The background is a dark blue gradient with a subtle pattern of small white dots. Overlaid on the left side is a large, semi-circular degree scale ranging from 150 to 260 degrees. Several concentric circles and arcs are drawn in a light blue/white color, some with arrows indicating a counter-clockwise direction of rotation. The main title and subtitle are positioned on the right side of the slide.

# ENGR 101 – Chapter 2

## Vectors and Matrices

Laura Alford, James Juett, Rick Niciejewski

9/3/2020

# Vectors and Matrices

- A key strength of MATLAB is support for working with vectors and matrices just as easily as scalar values.

- Scalar: A plain old number

- Vector: A one-dimensional sequence of numbers

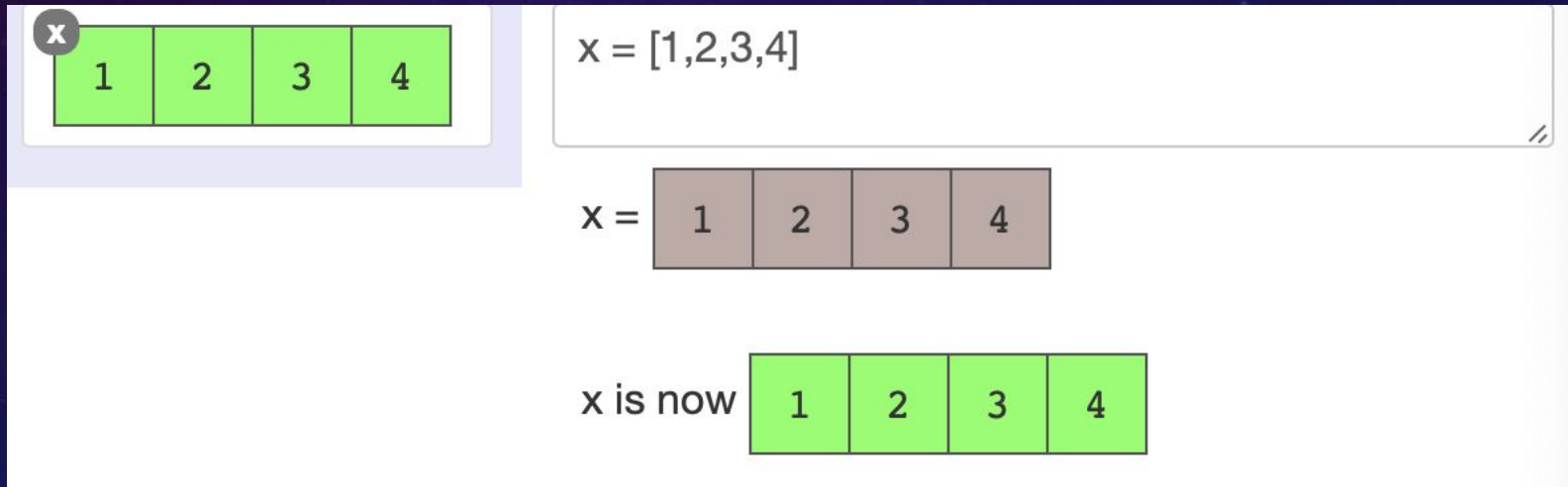
1	3	6	7	9
---	---	---	---	---

- Matrix: A two-dimensional grid of numbers

7	3	9
5	7	2

# Creating Vectors

- Use the square brackets `[ ]` to create a vector.
- Elements may be separated by spaces or commas.



The image shows a MATLAB-style interface for creating a vector. On the left, a variable `x` is shown with a value of `[1, 2, 3, 4]`, represented by a horizontal row of four green boxes containing the numbers 1, 2, 3, and 4. To the right, a text input field contains the code `x = [1,2,3,4]`. Below this, the variable `x` is shown with a value of `[1, 2, 3, 4]`, represented by a horizontal row of four brown boxes containing the numbers 1, 2, 3, and 4. At the bottom, the text `x is now` is followed by a horizontal row of four green boxes containing the numbers 1, 2, 3, and 4.

