Matrices

Storing two-dimensional numerical data

CS112 Scientific Computation
Department of Computer Science
Wellesley College

Analyzing table data

level	1998	1999	2000	2001	2002	2003	2004	2005
advanced	7	9	15	18	20	24	29	35
proficient	17	15	18	27	24	27	28	27
needs improvement	24	23	22	30	31	29	28	24
failing	52	53	45	25	25	20	15	15

Statewide results for MCAS Test in Mathematics, Grade 10

Medical imaging

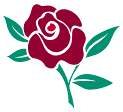
Matrices are particularly good for storing data that is inherently two-dimensional

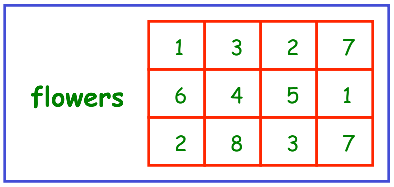
For example, the illustrated MRI slice is obtained from a two-dimensional grid of brightness measurements registered by an array of light sensitive elements

Matrices: The basics

- A matrix is a rectangular array of numbers
- We create a matrix of specific values with an assignment statement:

```
>> flowers = [1  3  2  7;  6  4  5  1;  2  8  3  7]
flowers =
      1   3   2   7
      6   4   5   1
      2   8   3   7
```





flowers

1	3	2	7
6	4	5	1
2	8	3	7

Dimensions

- Each row must contain the same number of values!

```
nums = [1  4  2;  6  8]
```

- **size** function returns the number of rows and columns in a matrix:

```
>> dims = size(flowers)
dims =
     3   4
>> rows = size(flowers, 1)
rows =
     3
>> cols = size(flowers, 2)
cols =
     4
```

flowers

1	3	2	7
6	4	5	1
2	8	3	7

How could you determine the total number of elements in a matrix?

Déjà vu all over again

Many computations can be performed on an entire matrix all at once

```
flowers = 2 * flowers + 1
```

flowers

1	3	2	7
6	4	5	1
2	8	3	7

flowers

3	7	5	15
13	9		

Element-by-element matrix addition

sumFlowers = flowers + addOns

flowers	3	7	5	15
	13	9	11	3
	5	17	7	15

+

addOns	2	1	3	4
	2	4	3	1
	2	0	1	4

=

sumFlowers	5	8	8	19
	15	13		

How do you perform element-by-element multiplication?

Vectors are matrices…
… with one row or column

```
rowNums = [1  2  3]
rowNumsSize = size(rowNums)
colNums = [1; 2; 3]
colNumsSize = size(colNums)
```

length function returns the largest dimension

rowNums	1	2	3

rowNumsSize	1	3

colNums	1
	2
	3

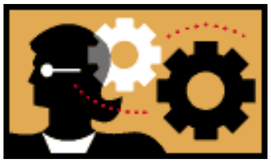
colNumsSize	3	1

Now I know what you're thinking…

You probably think that we can use functions like **sum**, **prod**, **min**, **max** and **mean** in the same way they were used with vectors:

```
numbers = [1 3 2 4; 4 1 2 3]
totalSum = sum(numbers)
totalProd = prod(numbers)
minVal = min(numbers)
maxVal = max(numbers)
meanVal = mean(numbers)
```

numbers	1	3	2	4
	4	1	2	3



Hmmm… that's not what I expected…

```
numbers = [1 3 2 4; 4 1 2 3]
totalSum = sum(numbers)
totalProd = prod(numbers)
minVal = min(numbers)
maxVal = max(numbers)
meanVal = mean(numbers)
```

numbers	1	3	2	4
	4	1	2	3

totalSum	5	4	4	7

totalProd	4	3	4	12

minVal	1	1	2	3

meanVal	2.5	2.0	2.0	3.5

maxVal	4	3	2	4

Processing and displaying images

An image is a two-dimensional grid of measurements of brightness

We will start with images with brightness ranging from black (0.0) to white (1.0) with shades of gray in between (0.0 < b < 1.0)

