Mixed bags

Working with strings and cell arrays

CS112 Scientific Computation
Department of Computer Science
Wellesley College

Strings

Program input/output

```
month = input('Enter a month: ');
disp(['There are ' num2str(numDays(month)) ' days in that month']);
myTitle = get(handles.titlebox, 'String');
set(handles.sourceToggle, 'String', 'consume');
```

Text labels and graph properties

```
plot(xcoords, ycoords, 'g:*', 'Linewidth', 2);
title('golden ratio data');
```

Cell arrays of strings

```
names = {'Varitek' 'Ortiz' 'Ramirez' 'Drew' 'Lowell' 'Lugo' ... 'Pedroia'};
```

Reading/Writing image files

```
>> im = imread('clinton.jpg');
>> imwrite(im, 'myImage.jpg');
```

Cell arrays
What lies beneath - the ASCII code

The ASCII code, established by the American Society for Communication and Information Interchange, is a numerical code for representing letters, digits, punctuation, and control signals.

The original ASCII code represents characters using a 7-bit code (numbers from 0 to 127).

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Chrs</th>
<th>Dec</th>
<th>Hex</th>
<th>Chrs</th>
<th>Dec</th>
<th>Hex</th>
<th>Chrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>&quot; &quot;</td>
<td>127</td>
<td>7F</td>
<td>&quot;~&quot;</td>
<td>255</td>
<td>FF</td>
<td>&quot; &quot;</td>
</tr>
</tbody>
</table>

MATLAB character strings

Character strings in MATLAB are stored in special numerical vectors of ASCII values.

The double and char functions translate between a character string and its ASCII values:

```
>> string1 = 'violet';
>> numcode = double(string1)
numcode =
    118  105  111  108  101  116
>> char(numcode)
ans =
    violet
```
String processing freebies

```matlab
>> string1 = 'spring break';
>> length(string1)
ans =
    12
>> string1(6)
ans =
    r
>> string1(3:6)
ans =
    pring
>> string1([6 3 11 8])
ans =
        9
>> string1(8:12) = 'fling'
ans =
    spfling
>> string1(20) = '*';
```

Conditional expressions & strings

```matlab
>> string1 == 'g'
ans =
    0 0 0 0 1 0 0 0 0 1
>> string1(string1 == 'f') = 'b'
string1 =
    spfling
>> string1 == 'CS112'

* remember strcmp
```
More fun with strings

```matlab
>> string2 = 'to be or not to be';
>> find(string2 == ' ')
ans =

>> string2Letters = string2(string2 ~= ' ')
string2Letters =

>> string2(string2 == ' ') = []
string2 =
```

String processing newbies

```matlab
>> lower('To Be Or Not To Be')
converts letters to lower case
ans =

>> upper(ans)
converts letters to upper case
ans =

>> strrep('to be or not to be', 'be', 'play')
replaces occurrences of 2nd input string with 3rd input string, in the 1st input string...
ans =

>> strfind(ans, 'ay')
finds all occurrences of second input string in first input string...
returns indices of first character
ans =
```
Time out exercises

What actions are performed by the following statements?

```
    newString = ' ';  % Initialize newString
    for letter = string1  % Iterate over each letter in string1
        newString = [letter newString];  % Append letter to newString
    end
```

What test is performed by the following function?

```
    function answer = test(str)
        str = str(str ~= ' ');  % Remove spaces from str
        str = lower(str);  % Convert str to lowercase
        answer = all(str == str(end:-1:1));  % Check if string is palindrome
    end
```

```
    >> answer = test('Murder for a jar of red rum')
    ans =
        1
```

Collecting strings with cell arrays

We have used a **cell array** to store a collection of strings

```
    >> myPets = {'mona' 'cleo'};
```

We can access the contents of individual locations of a cell array using an index placed **inside curly braces**:

```
    >> myPets{1}
    ans =
        mona

    for index = 1:length(myPets)
        disp(myPets{index});
    end
```

**Exercise:** Write a function with a single input that is a cell array, which prompts the user for a string and determines whether the user’s string is contained in the input cell array.
Collecting multiple types of data

The *real power* of cell arrays is that they allow us to store multiple types of data in one place:

```matlab
>> myCell = {'Ellen', 3.14159, [2 5 1 7], [1 2; 3 4]}
myCell =
    'Ellen' [3.14159] [1x4 double] [2x2 double]

>> celldisp(myCell)
myCell(1) = Ellen
myCell(2) = 3.1416
myCell(3) = 2 5 1 7
myCell(4) = 1 2
            3 4
```

Create a cell array from scratch with the `cell` function:

```matlab
>> newCell = cell(1,3);
>> newCell{1} = 'Stella';
>> newCell{2} = 'SCI E131';
>> newCell{3} = stellaImage;
```

A thousand words...

The `cellplot` function provides a high-level picture of a cell array:

```matlab
>> myCell = {'Ellen', 3.14159, [2 5 1 7], [1 2; 3 4]}
myCell =
    'Ellen' [3.14159] [1x4 double] [2x2 double]

>> cellplot(myCell)
```

Cell arrays

Cell arrays
Accessing the contents of cell arrays

Contents of individual locations of a cell array can be accessed with an index surrounded by curly braces:

\[
\text{myCell} = \{'Ellen' \quad 3.14159 \quad [2 \ 5 \ 1 \ 7] \quad [1 \ 2; \ 3 \ 4]\};
\]

\[
\text{disp}([\text{myCell}(1) \ 's \ favorite \ number \ is \ ' \ num2str(myCell(2))])
\]

ans =

\[
\text{myCell}(3)(2)
\]

ans =

\[
\text{sum(sum(myCell(4)))}
\]

Into thin air...

\[
\text{mountains} = \{'\text{Everest}' \quad 'K2' \quad 'Kanchenjunga' \quad 'Lhotse I' \quad 'Makalu I' \ ... \\
'Lhotse II' \quad 'Dhaulagiri' \quad 'Manaslu I' \quad 'Cho Oyu' \ ... \\
'Nanga Parbat' \quad 'Annapurna' \ ... \\
'\text{Himalayas}' \quad 'Karokoram' \quad 'Himalayas' \quad 'Himalayas' \quad 'Himalayas' \quad 'Himalayas' \quad 'Himalayas' \ ... \\
'\text{Himalayas}' \quad 'Himalayas' \quad 'Himalayas' \quad 'Himalayas' \ ... \\
'\text{Nepal-India}' \quad 'Nepal-China' \quad 'Nepal-China' \quad 'Nepal-China' \quad 'Nepal-China' \quad 'Nepal-China' \quad 'Nepal-China' \ ... \\
'\text{Kashmir}' \quad 'Nepal'} \ ...
\]

\[
\text{mount} = \text{input('Enter the name of a mountain: ', 's')}
\]

Exercise:

Write a loop that prints all of the information about the user’s input mountain