`x`

`x = [1,2,3,4]`

`x =` `[1, 2, 3, 4]`

`x is now` `[1, 2, 3, 4]`

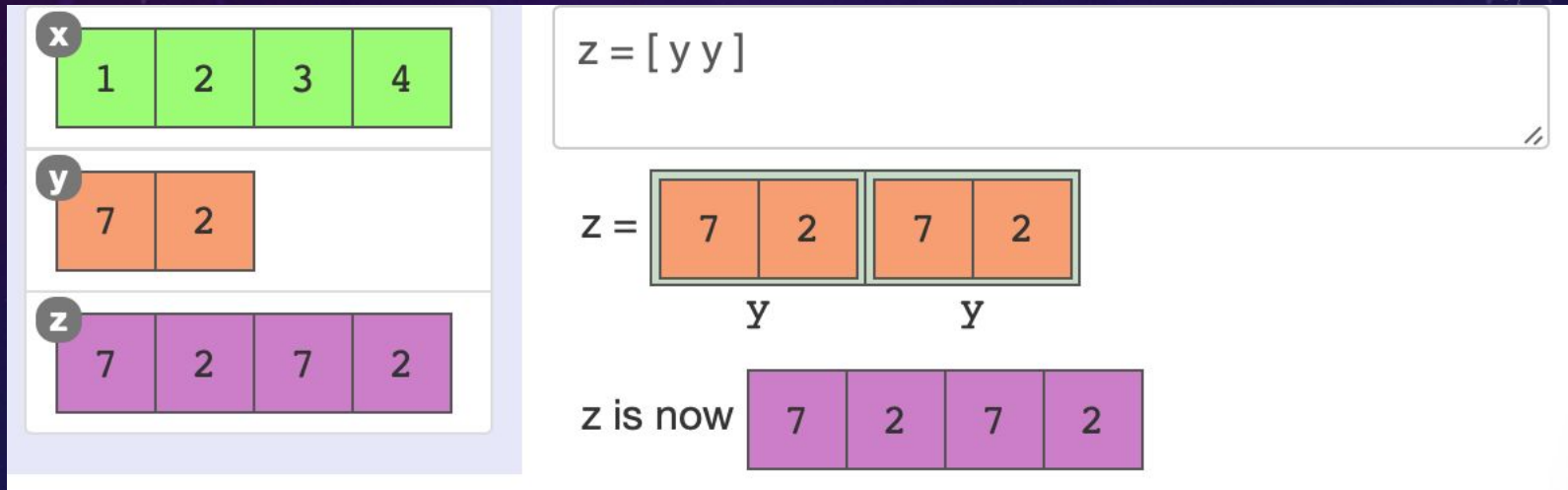
# Creating Vectors

- Use the square brackets `[ ]` to create a vector.
- Elements may be separated by spaces or commas.

The diagram illustrates the process of creating and updating a vector in MATLAB. On the left, a variable `x` is shown as a horizontal vector of four green boxes containing the values 1, 2, 3, and 4. Below it, a variable `y` is shown as a horizontal vector of two orange boxes containing the values 7 and 2. On the right, a code editor shows the command `y = [7 2]` being entered. Below the code editor, the updated state is shown: `y` is now a horizontal vector of two orange boxes containing the values 7 and 2.

# Creating Vectors

- An "element" might also be another vector, which is essentially pasted into the new one.





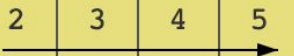
# Creating Ranges of Values

- Use the colon (:) operator to create evenly-spaced vectors.

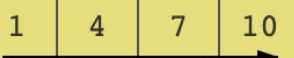
first:step:last

- step may be omitted. It defaults to 1.

