CS 1112 Prelim 2 Review

Prelim 2 Topics

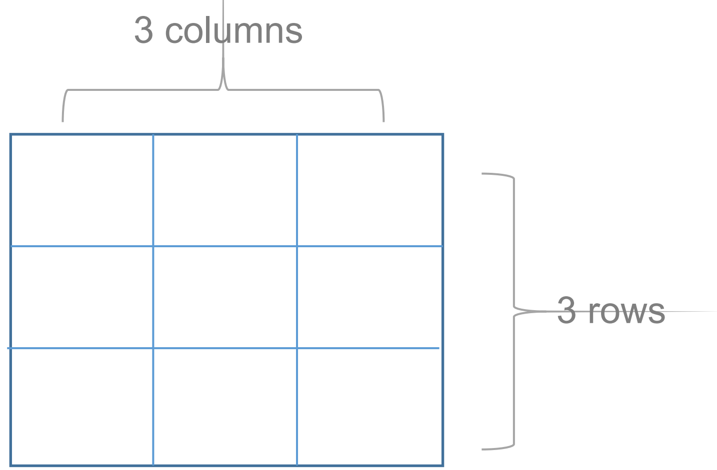
- 2-dimensional array (matrix)
  - 2-d array traverse with nested loops
  - Partial matrix traverse, e.g., triangular
- 3-dimensional array (e.g., color image data)
- character and char arrays (1-d, 2-d) (We do not use type String. No ASCII arithmetic.)
- the type uint8 (doing arithmetic with uint8 variables)
- Nested loops pattern for all non-duplicating combinations in a set (e.g., n-choose-k type situations)
- Linear search
- vectorized code (as discussed in lecture)

2-D and 3-D arrays

2-D array (also known as a matrix)

- A collection of data (e.g. numbers, or characters, but not both) stored in rows and columns
- Items are referenced using 2 numbers (row # and column #)

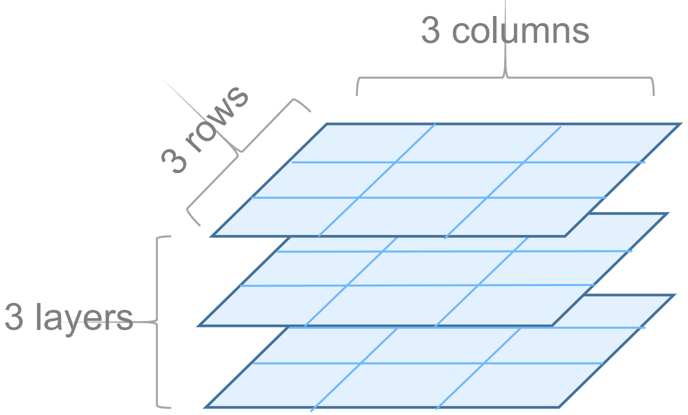
Example: 3x3 matrix



3-D array

- A series of 2-D arrays layered on each other
- Items are referenced by 3 numbers (row #, column #, layer #)

Example: 3x3x3 3-D array



2-D and 3-D arrays

Initialize a 2-D array, A

```
% Create 3x3 matrix of zeros
A = zeros(3,3);
```

Find size of 2-D array A

```
[nr,nc] = size(A);  % nr=3,nc=3
```

Pattern for *traversing* a 2-D array

```
for r = 1:nr
    for c = 1:nc
        % Do something to A(r,c)
    end
end
```

Initialize a 3-D array, B

```
% Create 3x3x3 matrix of zeros
B = zeros(3,3,3);
```

Find size of 3-D array B

```
[nr,nc,np] = size(B);  % nr=3,nc=3,np=3
```

Note: this would also work for a 2-D array; np would just be 1.

Pattern for *traversing* a 3-D array

```
for r = 1:nr
    for c = 1:nc
        for p = 1:np
            % Do something to A(r,c,p)
        end
    end
end
```

Remember the image of a 3-D array as a *stack* of 2-D arrays: this pattern works by examining *down* the layers, then *across* the columns, then *down* the rows

Matrix transposition

Given a matrix M, where

M = 1 2

       3 4

We'd like to create a matrix T, which stores the *transpose* of M.

T = 1 3

      2 4

How do we write code to do this?

Hints: the *rows* of M have become the *columns* of T. M(1, 2) is the same as T(2, 1).

Solution:

- We know that we need a *nested for-loop* to go through all elements of M.
- Relation between M and T: reverse positions in M to get positions in T

```
[nr,nc] = size(M);
for r = 1:nr
    for c = 1:nc
        T(c,r) = M(r,c);
    end
end
```

Mirror image of a **2-D array**

Given a matrix M, where

M = 1 2 3 4
    5 6 7 8

We'd like to create a matrix T, which is the *mirror image of M*.

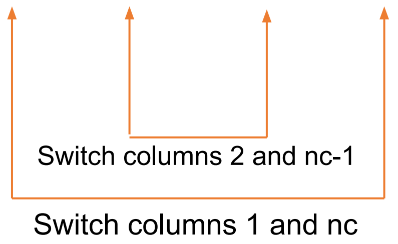
T = 4 3 2 1
    8 6 7 5

How do we write code to do this?

Hint: Reverse the order of the columns of M to get T.

How do we re-order the columns?

M = 1 2 3 4
    5 6 7 8



Relationship between a pair of columns that must be reversed:

Column c should be replaced by column (nc – c) + 1.

Mirror image of a **2-D array**

Given a matrix M, where

M = 1 2 3 4

5 6 7 8

We'd like to create a matrix T, which is the *mirror image of M*.

T = 4 3 2 1

8 6 7 5

How do we write code to do this?

Hint: Reverse the order of the columns of M to get T.

Non-vectorized solution:

Column c should be exchanged with column (nc – c) + 1.

```
[nr,nc] = size(M);
for r = 1:nr
    for c = 1:nc
        T(r,c) = M(r,(nc-c)+1));
    end
end
```

Mirror image of a **2-D array**

Given a matrix M, where

M = 1 2 3 4
5 6 7 8

We'd like to create a matrix T, which is the *mirror image of M*.

T = 4 3 2 1
8 6 7 5

How do we write code to do this?

Hint: Reverse the order of the columns of M to get T.

