- Today's topics
  - Characters and strings
  - Review of topics for Test 1

- Announcements/Reminders:
  - Assignment 1b due tonight 11:59pm
  - Test 1 in class on Thursday

# Characters & strings

- We have used strings already:
  - `n= input('Next number: ')`
  - `fprintf('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' | '3' | '2' | ' ' | 'r' | 'o' | 'c' | 'k' | 's' | '!' |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

- Can have 2-d array of characters as well

| 'C' | 'S' | '1' | '1' | '3' | '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 | 'C' | 'S' | '1' |
|---|---|---|---|

**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"

```
d = rand > .5
```

**d** is a scalar of the type **logical**. We call **d** a "boolean value"

# 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);     % x is 5
v(1)= 1;     % v is [1 0 5]
w= v(2:3);   % w is [0 5]
```

- : notation

```
v= 2:5;      % v is [2 3 4 5]
```

- Appending

```
v= [7 0 5];
v(4)= 2;     % 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);     % c is 'e'
s(1)= 'J';   % s is 'Jello'
t= s(2:4);   % t is 'ell'
```

- : notation

```
s= 'a':'g';  % s is 'abcdefg'
```

- Appending

```
s= 'duck';
s(5)= 's';    % 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**)

```
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
```

```
Look For   The   Spaces
```

See **caps.m**

```
function [str, nCaps] = caps(str)
% Post: Capitalize 1st letter of each word.
%   str= partially capitalized string
%   nCaps= no. of capital letters
% Pre: str= string with lower case letters and blanks only

nCaps= 0;
for k= 2:length(str)
   if (str(k-1)==' ' && isletter(str(k)))
      str(k)= upper(str(k));
      nCaps= nCaps + 1;
   end
end

if (isletter(str(1)))
   str(1)= upper(str(1));
   nCaps= nCaps + 1;
end
```

# ASCII characters

(American Standard Code for Information Interchange)

| *ascii code* | *Character* | *ascii code* | *Character* |
|---|---|---|---|
| : | : | : | : |
| : | : | : | : |
| 65 | 'A' | 48 | '0' |
| 66 | 'B' | 49 | '1' |
| 67 | 'C' | 50 | '2' |
| : | : | : | : |
| 90 | 'Z' | 57 | '9' |
| : | : | : | : |

# 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';  d2= 'Mar 9';
x1= d1(5);  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';  d2= 'Mar 29';
x1= d1(5:6);   x2= d2(5:6);
g= x2-x1;
```

A: the string '16'

B: the numeric value 16

C: Error in the subtraction operation

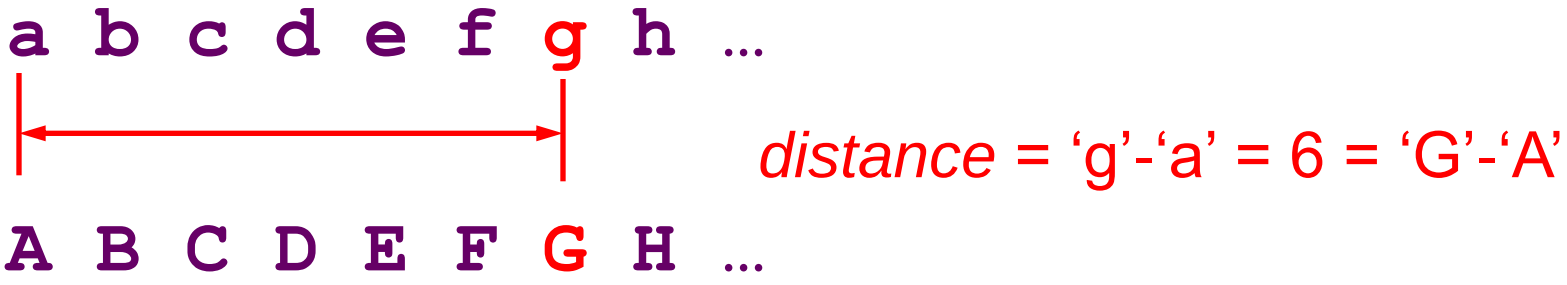
D: Error in assigning variables **x1**, **x2**

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.,




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' )

    % Find distance of cha from ‘a’
    offset= cha - 'a';

    % Go same distance 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' )

    % Find distance of cha from ‘a’
    offset= cha - 'a';

    % Go same distance from ‘A’
    up= char('A' + offset);
end
```