1887 Crew Team
Wellesley College

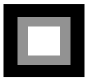
Creating a tiny image

```
>> tinyImage = [ 0.0  0.0  0.0  0.0  0.0  0.0; …
                 0.0  0.5  0.5  0.5  0.5  0.0; …
                 0.0  0.5  1.0  1.0  0.5  0.0; …
                 0.0  0.5  1.0  1.0  0.5  0.0; …
                 0.0  0.5  0.5  0.5  0.5  0.0; …
                 0.0  0.0  0.0  0.0  0.0  0.0]
```

tinyImage

0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.5	0.5	0.5	0.5	0.0
0.0	0.5	1.0	1.0	0.5	0.0
0.0	0.5	1.0	1.0	05	0.0
0.0	0.5	0.5	0.5	0.5	0.0
0.0	0.0	0.0	0.0	0.0	0.0

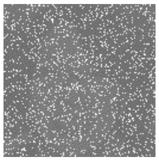
```
>> imshow(tinyImage)
```

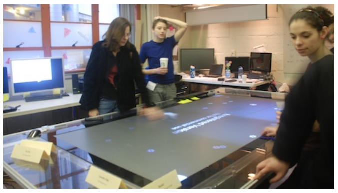


(not to scale)

A mystery: Who stole The Beast?

This very corrupted image was received by anonymous courier late last night





Let's figure out what's in it using the Image Processing Toolbox

```
>> imtool(image)
```