Vectorized solution:

Column c should be exchanged with column (nc – c) + 1.

```
[nr,nc] = size(M);
for c = 1:nc
    T(:,c) = M(:,(nc-c)+1));
end
```

The vectorized solution exchanges entire columns instead of individual elements.

Mirror image of a **3-D array**

Non-vectorized solution:

Column c should be exchanged with column (nc – c) + 1.

```
[nr, nc, np] = size(M);
for p = 1:np
   for r = 1:nr
      for c = 1:nc
         T(r,c,p) = M(r,(nc-c)+1,p);
      end
   end
end
```

Vectorized solution:

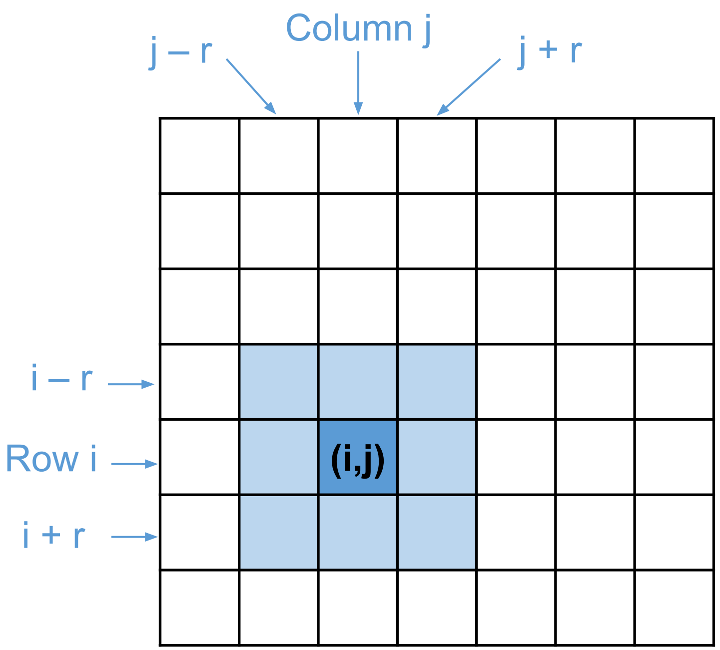
Column c should be exchanged with column (nc – c) + 1.

```
[nr, nc, np] = size(M);
for c = 1:nc
    T(:,c,:) = M(:,nc-c+1,:);
end
```

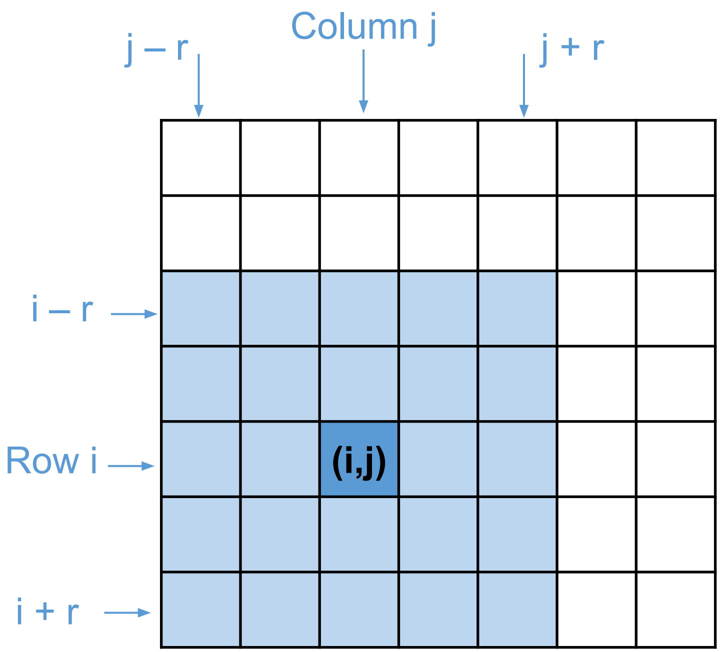
2-D and 3-D arrays: **Sub-arrays**

The **neighborhood** of a particular cell in an array is the set of cells that *surround* that one cell within a particular radius.

Neighborhood radius: r = 1



Neighborhood radius: r = 2



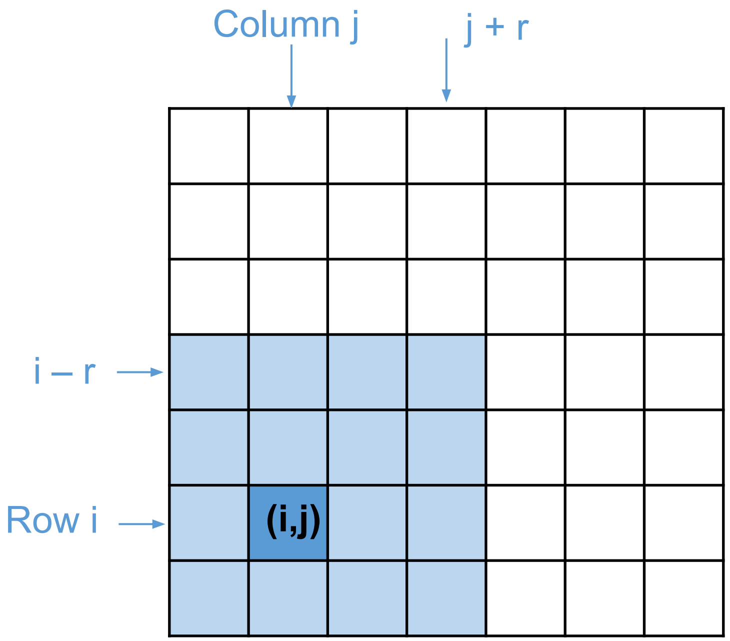
How do we *extract the sub-array* that corresponds to the highlighted neighborhood?

We need to select all rows between row i-r and row i+r, and all columns between columns j-r and j+r.

2-D and 3-D arrays: **Sub-arrays**

From previous slide: We need to select all rows between row i-r and row i+r, and all columns between columns j-r and j+r. What if the neighborhood goes out of bounds?

Neighborhood radius: r = 2



How do we **generalize** this approach so that it works when i-r or i+r or j-r or j+r are out of bounds?

