Text Manipulation

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

AE121: Computational Method

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-07-08

Text Terminology

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

Characters

- Both character vectors and string scalars are composed of groups of characters
- ***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

- Character vectors:
  - consist of any number of characters
  - are contained in single quotation marks
  - are displayed using single quotation marks
  - are of the type, or class, **char**
  - Examples: ‘x’, ‘ ’, ‘abc’, ‘hi there’
- Since these are vectors of characters, many built-in functions and operators that we’ve seen already work with character vectors as well as numbers – e.g., **length** to get the length of a character vector, or the transpose operator
- You can also index into a character vector to get individual characters or to get subsets

Dimensions of Character Vectors

- A single character is a 1 x 1 scalar

```
>> letter = 'x';
>> size(letter)
ans =
     1     1
```

- A character vector is 1 x n where n is the length

```
>> myword = 'Hello';
>> size(myword)
ans =
     1     5
```

Revisit: Lab05-Problem 4

l	s	t	p	q	u	g	g	m	g
u	o	q	o	h	q	p	f	e	y
v	g	v	v	s	y	r	g	o	d
v	o	x	h	d	q	c	c	w	e
j	f	z	o	r	c	r	r	x	f
b	a	n	a	n	a	k	i	i	f
p	d	c	x	s	e	d	p	g	r
e	w	w	q	t	w	o	s	p	q
e	p	r	w	x	o	g	o	r	d
f	y	z	t	z	w	e	p	l	y

```
% search columns
isRun = true;
for ii=1:puzzle_size
    col_vec = puzzle(:,ii);
    for jj=1:(puzzle_size-n_word+1)
        test_loc = jj:(jj+n_word-1);
        test_word = col_vec(test_loc);
        if isequal(word_db, test_word')
            word_loc(:,2) = ii;
            word_loc(:,1) = test_loc;
            isRun = false;
            break;
        end
    end
end
```

all(word_db == test_word’)

Revisit: Lab05-Problem 4 (Continue)

```
if isequal(word_db, test_word')
    word_loc(:,2) = ii;
    word_loc(:,1) = test_loc;
    isRun = false;
    break;
end
```

l
u
v
v
j
b

word_db => ‘luvvjb‘

b	a	n	a	n	a

test_word => ‘banana‘

```
is_word_same = 1;
for ii=1:6
    if word_db(ii)~=test_word(ii)
        is_word_same = 0;
    end
end

if is_word_same
    word_loc(:,2) = ii;
    word_loc(:,1) = test_loc;
    isRun = false;
    break;
end
```

String Scalars

- A string scalar is a single string, and is used to **store a group of characters such as a word**
- Strings can be created
  - using double quotes, e.g. “Awesome”
  - using the **string** function, e.g. string(‘Awesome’)
- Strings are displayed using double quotes
- The **class** of a string scalar is ‘string’

A string is a 1 x 1 scalar

```
>> mystr = "Awesome";
>> size(mystr)
ans =
     1     1
```

String Indexing

A character vector is contained in a string; curly braces can be used to index:

```
mystr = "AE121"
mystr(1)
mystr{1}
mystr{1}(1)

mycharvec = 'AE121'
mycharvec(1)
```

```
mystr = "AE121"
ans = "AE121"
ans = 'AE121'
ans = 'A'

mycharvec = 'AE121'
ans = 'A'
```

String Length

- We have seen that the **length** function can be used to find the number of characters in a character vector
- However, since a string is a 1 x 1 scalar, the length of a string is 1
- Use the **strlength** function for strings:
- Even an empty string is a 1 x 1 scalar, so the length is 1 (not 0)

```
str1 = "AE121"
length(str1)
strlength(str1)

charvec1 = 'AE121'
length(charvec1)

length('') % empty
length("") % empty

length('  ') % two space
length("  ") % two space
```

```
str1 = "AE121"
ans = 1
ans = 5

charvec1 = 'AE121'
ans = 5

ans = 0
ans = 1

ans = 2
ans = 1
```

