to the first row.

The second index accesses the first two columns.

Consider this example,

the first index corresponds to the first two rows.

The second index accesses the last column.

We can also add arrays,

the process is identical to matrix addition.

Consider the matrix X,

each element is colored differently.

Consider the matrix Y.

Similarly, each element is colored differently.

We can add the matrices.

This corresponds to adding

the elements in the same position,

i.e adding elements contained

in the same color boxes together.

The result is a new matrix that has

the same size as matrix Y or X.

Each element in this new matrix is the sum of

the corresponding elements in X and Y.

To add two arrays in numpy,

we define the array in this case X.

Then we define the second array Y, we add the arrays.

The result is identical to matrix addition.

Multiplying a numpy array by

a scalar is identical to

multiplying a matrix by a scalar.

Consider the matrix Y.

If we multiply the matrix by this scalar two,

we simply multiply every element in the matrix by two.

The result is a new matrix of

the same size where each element is multiplied by two.

Consider the array Y.

We first define the array,

we multiply the array by a scalar as

follows and assign it to the variable Z.

The result is a new array

where each element is multiplied by two.

Multiplication of two arrays corresponds to

an element-wise product, or Hadamard product.

Consider array X and array

Y. Hadamard product corresponds to multiplying each

of the elements in the same position i.e

multiplying elements contained in

the same color boxes together.

The result is a new matrix that is

the same size as matrix Y or X.

Each element in this new matrix is

the product of the corresponding elements in X and Y.

Consider the array X and Y.

We can find the product of

two arrays X and Y in one line,

and assign it to the variable Z as follows.

The result is identical to Hadamard product.

We can also perform

matrix multiplication with Numpy arrays.

Matrix multiplication is a little more

complex but let's provide a basic overview.

Consider the matrix A

where each row is a different color.

Also, consider the matrix B

where each column is a different color.

In linear algebra, before we

multiply matrix A by matrix B,

we must make sure that the number of

columns in matrix A in

this case three is

equal to the number of rows in matrix B,

in this case three.

From matrix multiplication, to obtain

the ith row and jth column of the new matrix,

we take the dot product of the ith row

of a with the jth columns of B.

For the first column,

first row we take the dot product of

the first row of A with the first column of B as follows.

The result is zero.

For the first row and the

second column of the new matrix,

we take the dot product of the first row of the matrix A,

but this time we use the second column of matrix B,

the result is two.

For the second row and

the first column of the new matrix,

we take the dot product of

the second row of the matrix A.

With the first column of matrix B,

the result is zero.

Finally, for the second row

and the second column of the new matrix,

we take the dot product of the second row of

the matrix A with the second column of matrix B,

the result is two.

In numpy, we can define the numpy arrays A and B.

We can perform matrix multiplication and

assign it to array C. The result is

the array C. It corresponds to

the matrix multiplication of array A and B.

There is a lot more you can do with it in numpy.

Checkout numpy.org.

Thanks for watching this video.

(Music)

Two Dimensional Numpy: Added to Selection. Press [CTRL + S] to save as a note

# Simple APIs (Part 1)

0:07

In this video we will discuss Application Program Interfaces, or APIs for short.

Specifically, we will discuss: What an API is, API Libraries, and

REST APIs, including: Request and Response and

an example with PyCoinGecko. An API lets two pieces

of software talk to each other. For example, you have your program,

you have some data, and you have other software components.

You use the API to communicate with other software via inputs and outputs.

Just like a function, you don’t have to know how the API works, just its inputs and outputs.

Pandas is actually a set of software components, much of which are not even written in Python.

You have some data. You have a set of software components.

We use the pandas API to process the data by communicating with the other Software Components.

Let’s clean up the diagram. When you create a dictionary,

and then create a pandas object with the Dataframe constructor, in API lingo, this is an “instance.”

The data in the dictionary is passed along to the pandas API.

You then use the dataframe to communicate with the API.

When you call the method head, the dataframe communicates with the API

displaying the first few rows of the dataframe. When you call the method mean

the API will calculate the mean and return the values.

Play video starting at :1:32 and follow transcript1:32

REST APIs are another popular type of API; they allow you to communicate through the

internet letting you to take advantage of resources like storage, access more data,

artificial intelligent algorithms, and much more. The RE stands for Representational,

the S stands for State, the T stand for Transfer.