Solution:

```
[m,n] = size(M);
iMin = max(1,i-r)
iMax = min(m,i+r)
jMin = max(1,j-r)
jMax = min(n,j+r)

% Now extract submatrix: the neighborhood
C = M(iMin:iMax, jMin:jMax)
```

2-D and 3-D arrays: **Resizing an array**

Interpolating on a matrix means:

- Inserting new rows/columns between existing rows/columns
- The new rows/columns are calculated from a pair of originally adjacent rows/columns

Example of expanding a matrix (original cells in gray):

1	2	3
4	5	6

→

1	1.5	2	2.5	3
2.5	3	3.5	4	4.5
4	4.5	5	5.5	6

2-D and 3-D arrays: **Resizing an array**

If the interpolation involves creating additional rows *and* columns, it is easier to work with one dimension at a time, i.e. columns first and then rows.

Example of expanding a matrix (original cells in gray):

1	2	3
4	5	6

Size: nr x nc

1	1.5	2	2.5	3
4	4.5	5	5.5	6

Size: nr x 2*nc-1

1	1.5	2	2.5	3
2.5	3	3.5	4	4.5
4	4.5	5	5.5	6

Size: 2*nr-1 x 2*nc-1

2-D and 3-D arrays: **Resizing an array**

1	2	3
4	5	6

Size: nr x nc

1	1.5	2	2.5	3
4	4.5	5	5.5	6

Size: nr x 2*nc-1

1	1.5	2	2.5	3
2.5	3	3.5	4	4.5
4	4.5	5	5.5	6

Size: 2*nr-1 x 2*nc-1

Step 1: interpolate on columns

```
for r= 1:nr
    for c= 1:nc-1
        % copy original data
        wideM(r,c*2-1)= M(r,c);
        % calculate average of adjacent columns
        wideM(r,c*2)= M(r,c)/2 + M(r,c+1)/2;
    end
    wideM(r,nc*2-1)= M(r,nc);
end
```

Step 2: interpolate on rows

```
for c= 1:nc*2-1
    for r= 1:nr-1
        % copy data from intermediate matrix
        newM(r*2-1,c)= wideM(r,c);
        % calculate average of adjacent rows
        newM(r*2,c)= wideM(r,c)/2 + wideM(r+1,c)/2;
    end
    newM(nr*2-1,c)= wideM(nr,c);
end
```

2-D and 3-D arrays: **Resizing an array**

How might we **reduce** a 2-D array by averaging each group of 4 cells?

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

3.5	5.5
11.5	13.5

Solution:

- Size of new matrix: ½ the size of the original one
- We can *pick one cell* in each group (for example, pick the cells containing the values 6, 8, 14, 16), and calculate the average of it and its *neighbors*.

```
[nr,nc] = size(M);
N = zeros(nr/2, nc/2);
for r = 2:2:nr
    for c = 2:2:nc
        avg = (M(r-1,c)+M(r-1,c-1)+M(r,c-1)+M(r,c))/4;
        N(r/2,c/2) = avg;
    end
end
```

Spring 2015 Prelim: **Question 1b**

```
function min = minAlongEvenCol(M)
%Find the minimum element contained in an even column in matrix M
```

Spring 2015 Prelim: **Question 1b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Go through a 2-D array	
2		
3		

Spring 2015 Prelim: **Question 1b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Go through a 2-D array	Nested for-loop
2		
3		

Spring 2015 Prelim: **Question 1b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Go through a 2-D array	Nested for-loop
2	Look at items only in the even columns	
3		

Spring 2015 Prelim: **Question 1b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Go through a 2-D array	Nested for-loop
2	Look at items only in the even columns	Inner for-loop should iterate as: c = 2:2:nc
3		

Spring 2015 Prelim: **Question 1b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Go through a 2-D array	Nested for-loop
2	Look at items only in the even columns	Inner for-loop should iterate as: c = 2:2:nc
3	Find the minimum of the set of cells being examined	

Spring 2015 Prelim: **Question 1b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Go through a 2-D array	Nested for-loop
2	Look at items only in the even columns	Inner for-loop should iterate as: c = 2:2:nc
3	Find the minimum of the set of cells being examined	"Best-so-far" pattern: 1. initialize a variable to hold minSoFar 2. update minSoFar as we examine each cell

Spring 2015 Prelim: **Question 1b**

Translating what we need to do into code:

```
[nr, nc]= size(M);

for r=
    for c=




    end
end
```

Connection to previous slide:
Red: nested for-loop
Green: only examine items in even columns
Blue: finding minimum so far

Spring 2015 Prelim: **Question 1b**

Translating what we need to do into code:

```
[nr, nc]= size(M);

for r= 1:nr
    for c= 2:2:nc % better than checking if each column is odd or even



    end
end
```

Connection to previous slide:
Red: nested for-loop
Green: only examine items in even columns
Blue: finding minimum so far

Spring 2015 Prelim: **Question 1b**

Translating what we need to do into code:

```
[nr, nc]= size(M);
minSoFar = inf;
for r= 1:nr
    for c= 2:2:nc % better than checking if each column is odd or even
        if M(r,c) < minSoFar
            minSoFar = M(r,c);
        end
    end
end
```

Connection to previous slide:
Red: nested for-loop
Green: only examine items in even columns
Blue: finding minimum so far

Working with images: **uint8 type**

What is uint8?

An integer that can hold values between 0 and 255 ($2^8 – 1 = 255$). Images can be represented with numbers in this range.

Working with images: **uint8 type**

What is uint8?

An integer that can hold values between 0 and 255 ($2^8 - 1 = 255$). Images can be represented with numbers in this range.

How to convert numeric data into uint8:

Given an array x of (regular) numbers,

```
y = uint8(x);
```

Working with images: **uint8 type**

What is uint8?

An integer that can hold values between 0 and 255 ($2^8 - 1 = 255$). Images can be represented with numbers in this range.

