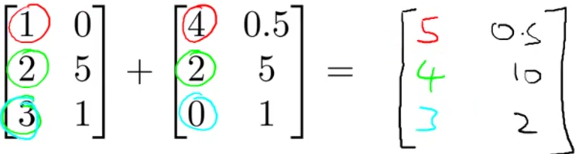
***LINEAR ALGEBRA REVIEW***

**I. Matrices and Vectors**

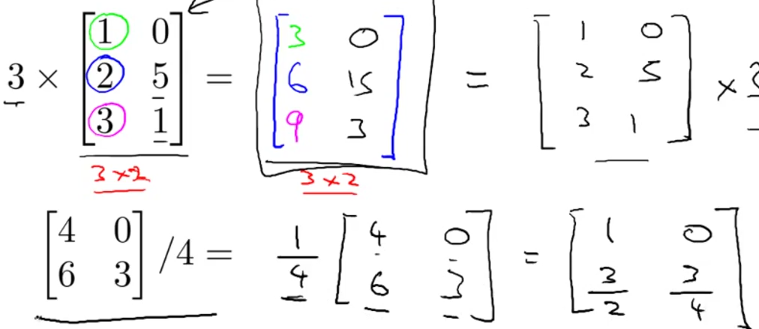
* **Matrix** = rectangular array of #’s written w/in square brackets 🡪 a 2D array w/ x rows + y cols
* **A(i,j) =** value in row i and column j 🡪 A(1,2) = value in 1st row, 2nd col
* 4\*2 matrix **=** 
* **Vector =** an n x 1 matrix (only 1 *column*) 🡪 y = [4, 6, 3, 0] = a **4D vector** = vector w/ 4 elements
* **Y(i) =** ith element of vector Y
* Can be 1 or 0 indexed (like arrays in programming languages)

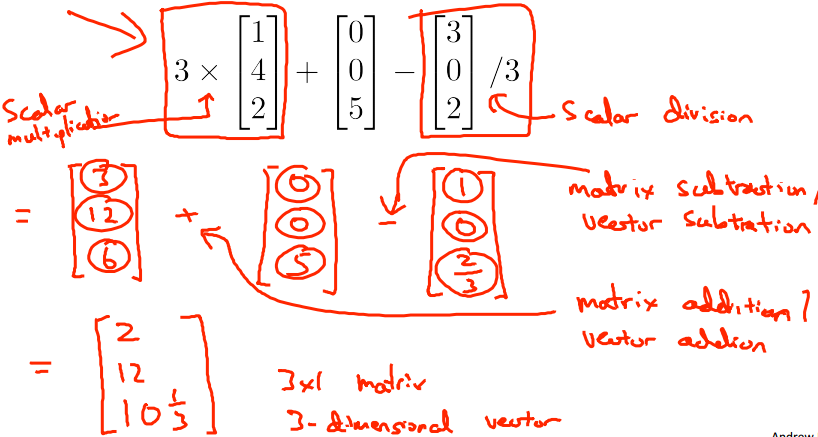
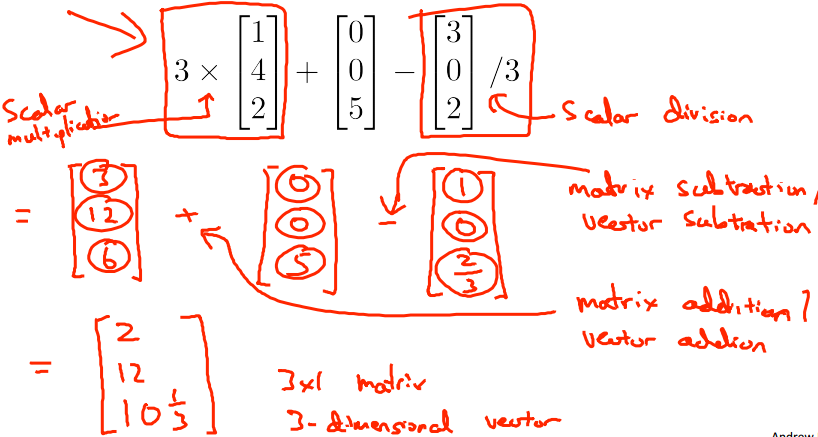
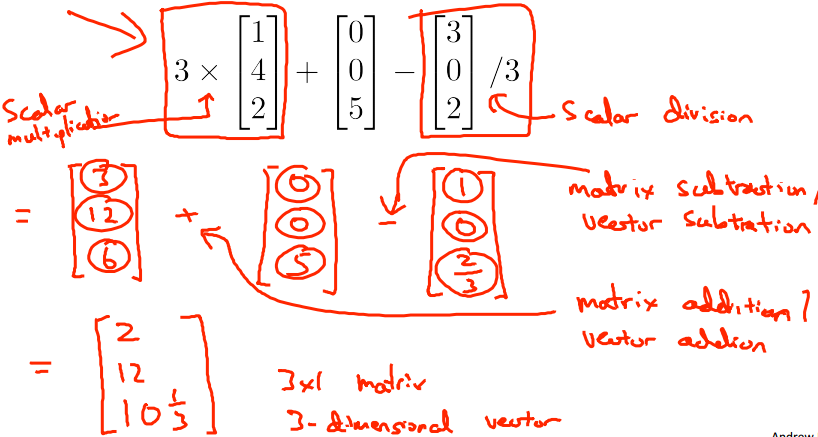
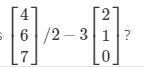
**II. Matrix Addition + SCALAR MULTIPLICATION**