In rest APIs your program is called the client. The API communicates with a web

service you call through the internet. There is a set of rules regarding Communication,

Input or Request, and Output or Response. Here are some common terms.

You or your code can be thought of as a client. The web service is referred to as a resource.

The client finds the service via an endpoint. We will review this more in the next section.

The client sends requests to the resource and the response to the client.

HTTP methods are a way of transmitting data over the internet

We tell the Rest APIs what to do by sending a request. The request

is usually communicated via an HTTP message. The HTTP message usually contains a JSON file.

This contains instructions for what operation we would like the service to perform.

This operation is transmitted to the webservice via the internet.

And the service performs the operation. In a similar manner, the webservice

returns a response via an HTTP message, where the information is usually returned

via a JSON file. This information is transmitted back to the client.

Crypto currency data is excellent to use in an API because it is constantly updated

and is vital to Crypto currency trading. We will use the PyCoinGecko Python

client or wrapper for the Coin Gecko API, updated every minute by CoinGecko

We use the wrapper, or client, because it is easy to use so you can focus on the task of collecting

data, we will also introduce pandas time series functions for dealing with time series data.

Using PyCoinGecko to collect data is simple. All we need is to install

and import the library, then create a client object, and finally, use a function to request our data.

In this function we are getting data on bitcoin, in US Dollars, for the past

30 days. In this case our response is a JSON expressed as a Python dictionary of nested lists

including price, market cap, and total volumes which contain the Unix timestamp and the price

at that time. We are only interested in price so that is what we will select using the key price.

To make things simple, we can convert our nested list to a DataFrame.

With the columns time stamp and price, it is difficult to understand the column time stamp.

We will convert it to a more readable format using the pandas function to\_datetime.

Play video starting at :4:28 and follow transcript4:28

Using this to\_datetime function, we create readable time data, the input is the time stamp

column, unit of time is set to milliseconds. We append the output to the new column date.

Play video starting at :4:43 and follow transcript4:43

Now we want to create a candle stick plot. To get the data for the daily candlesticks,

we will group by the date to find the minimum, maximum,

first, and last price of each day. Finally, we will use plotly to create

the candlestick chart and plot it.

Now we can view the candlestick chart by opening the HTML file and clicking

trust HTML in the top left of the tab. It should look something like this.

# Simple APIs (Part 2)

0:00

In this video, we will discuss

Application Program Interfaces that

use some kind of artificial intelligence.

We will transcribe an audio file using

the Watson Text to Speech API.

We will then translate the text to

a new language using the Watson Language Translator API.

In the API call,

you will send a copy of the audio file to the API.

This is sometimes called a POST request.

Then the API will send

the text transcription of what the individual is saying.

Under the hood, the API is making a GET request.

We then send the text we would like to translate

into a second language to a second API.

The API will translate the text

and send the translation back to you.

In this case, we translate English to Spanish.

We then provide an overview of API keys and endpoints,

Watson Speech to Text, and Watson Translate.

First, we will review API keys and endpoints.

They will give you access to the API.

An API key as a way to access the API.

It's a unique set of characters that the API

uses to identify you and authorize you.

Usually, your first call to the API includes the API key.

This will allow you access to the API.

In many APIs, you may get charged for each call.

So like your password,

you should keep your API key a secret.

An endpoint is simply the location of the service.

It's used to find the API on

the Internet just like a web address.

Now, we will transcribe an audio file

using the Watson Text to Speech API.

Before you start the lab,

you should sign up for an API key.

We will download an audio file into your directory.

First, we import SpeechToTextV1 from IBM Watson.

The service endpoint is based on

the location of the service instance.

We store the information in the variable url\_s2t.

To find out which URL to use,

view the service credentials.

You will do the same for your API key.

You create a speech-to-text adapter object.

The parameters are the endpoint and API key.

You will use this object to communicate

with the Watson Speech to Text service.

We have the path of the wav file

we would like to convert to text.

We create the file object wav

with the wav file using open.

We set the mode to rb,

which means to read the file in binary format.

The file object allows us

access to the wav file that contains the audio.

We use the method recognize from

the speech to text adapter object.

This basically sends the audio file

to Watson Speech to Text service.

The parameter audio is the file object.

The content type is the audio file format.

The service sends a response

stored in the object response.

The attribute result contains a python dictionary.

The key results value has

a list that contains a dictionary.

We are interested in the key transcript.

We can assign it to

the variable recognized\_text as follows.

Recognized\_text now contains a string

with a transcribed text.