How to convert numeric data into uint8:

Given an array x of (regular) numbers,

```
y = uint8(x);
```

Note about overflow:

If you need to perform an arithmetic operation on uint8 numbers, e.g. averaging, be careful that the numbers won't overflow, i.e. exceed 255.

```
goodAverage = x(1)/3 + x(2)/3 + x(3)/3;    % Do this
badAverage = (x(1) + x(2) + x(3))/3;   % Don't do this
```

Fall 2015 Prelim: **Question 5**

Question 5: (20 points)

Consider the radius s neighborhood of a pixel on one layer of a 3-d uint8 array Q (corresponding to a color JPEG image). Assuming that s is much smaller than the number of rows and number of columns of Q, an interior pixel of Q would have a "full-size" neighborhood while a pixel on an edge would have a reduced-size neighborhood. For example, if $s = 2$, then an interior pixel has a full-size neighborhood of 5-by-5 pixels while any of the four corner pixels has a 3-by-3—not full-size—neighborhood.

Implement the following function as specified:

```
function W = modifyColor(Q,s)
% Q is a 3-d array of type uint8 corresponding to a color jpeg image.
% W is a 3-d array of type uint8 the same size as Q.  W corresponds to a modified
%   color version of Q.
% s is the radius of the neighborhood, s is a positive integer smaller than the
%   number of rows and columns in Q.
% For the pixels in Q that each has a full-size radius s neighborhood:
%   - At each pixel (r,c), the red intensity value in W is the maximum of the red
%   intensity values in the neighborhood of pixel (r,c) in Q.
%   - Similarly, the green intensity in pixel (r,c) of W is the max of the green
%   intensities in the neighborhood in Q; the blue intensity in pixel (r,c)
%   of W is the max of the blue intensities in the neighborhood in Q.
% For the pixels in Q at or near the edges where the radius s neighborhood is not
% full-size:
%   - At each pixel (r,c) of W, calculate the gray value (average of red, green, and
%   blue) of that pixel (not the neighborhood) and assign that gray value to all
%   layers at that pixel.
% Only these functions are allowed: size, zeros, max, min, uint8.
% Note that function double is NOT allowed.
```

Fall 2015 Prelim: **Question 5**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Must loop through each pixel of the image before we can determine what is done with the colors	
2		
3		

Fall 2015 Prelim: **Question 5**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Must loop through each pixel of the image before we can determine what is done with the colors	Outer double for-loop for the pixels
2		
3		

Fall 2015 Prelim: **Question 5**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Must loop through each pixel of the image before we can determine what is done with the colors	Outer double for-loop for the pixels
2	Detect if a given pixel has a full-size neighborhood of radius s	
3		

Fall 2015 Prelim: **Question 5**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Must loop through each pixel of the image before we can determine what is done with the colors	Outer double for-loop for the pixels
2	Detect if a given pixel has a full-size neighborhood of radius s	Full sized if s+1 <= c <= nc-s and s+1 <= r <= nr-s
3		

Fall 2015 Prelim: **Question 5**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Must loop through each pixel of the image before we can determine what is done with the colors	Outer double for-loop for the pixels
2	Detect if a given pixel has a full-size neighborhood of radius s	Full sized if s+1 <= c <= nc-s and s+1 <= r <= nr-s
3	For full sized neighborhood, take the max intensity within the neighborhood for each color individually. Otherwise, compute the gray value at the pixel and assign to all layers	

Fall 2015 Prelim: **Question 5**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Must loop through each pixel of the image before we can determine what is done with the colors	Outer double for-loop for the pixels
2	Detect if a given pixel has a full-size neighborhood of radius s	Full sized if s+1 <= c <= nc-s and s+1 <= r <= nr-s
3	For full sized neighborhood, take the max intensity within the neighborhood for each color individually. Otherwise, compute the gray value at the pixel and assign to all layers	Full sized -> Correct use of max/min on the neighborhood Otherwise -> Correct averaging of uint8 values

Fall 2015 Prelim: **Question 5**

Translating what we need to do into code:

```
Q = P;
[nr, nc, nl]= size(P);
for r= 1:nr
    for c= 1:nc



    end
end
```

Connection to previous slide:
Red: Outer double for-loop for the pixels
Green: Determine if neighborhood full sized
Blue: Compute colors

Fall 2015 Prelim: **Question 5**

Translating what we need to do into code:

```
Q = P;
[nr, nc, nl]= size(P);
for r= 1:nr
    for c= 1:nc
        if c >= s+1 && c <= nc-s && r >= s+1 && r <= nr-s



        else

        end
    end
end
```

Connection to previous slide:
Red: Outer double for-loop for the pixels
Green: Determine if neighborhood full sized
Blue: Compute colors

Fall 2015 Prelim: **Question 5**

Translating what we need to do into code:

```
Q = P;
[nr, nc, nl]= size(P);
for r= 1:nr
    for c= 1:nc
        if c >= s+1 && c <= nc-s && r >= s+1 && r <= nr-s
            nbd = P( r-s:r+s, c-s:c+s, : );
            Q( r, c, 1) = max( max( nbd( : , : , 1 ) ) );
            Q( r, c, 2) = max( max( nbd( : , : , 2 ) ) );
            Q( r, c, 3) = max( max( nbd( : , : , 3 ) ) );
        else
            Q( r, c, : ) = P(r , c, 1) / 3 + P(r , c, 2) / 3 + P(r , c, 3) / 3
        end
    end
end
```

Connection to previous slide:
Red: Outer double for-loop for the pixels
Green: Determine if neighborhood full sized
Blue: Compute colors

Strings and characters

- A string is a 1-D array (vector) of characters, 1 character per cell
- A 2-D array with *n* rows could store *n* strings, with one string per row, as long as each string has the same number of characters (columns).

Strings and characters

- A string is a 1-D array (vector) of characters, 1 character per cell
- A 2-D array with *n* rows could store *n* strings, with one string per row, as long as each string has the same number of characters (columns).

‘m’	‘a’	‘t’	‘l’	‘a’	‘b’

1-D array of characters (1 row, 6 columns)

```
A = ‘matlab’;
A = ['m','a','t','l','a','b'];
disp(A(3))    % prints ‘t’
```

Strings and characters

- A string is a 1-D array (vector) of characters, 1 character per cell
- A 2-D array with *n* rows could store *n* strings, with one string per row, as long as each string has the same number of characters (columns).