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

**Source: www.LookupTables.com**

# Review

- Scrip and Function
- If statement
- While loop
- For loop
- Array
- Matrix
- Plot, hold on, legend

# The `if` construct

```
if  boolean expression1
    statements to execute if expression1 is true
elseif boolean expression2
    statements to execute if expression1 is false
    but expression2 is true
:
else
    statements to execute if all previous conditions
    are false
end
```

Can have any number of elseif branches but at most one else branch

# Generating random numbers

- `rand(m,n)` gives an m-by-n matrix of random values, each in interval (0,1)
- Generate a random number in the range (a,b)
- Generate a random integer in the range [a,b]

# Generating random numbers

- `rand(m,n)` gives an m-by-n matrix of random values, each in interval (0,1)

- Generate a random number in the range (a,b)

  rand * (b-a) + a

- Generate a random integer in the range [a,b]

  floor (rand * (b-a+1) + a )

  ceil (rand*(b-a+1) + a-1 )

Simulation problem:

- Ann and Bob take turns flipping an unfair coin—twice as likely to be heads than tails
- In one round, each player flips once
- Ann gets 1 point if she gets heads; Bob gets 2 points if he gets tails
- Game ends after the round in which at least one player gets 10 points. Display the final scores.

Simulation problem: Stop : pA >= 10 or pB >= 10

- Ann and Bob take turns flipping an unfair coin—twice as likely to be heads than tails
- In one round, each player flips once
- Ann gets 1 point if she gets heads; Bob gets 2 points if he gets tails
- Game ends after the round in which at least one player gets 10 points. Display the final scores.


```
pA = 0 ; pB = 0;
while  pA < 10  &&  pB < 10
    % 1 round : A flips ; B flips ; scoring
    r = rand
    if  r < 2/3
        pA = pA + 1 ;
    end
    r = rand;
    if  r ____
        pB = pB + 2
    end
end
[ D ] play
```

# Common loop patterns

Do something *n* times

```
for k= 1:1:n
    % Do something

end
```

Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
    % Do something

    % Update loop variables

end
```

# `for` loop examples

```
for k= 2:0.5:3
     disp(k)
end
```

`k` takes on the values 2,2.5,3

Non-integer increment is OK

```
for k= 1:4
     disp(k)
end
```

`k` takes on the values 1,2,3,4

Default increment is 1

```
for k= 0:-2:-6
     disp(k)
end
```

`k` takes on the values 0,-2,-4,-6

"Increment" may be negative

```
for k= 0:-2:-7
     disp(k)
end
```

`k` takes on the values 0,-2,-4,-6

Colon expression specifies a *bound*

```
for k= 5:2:1
     disp(k)
end
```

The set of values for k is the empty set: the loop body won't execute

```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

**Not** a condition (boolean expression) that checks whether k<=6.

It is an expression that specifies values:

# Built-in functions for creating/manipulating arrays

- Creation
  - zeros, ones, rand
  - linspace
- Manipulation
  - length
  - size

# Built-in functions for creating/manipulating arrays

- Creation
  - zeros, ones, rand
  - linspace
- Manipulation
  - length
  - size

#rows

#cols

zeros(3, 2)

x = linspace( 3 , 10 , 8 )'

3 = starting, 10 = end, 8 = # of values

[nr, nc] = size(M)

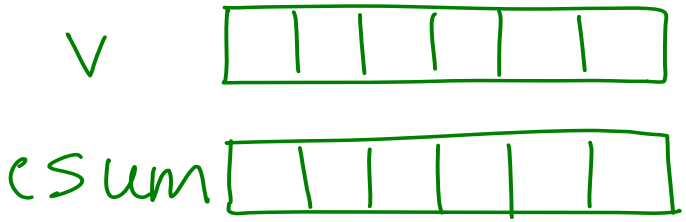
a = [1 2 3; ones(2,3)]