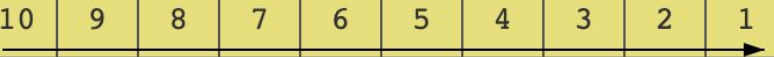
2:5



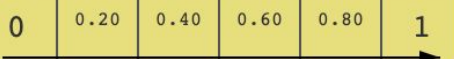
1:3:12



10:-1:1



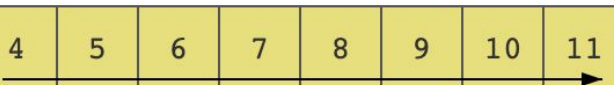
0:0.2:1



1:3:6



4:11



# Creating Matrices

- A matrix is also created with `[]`
- Rows are separated with a semicolon ";" (or a newline)

```
x = [3,7;2,8]
```

x =

3	7
2	8

x is now

3	7
2	8

```
y = [1:4;4:-1:1]
```

y =

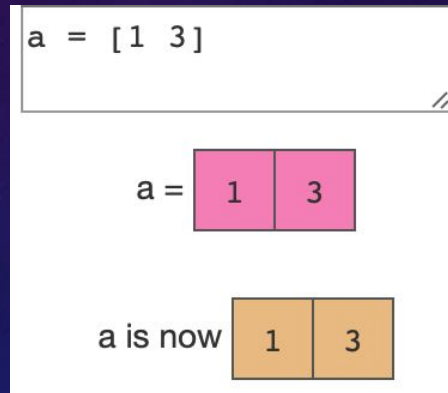
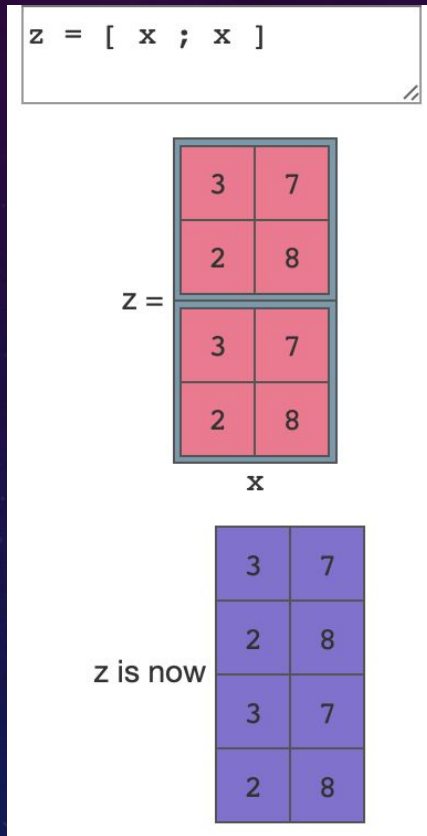
1	2	3	4
4	3	2	1

y is now

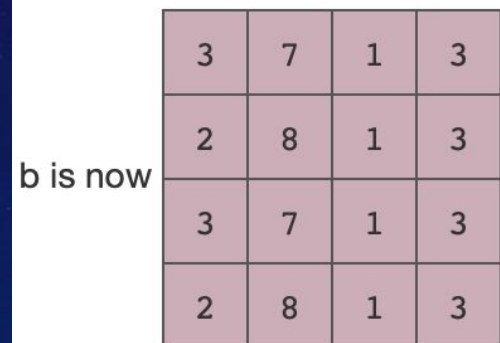
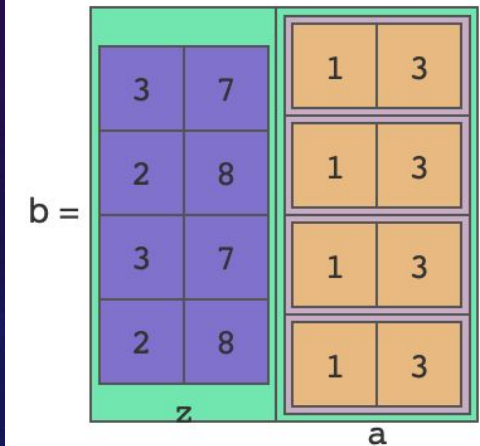
1	2	3	4
4	3	2	1

