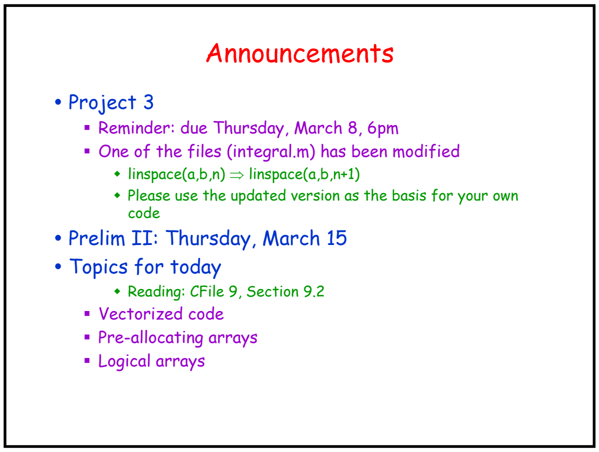
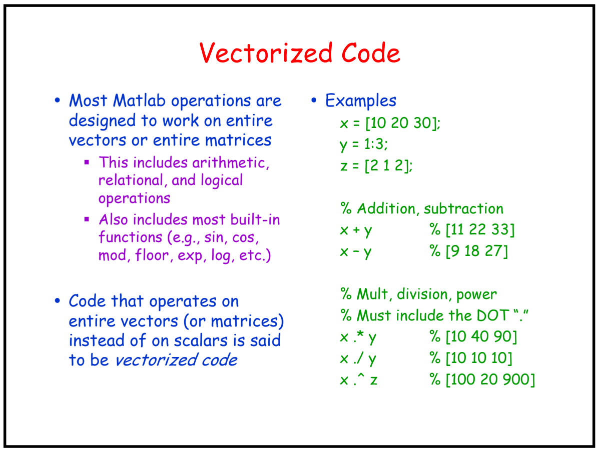


Announcements

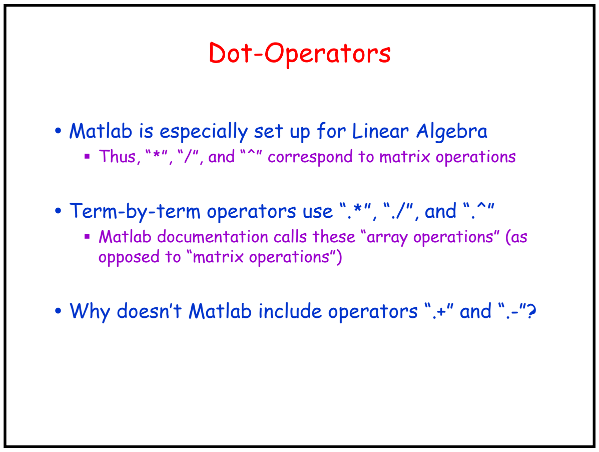
- Project 3
  - Reminder: due Thursday, March 8, 6pm
  - One of the files (integral.m) has been modified
    - linspace(a,b,n) ⇒ linspace(a,b,n+1)
    - Please use the updated version as the basis for your own code
- Prelim II: Thursday, March 15
- Topics for today
    - Reading: CFile 9, Section 9.2
  - Vectorized code
  - Pre-allocating arrays
  - Logical arrays



Vectorized Code

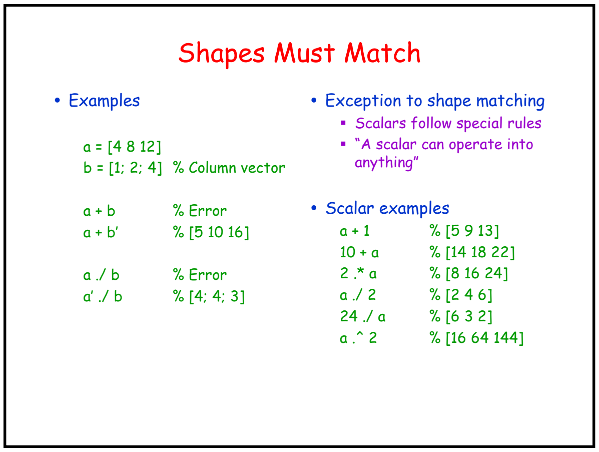
- Most Matlab operations are designed to work on entire vectors or entire matrices
  - This includes arithmetic, relational, and logical operations
  - Also includes most built-in functions (e.g., sin, cos, mod, floor, exp, log, etc.)
- Code that operates on entire vectors (or matrices) instead of on scalars is said to be *vectorized code*
- Examples

```
x = [10 20 30];
y = 1:3;
z = [2 1 2];

% Addition, subtraction
x + y          % [11 22 33]
x - y          % [9 18 27]

% Mult, division, power
% Must include the DOT "."
x .* y         % [10 40 90]
x ./ y         % [10 10 10]
x .^ z         % [100 20 900]
```



