B2 • Discussion #6 • 2021-03-05

Remind us ~10:38 am EST to stop Discussion for the quiz! (15 min + 5 upload)

Announcements

- Homework #3 was posted Monday and is done individually to get ready for the C component of the course
  → Due: Monday, March 15th by 10 am EST
- Exam 2 is two weeks from today! Friday, March 19th from 4:30-6:15 pm EST for all.

REMINDER: VIDEO ON FOR DISCUSSION FOR FULL ATTENDANCE GRADE CREDIT

Review Last Week's Quiz

Notes from class this week

- the + operator is not used to concatenate two character vectors
- the function iscellstr() despite its name, is determining whether or not a cell array contains only character vectors (not strings).
- When creating structures using the struct() function, as of R2018b, strings can be used for field names (before they had to be character vectors)

Review Lecture & Test Material

- strings and character vectors
- cell arrays
- structures — make sure you know how these are displayed when called in Command Window
- when to use cell arrays vs. structures
- vector of structures
- nested structures
- tables (used a lot in Machine Learning aka ML)
- categorical arrays (also used a lot in ML)

TEXT MANIPULATION FUNCTIONS + NOTES

strtok()

ex. strtok(strvar, delimiter)

note that default delimiter is always a space

```
ex. example = 'Well hello'
   [f l] = strtok(example)
      f =
         'Well'
      l =
         ' hello'
```

note that l includes delimiter (space)

strcat()

ex. strcat(var1, var2)

ex. strcat('Well', 'hello')

```
ans =
    'Wellhello'
```

ex. strcat('Well', ' hello')

```
ans =
    'Well hello'
```

strrep()

ex. strrep(string, oldstring, newstring)

ex. strrep('hello', 'lo', 'p!')

```
ans =
    'help!'
```

→ strrep('hello', 'lo', 'lp me!')

```
ans =
    'help me!'
```

sprintf()

```
ex. var = sprintf( )
    input(var)
```

Functions need to know that work on strings but <u>not</u> char. vectors

+ strings( ) strjoin( ) strsplit( ) join( )

Structs, Nested Structs, and printing in a nice format

```
fprintf('This program will create a struct called roster and then write the\n')
fprintf('contents of the struct in a nice format to sandlot.txt.\n\n')
pause(3)
fprintf('Remember, when creating structs always pre-allocate by filling in the\n')
fprintf('last ''row'' first!\n')
pause(3)

roster(9) = struct('Order', 9, 'Name', "Tommy Timmons", 'Player_Info', struct('Position', '3B', 'Nickname', 'Repeat'));
roster(8) = struct('Order', 8, 'Name', "Scott Smalls", 'Player_Info', struct('Position', 'LF', 'Nickname', 'Smalls'));
roster(7) = struct('Order', 7, 'Name', "Michael Palledorous", 'Player_Info', struct('Position', 'RF', 'Nickname', 'Squints'));
roster(6) = struct('Order', 6, 'Name', "Timmy Timmons", 'Player_Info', struct('Position', '1B', 'Nickname', 'Timmy'));
roster(5) = struct('Order', 5, 'Name', "Hamilton Porter", 'Player_Info', struct('Position', 'C', 'Nickname', 'Ham'));
roster(4) = struct('Order', 4, 'Name', "Kenny DeNunez", 'Player_Info', struct('Position', 'P', 'Nickname', 'Kenny'));
roster(3) = struct('Order', 3, 'Name', "Bertram Weeks", 'Player_Info', struct('Position', '2B', 'Nickname', 'Grover'));
roster(2) = struct('Order', 2, 'Name', "Benny Rodriguez", 'Player_Info', struct('Position', 'CF', 'Nickname', 'The Jet'));
roster(1) = struct('Order', 1, 'Name', "Alan McClennan", 'Player_Info', struct('Position', 'SS', 'Nickname', 'Yeah-Yeah'));

fprintf('\nWe have the following struct, roster:\n')
disp(roster)
pause(2)
fprintf('Note that the fieldnames ''Order'' is an integer, ''Name'' is a string, ''Player_Info''\n')
fprintf('is a struct, ''Position'' is a character vector, and ''Nickname'' is a character vector.\n\n')
pause(3)
fprintf('The struct essentially looks like this...\n')
pause(1)
fprintf('%-7s %-20s %10s\n', 'Order', 'Name', 'Player_Info')
fprintf('%-24s %-9s %-10s\n', ' ', 'Position', 'Nickname')
for i = 1:length(roster)
    pause(1)
    fprintf('%-5d %-22s %-5s %-10s\n', roster(i).Order, roster(i).Name, roster(i).Player_Info.Position, roster(i).Player_Info.Nickname)
end
```

pre allocating & making the vector of structs

If we were looping we would do: for i = 9:-1:1

always preallocate from the bottom

example of printing

roster

	Order	Name	Player_Info	
			Position	Nickname
1	1	Alan McClennan	SS	Yeah-Yeah
2				
3				
4				
5				
6				
7				
8				
9				

← try filling in the rest!

If we wanted to fill in info via a for loop:

```
roster(i).Name = 'Alan'

roster(i).Player_Info.Nickname = 'Yeah-Yeah'
```

If we wanted to change info in an already made struct

```
roster(1).Name = 'Leah'
```

cell array indexing

```
dogs = {'golden', 'aussie', 'lab'};
```

```
>> dogs(1:2)
ans =
  1 x 2 cell array
    {'golden', 'aussie'}
```

```
>> dogs{1:2}
ans =
    'golden'
ans =
    'aussie'
```

```
>> [d1, d2] = dogs{1:2}
d1 =
    'golden'
d2 =
    'aussie'
```

Advantage of cell array over struct?
→ you can index into a cell array, good for looping or vectorizing code

Advantage of struct over cell array?
→ fieldnames are mnemonic!

IMPORTANT TEXT MANIPULATION EXAMPLE!

sports = ["baseball" "football" "track and field"]

```
>> sports(1:2)
   1 x 2 string array
   "baseball" "football"
```

*not sequential commands

```
>> sports{1:2}
   ans =
         "baseball"
   ans =
         "football"
```

```
>> sports(2) = [ ]
   1 x 2 string array
   "baseball" "track and field"
```

```
>> sports{2} = [ ]
   1 x 3 string array
   "baseball" <missing> "track and field"
```

```
>> "I love " + sports
   ["I love baseball" "I love football" etc...]
```

```
>> sports{2}(1:3)
   'foo'
```

(sports(2){1}(1:3) also works)

```
>> sports(3)
   "track and field"
```

↑ 1x1 string scalar