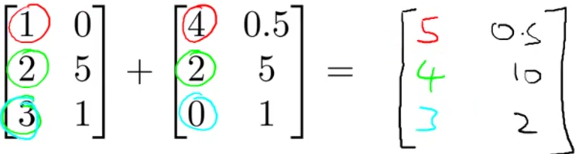
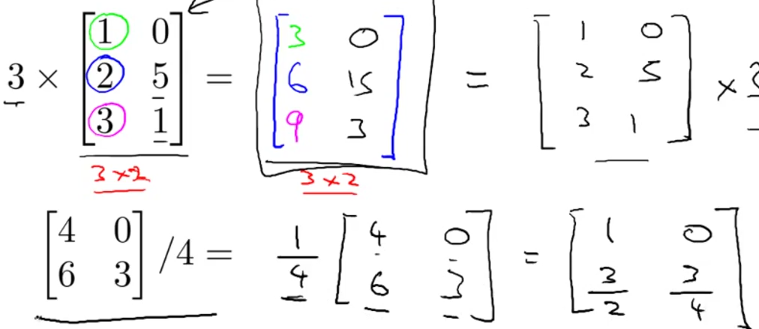
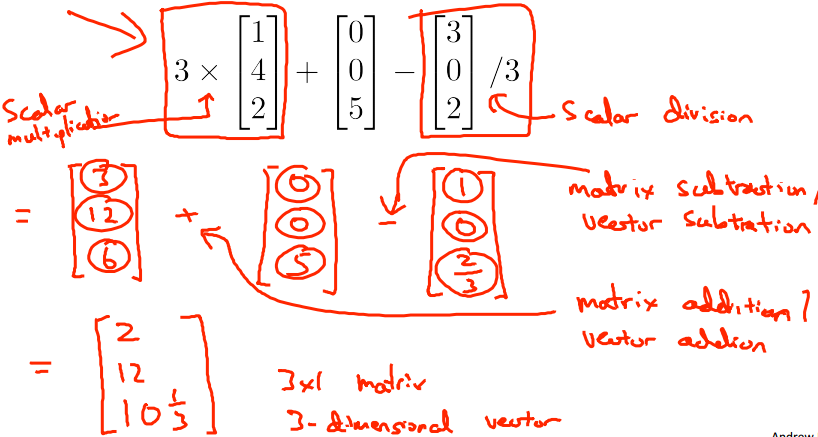
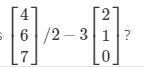
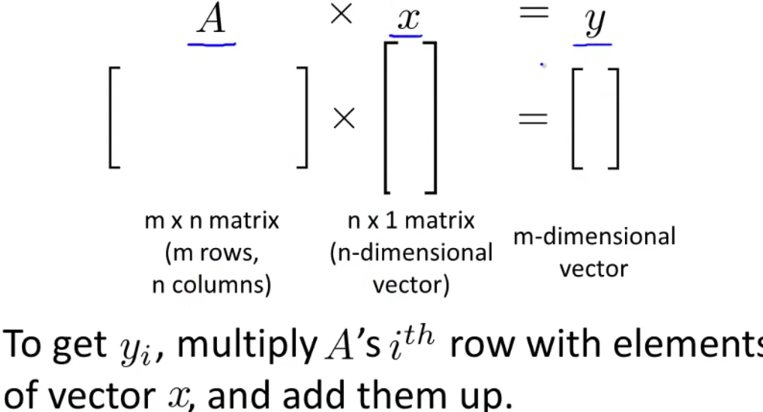
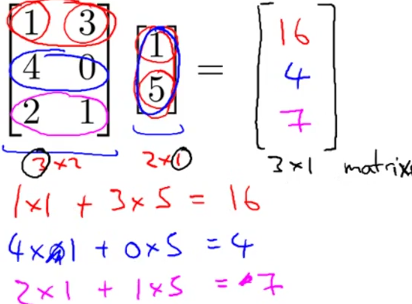
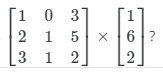
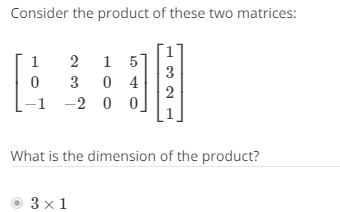
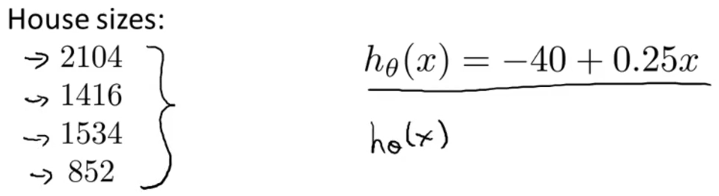
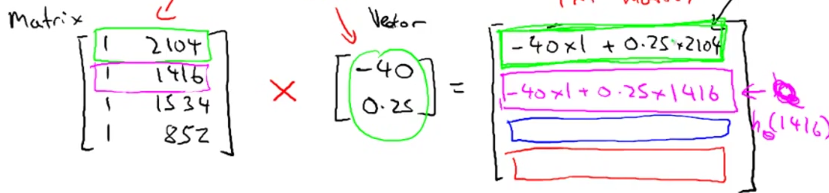
* **Matrix** = rectangular array of #’s w/in square brackets 🡪 a 2D array w/ x rows and y cols
* **A(I,j) =** value in row i and column j 🡪 A(1,2) = value in 1st row, 2nd col
* **Vector =** an n x 1 matrix (only 1 column) 🡪 y = [4, 6, 3 0] = a 4D vector = vector w/ 4 elements
* **Y(i) =** ith element of vector Y
* Can be 1 or 0 indexed (like arrays in programming languages)
* **Matrix Addition** 🡪 add up elements in same coordinates
* 
* *Can only add matrices w/ same dimensions*
* Matrices are usually denoted by uppercase names while vectors are lowercase.
* **Scalar (Real #, object that is a single value, not a vector or matrix ) Math** 🡪 multiple/divide matrix by 1 number 🡪 perform operation on each element
* 
* R refers to the set of scalar real numbers.
* Rn refers to the set of n-dimensional vectors of real numbers.
* Can combine operands
* 
*  *=* 
* **Matrix + Vector Multiplication**
* 
* *Columns in matrix must match rows in/dimension in vector* (n = n)
* *# of rows in matrix = # rows in resulting matrix/vector*
* To get 1st value of resulting vector, take 1st row of matrix, multiply each element by each element in vector, and then sum them up
* (1st # in row \* 1st # in vector) + (2nd # in row + 2nd # in vector) + (3rd # in row \* 3rd # in vector)