* **Matrix Addition** 🡪 add up elements in *same coordinates* {x,y} in each respective matrix

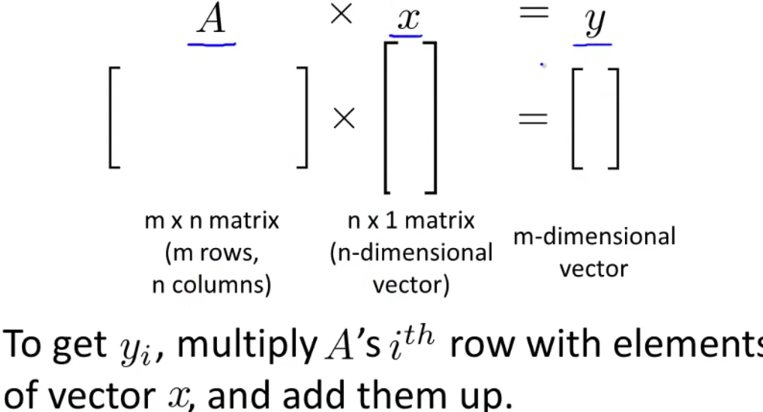


* *\*\*\*Can only add matrices w/ same dimensions*
* **Scalar (Real # 🡪 object that is a single value 🡪 NOT a vector or matrix ) Math** 🡪 multiply/divide matrix by ONE number 🡪 perform operation on *each element*

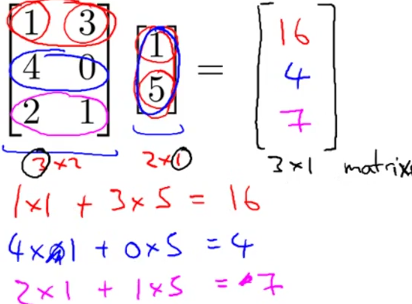


* R refers to the set of scalar real numbers.
* Rn refers to the set of n-dimensional vectors of real numbers.
* Can *combine operands*
* 
* 
* 
*  *=* 

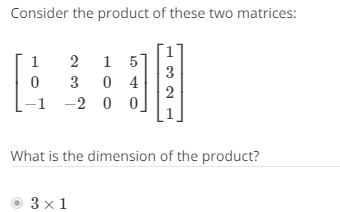
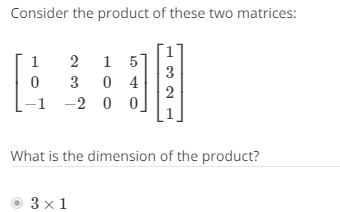
**III. Matrix + Vector Multiplication**