‘m’	‘a’	‘t’	‘l’	‘a’	‘b’

1-D array of characters (1 row, 6 columns)

A = ‘matlab’;

A = ['m','a','t','l','a','b'];

disp(A(3))   % prints ‘t’

‘m’	‘a’	‘t’	‘l’	‘a’	‘b’
‘i’	‘s’	‘ ’	‘ ’	‘ ’	‘ ’
‘f’	‘u’	‘n’	‘ ’	‘ ’	‘ ’

2-D array of characters (3 rows, 6 columns)

Note that empty spaces have to be appended onto the shorter words so that the strings on each row have the same length.

B = [‘matlab’; ‘is   ’; ‘fun   ’];

disp(B(1,:)) % prints ‘matlab’

Strings and characters: **Useful functions**

- strcmp(str1, str2)
  Returns 1 if str1 has all the same characters as str2, and 0 if not. Case-sensitive.

```
strcmp('matlab', 'mAtlab')  % 0
```

Strings and characters: **Useful functions**

- strcmp(str1, str2)
  Returns 1 if str1 has all the same characters as str2, and 0 if not. Case-sensitive.

```
strcmp('matlab', 'mAtlab')  % 0
```

- str2double(str1)
  If str1 is a string containing a number, the function returns that number as a numerical value. Returns NaN if str1 is something other than a number.

```
str2double('20') + 2     % 22
str2double('hi')      % NaN
```

Strings and characters: **Manipulating strings**

Given a string s, where

s = ‘hello’

We’d like to reverse the string to obtain

r = ‘olleh’

How to we write code to do this?

Hint: This is very similar to when we produced the mirror image of a matrix.

Strings and characters: **Manipulating strings**

Given a string s, where

s = 'hello'

We'd like to reverse the string to obtain

r = 'olleh'

How to we write code to do this?

Hint: This is very similar to when we produced the mirror image of a matrix.

Solution:

Character c should be exchanged with character (nc – c) + 1.

Strings and characters: **Manipulating strings**

Given a string s, where

s = 'hello'

We'd like to reverse the string to obtain

r = 'olleh'

How to we write code to do this?

Hint: This is very similar to when we produced the mirror image of a matrix.

Solution:

Character c should be exchanged with character (nc – c) + 1.

```
n = length(s);
for k = 1:n
   r(k) = s(n-k+1);
end
```

Strings and characters: **Manipulating strings**

Given a string s and a character c, we'd like to display the number of times c occurs in s.

Examples:

s = 'mathematics', c = 'a' → display 2.

How to we write code to do this?

Strings and characters: **Manipulating strings**

Given a string s and a character c, we'd like to display the number of times c occurs in s.

Examples:

s = 'mathematics', c = 'a' → display 2.

How to we write code to do this?

Solution:

```
for k = 1:length(s)



end
```

Strings and characters: **Manipulating strings**

Given a string s and a character c, we'd like to display the number of times c occurs in s.

Examples:

s = 'mathematics', c = 'a' → display 2.

How to we write code to do this?

Solution:

```
for k = 1:length(s)
   if strcmp(s(k), c)

   end
end
```

Strings and characters: **Manipulating strings**

Given a string s and a character c, we'd like to display the number of times c occurs in s.

Examples:

s = 'mathematics', c = 'a' → display 2.

How to we write code to do this?

Solution:

```
count = 0;
for k = 1:length(s)
   if strcmp(s(k), c)
      count = count+1;
   end
end
disp(count)
```

Strings and characters: **Manipulating strings**

Given a string s and a smaller string str, we'd like to display the number of times str occurs in s.

Examples:

s = 'mathematics', str = 'at' → display 2.

How to we write code to do this?

Solution:

```
count = 0;
for k = 1:length(s)-length(str)+1
   if strcmp(s(k : k+length(str)-1), str)
      count = count+1;
   end
end
disp(count)
```

Fall 2016 Prelim: **Question 5a**

Question 5a: (10 points)

Implement the following function as specified:

```
function v = getIndices(str, sep)
% str and sep are each a string and str is longer than sep. sep is the
%   separator string, i.e., the delimiting string.
% v is the vector of the indices where the separator begins in str.
%   Therefore the length of v is the number of times that sep occurs in str.
% Examples: If str is 'Hi!?Ann!?Bob' and sep is '!?' then v is [3 8].
%           If str is 'Hi!Ann!Bob' and sep is '!' then v is [3 7].
%           If str is 'Hi!Ann!Bob' and sep is '?' then v is [].
% Assume that the characters in sep are used only as the delimiter and not
%   in the separated substrings.  Assume that separators are always correctly
%   placed--never incomplete and never side-by-side not separating anything.
% DO NOT USE any built-in functions other than length and strcmp.
```

Fall 2016 Prelim: **Question 5a**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all substrings of str with the same length as sep	
2		
3		

Fall 2016 Prelim: **Question 5a**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all substrings of str with the same length as sep	A loop on the index of the leftmost character of the substring **for left=1:length(str)-length(sep)+1** (why?)
2		
3		

Fall 2016 Prelim: **Question 5a**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all substrings of str with the same length as sep	A loop on the index of the leftmost character of the substring **for left=1:length(str)-length(sep)+1** (why?)
2	For a given left index, extract substring with same length as sep and compare with sep	
3		

Fall 2016 Prelim: **Question 5a**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all substrings of str with the same length as sep	A loop on the index of the leftmost character of the substring **for left=1:length(str)-length(sep)+1** (why?)
2	For a given left index, extract substring with same length as sep and compare with sep	substr = str(left:left+length(sep)-1); (why?) strcmp(substr,sep);
3		

Fall 2016 Prelim: **Question 5a**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all substrings of str with the same length as sep	A loop on the index of the leftmost character of the substring **for left=1:length(str)-length(sep)+1** (why?)
2	For a given left index, extract substring with same length as sep and compare with sep	substr = str(left:left+length(sep)-1); (why?) strcmp(substr,sep);
3	Record the left index to the output variable whenever the substring matches sep	

Fall 2016 Prelim: **Question 5a**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all substrings of str with the same length as sep	A loop on the index of the leftmost character of the substring **for left=1:length(str)-length(sep)+1** (why?)
2	For a given left index, extract substring with same length as sep and compare with sep	substr = str(left:left+length(sep)-1); (why?) strcmp(substr,sep);
3	Record the left index to the output variable whenever the substring matches sep	Append the left to v each time the above condition is true

Fall 2016 Prelim: **Question 5a**

Translating what we need to do into code:

```
for left = 1 : length(str) – length(sep) + 1



end
```

