PSYCH 20A (PROF. FRANE) – LECTURE 5: Characters, Strings, and Text

Classes of objects

So far we have been using two *classes* of values. Mainly we've been using *numeric* values, or more specifically *double-precision* values (which basically are numeric values that can be positive, negative, or zero, and have digits before and after the decimal point). We've also used *logical* values (which look like zeros and ones, but are actually trues and falses, and take up less memory than numeric values). In this lecture, we'll learn about two more classes: *characters* and *strings*. A given array can only contain elements of a single class.

Character arrays

We can make arrays out of characters rather than out of numeric or logical values. Single-quotes indicate that the contents within the quotes are characters. For example, `'Hi there!'` is a 1 × 9 row vector containing the following characters: H, i, space, t, h, e, r, e, exclamation point.

We can type `['abc' ; 'def']` or `['a':'c' ; 'd':'f']` or `['a' 'b' 'c' ; 'd' 'e' 'f']` to make the following 2 × 3 matrix:

```
abc
def
```

We can do the same indexing, concatenation, replication, transposing, reshaping, etc. on character arrays that we do with numeric arrays. For example:

`myFaveFruit = 'apple'`	Defines `myFavoriteFruit` as a 5-character vector
`myFaveFruit(1:3)`	Outputs `app`
`['pine' 'apple']`	Outputs `pineapple`
`cat(2, 'pine', 'apple')`	Outputs `pineapple` (same as above)
`'apple' == 'p'`	Outputs the logical vector `0 1 1 0 0`
`repmat('apple', 1, 3)`	Outputs `appleappleapple`

Strings

Sometimes we want to treat a series of characters as a single value (called a *string*), rather than as a vector of individual values. We can do that by using double-quotes instead of single-quotes. For example, `'apple'` is a 1 × 5 vector of characters, whereas `"apple"` is a 1 × 1 string. And `['pine' 'apple']` is the 1 × 9 character vector `'pineapple'` (with no special separation between "pine" and "apple"), whereas `["pine" "apple"]` is a 1 × 2 string vector that considers "pine" and "apple" as 2 discrete elements.

Note that using strings allows us to concatenate words vertically in an array even if they have different numbers of letters. For example, the following code defines `fruits` as a 3 × 1 string array:

```
fruits = ["apple" ; "pear" ; "blood orange"]
```

We can't count the number of characters in a string by using the `numel` or `length` function like we can with a character array (e.g., remember that `"apple"` has dimensions 1 × 1). We use the `strlength` function instead. For example, `strlength("apple")` outputs 5, and `strlength(fruits)` outputs the following:

```
  5
  4
 12
```

Some functions that can be applied to both character arrays and strings

For the following examples, assume `myFaveFruit` is defined as the character array `'apple'` or as the string `"apple"` (the examples work for both).

Finding specific characters in an array or string

`strfind(myFaveFruit, 'e')`	Outputs `5` because there's an 'e' at the 5th index in the vector
`strfind(myFaveFruit, 'p')`	Outputs `[2 3]` because there's a 'p' at the 2nd and 3rd indexes
`strfind(myFaveFruit, 'z')`	Outputs `[]` (meaning empty) because there is no 'z' in the vector

Unlike the `find` function, `strfind` can locate a *series* of values rather than only a single value:

`strfind(myFaveFruit, 'le')`	Outputs 4 because there is an "le" starting at the 4th index

That works even for numeric vectors:

`strfind([24 12 36], [12 36])`	Outputs 2 because there is a [12 36] starting at the 2nd index

For more complex types of searches (such as finding words that start with sh and end in a vowel), you can use the `regexp` function (not covered here).

Asking whether two character arrays or strings are the same

`strcmp(myFaveFruit, 'apple')`	Outputs logical 1 (meaning TRUE)
`strcmp(myFaveFruit, 'pear')`	Outputs logical 0 (meaning FALSE)
`strcmp(myFaveFruit, 'AppLe')`	Outputs logical 0 because `strcmp` is case-sensitive
`strcmp('apple', "apple")`	Outputs logical 1, even though one input is a character array and the other is a string
`strcmpi(myFaveFruit, 'AppLe')`	`strcmpi` is like `strcmp`, but not case-sensitive, so this outputs 1 (the i in `strcmpi` stands for "ignore case")

Changing the case of letters

`lower('99 ApPLeS')`	Converts all letters to lower-case; this outputs `'99 apples'`
`upper('99 ApPLeS')`	Converts all letters to upper-case; this outputs `'99 APPLES'`

Replacing or removing characters

`strrep('organize and stylize', 'z', 's')`	Outputs `'organise and stylise'`
`strrep('irregardless', 'ir', '')`	Outputs `'regardless'`

Removing all punctuation marks using the erasePunctuation function (requires Text Analytics toolbox)

