- Previous Lecture:
  - Working with images

- Today’s Lecture:
  - Characters and strings

- Announcements:
  - Discussion this week in classrooms as listed on roster
  - Project 4 due Thurs 3/28 at 11pm
We have used strings already:

```python
n = input('Next number: ')
sprintf('Answer is %d', ans)
```

A string is made up of individual characters, so a string is a 1-d array of characters.

‘CS1112 rocks!’ is a character array of length 13; it has 7 letters, 4 digits, 1 space, and 1 symbol.

```
'C'S'1'1'1'2' 'r'o'c'k's'!'
```

Can have 2-d array of characters as well

```
'C'S'1'1'1'2'
'r'o'c'k's'!
```

2×6 matrix
Matlab types: char, double, uint8, logical

There is not a type “string”! What we call a string is a 1-d array of chars

\[ a = \begin{bmatrix} 'C' & 'S' & '1' \end{bmatrix} \]

\( a \) is a 1-d array with type char components. We call \( a \) a “string” or “char array”

\[ b = [3 \ 9] \]

\( b \) is a 1-d array with type double components. double is the default type for numbers in Matlab. We call \( b \) a “numeric array”

\[ c = \text{uint8}(b) \]

\( c \) is a 1-d array with type uint8 components. We call \( c \) a “uint8 array”

\[ d = \text{rand} > .5 \]

\( d \) is a scalar of the type logical. We call \( d \) a “boolean value”
Strings are important in computation

Numerical data is often encoded in strings. E.g., a file containing Ithaca weather data begins with the string `W07629N4226` meaning

Longitude: 76° 29’ West
Latitude: 42° 26’ North

We may need to grab hold of the substring `W07629`, convert 076 and 29 to the numeric values 76 and 29, and do some computation
Comparison of genomic sequences is another example of string computation

- E.g., looking for a pattern:
  Given the sequence `ATTCTGACCTCGATC…`
  Look for the pattern `ACCT`

- E.g., quantifying the difference between sequences:
  
  `ATTCTGACCTCGATC`
  
  `ATTCGTGACCTCGAT`
  
  `ATTCGTGACCTCGAT`

  What if this nucleotide is removed?
Single quotes enclose strings in Matlab

Anything enclosed in single quotes is a string (even if it looks like something else)

- ‘100’ is a character array (string) of length 3
- 100 is a numeric value
- ‘pi’ is a character array of length 2
- pi is the built-in constant 3.1416…
- ‘x’ is a character (vector of length 1)
- x may be a variable name in your program
Strings are vectors

**Vectors**

- **Assignment**
  
  \( v = [7 \ 0 \ 5] \);

- **Indexing**
  
  \( x = v(3); \) \hspace{2em} \% x is 5  
  \( v(1) = 1; \) \hspace{2em} \% v is [1 \ 0 \ 5]  
  \( w = v(2:3); \) \hspace{2em} \% w is [0 \ 5]  

- **: notation**
  
  \( v = 2:5; \) \hspace{2em} \% v is [2 \ 3 \ 4 \ 5]  

- **Appending**
  
  \( v = [7 \ 0 \ 5]; \)
  \( v(4) = 2; \) \hspace{2em} \% v is [7 \ 0 \ 5 \ 2]

- **Concatenation**
  
  \( v = [v [4 \ 6]]; \)
  \( \% v is [7 \ 0 \ 5 \ 2 \ 4 \ 6] \)

**Strings**

- **Assignment**
  
  \( s = 'hello'; \)

- **Indexing**
  
  \( c = s(2); \) \hspace{2em} \% c is ‘e’  
  \( s(1) = 'J'; \) \hspace{2em} \% s is ‘Jello’  
  \( t = s(2:4); \) \hspace{2em} \% t is ‘ell’

- **: notation**
  
  \( s = 'a':'g'; \) \hspace{2em} \% s is ‘abcdefg’

- **Appending**
  
  \( s = 'duck'; \)
  \( s(5) = 's'; \) \hspace{2em} \% s is ‘ducks’

- **Concatenation**
  
  \( s = [s \ ' quack']; \)
  \( \% s is ‘ducks quack’ \)
Some useful string functions

str = ‘Cs 1112’;

length(str) % 7
isletter(str) % [1 1 0 0 0 0 0]
isspace(str) % [0 0 1 0 0 0 0]
lower(str) % ‘cs 1112’
upper(str) % ‘CS 1112’

ischar(str)
    % Is str a char array? True (1)
strcmp(str(1:2), ‘cs’)
    % Compare strings str(1:2) & ‘cs’. False (0)
strcmp(str(1:3), ‘CS’)
    % False (0)
Example: capitalize 1st letter

Write a function to capitalize the first letter of each word in a string. Assume that the string has lower case letters and blanks only. (OK to use built-in function `upper`)

```matlab
function [str, nCaps] = caps(str)
% Post: Capitalize first letter of each word.
% str = partially capitalized string
% nCaps = no. of capital letters
% Pre: str = string with lower case letters & blanks only
```

Look For The Spaces
# ASCII characters

(American Standard Code for Information Interchange)