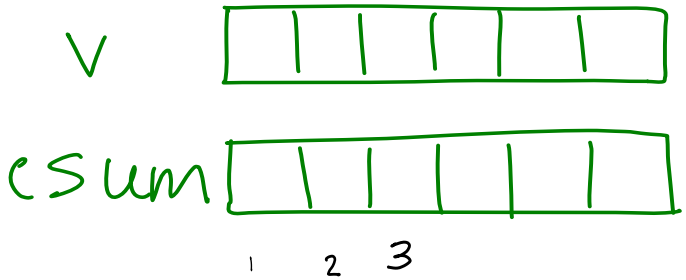
# Example

- Write a program fragment that calculates the cumulative sums of a given vector **`v`**.
- The cumulative sums should be stored in a vector of the same length as **`v`**.

1, 3, 5, 0 **`v`**

1, 4, 9, 9 cumulative sums of **`v`**


v
csum




$csum(k) = csum(k-1) + v(k)$

$csum(3) = v(1) + v(2) + v(3)$

$csum(4) = \underbrace{v(1) + v(2) + v(3)}_{csum(3)} + v(4)$

```
csum(1) = v(1);
for k = 2 : length(v)
    csum(k) = csum(k-1) + v(k);
end
```

Function header is the "contract" for how the function will be used (called)

You have this function:

```
function  [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r, theta) to
% Cartesian coordinates (x,y).  Theta in degrees.
…
```

Code to call the above function:

```
% Convert polar (r1,t1) to Cartesian (x1,y1)
r1= 1;   t1= 30;
[x1, y1]= polar2xy(r1, t1);
plot(x1, y1, 'b*')
…
```

Given this function:

```
function m = convertLength(ft,in)
% Convert length from feet (ft) and inches (in)
% to meters (m).

    . . .
```

How many proper calls to convertLength are shown below?

```
% Given f and n
d= convertLength(f,n);
d= convertLength(f*12+n);
d= convertLength(f+n/12);
x= min(convertLength(f,n), 1);
y= convertLength(pi*(f+n/12)^2);
```

A: 1 | B: 2 | C: 3 | D: 4 | E: 5 or 0

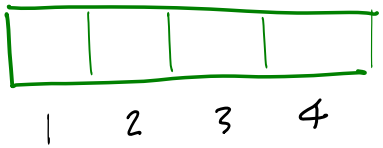
# Example

- Write a function **`evalPoly`** to evaluate an $n^{\text{th}}$ order polynomial of $x$ :

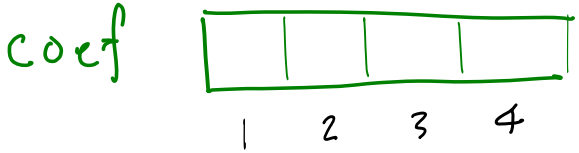
$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- Input parameter **`coef`** has length $n+1$, contains the coefficients of the polynomial
- **`coef(1)`** is the coefficient for the term $x^0$
- Input parameter **`x`**
- Return the value of the polynomial evaluated at $x$
- No Matlab predefined function other than **`length`**

coef




$$c_1 x^0 + c_2 x^1 + c_3 x^2 + c_4 x^3$$




$$c_1 x^0 + c_2 x^1 + c_3 x^2 + c_4 x^3$$

```
function  val = evalPoly(coef, x)
% val is polynomial evaluated at x
% coef is a vector where coef(1) is for term x^0
% xpow = 1;
  val = coef(1)
  for  k = 2:length(coef)
       val = val + coef(k)*x^(k-1);
       % xpow = xpow*x;
       % val = val + coef(k)*xpow;
end
```

# Other notes for the test (course)

- Read questions/instructions carefully
- Use Matlab syntax
- Do not use `break`, `continue`
- Do not use `randi`, instead use `rand`
- Many students make “index out-of-bounds” error

# Other notes for the test (course)

- Read questions/instructions carefully
- Use Matlab syntax
- Do not use **`break`**, **`continue`**
- Do not use **`randi`**, instead use **`rand`**
- Many students make "index out-of-bounds" error

```
% vector v
for k = 1 : length(v) - 1
    y(k) = v(k) + v(k+1)
end
```