Module 11: Text Manipulation

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Module 11: Learning Outcomes

- Distinguish the difference between a character vector and a string scalar
- Distinguish the difference between a string scalar and a string array.
- Explain how to read and extract individual characters in a string
- Apply built-in functions specially designed for operations using string or character vector
- Explain how to use operators (==, +) when string variables are involved.
- Convert strings or character vectors to numbers and vice versa.

Text Terminology

- Text in MATLAB can be represented using:
  - character vectors (in single quotes)
  - string arrays, introduced in R2016b (in double quotes)
- Prior to R2016b, the word “string” was used for what are now called character vectors.
- Many functions that manipulate text can be called using **either a character vector or a string.**
- Additionally, several new functions have been created for the string type

Review: Character vs Character Vector

- **Characters** include letters of the alphabet, digits, punctuation marks, white space, and control characters. Individual characters in **single quotation** marks are the type char (so, the class is char)
- Character vectors consist of any number of characters and contained in single quotation marks. Their type are **char.**
- Since these are **vectors** of characters, **built-in functions and operators that we've seen already work with character vectors as well as numbers**. You can also index into a character vector to get individual characters or to get subsets

```
>> letter = 'x';
>> size(letter)

ans =

     1     1

>> class(letter)

ans =

    'char'
```

```
>> myword = 'Hello';
>> size(myword)

ans =

     1     5

>> class(myword)

ans =

    'char'
```

String Scalar

- A string scalar is a data type of '**string**', a single string, and is used to store a group of characters.
- The group of characters defined as a string is a single scalar variable, not a vector.
- String can be created
  - using double quotes, e.g., "`Chul Min`"
  - using the string type cast function, e.g., `string('Chul Min')`
- Strings are displayed using double quotes.

```
>> myword = 'Hello';
>> size(myword)

ans =

     1     5

>> class(myword)

ans =

    'char'
```

```
>> myword = "Hello";
>> size(myword)

ans =

     1     1

>> class(myword)

ans =

    'string'
```

⚠: A character vector is an array of characters. A string is a single data type (dimension) that can contain a string of characters. It is not an array of characters!

String Indexing

Challenging

- You can index into a string array using curly braces {}, to access characters directly.
- Indexing with curly braces provides compatibility for code that could work with either string arrays or cell arrays of character vectors.

```
mystr = "CM121";

mycharv = 'CM121';

mystrarry = ["ENVE121",...
"GEOE121", "AE121"];
```

📖: `strlength` is to count the number of characters in a string value.

```
size(mystr)              [1  1]
mystr(1)                 "CM121"
mystr{1}                 'CM121'
mystr{1}(1)              'C'
mystr{1}(4)              '2'

size(mycharv)            [1  5]
mycharv(1)               'C'
mycharv(4)               '2'

size(mystrarry)          [1  3]
mystrArry(2)             "GEOE121"
mystrArry{1}             'ENVE121'
mystrArry{2}(1)          'G'
mystrArry{2}(5)          '1'

numel(mystr)             1
numel(char(mystr))       5
strlength(mystr)         5
```

Group of Strings

Challenging

- Groups of strings can be stored in (1) **string arrays**, (2) **character matrices**, (3) **cell arrays**.
- Strings are created using double quotes or by passing a character vector to the string function
- The plus function or operator can **join**, or **concatenate**, two strings together (e.g., `"abc" + "xyz" => "abcxyz“`)

`[char1 char2]`	`‘ENVE121’`
`char1 + char2`	`error`
`char1 + char1`	`[138 156 172 138]`
`[str1 str2]`	`"AE"  "Hello“`
`str1 + str2` `strcat(str1, str2)`	`“AEHello“` `“AEHello“`
`str1 + string(char1)`	`“AEENVE“`
`[char(str1) char(str2)]`	`‘AEHello’`

```
char1 = ‘ENVE’;
char2 = ‘121’;

str1 = "AE";
str2 = "Hello";
```

: You can initialize a string array using `strings`.

```
str = strings(sz1,...,szN)

str_arry = strings(1, 2);
str_arry(1,1) = "AE"
str_arry(1,2) = "Hello"
```

Example: Print Texts

Q. Print out the following texts in the command window when variables containing the first and last names are given.

```
Welcome ! Park, Ju An
Welcome ! Yeum, Chul Min
Welcome ! Gao, Noreen
Welcome ! Fierastrau, Vlad
Welcome ! Connelly, Jason
```

```
num_name = 5;
fname = ["Ju An", "Chul Min", "Noreen", "Vlad", "Jason"];
lname = ["Park", "Yeum", "Gao", "Fierastrau", "Connelly"];
```

```
for ii=1:num_name
    fprintf('Welcome ! %s, %s \n', lname(ii), fname(ii));
end
```

option1

```
for ii=1:num_name
    str = lname(ii) + ", " + fname(ii);
    fprintf('Welcome ! %s \n', str);
end
```

option2

⚠: '…%s,%s \n'
Here, the comma is not to separate arguments. It is in the text.