# Creating Matrices

□ You can use smaller matrices as components.



```
b = [ z [a;a;a;a] ]
```







# Exercise: Creating Matrices

- Use either MATLAB or [lobster.eecs.umich.edu/matlab](http://lobster.eecs.umich.edu/matlab)
- Create these matrices (reuse is your friend):

3	6
9	8

1	6	11	16	21
---	---	----	----	----

1	6	11	16	21	3	6
1	6	11	16	21	9	8

1
2
3
4

# Solution: Creating Matrices

3	6
9	8

$x = [3, 6; 9, 8]$

1	6	11	16	21
---	---	----	----	----

$y = 1:5:21$

1	6	11	16	21	3	6
1	6	11	16	21	9	8


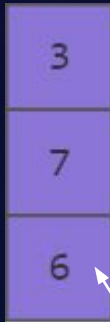
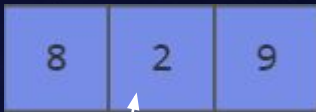
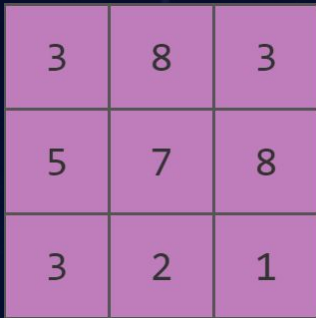
$z = [[y; y], x]$

1
2
3
4

$w = [1; 2; 3; 4]$

# Everything in MATLAB is an Array

□ MATLAB treats all data as a grid-like structure called an **array**.

	Scalar	Vector	Matrix	...
Number of Dimensions	0	1	2	...
Size	1x1	$m \times 1$ or $1 \times n$	$m \times n$	...
Examples	<div>1x1 </div>	<div><div>3x1 </div><div>1x3 </div><div>"row vector"</div><div>"column vector"</div></div>	<div>3x3 </div> <div>...</div>	

# Generalizing Your Code for Any Size Array

- `numel(x)` yields the # of elements in `x`.
- `length(x)` yields the # of elements along the **longest dimension** of `x`.
- `size(x)` yields a vector with the # of elements along each dimension of `x`.
  - For 2D matrices, this is a vector containing: [# of rows, # of cols]

2	1	4
1	3	7

```
numel: 6  
length: 3  
size: [2,3]
```

1
1
1

```
numel: 3  
length: 3  
size: [3,1]
```

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

```
numel: 16  
length: 4  
size: [4,4]
```



# Addition with Arrays and Scalars

- Adding a scalar to an array **just adds it to each element.**

Diagram illustrating scalar addition to an array. A 1D array  $z$  with elements  $[1, 2, 3]$  is added to a scalar  $5$ . The result is an array with elements  $[6, 7, 8]$ .

1	2	3
---	---	---

 + 5 = 

6	7	8
---	---	---

$z$

- Adding two arrays will add them **element-by-element.**
  - The dimensions of the matrices must match EXACTLY!

Diagram illustrating element-by-element addition of two 3x3 matrices. Matrix  $x$  (purple) is added to matrix  $y$  (orange) to produce the result matrix (green).

1	3	5
3	3	1
2	1	5

 + 

7	3	4
6	7	2
7	1	2

 = 

8	6	9
9	10	3
9	2	7

$x$   $y$

- This behavior is a specific case of an **array operation.**



# Arithmetic Array Operations

- "Array operations" work with arrays **element-by-element**.
- The dimensions of the two arrays must match EXACTLY!