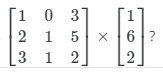


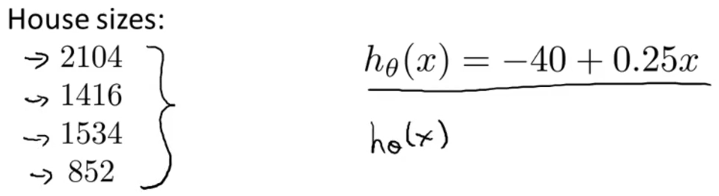
* *Columns in matrix MUST match rows in/dimension in vector* (n = n)
* *# of rows in matrix = # rows in resulting matrix/vector*
* To get 1st value of resulting vector, take 1st row of matrix + multiply each element by each element in vector, and then *sum them up*
* (1st # in row1 \* 1st # in vector) + (2nd # in row1 + 2nd # in vector) + (3rd ….. etc.)



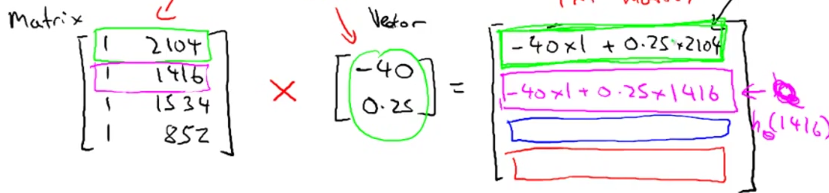
* Consider the product of these 2 matrices. What are the dimensions of the product?



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

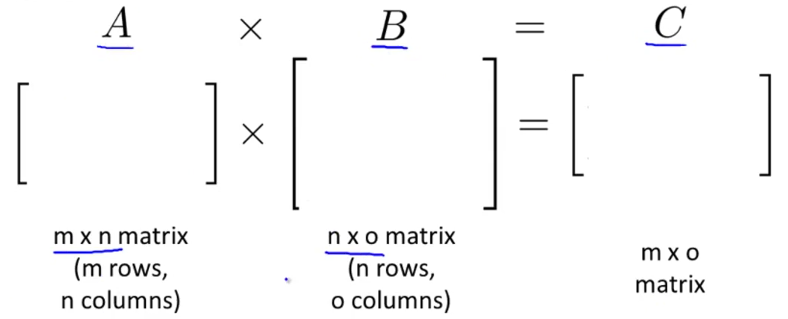


* Construct matrix where column 1 = 1 (the # multiplied by θ0) and column 2 = housing sizes (multiplied by θ1)
* Then have a vector w/ 2 elements, θ0 + θ1 + multiply them together to result in a 4x1 vector

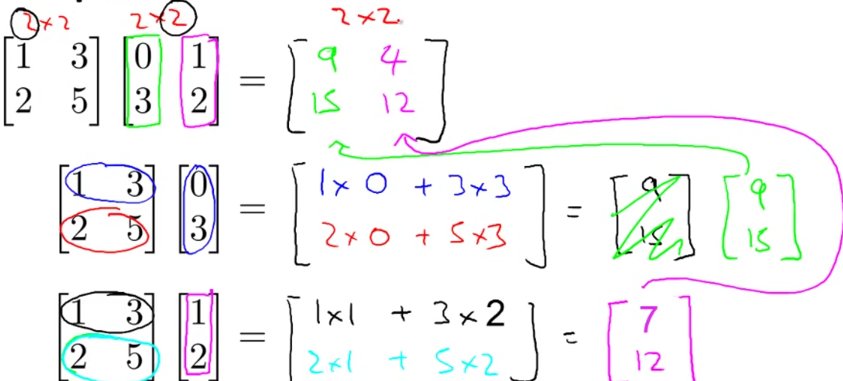
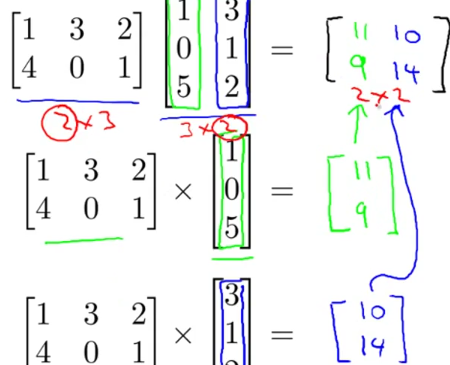