📖: %s is to print both strings and characters.

Built-in Functions for Text

Many functions can have either character vectors or strings as input arguments. Generally, **if a character vector is passed to the function, a character vector will be returned, and if a string is passed, a string will be returned**

😊: Again, you do not have to memorize the functions and their usage. You can simply search for their usage in google or type `doc fun_name` in command window.

Function	Description
`upper`	Convert strings to uppercase
`lower`	Convert strings to lowercase
`reverse`	Reverse order of characters in strings
`strcmp`	Compare strings
`strcmpi`	Compare strings (case insensitive)
`strncomp`	Compare first n characters of strings (case sensitive)
`strncmpi`	Compare first n characters of strings (case insensitive)
`contains`	Determine if pattern is in strings
`strfind`	Find strings within other strings
`strrep`	Find and replace substrings
`count`	Count occurrences of pattern in strings

Example: upper Built-in Function

Q. Write a custom function named '`MyUpper`' to replicate the same operation of `upper`. The input and output types are assumed to be string.

```
function newStr = MyUpper(str)

char_db = double(char(str));

lgv = 97 <= char_db & char_db <= 122;

char_db(lgv) = char_db(lgv)-32;

newStr = string(char(char_db));

end
```

```
>> str = "abc!ABC?";
>> upper(str)

ans =

    "ABC!ABC?"

>> MyUpper(str)

ans =

    "ABC!ABC?"
```

📖: Here is the syntax for `upper` built-in function: `newStr = upper(str)` converts all lowercase characters in `str` to the corresponding uppercase characters and leaves all other characters unchanged.

⚠: string type has no equivalent numeric values. Thus, we cannot do the type casting of strings to numeric values using `double()`.

Example: strfind and count Built-in Function

Q. Write a script to count the number of 'word' in 'char_vec'

option1

```
char_vec = 'abcardcsecar';
word = 'car';

vec_len = numel(char_vec);
n_char = numel(word);

n_word = 0;
for ii=1:vec_len-n_char+1
    test_loc = ii:ii+n_char-1;
    if isequal(char_vec(test_loc), word)
       n_word = n_word + 1;
    end
end
```

option2

```
char_vec = 'abcardcsecar';
word = 'car';

idx = strfind(char_vec, word);
n_word = numel(idx);
```

option3

```
char_vec = 'abcardcsecar';
word = 'car';

n_word = count(char_vec, word);
```

📖: `k = strfind(str,pattern)` searches str for occurrences of pattern. The output, `k`, indicates the starting index of each occurrence of pattern in `str`.

`A = count(str,pattern)` returns the number of occurrences of `pattern` in `str`.

📖: Again, the function support both **strings** and **character** vectors as the input for `str`.

Example: strfind Built-in Function (Word Finder Puzzle)

Q. Rewrite a script for a word finder puzzle using `strfind`.

```
word_loc = zeros(n_word, 2);
for ii=1:puzzle_size
    col_vec = puzzle(:,ii); % column
    row_vec = puzzle(ii,:); % row

    for jj=1:(puzzle_size-n_word+1)
        test_loc = jj:(jj+n_word-1);

        test_word_col = col_vec(test_loc);
        test_word_row = row_vec(test_loc);

        if isequal(test_word_col, word_db')
            word_loc(:,2) = ii;
            word_loc(:,1) = test_loc';
        elseif isequal(test_word_row, word_db)
            word_loc(:,1) = ii;
            word_loc(:,2) = test_loc';
        end
    end
end
```

```
for ii=1:puzzle_size
    col_vec = char(puzzle(:,ii)); % column
    row_vec = char(puzzle(ii,:)); % row

    id_col = strfind(col_vec', char(word_db));
    id_row = strfind(row_vec, char(word_db));

    if ~isempty(id_col)
        word_loc(:,2) = ii;
        word_loc(:,1) = (id_col:id_col+n_word-1)';
    elseif ~isempty(id_row)
        word_loc(:,1) = ii; % row
        word_loc(:,2) = (id_row:id_row+n_word-1)';
    end
end
```

📖: `k = strfind(str,pattern)` searches `str` for occurrences of `pattern`. The output, `k`, indicates the starting index of each occurrence of `pattern` in `str`.

