Video #14: Selecting and Sorting Data

This video explores the topics of selecting and sorting data - we mostly draw on things you've learned already, about strings, cell arrays, and reading text files, but you'll also learn about the built-in sort function in MATLAB, which comes in handy in many contexts. We'll work with data about birds that's stored in a text file - this picture shows a snippet of the data, which includes information like the name and family of different bird species, their habitat, average size and wingspan, coloration of different parts of their body, and so on. The textual data is carefully written in a way that provides a single string for each property even when there's multiple words like in the name and habitat. Suppose we'd like to select out information about all the birds in the heron family, or birds with large wingspans, or maybe we'd like to sort the birds by the numerical properties of size or wingspan, or sort them alphabetically by name.

To work with the data, we first need to load it into the MATLAB workspace, and we know how to do this - we open the text file, read in the content with textscan, and close the file. The structure of each line of the file is three strings, followed by two integers, then 5 more strings, and there's a header line of property names that we can skip over. The bird information gets stored in the variable birds here. Let's visualize the structure of birds - it's one large cell array with 10 things nested inside corresponding to the 10 items in the format string we provided for textscan. The strings in each row of the file will be stored in cell arrays that are highlighted in blue, and the integers will be stored in vectors highlighted in pink. Each nested cell array or vector will be organized as a column, with a row for each row of data in the text file. The names of the properties here are just shown for reference, and I abbreviated the color names. Given this structure of the birds variable, suppose we'd like to print out the content of birds in a nicely formatted way. Here's a call to printBirdInfo, and let's take a quick look at the function definition. There's a for loop with an index i from 1 to the length of one of the nested cell arrays, and we use fprintf to print each line using a format string with numbers to get nicely aligned columns, and \n to insert a carriage return at the end of each line, and we insert the data for each element of the format string, being careful to use curly braces around the index i for the strings stored in cell arrays, and parentheses around i for the two integers stored in vectors. Let's run this section and see the printout (there's 59 birds in this database).

Now let's return to the tasks of selecting a subset of the data, for example selecting the information about herons, or birds with large wingspans. The basic idea here is that we want to select a subset of the rows of data, for example, all the herons are in the first 5 rows, and we're going to define a function that returns a new cell array with the same structure of the original birds cell array, with 10 things nested inside that are the cell arrays and vectors of different properties. The only difference is that there will only be 5 rows in each of the nested cell arrays and vectors with the information about the herons. So how do we accomplish this?

To understand what we're going to do, we'll start with a much simpler example. In this code block, I created a cell array of two things, a cell array of word strings for numbers and a vector of numbers. Think of these as being analogous to the cell array of family names and vector of wingspans in our bird data. I've written them in a row instead of a column, but the idea is the same. Suppose we want to create a new cell array of two things that just include the middle three elements of the cell array and vector. We can use the cell function to create an initial cell array to store two things, and create a vector of the indices that we're interested in selecting from the content of numbers. At the two locations of the newNums cell array, at indices {1} and {2}, we'll copy content from the numbers cell array at those same indices {1} and {2}, but we'll only copy content from our subset of indices. The first thing in birds is a cell array and the second thing is a vector, but we use parentheses around the vector of indices in both cases. We want to create a container holding the content at the three indices, either a cell array containing the three middle strings or a vector containing the three middle numbers. In the case of a cell array, we use parentheses around indices when we want to create a cell array storing the content of these indices - we use curly braces when we want to refer to the individual elements inside the cell array, like an individual string. Let's run this section and see what's printed (cell gives initial two-element cell array with empty vectors at each location, after we place a subset of the original content of numbers into newNums, the first element is a cell array {'two' 'three' 'four'} and second element is vector [5 0 2]).

