Video #11: Reading External Text Files

You've had experience earlier in the semester with saving and loading .mat files, which are a special kind of file that only MATLAB understands. Text files are a kind of external file that are human readable, and they can store information for our program to use, or store the results of our program in a permanent way. This information can be textual or numerical, and can be created and read in MATLAB as well as in other applications. Text files have a file name extension of .txt, and in my editor here I have a short text file with some opening lines from the first Harry Potter novel. I created this by opening a new MATLAB script, typing in the content, and when saving, I selected All Files as the format for the file and I wrote the full name harryPotter.txt for the file I wanted to save.

How can we read the content of this file into the MATLAB workspace? There are three steps to the process - we open the file for reading with the fopen function, then read its contents using textscan, and finally close the file with fclose. The fopen function returns a numerical ID that refers to this particular file, and we assign that value to a variable fid that we can then provide as an input for textscan and fclose. textscan returns the content of the text file inside a cell array, and the structure of the stuff inside that cell array depends on the inputs we provide to textscan. Here we provided a special combination of characters %s that tell MATLAB that we expect to read the content of the file as strings. We assign the results to the variable words, and let's look at a picture of what words will look like in this case. words is a cell array at the top level and will have a cell array nested inside it that contains the individual words of the text, including the punctuation attached to these words, like the commas at the end of Dursley and four, and the period at the end of nonsense. The words are organized in a column. A cell array can be two-dimensional, and in this case, it has a large number of rows (the total number of words in the text) and one column. Let's see what MATLAB prints for the content. Here words{1}, which should give me that inner cell array, and then a couple references to individual words within that cell array. <run section>

That second input to textscan, which was '%s' in our harryPotter example, is referred to as a format specifier, and there are different types of information that we can specify, in addition to strings, such as different kinds of numbers. But just considering strings, it's not clear what we actually mean by a string - we saw in the harryPotter example that each word was a separate string in the cell array of content that was returned, but suppose we instead want full lines of text to be stored together in the cell array that's returned by textscan. There are multiple ways that we can control the structure of how the textscan function collects and stores the information from the text file. One thing we can specify is the delimiters that are used to separate the content into strings. By default, the spaces within lines and the line breaks at the end of each line are both used to separate the text into individual strings. This is why we got the individual words with their punctuation stored in the result for our harryPotter example. In this next example, we add the Delimiter property when we call textscan, and specify this special character combination of backslash-n (\n) which tells MATLAB that we only want to divide the content into strings using the hidden line break characters at the end of each line.

Let's see what this gives us. words will again be a cell array with a cell array nested inside at the first location, but now this inner cell array contains the entire lines of text.

So far, our text file just contained strings, but what if it's a mix of text and numbers. We'll first consider an example where the lines of the text file have a fixed format. Suppose we have a file named elements.txt (and I'll go to the actual file in the editor). Every line of the file has exactly the same format, there's an integer that's the atomic number for the element, then the name and atomic symbol (both strings) and then a floating point number that's the atomic mass. Let's suppose we want to read in the content of this text file in such a way that we store the four pieces of information separately, and the two numerical columns of information (atomic number and mass) are stored in vectors, while the textual information (name and symbol) are stored in cell arrays. <back to Live Script> We can expand the format specifier that we provide for textscan to include the pattern that we expect for each line of the text file, and that pattern can contain different types of values. Each type is indicated by a combination of the % symbol followed by a letter, and we're saying here that the content of the file is a repeated pattern of integer, string, string, floating point number. As textscan reads the content, it will apply this pattern over and over again, expecting to see integer, string, string, float, etc. And by default, it will assume that these entities are separated by either spaces or line breaks, because we haven't provided any additional input about the delimiters to use. When it reads the content of the file, it will collect the integers together in a column vector, the strings will each be stored in a cell array arranged as a column, and the masses will be stored in a column vector, and these four entities will be nested inside one large cell array. Let's view the result that we get for our elements text file. We call textscan with this extended format string, and we're going to view the high-level structure of elements and then the content of the four things contained inside at indices 1-4. Let's run the section - we can see that elements is a cell array of four things - MATLAB shows us a table of the four things - and if we look at the four individual cells, we see column vector of atomic numbers, cell arrays of names/symbols, and vector of masses.

So we've illustrated how we can read the content of a text file that has a fixed format for the information that's stored in the file. Sometimes a file might look at first glance, like it has a fixed format, but not really be a uniform format. Suppose we have this file that provides an inventory of the price and quantity of different toys in a toy store, and we want to know the total value of the current stock of toys. This looks like a three-column format with a name string, numerical price and quantity. But if we look closer, we have a problem, the names have different numbers of words, so if we're using spaces to delineate the different pieces of information, the names vary between one, two, or three strings (for mr. potato head). And prices are numerical, but they have this dollar sign at the beginning that we need to deal with. So in this case, we'll resort to reading everything as strings, like in our harryPotter example, and then process the strings afterwards to find the prices and quantities that we need to compute the value of the inventory. We'll look at the full computeToyValue script in the editor, but here let's first take a peak at the first step, which is to read the file with fopen, textscan,

and fclose. We'll assume strings, separated by the default delimiters of spaces and line breaks, but the first line of the text file was a set of headers that we don't need to preserve, and MATLAB provides a way to skip over lines at the top of a file, with the HeaderLines property that's followed by an integer that indicates how many lines to skip (here just one). Similar to our original words cell array for the harryPotter example, tokens will be a cell array with a long cell array nested inside it, at index 1, and if we run the section, we can see that the inner cell array has all the individual strings and numbers in the inventory, all stored as strings.

So now let's look at the script to see how we can process this content to get the overall value of the toys. After reading the text file, the tokens variable is reassigned to the inner cell array with all the individual strings. We initialize the total value to 0 and then loop through our cell array of strings using an index, and assign a variable named token (singular) to the next token in the cell array. token is a string, and if the first character of the string is a dollar sign, we'll extract the rest of the string (at indices 2 to end), which should be the price, convert it to a number with the str2double function, and store this in the price variable. We know that every price is followed immediately by a quantity, so we'll get tokens{index+1}, convert that to a number as well, and store that in the quantity variable. Finally we'll update our total value by adding in the product of this new price*quantity. We'll display the totalValue at the end. So here we used what we learned about string processing to analyze the content of the strings in more detail, to get the value of our toy inventory. <run and get result $743.90>

There's one final note I'd like to add here - you learned about the HeaderLines property that allows us to skip over lines at the top of a text file. But suppose we instead want to preserve the information about the headers. The textscan function has an optional third input that's an integer N, and this input specifies the number of times to apply the format string, so we don't need to read the entire file all at once. Here we just read the first three strings (headers in the first line), store them in the variable headers, and then go on to read the rest of the text file. Let's run and see what the two parts look like, headers{1} has the three header labels and tokens{1} has the rest of the file content.

That's it for reading text files - the next video focuses on the opposite task, which is writing information, like the results of a program, to a new text file.