Connection to previous slide:
Red: for-loop on left index
Green: use strcmp on substr
Blue: append left index to v

Fall 2016 Prelim: **Question 5a**

Translating what we need to do into code:

```
for left = 1 : length(str) – length(sep) + 1
    substr = str(left : left + length(sep) – 1);
    if strcmp( substr, sep ) == 1

    end
end
```

Connection to previous slide:
Red: for-loop on left index
Green: use strcmp on substr
Blue: append left index to v

Fall 2016 Prelim: **Question 5a**

Translating what we need to do into code:

```
v= [];
for left = 1 : length(str) – length(sep) + 1
    substr = str(left : left + length(sep) – 1);
    if strcmp( substr, sep ) == 1
        v = [v, left];
    end
end
```

Connection to previous slide:
Red: for-loop on left index
Green: use strcmp on substr
Blue: append left index to v

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

- **uint8:** integers ranging from 0 to 255
  Array of uint8 numbers: y = uint8(x);

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

- **uint8:** integers ranging from 0 to 255
  Array of uint8 numbers: y = uint8(x);

- **char:** standard characters, including letters, digits, symbols. Multiple chars together form a *string*, but a string is *not* a type – it is just an array of characters.
  Array of characters: s = ‘CS1112’; s = [‘c’, ’s’, ’1’, ’1’, ’1’, ’2’];

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

- **uint8:** integers ranging from 0 to 255
  Array of uint8 numbers: y = uint8(x);

- **char:** standard characters, including letters, digits, symbols. Multiple chars together form a *string*, but a string is *not* a type – it is just an array of characters.
  Array of characters: s = ‘CS1112’; s = [‘c’, ’s’, ’1’, ’1’, ’1’, ’2’];

- **logical:** also known as a boolean. Can be true/false or 0/1.
  Creating a logical: z = rand > 0.5

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

- **uint8:** integers ranging from 0 to 255
  Array of uint8 numbers: y = uint8(x);

- **char:** standard characters, including letters, digits, symbols. Multiple chars together form a *string*, but a string is *not* a type – it is just an array of characters.
  Array of characters: s = 'CS1112'; s = ['c', 's', '1', '1', '1', '2'];

- **logical:** also known as a boolean. Can be true/false or 0/1.
  Creating a logical: z = rand > 0.5

An array can only hold values of one type. A *cell array* is a special kind of array that can hold data of different types. Yay!

Cell arrays

Arrays (e.g. vectors, matrices, 3-D arrays, etc.)

- Can hold *one scalar* value in each of its components, e.g. one double, one char, one uint8.
- Data of all components must be the same type

Cell arrays

Arrays (e.g. vectors, matrices, 3-D arrays, etc.)

- Can hold *one scalar* value in each of its components, e.g. one double, one char, one uint8.
- Data of all components must be the same type

Cell arrays

- Each cell can store something "larger" than a scalar (but doesn't have to). Can store a vector in a single component, or a matrix, or a string, etc.
- Each cell can store something of a different type

Cell arrays

Initialize a cell array with cell(…) function	c = cell(1,3); % Cell array with 1 row, 3 columns
Obtain number of rows and columns	[nr, nc] = size(c); % Same as for other arrays
Put items (strings, in this case) into the cell array	c = {'matlab', 'is', 'fun'}; % Commas optional
Display first item (string in this example)	disp(c{1}) % Note the use of curly braces
Display first two items (vectorized)	disp(c(1:2)) % Note the use of parentheses
Display first three letters of first string	disp(c{1}(1:3)) % Note the use of curly braces *and* parentheses
Concatenate the strings (produces ‘matlab is fun’)	s = [c{1} ‘ ’ c{2} ‘ ’ c{3}] % Note the use of square brackets to create a string

Fall 2016 Prelim: **Question 5b**

Question 5b: (25 points)

Assume that function `getIndices` of Question 5a has been correctly implemented; make effective use of it in implementing function `aveScores` below. Note the example at the bottom of the page.

```
function CA = aveScores(M)
% M is a 2-d array of characters. Each row of M stores the scores of one student:
%   a netID followed by one or more scores and these data items are separated by
%   commas.  There may be trailing spaces in a row of M.
% CA is an n-by-2 cell array where n is the number of students whose record includes
%   at least two scores. In each row of CA, the first cell stores the netID of a
%   student who has at least two scores and the second cell stores the average score
%   of that student.  If no student has at least two scores then CA is an empty cell
%   array.
% ONLY these built-in functions are allowed: length, size, str2double, sum, mean
% Recall that str2double can handle leading and trailing spaces, e.g.,
%   str2double('87    ') returns the type double scalar 87.
```

Example: Suppose `M` is

```
['vaf34,80,100,90';...
 'aaj91,100       ';...
 'rt2253,75,95    ']
```

Then `aveScores(M)` should return a 2×2 cell array `CA`:

- In row 1 column 1 is `'vaf34'` and in row 1 column 2 is the type double scalar 90.
- In row 2 column 1 is `'rt2253'` and in row 2 column 2 is the type double scalar 85.

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	
2		
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2		
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there are at least two test scores in the row, extract the netID and store it in the first column of CA	
4		
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there are at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1**);
4		
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there are at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1**);
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there are at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1**);
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	Use another for-loop (nested inside the first) that iterates from **k = 1:length(comma_idx)**, and determine indices of substring
5		

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there are at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1**);
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	Use another for-loop (nested inside the first) that iterates from **k = 1:length(comma_idx)**, and determine indices of substring
5	Store as a running sum in the second column of CA, and take the average after all scores have been extracted.	

Fall 2016 Prelim: **Question 5b**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there are at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1**);
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	Use another for-loop (nested inside the first) that iterates from **k = 1:length(comma_idx)**, and determine indices of substring
5	Store as a running sum in the second column of CA, and take the average after all scores have been extracted.	Initialize the second column to zero outside the for-loop of step 4, then convert the substring to a double and add to the second column.