* When implementing this in software, it can be written in 1 line of code (simpler + more computationally efficient than a FOR loop for prediction(i) through i in n:m)
* **Prediction = DataMatrix \* Parameters**

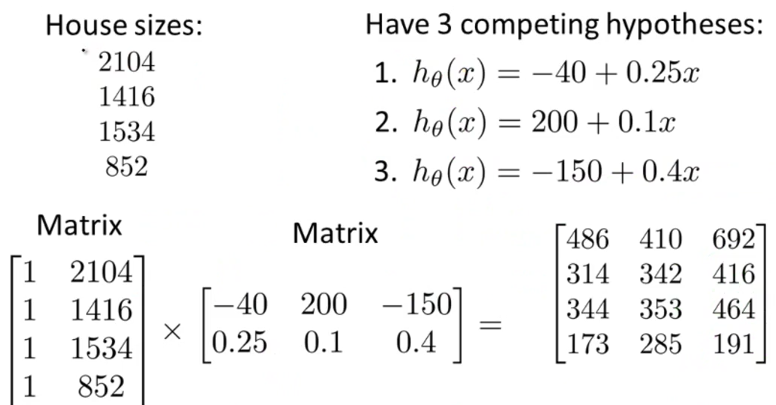
**IV. Matrix + Matrix Multiplication**

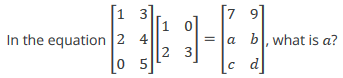


* *# of COLS in 1st matrix MUST EQUAL # of ROWS in 2nd matrix*



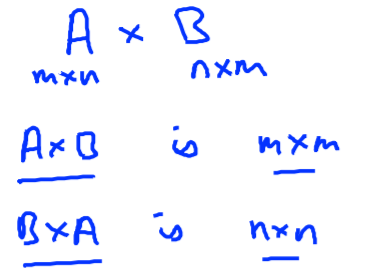
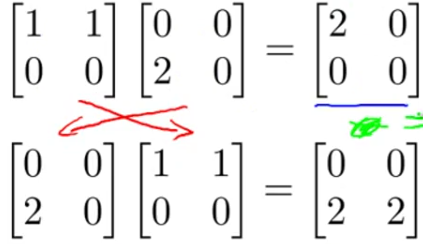
* Let’s compute multiple hθ(x)’s for a vector of house sizes



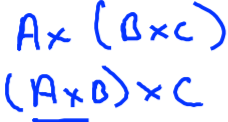
* 1st column in result = predictions of 1st hθ(x), 2nd column = predictions of 2nd hθ(x), etc.
*  🡺 **10, b = 12, c = 10, d = 15**

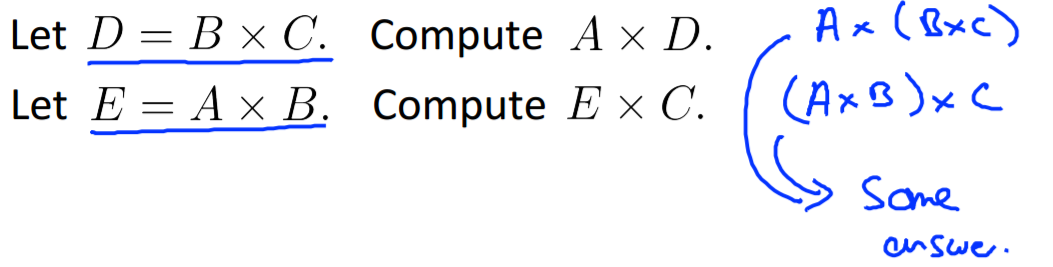
**IV. Matrix Multiplication Properties**