	Array Operator	Matrix Operator
Addition	+	+
Subtraction	-	-
Multiplication	.*	*
Division	./	/
Exponentiation	.^	^

# Operating on Matrices and Vectors

*In MATLAB, anything that you can do with scalars, you can do with arrays (i.e. vectors and matrices).*

**ans**

3	3	4
5	6	6

**a**

3	4	2
6	5	1

**b**

2	1	4
1	3	7

**c**

1
---

**d**

1	7
2	5

$a + b - 2$

3	4	2
6	5	1

a

+

2	1	4
1	3	7

b

-

2
---

ans =

3	3	4
5	6	6



4 min

## Exercise: Arithmetic Array Operations

□ Given these variables:

3	4	2
6	5	1

a

2	1	4
1	3	7

b

1
---

c

1	7
2	5

d

□ Find the result of the following expressions:

$$a * b$$

$$b - c .* 2$$

$$d ./ c$$

$$a + d$$

$$10 - d$$

$$a .* b$$

# Solution: Arithmetic Array Operations

□ Given these variables:

3	4	2
6	5	1

a

2	1	4
1	3	7

b

1
---

c

1	7
2	5

d

□ Find the result of the following expressions:

$a * b$

	4	
	15	

Don't forget the dot in .\*!

$b - c .* 2$

0	-1	2
-1	1	5

$d ./ c$

1	7
2	5

$a + d$

Error:  
Dimension mismatch!

$10 - d$

9	3
8	5

$a .* b$

6	4	8
6	15	7

# Break Time

We'll start again in 5 minutes.

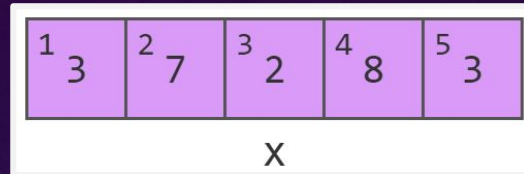




# Vector Indexing

- All elements in a vector are numbered by their **index**, starting at 1.

$x = [3, 7, 2, 8, 3]$



- To select elements, use the **indexing** operator, `( )`.
- Specify the index of the element you want in order to select it.

Left Screenshot:

```
ans
7
```

```
x
3 7 2 8 3
```

Right Screenshot:

```
x(2)
```

```
x
1 3 2 7 3 2 4 8 5 3
```

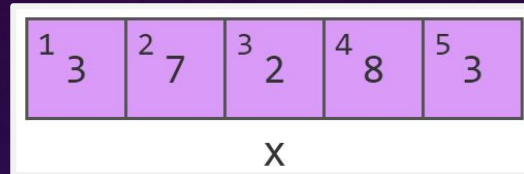
x

```
ans =
7
```

# Vector Indexing

- All elements in a vector are numbered by their **index**, starting at 1.

$x = [3, 7, 2, 8, 3]$



- To select elements, use the **indexing** operator, ( ).
  - Specify the index of the element you want in order to select it.
  - You can use the result in an expression ...

ans

5

x

3 7 2 8 3

x(5) + x(3)

1 3 2 7 3 2 4 8 5 3 + 1 3 2 7 3 2 4 8 5 3

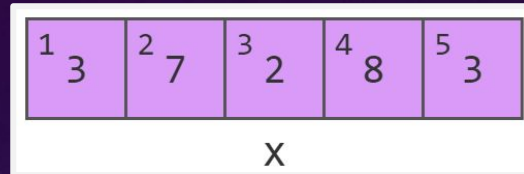
x x

ans = 5

# Vector Indexing

- All elements in a vector are numbered by their **index**, starting at 1.

$x = [3, 7, 2, 8, 3]$



- To select elements, use the **indexing** operator, `( )`.
  - Specify the index of the element you want in order to select it.
  - You can use the result in an expression ... and/or store it into a variable.

```
ans
5

x
3 7 2 8 3

y
5

y = x(5) + x(3)

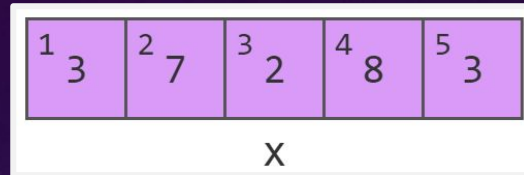
y =
3 7 2 8 3 + 3 7 2 8 3
x x

y is now
5
```

# Vector Indexing

- All elements in a vector are numbered by their **index**, starting at 1.