Now let's see how to translate

the text using the Watson Language Translator.

First, we import LanguageTranslatorV3 from ibm\_watson.

We assign the service endpoint to the variable url\_lt.

You can obtain the service in the lab instructions.

You require an API key,

see the lab instructions on how to obtain the API key.

This API request requires

the date of the version, see the documentation.

We create a language translator object,

LanguageTranslator.

We can get a list of the languages that

the service can identify as follows.

The method returns the language code.

For example, English has a symbol en to Spanish,

which has the symbol es.

In the last section, we assigned

the transcribed texts to the variable to recognized\_text.

We can use the method translate.

This will translate the text.

The result is a detailed response object.

The parameter text is the text.

model\_id is the type of model we would like to use.

In this case, we set it to en-es for English to Spanish.

We use the method get result to get

the translated text and

assign it to the variable translation.

The result is a dictionary that includes

the translation word count and character count.

We can obtain the translation and assign it to

the variable spanish\_translation as follows.

Using the variable spanish\_translation,

we can translate the text back to English as follows.

The result is a dictionary.

We can obtain the string with the text as follows.

We can then translate the text to French as follows.

Thanks for watching this video.

(Music)

# REST APIs & HTTP Requests - Part 1

0:07

In this video, we will discuss the HTTP protocol.

Specifically, we will discuss:

Uniform Resource Locator: URL

Request

Response

We touched on REST APIs in the last section.

The HTTP protocol can be thought of as a general protocol of transferring information through

the web.

This includes many types of REST APIs. Recall that REST API’s function by sending a request,

and the request is communicated via HTTP message.

The HTTP message usually contains a JSON file.

When you, the client, use a web page your browser sends an  HTTP request to the

server where the page is hosted. The server tries to find the desired resource by

default "index.html". I\

if your request is successful,

the server will send the object to the client in an HTTP response; this includes information

like the type of the resource, the length of the resource, and other information.

The table under the Web server represents a list of resources stored in the web server.

In this case, an HTML file, png image, and txt file .

When the request is made for the information, the web servers sends the the requested information,

i.e., one of the files.

Uniform Resource Locator: URL

Uniform resource locator (URL) is the most popular way to find resources on the web.

We can break the URL into three parts.

the scheme this is the protocol, for this lab it will always be http://

Internet address or Base URL this will be used to find the location; some examples include www.ibm.com and www.gitlab.com

route: this is the location on the web server; for example: /images/IDSNlogo.png

Let’s review the request and Response process.

The following is an example of the request message for the get request method.

There are other HTTP methods we can use.

In the start line we have the GET method. This is an HTTP method.

In this case, it’s requesting the file index.html

The Request header passes additional information with an HTTP request.

In the GET method the Request header is empty.

Some Requests have a body; we will have an example of a request body later.

The following table represents the response.

The response start line contains the version number followed by a descriptive phrase.

In this case, HTTP/1.0 a status code (200) meaning success, and the descriptive phrase,

OK.

We have more on status codes later.

The response header contains information.

Finally, we have the response body containing the requested file, in this case an HTML document.

Lets look at other status codes.

Some status code examples are shown in the table below. The prefix indicates the class;

for example, the 100s are informational responses; 100 indicates that everything is OK so far.

The 200s are Successful responses: For example, 200 The request has succeeded.

Anything in the 400s is bad news. 401 means the request is unauthorized.

500’s stands for server errors, like 501 for not Implemented.

When an HTTP request is made, an  HTTP method is sent.

This tells the server what action to perform.

A list of several HTTP methods is shown here.

In the next video, we will use Python to apply the GET method that Retrieves data from the

server and the post method that sends data to the server.

# REST APIs & HTTP Requests - Part 2

Notes

[**Discuss**](https://www.coursera.org/learn/python-for-applied-data-science-ai/discussions/weeks/5)

0:07

In this video, we will discuss the HTTP protocol using the Requests Library a popular method

for dealing with the HTTP protocol in Python

In this video, we will review Python library requests for Working with the HTTP protocols.

We will provide an overview of

Get Requests and Post Requests

Let’s review the Request Module in Python.

This is one of several libraries including: httplib, urllib, that can work with the HTTP

protocol.

Requests is a python Library that allows you to send HTTP/1.1 requests easily. We can

import the library as follows:

You can make a GET request via the method get to www.ibm.com:

We have the response object ’r’ , this has information about the request, like

the status of the request. We can view the status code using the attribute status\_code,

which is 200 for OK.