Groups of strings

- Groups of strings can be stored in
  - string arrays
  - character matrices
  - cell arrays (in Chapter 8)
- **String arrays** are very easy to use:

```
>> greets = ["hi", "ciao"]
greets =
  1×2 string array
    "hi"    "ciao"
>> greets(1)
ans =
    "hi"
```

Example: Simple Example 1

```
name_list = ["Chul Min", "Jason", "Robert", "Susannah", ...
    "Serena", "Amy", "Rose"];

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

```
Welcome! Chul Min
Welcome! Jason
Welcome! Robert
Welcome! Susannah
Welcome! Serena
Welcome! Amy
Welcome! Rose
```

Example: Simple Example 2

```
name_list = ["Chul Min", "Instructor";
             "Jason", "TA";
             "Robert", "Student";
             "Susannah", "Student"];

num_name = size(name_list,1);
for ii=1:num_name
   fprintf('Welcome! %s (%s) \n', name_list(ii,1), name_list(ii,2));
end
```

```
Welcome! Chul Min (Instructor)
Welcome! Jason (TA)
Welcome! Robert (Student)
Welcome! Susannah (Student)
```

Operations on Strings

- Strings are created using double quotes or by passing a character vector to the **string** function
- Without any arguments, the **string** function creates an empty string (which, don't forget, is still a 1 x 1 scalar so the length is 1)
- The **plus** function or operator can join, or concatenate, two strings together

```
>> "abc" + "xyz"
ans =
    "abcxyz"
```

Text Functions

- 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
- For example, changing case using **upper** or **lower**:

```
>> upper('abc')
ans =
    'ABC'
>> lower("HELP")
ans =
    "help"
```

String Concatenation

- There are several ways to ***concatenate***, or join, strings (using +) and character vectors (using [ ])

```
>> "abc"+"d"
ans =
    "abcd"
>> ['abc' 'd']
ans =
    'abcd'
```

- The **strcat** function can be used; it will remove trailing blanks for character vectors but not for strings

```
>> strcat('Hi  ', 'everyone')
ans =
    'Hieveryone'
>> strcat("Hi   ", "everyone")
ans =
    "Hi   everyone”
```

The sprintf function

- **sprintf** works just like **fprintf**, but instead of printing, it creates text– so it can be used to customize the format of text.
- Any time text is required as an input, **sprintf** can create customized text.

sprintf

Format data into string or character vector

col

Syntax

```
str = sprintf(formatSpec,A1,...,An)
[str,errmsg] = sprintf(formatSpec,A1,...,An)
str = sprintf(literalText)
```

Description

`str = sprintf(formatSpec,A1,...,An)` formats the data in arrays `A1,...,An` using the formatting operators specified by `formatSpec` and returns the resulting text in `str`. The `sprintf` function formats the values in `A1,...,An` in column order. If `formatSpec` is a string, then so is the output `str`. Otherwise, `str` is a character vector.

To return multiple pieces of formatted text as a string array or a cell array of character vectors, use the `compose` function.

Example: Simple Example

```
name_list = ["Chul Min", "Instructor";
             "Jason", "TA";
             "Robert", "Student";
             "Susannah", "Student"];

num_name = size(name_list,1);
str = [];
for ii=1:num_name
   str = [str sprintf('Welcome! %s (%s) \n', name_list(ii,1), name_list(ii,2))];
end
fprintf('%s', str)
```

```
Welcome! Chul Min (Instructor)
Welcome! Jason (TA)
Welcome! Robert (Student)
Welcome! Susannah (Student)
```

```
name_list = ["Chul Min", "Instructor";
             "Jason", "TA";
             "Robert", "Student";
             "Susannah", "Student"];

num_name = size(name_list,1);
str = [];
for ii=1:num_name
   str = strcat(str, sprintf("Welcome! %s (%s) \n", name_list(ii,1), name_list(ii,2)));
end
fprintf('%s', str)
```

Example: Poker Game