$x = [3, 7, 2, 8, 3]$



- To select elements, use the **indexing** operator, ( ).
  - Specify the index of the element you want in order to select it.
  - You can use the result in an expression or store it into a variable.
  - The **end** keyword gives the last index. (Why is this useful?)

ans

5

x

3

7

2

8

3

y

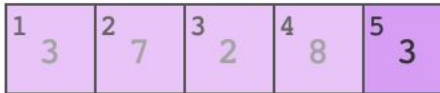
5

z

3

z = x(end)

z =



x

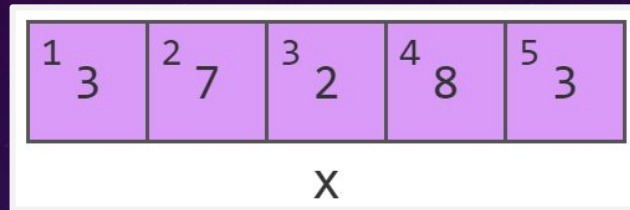
z is now

3

# Selecting Multiple Elements

- All elements in a vector are numbered, starting at 1.

$x = [3, 7, 2, 8, 3]$



- To select **multiple elements**, put a **vector of indices** in the ( ).

The image shows two side-by-side examples of selecting elements from a vector  $x$ .

**Left Example:** A light blue box contains two parts. The top part shows a green box labeled 'ans' containing the values 3, 7, and 8. The bottom part shows a purple box labeled 'x' containing the values 3, 7, 2, 8, and 3.

**Right Example:** A white box contains the MATLAB code `x([1,2,4])`. Below this, a diagram shows the vector  $x$  with indices 1 to 5 and values 3, 7, 2, 8, 3. The element at index 3 (the value 2) is faded. Below the vector, the result is shown as `ans =` followed by a green box containing the values 3, 7, and 8.



# Selecting Multiple Elements

- All elements in a vector are numbered, starting at 1.

$x = [3, 7, 2, 8, 3]$

1	3	2	7	3	2	4	8	5	3
x									

- To select **multiple elements**, put a **vector of indices** in the ( ).