You can view the request headers:

You can view the request body in the following line. As there is no body for a GET request,

we get a None.

You can view the HTTP response header using the attribute headers. This returns

a python dictionary of HTTP response headers. We can look at the dictionary values.

We can obtain the date the request was sent by using the key Date.

The key Content-Type indicates the type of data.

Using the response object ‘r’ , we can also check the encoding:

As the Content-Type is text/html, we can use the attribute text to display

the HTML in the body. We can review the first 100 characters. You can also download

other content, see the lab for more.

You can use the GET method to modify the results of your query. For example, retrieving

data from an API. In the lab we will use httpbin.org

A simple HTTP Request & Response Service.

We send a GET request to the server. Like before, we have the Base URL in the Route; we

append /get. This indicates we would like to preform a GET request. This is demonstrated

in the following table:

After GET is requested we have the query string. This is a part of a uniform resource

locator (URL) and this sends other information to the web server.

The start of the query is a ?, followed by a series of parameter and value pairs,

as shown in the table below.

The first parameter name is ”name” and the value is ”Joseph.”

The second parameter name is ”ID” and the Value is ”123.”

Each pair, parameter, and value is separated by an equal sign, ”=.” The series of

pairs is separated by the ampersand, ”&.”

Let’s complete an example in python.

We have the Base URL with GET appended to the end.

To create a Query string, we use the dictionary payload. The keys are the parameter names,

and the values are the value of the Query string.

Then, we pass the dictionary payload to the params parameter of the get() function.

We can print out the URL and see the name and values.

We can see the request body. As the info is sent in the URL, the body has a value of None.

We can print out the status code.

We can view the response as text:

We can look at the key 'Content-Type’ to look at the content type.

As the content 'Content-Type' is in the JSAON, we format it using the method json() . It

returns a Python dict:

The key 'args' has the name and values for the  query string.

Like a GET request a POST request is used to send data to a server, but the POST request

sends the data in a request body, not the url.

In order to send the Post Request in the URL, we change the route to POST: This endpoint

will expect data and it is a convenient way to configure an HTTP request to send data

to a server.

We have The Payload dictionary.

To make a POST request, we use the post() function. The variable payload is passed to the

parameter data :

Comparing the URL using the attribute url from the response object of the GET and POST request,

we see the POST request has no name or value pairs in it’s url.

We can compare the POST and GET request body. We see only the POST request has

a body:

We can view the key form to get the payload.

# Optional: HTML for Webscraping

0:07

In this video we will review Hypertext Markup Language or HTML

for Webscraping.

Lots of useful data is available on web pages,

such as real estate prices

and solutions to coding questions.

The website Wikipedia is a repository of the world's information.

If you have an understanding of HTML, you can use Python to extract this information.

In this video, you will:

review the HTML of a basic web page;

understand the Composition of an HTML Tag;

understand HTML Trees;

and understand HTML Tables.

Let’s say you were asked to find the name and salary of players in

a National Basketball League from the following web page.

The web page is comprised of HTML. It consists of text surrounded by a series of blue

text elements enclosed in angle brackets called tags. The tags tells the browser

how to display the content. The data we require is in this text.

The first portion contains the "DOCTYPE html” which declares this document is an HTML document.

Play video starting at :1:16 and follow transcript1:16

<html> element is the root element of an HTML page,  and

<head> element contains meta information about the HTML page.

Next, we have the body, this is what's displayed on the web page.

This is usually the data we are interested in, we see the elements with an “h3”,

this means type 3 heading, makes the text larger and bold.

These tags have the names of the players, notice the data is enclosed in the elements.

It starts with a h3 in brackets and ends in a slash h3 in brackets.

There is also a different tag “p”, this means paragraph, each p tag contains a player's salary.

Let’s take a closer look at the composition of an HTML tag.

Here is an example of an HTML Anchor tag. It will display IBM and when you click it,

it will send you to IBM.com.

We have the tag name, in this case “a”.

This tag defines a hyperlink, which is used to link from one page to another.

It’s helpful to think of each tag name as a class in Python and each individual tag as an instance.

We have the opening or start tag

and we have the end tag.

This has the tag name preceded by a slash.

These tags contain the content, in this case what’s displayed on the web page.

We have the attribute, this is composed of the

Attribute Name and

Attribute Value.