`erasePunctuation('Hi, how are you???')`	Outputs `'Hi how are you'`

Converting between classes of values

`char("apple")`	Inputs `"apple"` as a 1 × 1 string, outputs `'apple'` as a 1 × 5 character vector
`string('apple')`	Inputs `'apple'` as a 1 × 5 character vector, outputs `"apple"` as a 1 × 1 string
`string([4 16])`	Inputs 1 × 2 numeric vector [4 16], outputs 1 × 2 string vector ["4" "16"]
`num2str([4 16])`	Inputs 1 × 2 numeric vector [4 16], outputs 1 × 3 character vector `'416'`
`double(y > 2)`	Inputs logical vector y > 2, outputs numeric zeros and ones in place of the logical zeros and ones
`logical([0 1 3])`	Inputs 1 × 3 numeric vector [0 1 3], outputs logical vector [0 1 1] meaning false true true (any inputted nonzero real number is converted to true)

`str2double(x)`	Inputs x as a character array or string, outputs the numeric value of x Ignores commas (e.g., if x is `"1,000"`, it outputs 1000) Can't do mathematical operations (e.g., if x is `"500 + 500"`, it outputs `NaN`)
`str2num(x)`	Inputs x as a character array or string, outputs the numeric value of x. Evaluates x as an "expression." Thus, it will interpret commas and semicolons as separating elements in an array (e.g., if x is `"1,0"` it outputs `[1 0]`), and it can perform mathematical operations (e.g., if x is `"500 + 500"` it outputs 1000). `str2num` is slightly slower than `str2double`.

Splitting a string into multiple strings

Let's say we have a complete sentence that's defined as a string:

```
mySentence = "I like apples very much!" ;
```

We can split that sentence into a string vector, in which each string is a word:

`strsplit(mySentence)` Outputs `["I" "like" "apples" "very" "much!"]`

By default, the `strsplit` function uses a blank space as the "delimiter" (the indicator of where the breaks are). But you can specify a different delimiter by putting it as the second input to the `strsplit` function. For example, to use a comma:

```
strsplit(mySentence, ",")
```

You can specify multiple delimiters by inputting a string array of delimiters. For example, to interpret spaces *and* slashes as delimiters:

```
strsplit(mySentence, [" ", "/"])
```

Note that when you use the `strsplit` function, the delimiter itself doesn't appear in the output.

In a column-vector of strings that all have the same number of words, the `split` function can be applied to each string, producing a matrix. For example, let's say we have a 3×1 column vector called `names` that looks like this:

```
"Ashley Barnes"
"Stacy Moore"
"Oren Hu"
```

Typing `firstAndLastNames = split(names)` would then define `firstAndLastNames` as the following 3×2 matrix of strings:

```
"Ashley"   "Barnes"
"Stacy"    "Moore"
"Oren"     "Hu"
```

But when applied to a single string, the `split` function outputs a column vector, so the `strsplit` function is typically used instead in that context.

Joining string vectors into single strings

The `strjoin` and `join` functions are basically the opposites of the `strsplit` and `split` functions, respectively. For example, `strjoin` combines a string row-vector into a single string:

```
strjoin(["I" "like" "apples!"])
```
Outputs `"I like apples!"`

`join` can be used to combine each row in a matrix of strings. For example, `join(firstAndLastNames)` reverses the effect of `split` and thus outputs the original names vector:

```
"Ashley Barnes"
"Stacy Moore"
"Oren Hu"
```

Notice that the default delimiter for `strjoin` and `join` is a space. But other delimiters can be used. For example, we can enter an empty delimiter by typing `join(firstAndLastNames, "")` to output the following column vector:

```
"AshleyBarnes"
"StacyMoore"
"OrenHu"
```

Joining separate strings into a single string

We've talked about using strjoin to join a string vector into a single string. But what if we have strings that aren't arranged in a vector (they're just separate objects), and we want to join them into a single string? One method is to use the `strcat` ("string concatenate") function:

```
strcat("Ashley","Barnes")
```
Outputs string `"AshleyBarnes"`

But the simplest way to join strings is using the + symbol:

```
"Ashley" + "Barnes"
```
Outputs string `"AshleyBarnes"`

A handy feature of using the + symbol that way is that Matlab will automatically convert any non-string elements. For example, if we set `age` equal to 24, then:

```
"I am" + age + "years old."
```
Outputs string `"I am 24 years old."`

Special characters

To insert Greek letters or other symbols in a character array or string, you can copy and paste them in from an outside source.