Dot-Operators

- Matlab is especially set up for Linear Algebra
  - Thus, "*", "/", and "^" correspond to matrix operations
- Term-by-term operators use ".*", "./", and ".^"
  - Matlab documentation calls these "array operations" (as opposed to "matrix operations")
- Why doesn't Matlab include operators ".+" and ".-"?



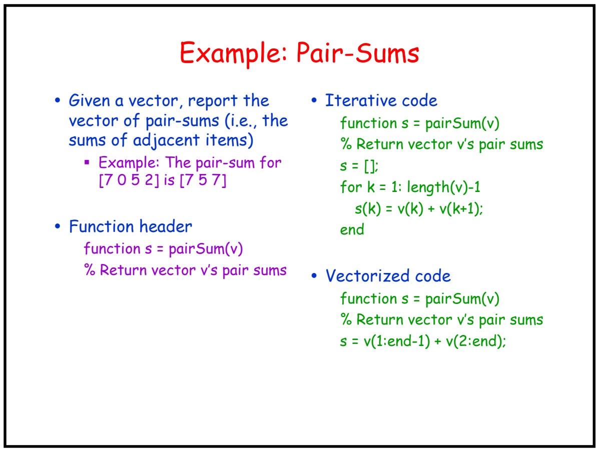
Shapes Must Match

- Examples

```
a = [4 8 12]
b = [1; 2; 4]  % Column vector

a + b          % Error
a + b'         % [5 10 16]

a ./ b         % Error
a' ./ b        % [4; 4; 3]
```

- Exception to shape matching
  - Scalars follow special rules
  - "A scalar can operate into anything"
- Scalar examples

```
a + 1          % [5 9 13]
10 + a         % [14 18 22]
2 .* a         % [8 16 24]
a ./ 2         % [2 4 6]
24 ./ a        % [6 3 2]
a .^ 2         % [16 64 144]
```



Example: Pair-Sums

- Given a vector, report the vector of pair-sums (i.e., the sums of adjacent items)
  - Example: The pair-sum for [7 0 5 2] is [7 5 7]
- Function header

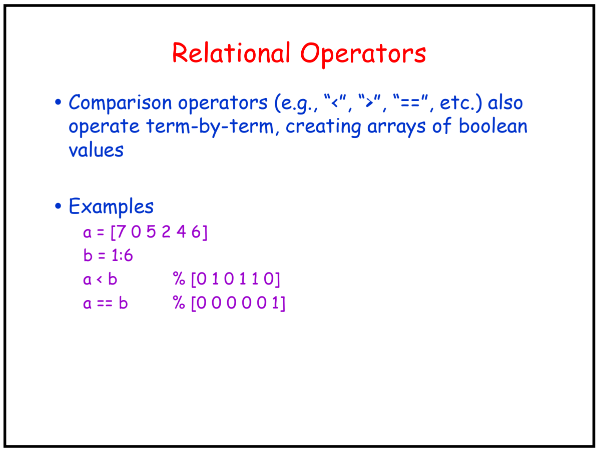
```
function s = pairSum(v)
% Return vector v's pair sums
```

- Iterative code

```
function s = pairSum(v)
% Return vector v's pair sums
s = [];
for k = 1: length(v)-1
   s(k) = v(k) + v(k+1);
end
```