```
soln_ranks = zeros(numTest, 1);

for ii = 1:numTest
    your_cards = cards_all(:,ii);


    is_ryl_str_fls = ChckRylFls(your_cards); % Royal Straight Flush (Rank 1)
    is_str_fls = ChckStrFls(your_cards); % Straight Flush (Rank 2)
    is_fr_knd = ChckFrKnd(your_cards); % Four of a Kind (Rank 3)
    is_fll_hs = ChckFllHs(your_cards); % Full House (Rank 4)
    is_fls = ChckFls(your_cards); % Flush (Rank 5)
    is_str = ChckStr(your_cards); % Straight (Rank 6)
    is_thr_knd = ChckThrKnd(your_cards); % Three of a Kind (Rank 7)
    is_tw_prs = ChckTwPrs(your_cards); % Two Pairs (Rank 8)
    is_pr = ChckPr(your_cards); % Pair (Rank 9)
    % If your card has no above ranks, in rule, the highest number is picked in
    % your cards. Thus, the rank is called "High Card" (Rank 10).

    % ChckPr is provided to help your understand desigining functions. Please
    % review it carefully. Also, designing ChckStr is very challenging
    % that we provide this code as well. However, you need to understand how it
    % works to design both ChckRylFls and ChckStrFls.

    % a value in 'rank' indicates:
    % 1: Royal Straight Flush
    % 2: Straight Flush
    % 3: Four of a Kind
    % 4: Full House
    % 5: Flush
    % 6: Straight
    % 7: Three of a Kind
    % 8: Two Pairs
    % 9: Pair
    % 10: High card
```

```
    % 9: Pair
    % 10: High card

    if is_ryl_str_fls == 1
        rank = 1;
    elseif is_str_fls == 1
        rank = 2;
    elseif is_fr_knd == 1
        rank = 3;
    elseif is_fll_hs == 1
        rank = 4;
    elseif is_fls == 1
        rank = 5;
    elseif is_str == 1
        rank = 6;
    elseif is_thr_knd == 1
        rank = 7;
    elseif is_tw_prs == 1
        rank = 8;
    elseif is_pr == 1
        rank = 9;
    else
        rank = 10;
    end

    soln_ranks(ii) = rank;
end
```

Example: Poker Game (Continue)

```
numTest = referenceVariables.numTest;
true_soln = referenceVariables.soln_ranks;

rank_str = ["Royal Straight Flush", "Straight Flush", "Four of a Kind", ...
"Full House", "Flush", "Straight", "Three of a Kind", "Two Pairs", ...
"Pair", "High Card"];



for ii=1:numTest
    is_same = isequal(true_soln(ii), soln_ranks(ii));
    if ~is_same
       stu_ans = sprintf('Your rank is %s\n', rank_str(soln_ranks(ii)));
       ref_ans = sprintf('The correct rank is %s\n', rank_str(true_soln(ii)));
       str =  sprintf('Your code ran incorrectly at test %d\n ', ii);
       cds = sprintf('Your cards are %s', PrntCards(cards_all(:,ii)));
       error([stu_ans ref_ans str cds]);
    end

end
```

Code Tester

Your rank is Straight Flush

The correct rank is Royal Straight Flush

Your code ran incorrectly at test 1

Your cards are 1C, 6C, 2H, 10C, 11C, 12C, 13C.

Example: Poker Game (Continue)

```
function str_cards = PrntCards(cards)

str_cards = [];
suits = 'CDHS'; % C = Clubs, D = Diamonds, H = Hearts, S = Spades
for ii = 1:numel(cards)
    card_num = ceil(cards(ii)/4);
    % Card number is equal to number times card number is divisible by
    % rounding up

    card_rem = rem(cards(ii),4);
    % Top row (suit) has rem of 1 when divided by 4 and the second row has
    % a remainder of 2...

    if card_rem==0
        card_rem=4;
    end
    % when card_rem is 0, it indicates the fourth row.