Or you can generate the desired symbol by inputting the symbol's *numeric html character code* (which can be found with a simple Google search) to the `char` function. For example, `char(176)` gives the ° (degree) symbol. That symbol can be concatenated into a character array:

```
['It is 80 ' char(176) 'F today']
```
It is 80 °F today

Or into a string:

```
"It is 80 " + char(176) + "F today"
```
It is 80 °F today

Quotes

When defining a character array that contains single quotes or a string that contains double quotes, the quotes cannot simply be typed in like any other character, because Matlab will get confused about where the array or string starts and ends. For instance, `'I'm happy'` is not a valid character array, and `"Say "hi" to me"` is not a valid string (but `'Say "hi" to me'` is a valid character array, and `"I'm happy"` is a valid string). One solution is to generate the quotes using the `char` function. But a simpler approach is to enter the quote twice. For example:

```
'I''m happy'                Outputs   'I'm happy'
"Say ""hi"" to me"          Outputs   "Say "hi" to me"
```

Be careful when pasting code containing quotes into Matlab from sources outside of Matlab (e.g., from websites or Word documents). Often those quotes are formatted as "smart quotes" (angled to be "open" or "closed"), which Matlab won't interpret as the standard quotes that are used to enclose a character array or string.

Getting keyboard input

One way to get data is to prompt the user to enter something in the command window. We can do this using the `input` function. For example, the following code prompts the user to enter something, and then defines `age` as whatever they type (the semicolon keeps Matlab from repeating back what the user just entered).

```
age = input('Please enter your age: ') ;
```

By default Matlab evaluates the entry as an expression, so if you enter 20+4 age will be 24, and if you enter [20 30] age will be a vector. To instead get the entry as a character vector (without the user having to type the single-quotes), put `'s'` in the parentheses:

```
name = input('Please enter your name: ', 's') ;
```

The `inputdlg` function is similar to the `input` function but uses a pop-up dialog box.

Printing text to the command window

There are various ways to print text to the command window. The most versatile is to use the `fprintf` function, which can take either a character array or a string. For example:

```
fprintf('Hi. Here is some text. Amazing!')
```

We can create line-breaks using `\n` (meaning "new line"). We don't put extra spaces before or after the `\n`. For example, typing `fprintf('Hi.\nHere is some text.\n\nAmazing!\n')` outputs this:

```
Hi.
Here is some text.

Amazing!
```

Now let's define a character array and a number:

```
name = 'Alejandra' ;
age  = 24 ;
```

Now, we can use square brackets `[]` to concatenate those into a character vector that we input to `fprintf`:

```
fprintf(['My name is ' name '.\nI am ' num2str(age) ' years old.\n'])
```

Or we can join them into a string, in which case the number-to-string conversion is done automatically when we use the + symbol:

```
fprintf("My name is " + name + ".\nI am " + age + " years old.\n")
```

Either way, the output looks like this:

```
My name is Alejandra.
I am 24 years old.
```

Another way to achieve the same result is to use the % symbol to create placeholders in the text for the values you want to insert, and enter those values as the subsequent inputs to the function. `%d` is a placeholder for a number (`%d` stands for "decimal format") and `%s` is a placeholder for a string or a character array. Thus, we could show the same output using the following command:

```
fprintf('My name is %s.\nI am %d years old.\n', name, age)
```

When using `%d` Matlab will automatically choose the formatting of the number. But you can use other codes to specify how the number is displayed. For example, `%.3f` displays the number to 3 decimal places (f stands for "fixed point"), `%e` displays the number in scientific notation, and `%.3e` displays the number to 3 decimal places in scientific notation (e.g, 4.123ee+00 means 4.123×10^0). See the documentation for other examples.

There are many other ways to customize the output of `fprintf`. For example, you can make hyperlinks (text that opens a webpage when you click on it). See the documentation for more info if you're interested.

The `disp` function displays values or text in the command window, e.g., `disp(age)` or `disp(name)` etc., but isn't as versatile as `fprintf`. For instance, `disp` doesn't allow you to embed formatting or line-breaks.

The % and \ symbols
Because the % and \ symbols have special uses in `fprintf` they cannot simply be typed into the text like any other character. We could use the `char` function as described earlier. But a simpler solution is to enter the character twice. For example, typing `fprintf('I have 95%% confidence')` prints the following:

```
I have 95% confidence
```

And typing `fprintf('We use \\n to indicate a line-break\n')` prints the following:

```
We use \n to indicate a line-break
```

Printing text to a text file

First we use the `fopen` ("file open") function to create a text file called myFilename.txt and have Matlab assign a temporary "file ID" number that will be used to reference it (the 'w' stands for "write," meaning we want to write in the file):

```
fileID = fopen('myFilename.txt', 'w') ;
```

Now we use `fprintf` as before to print whatever text we want, but we insert the file ID as the first input before the text:

```
fprintf(fileID, 'Hi.\nHere is some text.\n\nAmazing!\n') ;
```

The semicolon at the end of that command suppresses the output, which is the size of the file in bytes. Now we close the file using the `fclose` ("file close") function because we're done writing in the file:

```
fclose(fileID) ;
```

Or to close multiple files that we've opened:

```
fclose('all') ;
```

The semicolon at the end of that command suppresses the output, which is a zero confirming that the file was closed or a -1 if the file failed to close.

To verify that the text file came out correctly, you can open the text file (e.g., using a notepad or word processing application). Or you can use the `type` function followed by the filename (or full path) to output the file's contents to the Matlab command window, e.g.:

```
type 'myFilename.txt'
```