- Vectorized code

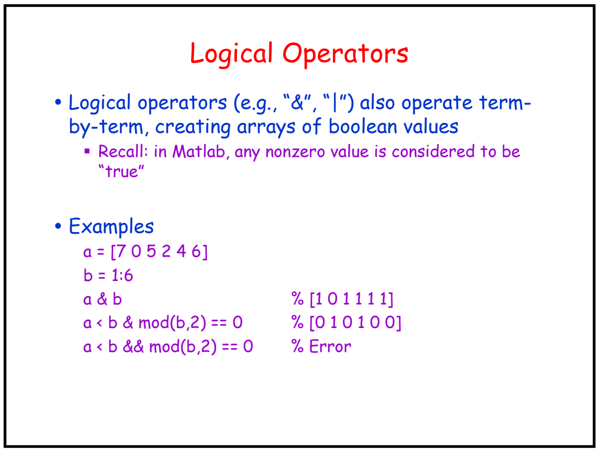
```
function s = pairSum(v)
% Return vector v's pair sums
s = v(1:end-1) + v(2:end);
```



Relational Operators

- Comparison operators (e.g., "<", ">", "==", etc.) also operate term-by-term, creating arrays of boolean values
- Examples

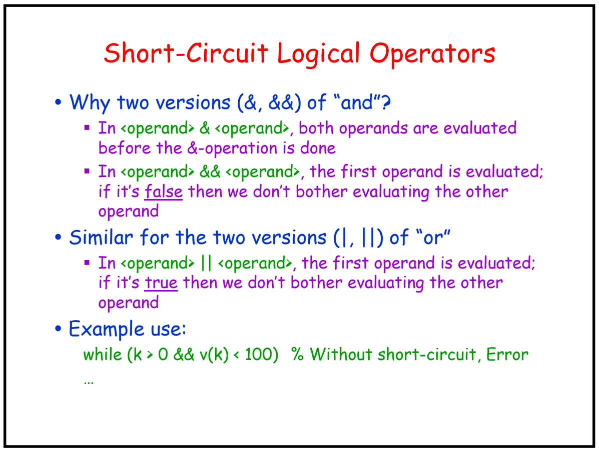
```
a = [7 0 5 2 4 6]
b = 1:6
a < b          % [0 1 0 1 1 0]
a == b         % [0 0 0 0 0 1]
```



Logical Operators

- Logical operators (e.g., “&”, “|”) also operate term-by-term, creating arrays of boolean values
  - Recall: in Matlab, any nonzero value is considered to be “true”
- Examples

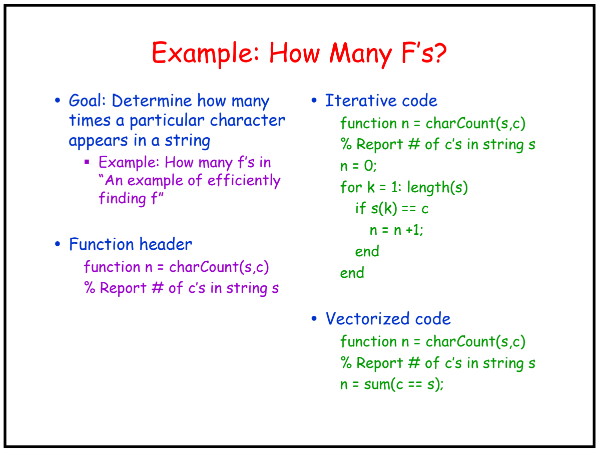
```
a = [7 0 5 2 4 6]
b = 1:6
a & b                    % [1 0 1 1 1 1]
a < b & mod(b,2) == 0    % [0 1 0 1 0 0]
a < b && mod(b,2) == 0   % Error
```



Short-Circuit Logical Operators

- Why two versions (&, &&) of “and”?
  - In <operand> & <operand>, both operands are evaluated before the &-operation is done
  - In <operand> && <operand>, the first operand is evaluated; if it’s false then we don’t bother evaluating the other operand
- Similar for the two versions (|, ||) of “or”
  - In <operand> || <operand>, the first operand is evaluated; if it’s true then we don’t bother evaluating the other operand
- Example use:

```
while (k > 0 && v(k) < 100)   % Without short-circuit, Error
…
```



Example: How Many F’s?