Now let's apply these ideas to select information about birds in the heron family - we define a function getHerons that takes one input that's the cell array with info about all 59 birds, and returns a cell array that we call newBirds, that just has herons. The first step is to determine the indices of rows of birds in the heron family - this uses familiar code. The second item in birds is the cell array of family names, and we'll compare that whole cell array of strings to the string 'heron', and use find to get the indices where this is true. For our bird data, this will end up being a vector of indices from 1 to 5. The rest of the statements create the newBirds cell array - it will have 10 things inside, so we can use the cell function to create an initial 1 x 10 cell array of empty content at each location. The for loop creates the nested cell arrays and vectors inside newBirds, like we did in our numbers example. We loop through an index i from 1 to 10, and at each of the 10 locations of newBirds we copy content from the corresponding 10 things in the larger birds cell array, and we just pull out content from the rows stored in the indices variable. Just like the numbers example, we write indices in parentheses, regardless of whether we're taking this content from a cell array or a vector.

There's an advantage to how we defined this function - suppose we instead want to select out the birds with large wingspans, let's say, with an average wingspan of at least 4 feet (48 inches). Here's a new function to do that task, it has a different name and comment at the top, but otherwise it's almost exactly the same as getHerons - the only change is highlighted in blue - we'll instead get the indices of birds where the wingspan vector, that's stored in the 5th location of the birds cell array, is greater than 48. The rest of the code is the same. Let's test the two new functions, and also note that an advantage of creating a new cell array that has the same structure as the original birds cell array, is that we can call our function printBirdInfo on the new cell array to view its content. <run section and view results>

The other tasks we mentioned at the beginning involved sorting our bird data, for example using a numerical property like size or wingspan, or a textual property, like ordering the birds alphabetically. To do this, we need to learn about the MATLAB sort function. We'll first sort a vector of numbers - by default, numbers will be sorted in increasing or ascending order, but there's an optional second input that we can specify if we want to sort the numbers in descending order instead. There's also a second optional output that we'll look at in this third example. Let's run the section. Here are our numbers in increasing and decreasing order, and there's no problem with duplicate numbers like the 3 and 7 here. Let's see what the optional output is - I called it sortIndices and it tells us where each of the sorted numbers comes from in the original vector of unsorted numbers. So the 1 here came from index 8 of the original nums vector, the 2 came from index 2, the two 3s came from the indices 6 and 9, and so on. Before I reveal the answer, what do you think would be returned by the expression nums(sortIndices)? This says collect together the content of nums, starting with number at index 8, then the number at index 2, then indices 6 and 9… this gives us the sorted numbers.

We can also sort strings - let's start with a cell array of words (odd collection) and sort them in the same way that we did for numbers. <run section> The strings are sorted alphabetically (a black cloud descended early from great heights), and the sortIndices tell us where the words came from - index 4, then 5, then 2, and so on. Let's consider a slight variation of words, and sort. First, what's different about the new collection of words? Half are capitalized, and notice that the capitalized words all appear before the lowercase words in the sorted ordering. Why is that? When ordering strings, MATLAB uses the ASCII code - that numerical code for characters that we spoke about long ago in the strings lecture. In the ASCII code, all the capital letters appear before all the lowercase letters. We can get around this by converting all the strings to lowercase. <run section> The capitalization isn't preserved here - you might think about how you could maintain the capitalization (it will take two MATLAB statements).

Finally let's return to sorting the bird data - the nice thing here is that the functions we write for sorting will be almost identical to the earlier functions we wrote for selecting the data. We'll start with a function sortByWingspan that takes all of the bird data as input and returns a rearrangement of the data as output, and what's different here is how we construct the indices. And I'd also like to introduce a new thing that you might see in other MATLAB code. We're sorting the 5th element of birds, which is the vector of wingspans, and we don't care about what the sorted wingspan values look like on their own, so this tilde (~) tells MATLAB, we don't care about the first value that's returned, we're just interested in the sequence of indices - in this case, it's a sequence of indices that would rearrange the rows of data in increasing order of the wingspan size so when we create newBirds, it creates them with rows in the order specified in the indices, with all the nested elements sorted in the same way. The function to sort alphabetically by name is exactly the same, but in this case, we sort a lowercase version of the cell array of names in the first location of birds. <run tests> - see ordered by wingspan and alphabetically.

So that's a taste of how you can select and sort data, where the data might consist of a collection of properties for a large number of entities like birds. We mostly drew on things you know about strings, cell arrays, and reading files, and also introduced the built-in sort function in MATLAB.