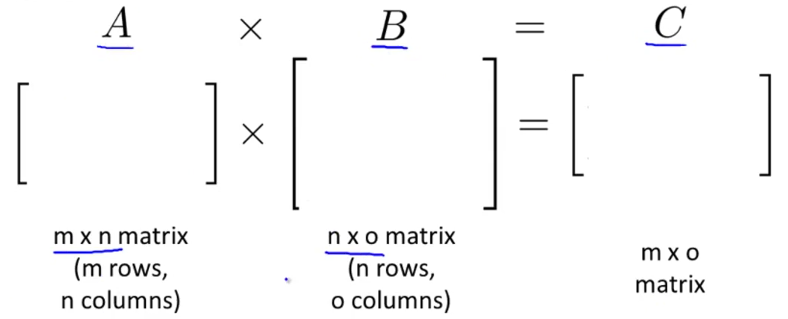


 **=** 

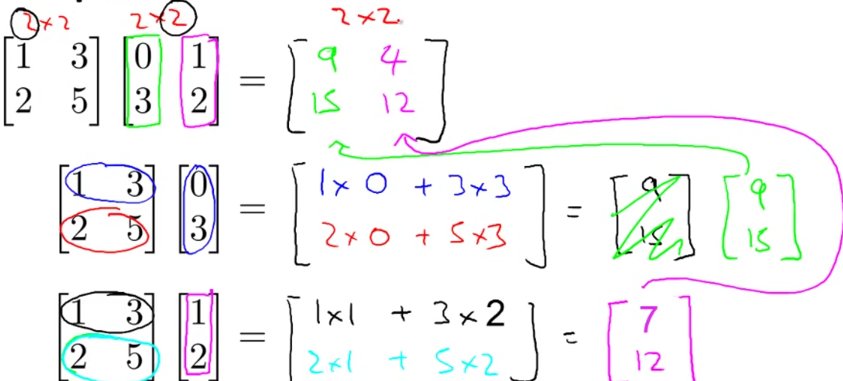
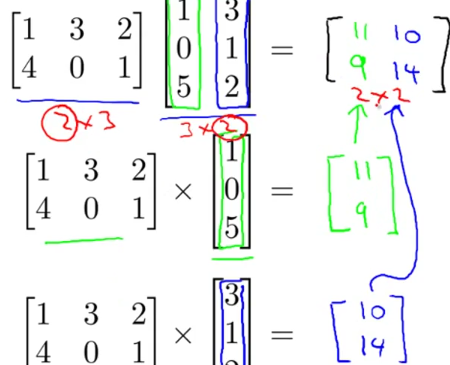
* Let’s compute h(x) = -40 + 0.25x for a vector of house sizes



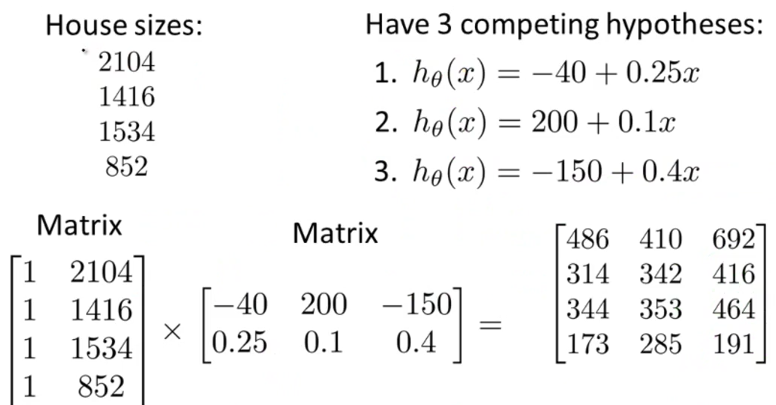
* Construct matrix of col 1 = 1 (to multiply by θ0) and col 2 = housing sizes (to multiple by θ1) and a vector w/ 2 elements, θ0 and θ1, and multiply them together to result in a 4x1 vector
* 
* When implementing this in software, it can be written in 1 line of code (simpler and more computationally efficient than a for loop for prediction(i) through i in n:m
* **Prediction = DataMatrix \* Parameters**
* **Matrix + Matrix Multiplication**



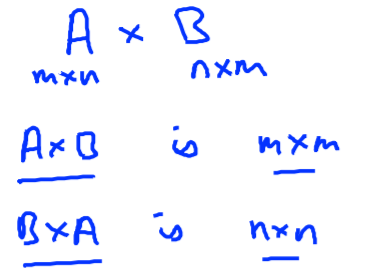
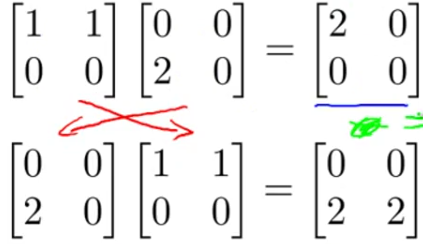
* *# of cols in 1st matrix MUST EQUAL # of rows in 2nd matrix*



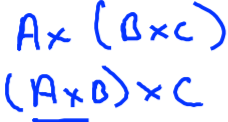
* Let’s compute multiple h(x)’s for a vector of house sizes