<table>
<thead>
<tr>
<th>ascii code</th>
<th>Character</th>
<th>ascii code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>65</td>
<td>‘A’</td>
<td>48</td>
<td>‘0’</td>
</tr>
<tr>
<td>66</td>
<td>‘B’</td>
<td>49</td>
<td>‘1’</td>
</tr>
<tr>
<td>67</td>
<td>‘C’</td>
<td>50</td>
<td>‘2’</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>90</td>
<td>‘Z’</td>
<td>57</td>
<td>‘9’</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
Character vs ASCII code

str = 'Age 19'

% a 1-d array of characters

code = double(str)

% convert chars to ascii values

str1 = char(code)

% convert ascii values to chars
Arithmetic and relational ops on characters

- `'c' - 'a'` gives 2
- `'6' - '5'` gives 1
- `letter1='e'; letter2='f';
- `letter1-letter2` gives -1
- `'c'>'a'` gives true
- `letter1==letter2` gives false
- `'A' + 2` gives 67
- `char('A'+2)` gives ‘C’
What is in variable \( g \) (if it gets created)?

\[
d1 = 'Mar 3'; \quad d2 = 'Mar 9'; \\
x1 = d1(5); \quad x2 = d2(5); \\
g = x2 - x1;
\]

A: the character ‘6’
B: the numeric value 6
C: Error in the subtraction operation
D: Error in assigning variables \( x1, x2 \)
E: Some other value or error
What is in variable \( g \) (if it gets created)?

\[
d1 = 'Mar 13'; \quad d2 = 'Mar 29'; \\
x1 = d1(5:6); \quad x2 = d2(5:6); \\
g = x2 - x1;
\]

A: the string ‘16’

B: the numeric value 16

C: Error in the subtraction operation

D: Error in assigning variables \( x1, x2 \)

E: Some other value or error
Example: toUpper

Write a function `toUpper(cha)` to convert character `cha` to upper case if `cha` is a lower case letter. Return the converted letter. If `cha` is not a lower case letter, simply return the character `cha`.

**Hint:** Think about the distance between a letter and the base letter ‘a’ (or ‘A’). E.g.,

\[
\begin{array}{cccccc}
a & b & c & d & e & f & g & h & \ldots \\
A & B & C & D & E & F & G & H & \ldots \\
\end{array}
\]

\[\text{distance} = \text{‘g’-‘a’} = 6 = \text{‘G’-‘A’}\]

Of course, do not use Matlab function `upper`!
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.
function up = toUpper(cha)

% up is the upper case of character cha.
% If cha is not a letter then up is just cha.

up = cha;

cha is lower case if it is between 'a' and 'z'
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.

up = cha;

if ( cha >= 'a' && cha <= 'z' )

    % Find distance of cha from 'a'

end
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.

up = cha;

if ( cha >= 'a' && cha <= 'z' )
    offset = cha - 'a';
    up = char(offset + 'A');

end
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.

up= cha;

if ( cha >= 'a' && cha <= 'z' )
    % Find distance of cha from ‘a’
    offset= cha - 'a';
    % Go same distance from ‘A’
    up= char('A' + offset);
end
Example: censoring words

function D = censor(str, A)

% Replace all occurrences of string str in
% character matrix A with X’s, regardless of
% case.
% Assume str is never split across two lines.
% D is A with X’s replacing str.

A

Use MATLAB
in that lab.

censor ‘lab’

D

Use MATRIX
in that XXX.
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A, 
% regardless of case, with X's. 
% A is a matrix of characters. 
% str is a string. Assume that str is never split across two lines. 
% D is A with X's replacing the censored string str.

D= A;
B= lower(A);
s= lower(str);
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length

% Traverse the matrix to censor string str
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A, 
% regardless of case, with X's. 
% A is a matrix of characters. 
% str is a string.  Assume that str is never split across two lines. 
% D is A with X's replacing the censored string str.

D= A;
B= lower(A);
s= lower(str);
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length
Xs= char( zeros(1,ns));
for k= 1:ns
    Xs(k)= 'X';
end

% Traverse the matrix to censor string str
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string.  Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

D= A;
B= lower(A);
s= lower(str);
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length
Xs= char( zeros(1,ns));
for k= 1:ns
    Xs(k)= 'X';
end

% Traverse the matrix to censor string str
for r= 1:nr
    for c= 1:nc-ns+1
        if strcmp( s , B(r, c:c+ns-1) )==1
            D(r, c:c+ns-1)= Xs;
        end
    end
end
Example: removing all occurrences of a character

- From a genome bank we get a sequence
  
  ATTG  CCG  TA  GCTA  CGTACGC  AACTGG
  
  AAATGGGC  CGTAT...

- First step is to “clean it up” by removing all the blanks. Write this function:

  ```matlab
  function s = removeChar(c, s)
  % Return string s with all occurrences
  % of character c removed
  ```
Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

```matlab
function s = removeChar_loop(c, s)
    % Return string s with all occurrences of character c removed.
```
Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

\[
\text{function } s = \text{removeChar\_loop}(c, s) \\
\% Return string } s \text{ with all occurrences of } \\
\% character } c \text{ removed.} \\
\]

\[
t = ''; \\
\text{for } k = 1:\text{length}(s) \\
\]

\[
\text{end} \\
s = t;
\]
Example: removing all occurrences of a character

Can solve this problem using iteration—check one character (one component of the vector) at a time

```matlab
function s = removeChar_loop(c, s)
% Return string s with all occurrences of % character c removed.

t= '';
for k= 1:length(s)
    if s(k)~=c
        t= [t s(k)];
    end
end
s= t;
```