Creating matrices with constant values

To create a matrix of all ones:

```
nums1 = ones(2,3)
nums2 = ones(1,5)
```

To create a matrix of all zeros:

```
nums3 = zeros(3,1)
nums4 = zeros(4,3)
```

nums1

1	1	1
1	1	1

nums2

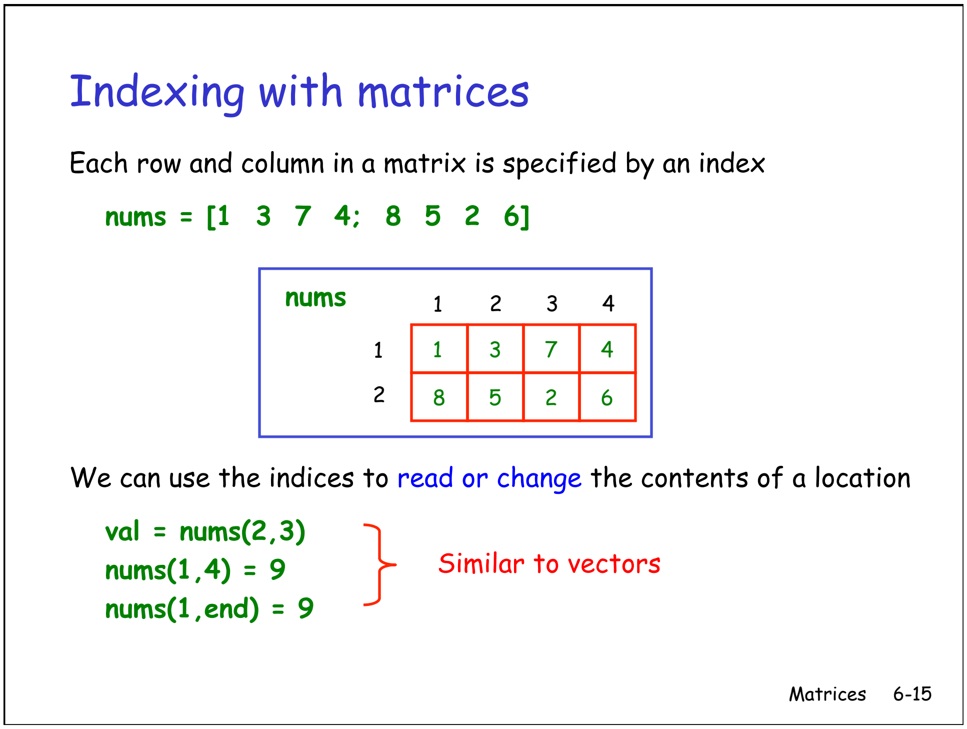
1	1	1	1	1

nums3

0
0
0

nums4

0	0	0
0	0	0
0	0	0
0	0	0



Indexing with matrices

Each row and column in a matrix is specified by an index

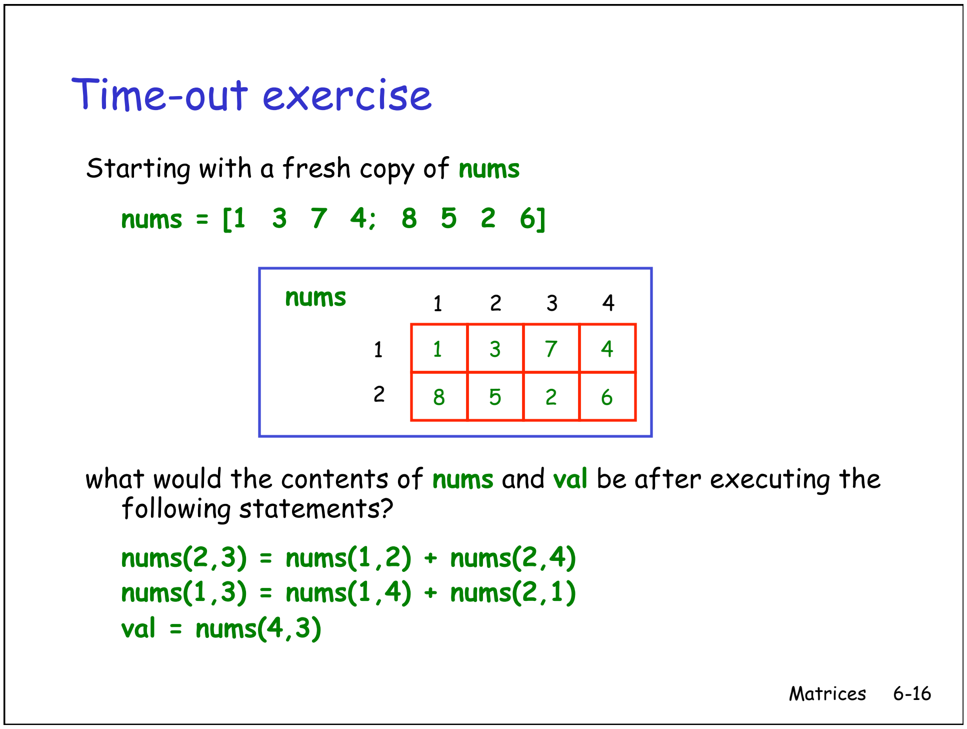
```
nums = [1  3  7  4;  8  5  2  6]
```

nums	1	2	3	4
1	1	3	7	4
2	8	5	2	6

We can use the indices to read or change the contents of a location

```
val = nums(2,3)
nums(1,4) = 9
nums(1,end) = 9
```

Similar to vectors



Time-out exercise

Starting with a fresh copy of nums

```
nums = [1  3  7  4;  8  5  2  6]
```

nums	1	2	3	4
1	1	3	7	4
2	8	5	2	6

what would the contents of nums and val be after executing the following statements?

```
nums(2,3) = nums(1,2) + nums(2,4)
nums(1,3) = nums(1,4) + nums(2,1)
val = nums(4,3)
```

Auto expansion of matrices

```
>> nums = [1 3 7 4; 8 5 2 6]
```

nums	1	2	3	4
1	1	3	7	4
2	8	5	2	6

```
>> nums(4, 7) = 3
```

nums	1	2	3	4	5	6	7
1	1	3	7	4	0	0	0
2	8	5	2	6	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	3

Analyzing table data

level	1998	1999	2000	2001	2002	2003	2004	2005
advanced	7	9	15	18	20	24	29	35
proficient	17	15	18	27	24	27	28	27
needs improvement	24	23	22	30	31	29	28	24
failing	52	53	45	25	25	20	15	15

Statewide results for MCAS Test in Mathematics, Grade 10

Indexing with colon notation

To refer to an *entire column* of a matrix, provide **:** as the first index and the column number as the second index

```
>> nums(:, 3)
ans =
     3
     8
    13
    18
```

nums	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20

To refer to an *entire row* of a matrix, provide **:** as the second index and the row number as the first index

```
>> nums(2, :)
   ans =
        6  7   8   9  10
```