    your_suit = suits(card_rem);
    % The remainder is equal to the index of suits

    if ii~=numel(cards)
        str_cards = [str_cards sprintf('%d%s,  ', card_num, your_suit)];
    else
        str_cards = [str_cards sprintf('%d%s. \n', card_num, your_suit)];
    end
end
```

Code Tester

Your rank is Straight Flush

The correct rank is Royal Straight Flush

Your code ran incorrectly at test 1

Your cards are 1C, 6C, 2H, 10C, 11C, 12C, 13C.

Comparing Strings

- **strcmp** compares two strings or character vectors and returns logical 1 if they are identical or 0 if not (or not the same length)
- Variations:
  - **strncmp** compares only the first n characters
  - **strcmpi** ignores case (upper or lower)
  - **strncmpi** compares n characters, ignoring case

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. For strings, however, it will simply return 1 or 0:

```
>> 'cat' == 'car'
ans =
   1   1   0
>> 'cat' == 'mouse'
Matrix dimensions must agree.
>> "cat" == "car"
ans =
   0
>> "cat" == "mouse"
ans =
   0
```

Revisit: Lab05-Problem 4

```
if isequal(word_db, test_word')
    word_loc(:,2) = ii;
    word_loc(:,1) = test_loc;
    isRun = false;
    break;
end
```

l
u
v
v
j
b

word_db => ‘luvvjb‘

b	a	n	a	n	a

test_word => ‘banana‘

```
word_db = 'luvvjb';
word_db_true = 'banana';
test_word = 'banana';

string(word_db)

fprintf('1====================')
word_db == test_word
word_db_true == test_word

fprintf('2====================')
isequal(word_db, test_word)
isequal(word_db, test_word)

fprintf('3====================')
string(word_db) == string(test_word)
string(word_db_true) == string(test_word)

fprintf('4====================')
strcmp(word_db, test_word)
strcmp(word_db_true, test_word)
```

```
ans = "luvvjb"
1====================
ans = 1×6 logical array
     0   0   0   0   0   0

ans = 1×6 logical array
     1   1   1   1   1   1

2====================
ans = logical
   0
ans = logical
   0
3====================
ans = logical
   0
ans = logical
   1
4====================
ans = logical
   0
ans = logical
   1
```

Find and replace functions

Note that the word “string” here can mean string or character vector

- **strfind(string, substring)**: finds all occurrences of the substring within the string; returns a vector of the indices of the beginning of the strings, or an empty vector if the substring is not found
- **strrep(string, oldsubstring, newsubstring)**: finds all occurrences of the old substring within the string, and replaces with the new substring
  - the old and new substrings can be different lengths
- **count(string, substring)**: counts the number of occurrences of a substring within a string

Revisit: HW03-Problem 3

```
% find a word location
word_loc = zeros(n_char, 2, n_word);
for ii = 1:n_word
    word_test = words(ii, :);

    for jj = 1:puzzle_size
        test_row = strfind(char(puzzle(jj, :)), word_test); % search rows
        test_col = strfind(char(puzzle(:, jj))', word_test); % search a column
        if ~isempty(test_row)
            word_loc(:,1,ii) = jj;
            word_loc(:,2,ii) = test_row:(test_row+n_char-1);
            break
        elseif ~isempty(test_col)
            word_loc(:,1,ii) = test_col:(test_col+n_char-1);
            word_loc(:,2,ii) = jj;
            break
        end
    end
end
```

String/Number Functions

Converting from strings to numbers and vice versa:

- **int2str** converts from an integer to a character vector storing the integer
- **num2str** converts a real number to a character vector containing the number
- **string** converts number(s) to strings
- **str2num (**and **str2double)** converts from a string or character vector containing number(s) to a number array

Common Pitfalls

- Trying to use == to compare character vectors for equality, instead of the **strcmp** function (and its variations)

- Confusing **sprintf** and **fprintf**. The syntax is the same, but **sprintf** creates a string whereas **fprintf** prints

Slide Credits and References

- Stormy Attaway, 2018, Matlab: A Practical Introduction to Programming and Problem Solving, 5th edition
- Lecture slides for “Matlab: A Practical Introduction to Programming and Problem Solving”
- Holly Moore, 2018, MATLAB for Engineers, 5th edition