**Reading Files with Open**

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.

**Writing Files with Open**

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.

**Loading Data with Pandas**

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.

**Pandas: Working with and Saving Data**

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.

**One Dimensional Numpy**

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.

**Two Dimensional Numpy**

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)