Fall 2016 Prelim: **Question 5b**

```
for rowM = 1:size(M,1)

end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2




    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
rowCA = 0;
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2
        rowCA = rowCA+1;
        CA{ rowCA, 1 } = M( rowM, 1:comma_idx(1)-1 );




    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
rowCA = 0;
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2
        rowCA = rowCA+1;
        CA{ rowCA, 1 } = M( rowM, 1:comma_idx(1)-1 );

        for k = 1:length(comma_idx)
            left = comma_idx(k)+1;
            if k < length(comma_idx)
                right = comma_idx(k+1)-1;
            else
                right = size(M,2); % After last comma, take all remaining characters
            end

        end

    end
end
```

Connection to previous slide:

Red: for-loop to look at each string in M

Orange: extract commas

Green: extract and store netID

Blue: extract test score indices

Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
rowCA = 0; CA = {};
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2
        rowCA = rowCA+1;
        CA{ rowCA, 1 } = M( rowM, 1:comma_idx(1)-1 );
        CA{ rowCA, 2 } = 0;
        for k = 1:length(comma_idx)
            left = comma_idx(k)+1;
            if k < length(comma_idx)
                right = comma_idx(k+1)-1;
            else
                right = size(M,2); % After last comma, take all remaining characters
            end
            CA{rowCA,2} = CA{rowCA,2} + str2double( M( rowM, left:right ) );
        end
        CA{rowCA,2} = CA{rowCA,2}/length(comma_idx);
    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Structures and Structure Arrays

- Structures allow you to group together variables of any type you wish into a single entity and represent more complex objects.
    - Structures can contain any combination of different types that you wish (numeric, string fields, arrays, cells, even structs)

Structures and Structure Arrays

- Structures allow you to group together variables of any type you wish into a single entity and represent more complex objects.
  - Structures can contain any combination of different types that you wish (numeric, string fields, arrays, cells, even structs)
- Structures offer more generality than cell arrays, as you explicitly indicate the components of a structure through field names instead of indices
  - **sName.fName** accesses the field with name **fName** inside struct **sName**

Structures and Structure Arrays

- Structures allow you to group together variables of any type you wish into a single entity and represent more complex objects.
  - Structures can contain any combination of different types that you wish (numeric, string fields, arrays, cells, even structs)
- Structures offer more generality than cell arrays, as you explicitly indicate the components of a structure through field names instead of indices
  - **sName.fName** accesses the field with name **fName** inside struct **sName**
- Structures can be initialized either through the ‘struct’ function or through manually adding values to fields individually
  - Initialization by ‘struct’: location = struct(‘city’, ‘Ithaca’, ‘zipcode’, 14850);
  - Manual initialization: location.city = ‘Ithaca’; location.zipcode = 14850;

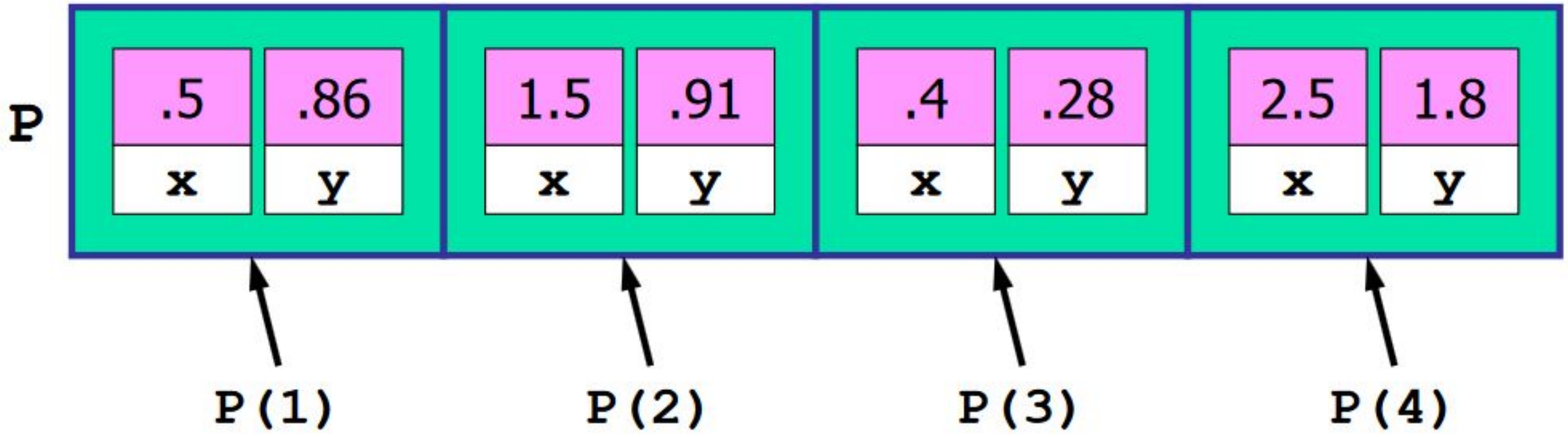
Structures and Structure Arrays

- You can create arrays of structures, and each component of the array will have exactly the same fields.

- You create structure arrays by supplying an index to the structure array and assigning fields or using the struct function.
  - P(1).x = 0.2;   P(1).y = 0.5;   P(2).x = 0.1;   P(2).y = 0.3
  - P(1) = struct(‘x’, 0.2, ‘y’, 0.5);   P(2) = struct(‘x’, 0.1, ‘y’, 0.3);

- Note that you should **not** initialize struct arrays as empty arrays, such as P = []; since that will make P into a numeric array.

Structure Array Syntax



`P`	is a struct array
`P(1)`	is a single struct
`P(1).x`	is a field value (0.5 in this case)

Spring 2016 Prelim: **Question 4**

Question 4: (25 points)

A 3-city cycling race is to be organized given the data of n cities, $n > 3$. Each struct in the length n struct array `S` stores the data of an individual city and has these fields:

- `cname`: the name of the city, a string
- `budget`: the amount of money that the city is willing to contribute to the race, a positive scalar

An n-by-n matrix `D` stores the distances between cities: `D(i,j)` is the distance between cities i and j in miles. `D` is symmetric, so `D(i,j)` equals `D(j,i)`.

In a 3-city cycling race, the cyclists start in one city, ride to a second city, ride to a third city, and then ride directly to the starting city. Given our symmetric matrix `D`, for a particular combination of three cities the race distance is the same regardless of the itinerary (the order of the three cities in the race). Elevation is not a concern in this race!