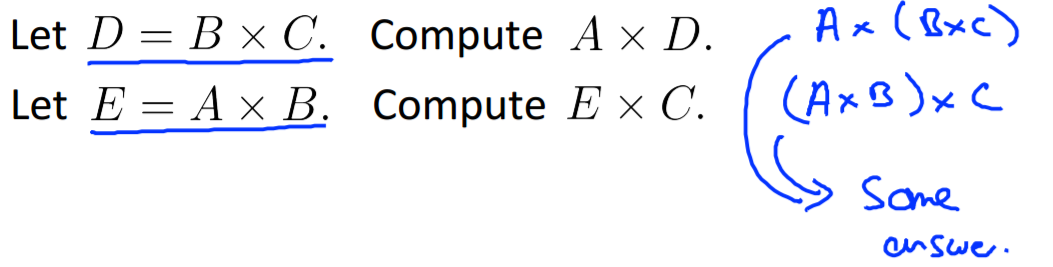


* 1st col in result = predictions of 1st h(x), etc.
* **Commutative Property of Multiplication of Real #’s** 🡪 doesn’t matter order (3x5 = 5x3)
* *NOT TRUE FOR MATRICES*

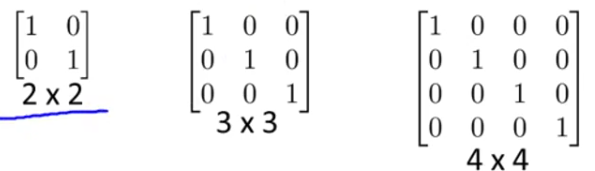


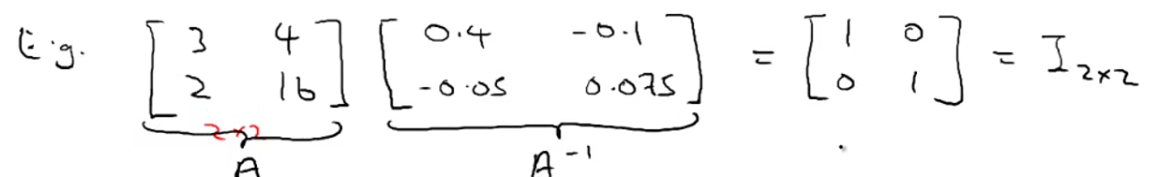
* **Associative Property**

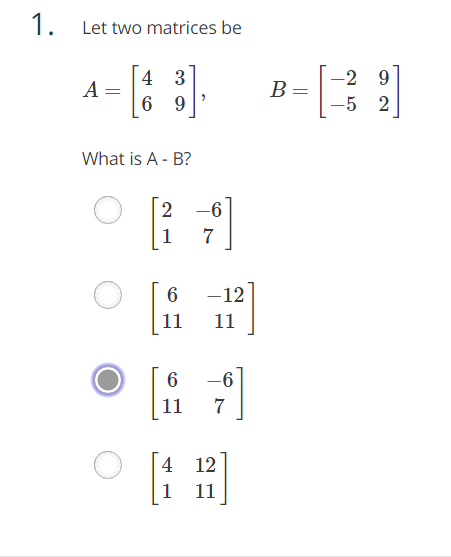
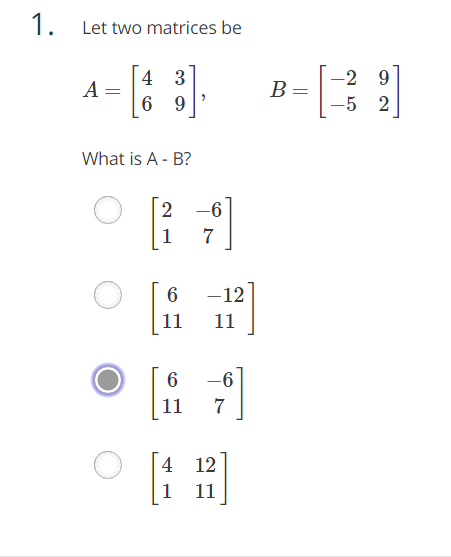