Plotting trends in performance levels

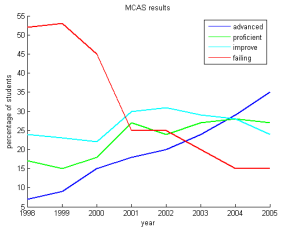
We begin our analysis by plotting the data for each performance level over the 8 years

```
% create matrices that store data and years
results =  [ 7    9  15  18  20  24  29  35; ...
            17  15  18  27  24  27  28  27; ...
            24  23  22  30  31  29  28  24; ...
            52  53  45  25  25  20  15  15];
years = [1998 1999 2000 2001 2002 2003 2004 2005];
```

Each row of the table corresponds to a performance level. How do we plot the resulting trend over the given years?

Plotting the data

```
% plot the data for each performance level vs. years
hold on
plot(years, results(1,:), 'b‘, ‘LineWidth’, 2);
plot(years, results(2,:), 'g‘, ‘LineWidth’, 2);
plot(years, results(3,:), 'c‘, ‘LineWidth’, 2);
plot(years, results(4,:), 'r‘, ‘LineWidth’, 2);
hold off
```

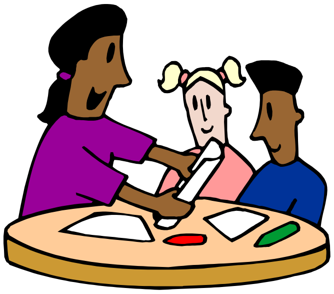


```
xlabel('year‘)
ylabel('percentage of students‘)
title('MCAS results‘)
legend('advanced‘, 'proficient‘, 'improve‘, 'failing‘ );
```

Finally, ...

Suppose we want to print the *change in results* between 1998 and 2005 for each performance level…

How do we do this?



Printing changes in results

```
% print total change in results between 1998 and 2005

totalChange = results(:, end) - results(:, 1);

disp('Change in performance between 1998 and 2005: ‘);
disp(['advanced: ' num2str(totalChange(1)) '% ‘]);
disp(['proficient: ' num2str(totalChange(2)) '% ‘]);
disp(['needs improvement: ' num2str(totalChange(3)) '% ‘]);
disp(['failing: ' num2str(totalChange(4)) '% ‘]);
```

```
Change in performance between 1998 and 2005:
advanced: 28%
proficient: 10%
needs improvement: 0%
failing: -37%
```

Time-out exercise

For each year, compute a *weighted sum* of the four percentages, using a weight of 1 for “advanced”, 2 for “proficient”, 3 for “needs improvement” and 4 for “failing”*

```
overallPerformance =
```

Add a new row to the **results** matrix that stores these weighted sums

* The resulting sum can range from 100 (great!) to 400 (not so good…)

More indexing with colon notation

We can use colon notation to refer to a *range of indices* within a column or row of a matrix

```
>> nums(1:3, 4)
ans =
      4
      9
     14
>> nums(3, 3:5)
ans =
     13  14  15
>> nums(2:3, 2:4)
ans =
      7   8   9
     12  13  14
```

nums	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15
4	16	17	18	19	20

Conditional operations on matrices

A conditional expression can be *applied to an entire matrix all at once* producing a new matrix of the same size that contains logical values

```
ages = [13  52  19  21;  18  47  23  15;  60  38  16  12];
teens = (ages >= 13) & (ages <= 19);
```

ages			
13	52	19	21
18	47	23	15
60	38	16	12

teens			
1	0	1	0
1	0	0	1
0	0	1	0

Using logical vectors

```
>> ages(teens) = 0
ages =
     0   52    0   21
     0   47   23    0
    60   38    0   12

>> overTheHill = ages(ages>40)
overTheHill =
    60
    52
    47
```

ages

13	52	19	21
18	47	23	15
60	38	16	12

teens

1	0	1	0
1	0	0	1
0	0	1	0

Time-out exercise

Given the original **ages** matrix, write two statements that each assign the variable **numAdults** to the total number of age values that are 18 or over

One statement should use **sum** and the other should use **length**