Given the struct array `S` and distance matrix `D`, consider all possible 3-city combinations that give a race distance of at least 100 miles and no more than 200 miles. Among those combinations find the one that has the largest total budget. (Assume that only one combination has this largest total budget value.) Store the data of this combination in a 1-d cell array `R` of length 4 where the first 3 cells store the names of the three cities and the 4th cell stores the total budget. If no 3-city combination meets the distance criteria, then `R` is an empty cell array.

For full credit, your code should be efficient. Built-in functions `max` and `min` are *not* allowed.

```
% Assume that struct array S and matrix D are given and are as described above.
% Write your code below.  Functions max and min are NOT allowed.
```

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	
2		
3		
4		

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2		
3		
4		

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2	Compute the distance of the itinerary and determine if the budget should be calculated	
3		
4		

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2	Compute the distance of the itinerary and determine if the budget should be calculated	Use an if statement on the total distance **D(city1,city2)+D(city2,city3)+D(city3,city1)**
3		
4		

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2	Compute the distance of the itinerary and determine if the budget should be calculated	Use an if statement on the total distance **D(city1,city2)+D(city2,city3)+D(city3,city1)**
3	If the itinerary is valid, compute the current budget and compare to the max budget so far	
4		

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2	Compute the distance of the itinerary and determine if the budget should be calculated	Use an if statement on the total distance **D(city1,city2)+D(city2,city3)+D(city3,city1)**
3	If the itinerary is valid, compute the current budget and compare to the max budget so far	Compute budget of current itinerary by accessing the structure array **curBudget = S(city1).budget + S(city2).budget + S(city3).budget** Initialize a **maxBudgetSoFar** and use an if statement comparing it to the budget of the current itinerary
4		

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2	Compute the distance of the itinerary and determine if the budget should be calculated	Use an if statement on the total distance **D(city1,city2)+D(city2,city3)+D(city3,city1)**
3	If the itinerary is valid, compute the current budget and compare to the max budget so far	Compute budget of current itinerary by accessing the structure array **curBudget = S(city1).budget + S(city2).budget + S(city3).budget** Initialize a **maxBudgetSoFar** and use an if statement comparing it to the budget of the current itinerary
4	If the current budget exceeds the max so far, then update the cell array **R** with information from **S**	

Spring 2016 Prelim: **Question 4**

Designing an algorithm

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through all possible three city combinations. Use the fact that order does not matter to minimize redundancy	A triple nested for loop. Since the order of the cities does not matter, we should set up the loops so that the indices are in increasing order **for city1 = 1:n-2** **for city2 = city1+1:n-1** **for city3 = city2+1:n**
2	Compute the distance of the itinerary and determine if the budget should be calculated	Use an if statement on the total distance **D(city1,city2)+D(city2,city3)+D(city3,city1)**
3	If the itinerary is valid, compute the current budget and compare to the max budget so far	Compute budget of current itinerary by accessing the structure array **curBudget = S(city1).budget + S(city2).budget + S(city3).budget** Initialize a **maxBudgetSoFar** and use an if statement comparing it to the budget of the current itinerary
4	If the current budget exceeds the max so far, then update the cell array **R** with information from **S**	**R{1} = S(city1).cname;** **R{2} = S(city2).cname;** **R{3} = S(city3).cname;** **R{4} = curBudget**

Spring 2016 Prelim: **Question 4**

```
n = length(S);


for city1 = 1:n-2
    for city2 = city1+1:n-1
        for city3 = city2+1:n









        end
    end
end
```

Connection to previous slide:
Red: for-loops across all itineraries
Orange: compute and compare distance
Green: compute and compare budget
Blue: store in R the max budget so far

Spring 2016 Prelim: **Question 4**

```
n = length(S);


for city1 = 1:n-2
    for city2 = city1+1:n-1
        for city3 = city2+1:n
            dist = D(city1,city2)+D(city2,city3)+D(city3,city1);
            if dist >= 100 && dist <= 200







            end
        end
    end
end
```

Connection to previous slide:
Red: for-loops across all itineraries
Orange: compute and compare distance
Green: compute and compare budget
Blue: store in R the max budget so far

Spring 2016 Prelim: **Question 4**

```
n = length(S);
maxBudgetSoFar = 0;

for city1 = 1:n-2
    for city2 = city1+1:n-1
        for city3 = city2+1:n
            dist = D(city1,city2)+D(city2,city3)+D(city3,city1);
            if dist >= 100 && dist <= 200
                curBudget = S(city1).budget + S(city2).budget + S(city3).budget;
                if curBudget > maxBudgetSoFar



                end
            end
        end
    end
end
```

Connection to previous slide:
Red: for-loops across all itineraries
Orange: compute and compare distance
Green: compute and compare budget
Blue: store in R the max budget so far

Spring 2016 Prelim: **Question 4**

```
n = length(S);
maxBudgetSoFar = 0;
R = { };
for city1 = 1:n-2
    for city2 = city1+1:n-1
        for city3 = city2+1:n
            dist = D(city1,city2)+D(city2,city3)+D(city3,city1);
            if dist >= 100 && dist <= 200
                curBudget = S(city1).budget + S(city2).budget + S(city3).budget;
                if curBudget > maxBudgetSoFar
                    R{1} = S(city1).cname;
                    R{2} = S(city2).cname;
                    R{3} = S(city3).cname;
                    R{4} = curBudget;
                end
            end
        end
    end
end
```

Connection to previous slide:

Red: for-loops across all itineraries

Orange: compute and compare distance

Green: compute and compare budget

Blue: store in R the max budget so far

Common Student Errors

- Getting the size of an array/Initializing arrays

  `size(A) = [nr, nc];` vs `[nr, nc] = size(A);`

- For loops based on array size

  `for k = 1:length(nr)` vs `for k = 1:nr`

- Accessing struct arrays

  `SA.x(1)` vs `SA(1).x`

- Initializing structs

  ```
  s = struct('vec', [1,2,3,4,5], 'num', 6, 'string', 'Hello')
  ```

- 2D Cell Array vs. Arrays in Cells

`A{1, 2}`	`A{1}(2)`
`{1, 2, 3;` ` 4, 5, 6}`	`{[1, 2, 3],` ` [4, 6]}`
has 6 cells	has 2 cells