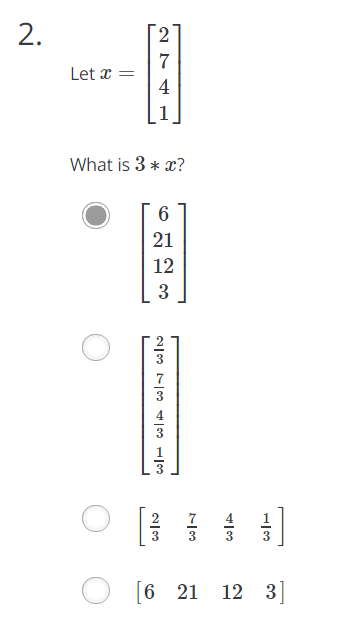
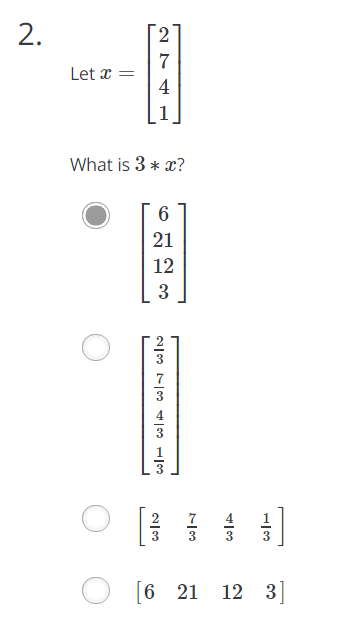


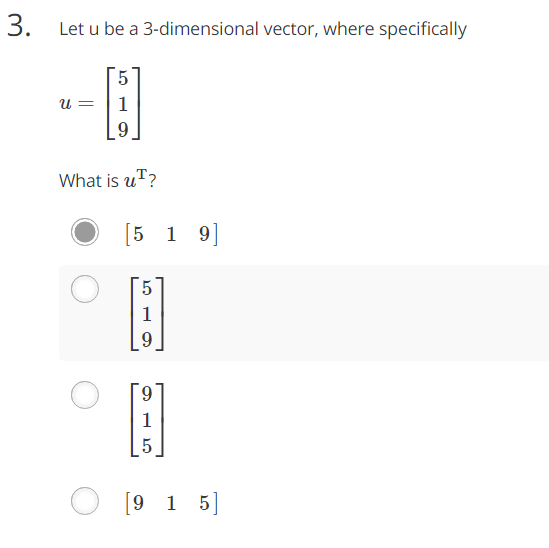
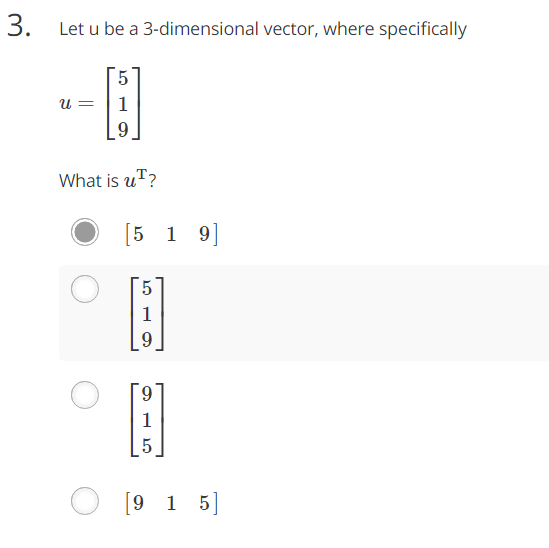
* **Identity Matrix (*I)***
* **1 = identity** 🡪 1\*z = z\*1 = z for any number z
* Matrix where diagonals = 1