* **Commutative Property of Multiplication of Real #’s** 🡪 multiplication order doesn’t matter for real numbers (3 x 5 = 5 x 3), which is *NOT TRUE FOR MATRICES*
* rows \* cols = final dimensions

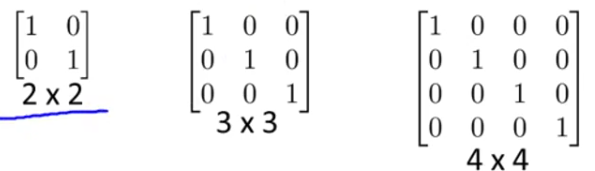


* **Associative Property**
* Again, order doesn’t matter for real numbers 🡪 For 3\*5\*2, we can do 3\*5 *OR* 5\*2 first
* Matrix multiplication IS associative, unlike how it is *not commutative*

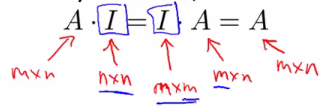




* B\*C has same column as (A\*B)\*C
* A has same columns as (A\*B)
* **Identity Matrix (*I)***
* **“1 is the identity operation”**🡪 For any number z, 1\*z = z\*1 = z
* **Identity Matrix** 🡪 where diagonals = 1
* Denoted as ***I*** or ***I(n\*n)***

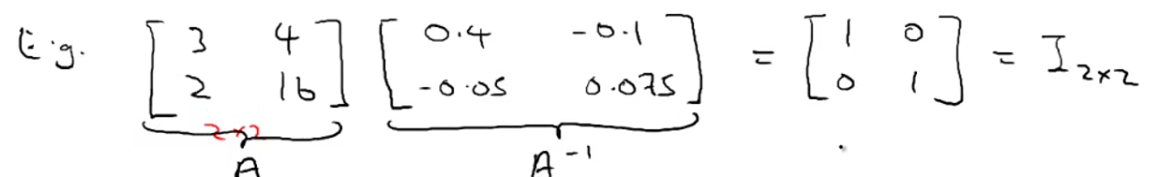


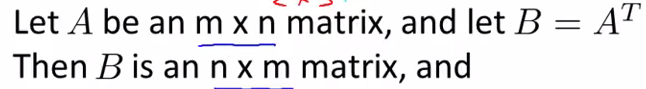
* For any matrix A 🡪 A\**I* = *I*\*A = A 🡪 \*\**The* *only time commutative property is true for matrices*

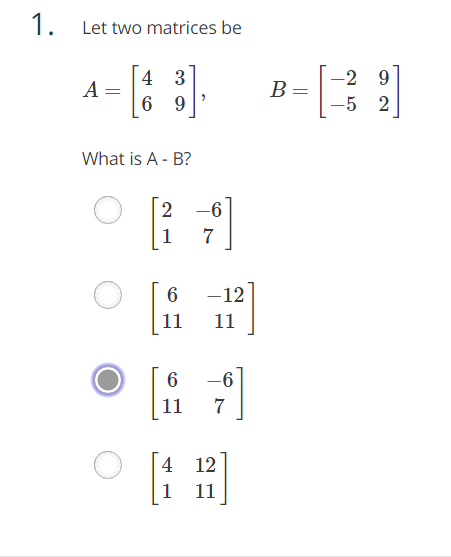
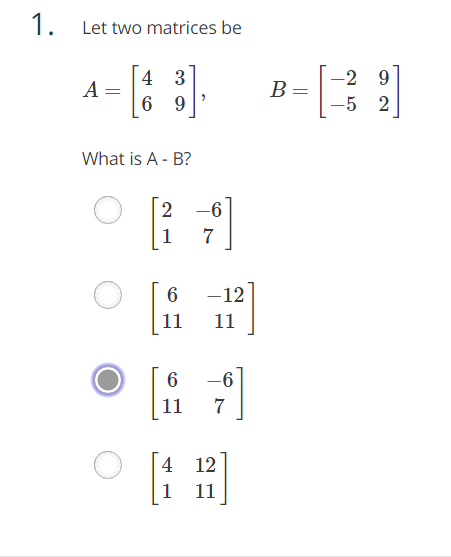