In this case it the url to the destination web page.

Real web pages are more complex, depending on your browser you can select the HTML element, then

click Inspect. The result will give you the ability to inspect the HTML. There is also

other types of content such as CSS and JavaScript that we will not go over in this course.

The actual element is shown here.

Each HTML document can actually be referred to as a document tree. Let's go over a simple example.

Tags may contain strings and other tags.

These elements are the tag’s children.

We can represent this as a family tree. Each nested tag is a level in the tree.

The tag HTML tag contains the head and body tag.

The Head and body tag are the descendants of the html tag.

In particular they are the children of the HTML tag.

HTML tag is their parent.

The head and body tag are siblings as they are on the same level.

Title tag is the child of the head tag and its parent is the head tag.

The title tag is a descendant of the HTML tag but not its child.

The heading and paragraph tags are the children of the body tag;

and as they are all children of the body tag they are siblings of each other.

The bold tag is a child of the heading tag,

the content of the tag is also part of the tree but this can get unwieldy to draw.

Play video starting at :4:13 and follow transcript4:13

Next, let’s review HTML tables.

To define an HTML table we have the table tag.

Each table row is defined with a  <tr>  tag, you can also use a table header tag for the first row.

The table row cell contains a set of <td> tags, each defines a table cell.

For the first row first cell we have;

for the first row second cell we have;

and so on.

For the second row we have;

and for the second row first cell we have;

for the second row second cell we have;

and so on. We now have some basic knowledge of HTML.

Now let's try and extract some data from a web page.

# Webscraping

0:07

In this video we will cover Webscraping.

After watching this video you will be able to: define web scraping;

understand the role of BeautifulSoup Objects; apply the find\_all method;

and webscrape a website. What would you do if you wanted to analyze hundreds of points of data

to find the best players of a sports team?

Would you start manually copying and pasting information from different websites into a

spreadsheet?

Spending hours trying to find the right data, and eventually giving up because the task

was to overwhelming?

That’s where webscraping can help. Webscraping is a process that can be used to automatically

extract information from a website, and can easily be accomplished within a matter of

minutes and not hours. To get started we just need a little Python code and the help of

two modules named Requests and Beautiful Soup.

Let’s say you were asked to find the name and salary of players in a National Basketball

League, from the following webpage.

Play video starting at :1:10 and follow transcript1:10

We import BeautifulSoup.

We can store the webpage HTML as a string in the variable HTML.

To parse a document, pass it into the BeautifulSoup constructor. We get the Beautiful

Soup object , soup, which represents the document as a nested data structure.

BeautifulSoup represents HTML as a set of Tree like objects with methods used to parse

the HTML. We will review the BeautifulSoup object

Using the BeautifulSoup object, soup, we created

The tag object corresponds to an HTML tag in the original document. For example,

the tag “title.”

Consider the tag <h3>. If there is more than one tag with the same name, the first

element with that tag is selected.

In this case with Lebron James, we see the name is Enclosed in the bold attribute "b".

To extract it, use the Tree representation.

Let’s use the Tree representation.

The variable tag-object is located here.

We can access the child of the tag or navigate down the branch as follows:

You can navigate up the tree by using the parent attribute.

The variable tag child is located here.

We can access the parent.

This is the original tag object.

We can find the sibling of “tag object.”

We simply use the next sibling attribute.

We can find the sibling of sibling one.

We simply use the next sibling attribute.

Consider the tag child object.

You can access the attribute name and value as a key value pair in a dictionary as follows.

You can return the content as a Navigable string, this is like a Python string that

supports BeautifulSoup functionality.

Let's review the method find\_all. This is a filter, you can use filters to filter based

on a tag’s name, it’s attributes, the text of a string, or on some combination of

these.

Consider the list of pizza places.

Like before, create a BeautifulSoup object. But this time, name it table.

The find\_all () method looks through a tag’s descendants and retrieves all descendants

that match your filters. Apply it to the table with the tag <tr>.

The result is a Python iterable just like a list,

each element is a tag object for <tr>.

This corresponds to each row in the list-

including the table header.

Each element is a tag object. Consider the first row.

For example, we can extract the first table cell.

We can also iterate through each table cell.

First, we iterate through the list “table rows,” via the variable row.

Each element corresponds to a row in the table.

We can apply the method find all to find all the table cells,

then we can iterate through the variable cells for each row.

For each iteration,

the variable cell corresponds to

an element in the table for that particular row.