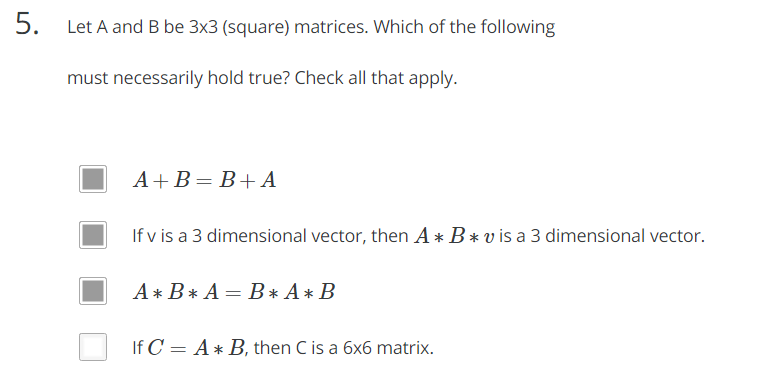
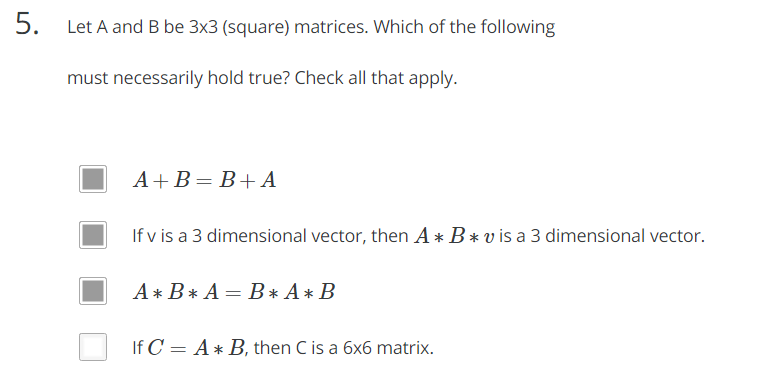


* For any matrix A 🡪 A\**I* = *I*\*A = A 🡪 *only time commutative property is true for matrices*
* **Inverse** of a number = the number that when multiplied by the original number = 1
* 3 \* 3^-1 = 1
* Not all #’s have an inverse (0^-1 = 0
* Matrices have inverses 🡪 A\*A^-1 = A^-1\*A = *I* (identity matrix)
* Must be a **square matrix** (MxM)
* 
* **Tranpose**
* 

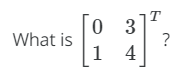
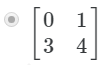


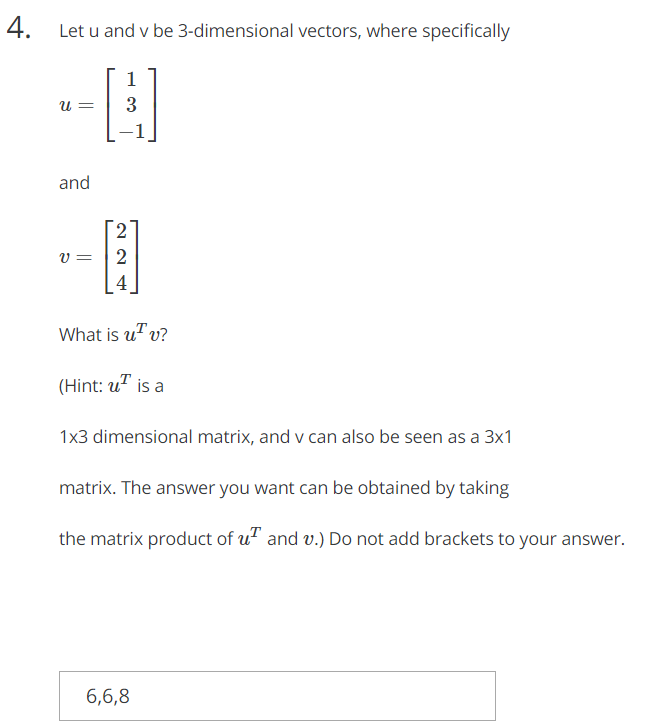










.