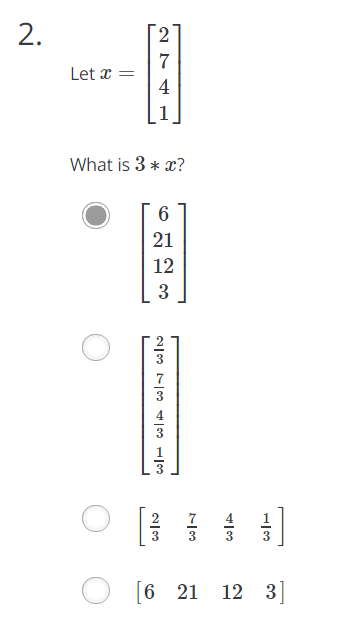
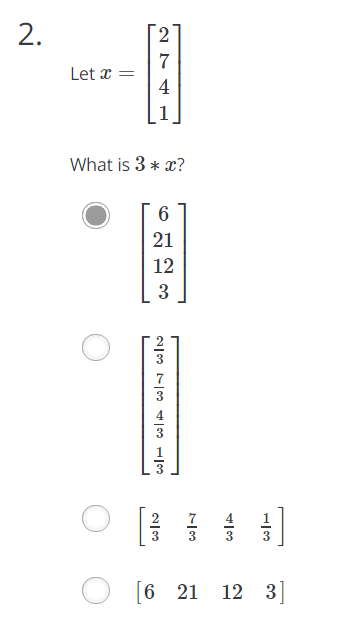
**V. Matrix Operations: Inverse and Transpose)**

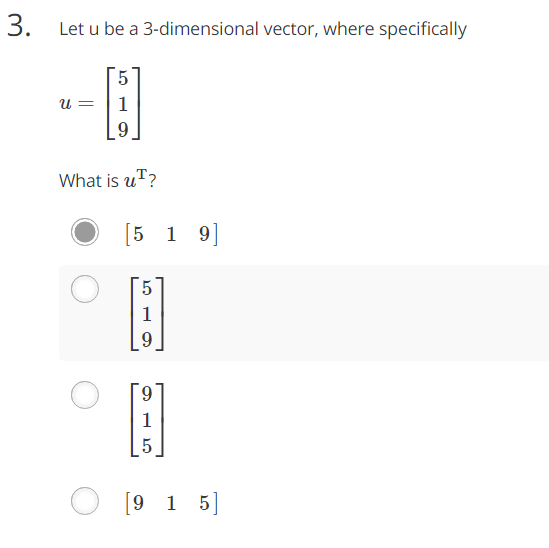
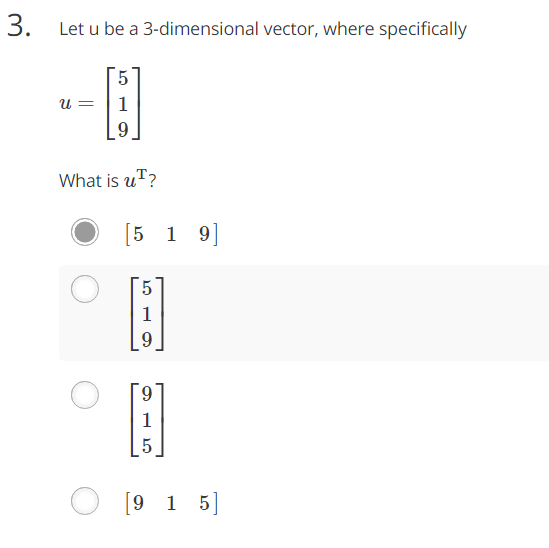
* **Inverse** of a number = the number that, when multiplied by the *original* number, = 1
* i.e. there exists some number for x such that x multiplied by this number = 1 🡪 3\*3^(-1) = 1
* *Not all #’s have an inverse (0^-1 = 0) 🡪 matrix of all 0’s does not have an inverse*
* Matrices w/out inverses are **degenerate matrices** or **matrices**
* If A is an m\*m matrix (**a square matrix**), and if it has an inverse, then these multiplied together give the m\*m Identity matrix ***I***
* A\*(A^-1) = (A^-1)\*A = *I*
* *Must be a* ***square matrix*** *(MxM) to have an inverse: 2\*2 matrix:*