We continue to iterate through each element and repeat the process for each row.

Let’s see how to apply BeautifulSoup to a webpage.

To scrape a webpage we also need the Requests library.

The first step is to import the modules that are needed.

Use the get method from the requests library to download the webpage. The input is the

URL. Use the text attribute to get the text and assign it to the variable page.

Then, create a BeautifulSoup object ‘soup’ from the variable page. It will allow you

to parse through the HTML page.

You can now scrape the Page. Check out the labs for more.

# Working with different file formats (csv, xml, json, xlsx)

0:07

hello

welcome to working with different file

formats

after watching this video you will be

able to define different file formats

such as csv xml and json

write simple programs to read and output

data

and list what python libraries are

needed to extract

data when collecting data you will find

there are many different file formats

that need to be collected

or read in order to complete a data

driven story or analysis

when gathering the data python can make

the process simpler with its predefined

libraries

but before we explore python let us

first check out some of the various file

formats

looking at a file name you will notice

an extension at the end of the title

these extensions let you know what type

of file it is

and what is needed to open it for

instance if you see a title like

file example.csv you will know this is a

csv

file but this is only one example of

different file

types there are many more such as json

or

xml when coming across these different

file formats and trying to access their

data

we need to utilize python libraries to

make this process

easier the first python library to

become familiar with

is called pandas by importing this

library in the beginning of the code

we are then able to easily read the

different file types

since we have now imported the panda

library let us use it to read the first

csv file

in this instance we have come across the

file example.csv file

the first step is to assign the file to

a variable

then create another variable to read the

file with the help

of the panda library we can then call

read underscore csv function to output

the data to the screen

with this example there were no headers

for the data

so it added the first line as the header

since we do not want the first line of

data as the header

let us find out how to correct this

issue

now that we have learned how to read and

output the data from a csv

file let us make it look a little more

organized

from the last example we were able to

print out the data

but because the file had no headers it

printed the first line of data as the

header

we easily solve this by adding a data

frame attribute

we use the variable df to call the file

then add the columns attribute by adding

this

one line to our program we can then

neatly organize the data output into the

specified headers for each column

the next file format we will explore is

the json file format

in this type of file the text is written

in a language independent data format

and is similar to a python dictionary

the first step in reading this type of

file is to import

json after importing json we can add a

line to open the file

call the load attribute of json to begin

and read the file and lastly we can then

print the file

the next file format type is xml

also known as extensible markup language

while the pandas library does not have

an attribute to read this type of file

let us explore how to parse this type of

file

the first step to read this type of file

is to import

xml by importing this library we can

then use the e-tree attribute to parse

the xml file

we then add the column headers and

assign them to the data frame

then create a loop to go through the

document to collect the necessary data

and append the data to a data frame

in this video you learned how to

recognize different file

types how to use python libraries to

extract data

and how to use data frames when

collecting data

# Next Steps

Congratulations on completing this course. We hope you enjoyed it.

As a next step, you can take the appropriate follow-on Python Project from the list below to apply your new found skills in a real-world scenario.

* [Python Project for Data Science](https://www.coursera.org/learn/python-project-for-data-science)
* [Python Project for Data Engineering](https://www.coursera.org/learn/python-project-for-data-engineering)
* [Python Project for AI & Application Development](https://www.coursera.org/learn/python-project-for-ai-application-development)

**Note: Successful completion of this course is a pre-requisite for these Python Project courses.**

You can explore the courses below to further hone and develop your skills for working with Data and Python:

* [Databases and SQL for Data Science with Python](https://www.coursera.org/learn/sql-data-science)

If you are looking to start a career in Data Science, Data Engineering or AI & Application Development, note that this course is part of the following Professional Certificates which are designed to empower you with the skills to become job-ready in these fields.

* [IBM Applied AI Professional Certificate](https://www.coursera.org/professional-certificates/applied-artifical-intelligence-ibm-watson-ai)
* [IBM Data Analyst Professional Certificate](https://www.coursera.org/professional-certificates/ibm-data-analyst)
* [IBM Data Science Professional Certificate](https://www.coursera.org/professional-certificates/ibm-data-science)
* IBM Data Engineering Professional Certificate
* [IBM Full Stack Cloud Developer Professional Certificate](https://www.coursera.org/professional-certificates/ibm-full-stack-cloud-developer)

We encourage you to leave your feedback and rate this course.

Good luck!