Comparing Strings: Equality Operator (String)

To use the equality operator == with character vectors, they must be the same length, and each element will be compared. The output is the same size as the input vector size. However, for strings, it will simply return 1 or 0 instead of comparing each element.

```
char1 = 'ENVE';
char2 = 'GEOE';
char3 = 'ENVE';
char4 = 'AE';
char5 = 'G';

str1 = "AE";
str2 = "Hello";
str3 = "AE";
str4 = "A";
```

`char1 == char2`	`[0 0 0 1]`
`char1 == char3`	`[1 1 1 1]`
`char1 == char4`	`error`
`char2 == char5`	`[1 0 0 0]`
`char2 ~= char5`	`[0 1 1 1]`
`str1 == str2`	`0`
`str1 == str3`	`1`
`str1 == str4`	`0`
`str1 ~= str4`	`1`
`[str1 str2] == str3`	`[1 0]`
`[str1 str2] == str4`	`[0 0]`
`string(char1) == ... string(char3)`	`1`

⚠: You need to clearly understand the usage of the equality operator when character or strings are involved.

Create Strings using sprintf

- `sprintf` creates text so it can be used to customize the format of text.

```
str = sprintf(formatSpec,A1,...,An)
```

- Formats the data in arrays `A1,...,An` using the formatting texts specified by `formatSpec` and returns the resulting text.
- In `formatSpec`, if you use **a single quote**, the output becomes a **character vector**. If it has **double quote**, the output becomes a **string**.

⚠: `sprintf` and `fprintf` have the same syntax for the input. The difference is `sprintf` creates a string or character vector whereas `fprintf` prints it in a command window or file.

```
>> str1 = "AEG";
>> str2 = "121";

>> "Welcome! " + str1 + str2

ans =

    "Welcome! AEG121"

>> sprintf("Welcome! %s%s", str1, str2)

ans =

    "Welcome! AEG121"

>> sprintf('Welcome! %s%s', str1, str2)

ans =

    'Welcome! AEG121'
```

Example: Generate a Card Sequence String

The standard 52-card deck has 13 numbers and 4 different suits. The suit order is 'Clubs (♣)', 'Diamonds (♦)', 'Hearts (♥)', and 'Spades (♠)'. Each integer from 1 to 52 will represent the value and suit of a card, where from 1 to 52, and the value and suit of the cards will proceed in the following order:

Num	Card
1	1C
2	1D
3	1H
4	1S
5	2C
6	2D
:	:
50	13D
51	13H
52	13S

```
function str_cards = StrCard(cards)

suits = 'CDHS';
str_cards = "";
for ii = 1:numel(cards)
    card_num = ceil(cards(ii)/4);
    card_rem = rem(cards(ii),4);

    if card_rem==0
        card_rem=4;
    end

    card_suit = suits(card_rem);

    str_cards = str_cards +  ...
        sprintf("%d%s ", card_num, card_suit);
end

str_cards{1}(end) = [];

end
```

Q. Create a function named '`StrCard`' that generate a card sequence string. The input named '`cards`' is a 1 x 7 numeric vector and include the integer card number. The output named '`str_cards`' is a string scalar and include all name of the cards separated by a space. The text for suits is 'C', 'D', 'H' and 'S', corresponding to 'Clubs', 'Diamonds', 'Hearts', and 'Spades'.

```
>> cards = [1 2 3 4 5 6 7];
>> str_cards = StrCard(cards);
>> str_cards

str_cards =

    "1C 1D 1H 1S 2C 2D 2H"
```

String/Number Functions

Challenging

Converting from strings or character vectors to numbers and vice versa:

- `num2str` converts a real number to a character vector containing the number
- `string` converts number(s) to strings
- `str2double` converts from a string or character vector containing number(s) to a number array

```
num1 = 75;
num2 = 12345;

char1 = ‘abcd’
char2 = ‘678910’
```

`char(num1)`	`'K'`
`char(num2)`	`‘#’`
`char(num1)`	`'K’`
`num2str(num1)`	`'75’`
`string(num1)`	`"75"`
`char(num2)`	`'#’`
`num2str(num2)`	`'12345'`
`string(num2)`	`"12345"`
`double(char1)`	`[97 98 99 100]`
`str2double(char1)`	`NaN`
`double(char2)`	`[54 55 56 57 49 48]`
`str2double(char2)`	`678910`

⚠: This is very confusing!! You should not be confused with **type casting** and number conversion from **string or character vectors**.

Example: How Many Digit in Your Number?

Q. Create a function named 'CountNum' to count the appearance of the given digit in a single numeric number. The digit should be within 0 and 9. For example:

`CountNum(454214, 4)`	`3`
`CountNum(454221, 2)`	`2`

```
function countd = CountNum(num, digit)

chard_num = num2str(num);
chard_digit = num2str(digit);
lg_vec = chard_num == chard_digit;

countd = sum(lg_vec);
end
```

option1

```
function countd = CountNum(num, digit)

countd = count(num2str(num), num2str(digit));
% countd = count(string(num), string(digit));

end
```

option2

```
CountNum(454214, 4)
```

Name	Value
num	454214
digit	4
card_num	'454214'
card_digit	'4'
lg_vec	[1 0 1 0 0 1]
countd	3