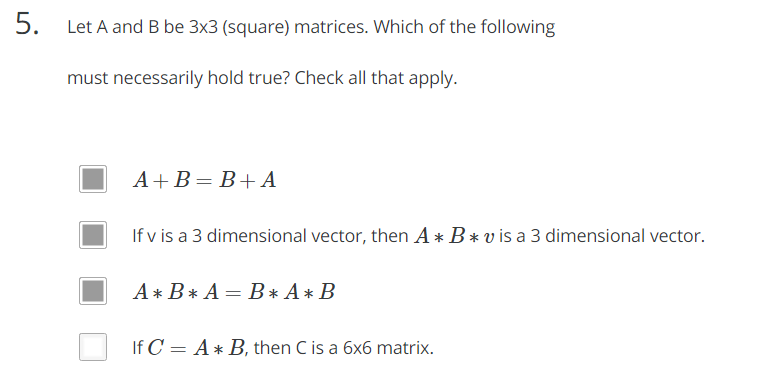
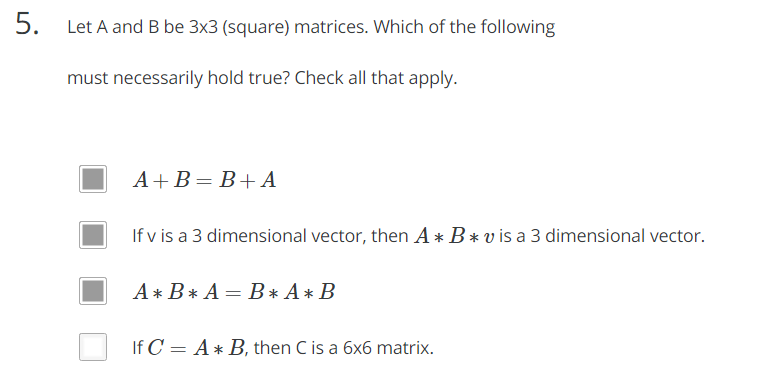


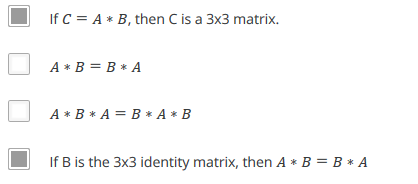
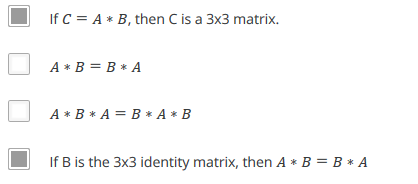
* **Transpose**
* Reverse/rotate the dimensions (rows 🡪 cols, cols 🡪 rows)
* 
* 
* A(1,2) = A(t)(2,1) A(2,3) = A(t)(3,2)

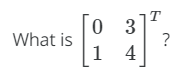
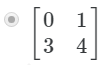








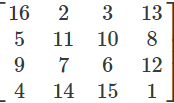
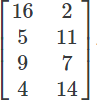


***OCTAVE***

* 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;**