Let two matrices be

A=[4639],B=[−2−592]

What is A - B?

[21−67]

[611−1211]

[611−67]

[411211]

1

point

2.

Let x=⎡⎣⎢⎢2741⎤⎦⎥⎥

What is 3∗x?

⎡⎣⎢⎢621123⎤⎦⎥⎥

⎡⎣⎢⎢⎢⎢⎢⎢23734313⎤⎦⎥⎥⎥⎥⎥⎥

[23734313]

[621123]

1

point

3.

Let u be a 3-dimensional vector, where specifically

u=⎡⎣519⎤⎦

What is uT?

[519]

⎡⎣519⎤⎦

⎡⎣915⎤⎦

[915]

1

point

4.

Let u and v be 3-dimensional vectors, where specifically

u=⎡⎣13−1⎤⎦

and

v=⎡⎣224⎤⎦

What is uTv?

(Hint: uT is a

1x3 dimensional matrix, and v can also be seen as a 3x1

matrix. The answer you want can be obtained by taking

the matrix product of uT and v.) Do not add brackets to your answer.

6,6,8

1

point

5.

Let A and B be 3x3 (square) matrices. Which of the following

must necessarily hold true? Check all that apply.

A+B=B+A

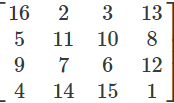
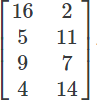
If v is a 3 dimensional vector, then A∗B∗v is a 3 dimensional vector.

A∗B∗A=B∗A∗B

If C=A∗B, then C is a 6x6 matrix.

* A = [1 2; 3 4; 5 6]; B = [1 2 3; 4 5 6];

Which of the following are valid commands?

* **C = A \* B;** (# of cols in left must = # of rows in right 🡪 3x2 \* 2x3)
* **C = B' + A;** (only matching indexes will add up)
* **C = A' + B;** (only matching indexes will add up)
* **C = B \* A;** (# of cols in left must = # of rows in right 🡪 2x3 \* 3x2)
* Let A=  Which of the following indexing expressions gives B= ?
* **B = A(:, 1:2); 🡪** all rows, cols 1 + 2
* **B = A(1:4, 1:2); 🡪** rows 1-4, cols 1 + 2
* Let A be a 10x10 matrix and x be a 10-element vector. Your friend wants to compute the product **Ax** and writes the following code:

v = zeros(10, 1);

for i = 1:10

for j = 1:10

v(i) = v(i) + A(i, j) \* x(j);

end

end

How would you vectorize this code to run without any for loops?

* **v = A \* x;** (# of cols in left = # of rows in right)
* Say you have 2 column vectors v and w, each w/ 7 elements (i.e., dimensions 7x1). Consider the following code:

z = 0;

for i = 1:7

z = z + v(i) \* w(i)

Which of the following vectorizations correctly compute z? Check all that apply.

* **z = sum (v .\* w);**
* **z = v' \* w;**
* **z = w' \* z;**
* In Octave/Matlab, many functions work on single #’s, vectors, + matrices. For example, **sin()**, when applied to a matrix, will return a new matrix w/ the sin of each element. But you have to be careful, as certain functions have different behavior. Suppose you have an 7x7 matrix X + you want to compute the log of every element, the square of every element, add 1 to every element, and divide every element by 4 + store the results in 4 matrices, A,B,C,D. 1 way to do so is the following code:

for i = 1:7

for j = 1:7

A(i, j) = log(X(i, j));

B(i, j) = X(i, j) ^ 2;

C(i, j) = X(i, j) + 1;

D(i, j) = X(i, j) / 4;

end

end

Which of the following correctly compute A,B,C, or D?

* **C = X + 1;**
* **D = X / 4;**
* **B = X .^ 2;**