- Goal: Determine how many times a particular character appears in a string
  - Example: How many f’s in “An example of efficiently finding f”
- Function header

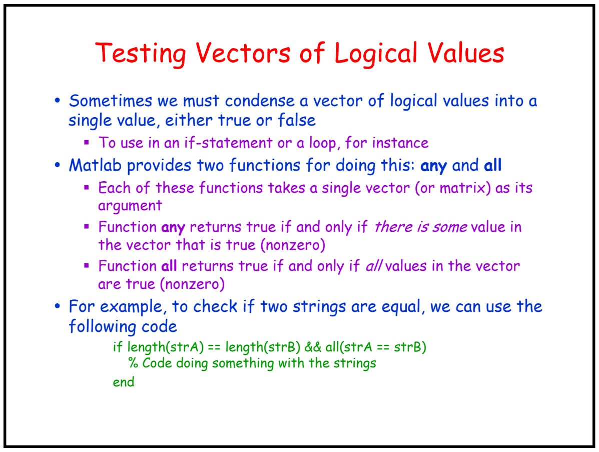
```
function n = charCount(s,c)
% Report # of c’s in string s
```

- Iterative code

```
function n = charCount(s,c)
% Report # of c’s in string s
n = 0;
for k = 1: length(s)
   if s(k) == c
      n = n +1;
   end
end
```

- Vectorized code

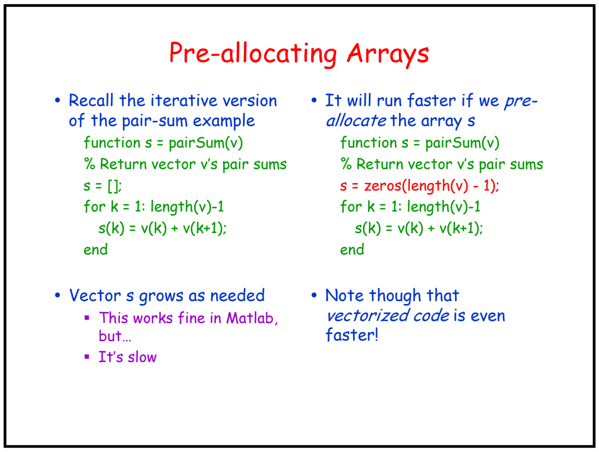
```
function n = charCount(s,c)
% Report # of c’s in string s
n = sum(c == s);
```



Testing Vectors of Logical Values

- Sometimes we must condense a vector of logical values into a single value, either true or false
  - To use in an if-statement or a loop, for instance
- Matlab provides two functions for doing this: **any** and **all**
  - Each of these functions takes a single vector (or matrix) as its argument
  - Function **any** returns true if and only if *there is some* value in the vector that is true (nonzero)
  - Function **all** returns true if and only if *all* values in the vector are true (nonzero)
- For example, to check if two strings are equal, we can use the following code

```
if length(strA) == length(strB) && all(strA == strB)
   % Code doing something with the strings
end
```



Pre-allocating Arrays

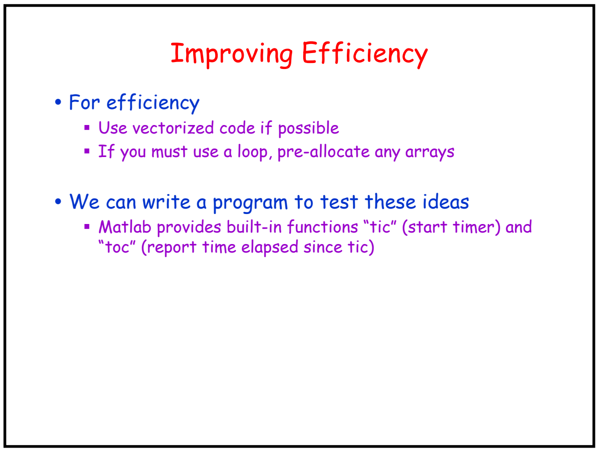
- Recall the iterative version of the pair-sum example

```
function s = pairSum(v)
% Return vector v’s pair sums
s = [];
for k = 1: length(v)-1
   s(k) = v(k) + v(k+1);
end
```

- Vector s grows as needed
  - This works fine in Matlab, but…
  - It’s slow
- It will run faster if we *pre-allocate* the array s

```
function s = pairSum(v)
% Return vector v’s pair sums
s = zeros(length(v) - 1);
for k = 1: length(v)-1
   s(k) = v(k) + v(k+1);
end
```

- Note though that *vectorized code* is even faster!



Improving Efficiency

- For efficiency
  - Use vectorized code if possible
  - If you must use a loop, pre-allocate any arrays
- We can write a program to test these ideas
  - Matlab provides built-in functions “tic” (start timer) and “toc” (report time elapsed since tic)