String Variables

Leigh J. Little SUNY Brockport November 14, 2018

Contents

1 Introduction

Not all variables you encounter in a computer program are numbers. Some variables are words (for example, names, addresses) or non-number numbers (phone numbers, social security numbers). In these cases, *string* variables are the most appropriate way to represent the data. In a string variable, the entities that make the string are treated as separate characters.

Unfortunately strings can be unpleasant to work with (especially if you are trying to work with multiple languages).

2 String Basics

We have seen string variables in MATLAB, but not in an organized way. To create a string variable in MATLAB, enclose a set of characters in single quotes (the quote key is usually next to the enter key on your keyboard).

```
>> s = '987654321'
s =
    '987654321'
```

MATLAB treats this string as a vector of length 9 and we can reference/change/display characters using the usual subscript notation, for example

```
>> s(4)        % Get 4th character of s
ans =
    '6'
>> s(2:7)      % Get characters 2 though 7
ans =
    '876543'
>> size(s)     % Get size of s
ans =
     1     9
>> length(s)   % Get length of s
ans =
     9
>> s(7:-1:3)   % Get characters 7 though 3 in reverse order
ans =
    '34567'
>> s(1:2:9)    % Get characters 1,3,5,7,9
```

```
ans =
    '97531'
>> s(6) = 'A'  % Set 6th character to A.
s =
    '98765A321'
```

Any sequence of consecutive characters from a string is called a *substring*.

2.1 A String With a Quote

What if your string has a quote in it? How can you enter a string that contains a contraction? In this case, you would put 2 quotes in sequence (no spaces between them).

```
>> s4 = 'You can''t do that'
s4 =
    'You can't do that'
```

2.2 Concatenation of Strings

Concatenation refers to building a larger string by placing smaller strings together. In MATLAB, you can do this using the [ ] constructor operators we have seen before

```
>> s1 = '12345';
>> s2 = 'ABCDE';
>> s3 = [s1 s2]
s3 =
    '12345ABCDE'
```

2.3 Converting a Number to a String

It is helpful to be able to convert a numerical value to a string. For example, this is useful when creating more descriptive plot titles. To convert a number to a string, use the `num2str` command. Consider the example

```
>> p = num2str(pi)
p =
    '3.1416'
>> ts = ['The value of pi is ' p]
ts =
    'The value of pi is 3.1416'
```

Note the string `ts` is built using concatenation of a literal string (the part in quotes) and a variable string `p`. Spaces are included as part of the string. A space is given at the end of the literal string to avoid having the number appear immediately after the last non-space character. You can alter the precision and format of the string version of the number. See `help num2str` for more information.

2.4 Converting a String to a Number

It is also useful to be able to convert a string to a number. This is done using the `str2num` function.

```
>> s = '1.232';
>> x = str2num(s)
x =
    1.2320
>> x + 1
ans =
    2.2320
>> s = 'cow'
s =
    'cow'
```

```
>> x = str2num(s)
x =
     []
```

Notice that if your input string is not actually a number, the output is an array of length 0.

2.5 The `strfind` Command

The `strfind` command is used to determine if some character or sequence of characters is contained within a string. For example,

```
>> s = 'ABCBDBEBFBGB';
s =
    'ABCBDBEBFBGB'
>> i = strfind(s,'B')
i =
     2     4     6     8    10    12
>> s(i)
ans =
    'BBBBBB'
```

Here, we are testing if the string `'B'` is contained in the string `s`. In this case, `'B'` appears 6 times. The output `i` from the `find` command is an array of indicies where the requested string is located. In this case, `'B'` appears in positions 2, 4, 6, 8, 10 and 12 of `s`.

For another example, consider

```
>> s = '123412671289';
>> i = strfind(s,'12')
i  =
     1     5     9
```

Here, we are testing if the string `'12'` appears in the string `s`. It occurs three times. The output indicates the starting position of each occurrence of `'12'`. The first occurrence is at `s(1)` and consists of characters `s(1:2)`. Similarly, the second and third occurrences are located at `s(5:6)` and `s(9:10)`.

2.6 Conditional Testing

We can use all of the usual conditional tests on strings or parts of strings, but the answer is not what you would first expect.

```
>> s = 'ABCBDBEBFBGB';
>> s(1:2) == 'AB'
ans =
    1x2 logical array
    1   1
```

Here we are testing if the first 2 characters of `s` are equal to `AB`. This is true however, instead of returning a 1 (for true) it returns a 1 for each individual character. When we use the conditional operators, the comparisons are performed character wise (meaning character by character). Here the result is a vector of two ones because each comparison `s(1) == 'A'` and `s(2) == 'B'` is true.

```
>> s(1:2) == 'AC'
ans =
   1x2 logical array
   1   0              % Second element is 0 because 'C' is not 'B'
```

2.7 The `strcmp` Command

In the above example, what we probably wanted to use is the `strcmp` command instead.

```
>> s = 'ABCBDBEBFBGB';
>> i = strcmp(s(1:2),'AB')
i =
  logical
   1
```

This command returns a value of 1 (or true) if the first and second strings are the same and a 0 (or false) if they are different.

The `strcmp` command is case sensitive. If you want to use a case insensitive comparison, you can use the `strcmpi` command.

2.8 The `eval` Command

One of the most powerful commands in MATLAB is the `eval` command. We will not have much, if any, opportunity to use it in this course though. The `eval` command will evaluate a string as if it was a MATLAB command. For example,

```
>> clear
>> s = 'x = sqrt(2)';
>> eval(s)
x =
    1.4142
```

2.9 Other String Commands

Here is a list of other commonly used string functions. You can use the help function for more information on these.

- `strcat` - another way to concatenate strings.
- `isletter` - returns an array indicating which input characters are letters.
- `isspace` - returns an array indicating which input characters are a spaces.
- `strrep` - find and replace a substring.
- `strsplit` - split a string at a specified location.
- `strncmp` - compares the first n characters of strings.
- `deblank` - removes trailing blanks from a string.
- `strtrim` - removes leading and trailing blanks from a string.
- `lower` - converts a string to all lower case.
- `upper` - converts a string to all upper case.