**ans**

3	3	2	8
---	---	---	---

**x**

3	7	2	8	3
---	---	---	---	---

`x([5, 5, 3, 4])`

1	3	2	7	3	2	4	8	5	3
x									

`ans =`

3	3	2	8
---	---	---	---

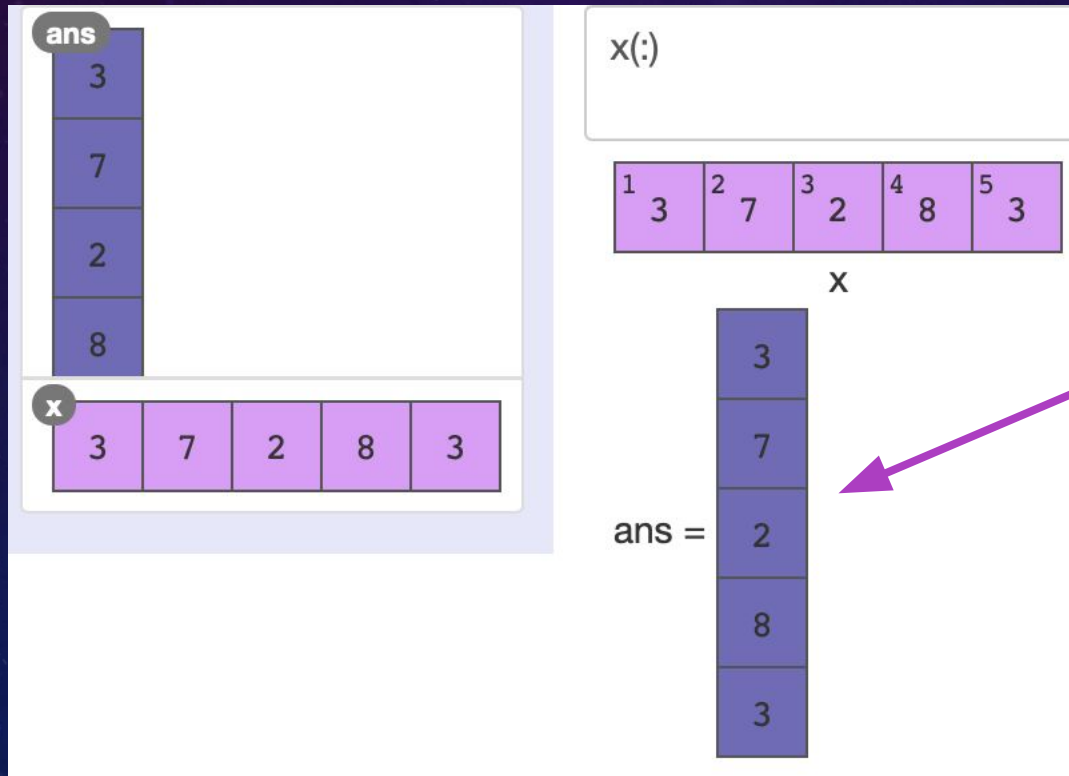
# Selecting Multiple Elements

- All elements in a vector are numbered, starting at 1.

$x = [3, 7, 2, 8, 3]$

<sup>1</sup> 3	<sup>2</sup> 7	<sup>3</sup> 2	<sup>4</sup> 8	<sup>5</sup> 3
x				

- To select **all elements**, use the **:** operator in the ( ).



Note that this yields  
a column matrix  
when read out



1 min

# Minute Exercise: Matching

□ Which code snippet goes with which output?

1	3	2	7	3	2	4	8	5	3
x									

A	<code>x([5,1,1])</code>
B	<code>x(2)</code>
C	<code>x(2:4)</code>
D	<code>x([5:-1:1])</code>

1

7	2	8
---	---	---

2

3	8	2	7	3
---	---	---	---	---

3

3	3	3
---	---	---

4

7
---

# Indexing and Assignment

- Use indexing in the LHS of an assignment to select the elements you want to write over.
- The RHS specifies the new values.

<sup>1</sup> 3	<sup>2</sup> 7	<sup>3</sup> 2	<sup>4</sup> 8	<sup>5</sup> 3
x				

<b>x</b>	3	0	2	8	3
----------	---	---	---	---	---

$x(2)=0$

<sup>1</sup> 3	<sup>2</sup> 7	<sup>3</sup> 2	<sup>4</sup> 8	<sup>5</sup> 3	=	0
----------------	----------------	----------------	----------------	----------------	---	---

x

x is now

3	0	2	8	3
---	---	---	---	---



1 min

## Minute Exercise: Vector Indexing

- Replace the elements with odd indices by 0.

1	3	2	7	3	4	4	9	5	2	6	8	7	3
X													



# Minute Solution: Vector Indexing

□ There are lots of ways to do this. A few examples:

□ `x(1) = 0;`

`x(3) = 0;`

`x(5) = 0;`

`x(7) = 0;`

□ `x([1,3,5,7]) = 0;`

□ `x(1:2:7) = 0;`

□ `x(1:2:end) = 0;`

The **end** keyword gives the last index of the array. This is equivalent to `x(1:2:7)` in our specific case.

# Matrices and Row/Column Indexing

- Specify separate row/column indices.
- Rows first, then columns, separated by a comma.
- Again, we can use either a **single number**,

**ans**

4
---

**x**

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

**x(2,3)**

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

**x**

**ans =**

4
---

# Matrices and Row/Column Indexing

- Specify separate row/column indices.
- Rows first, then columns, separated by a comma.
- Again, we can use either a **single number**, a **matrix of several**,

ans

4	3
4	1

x

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

x([1,3], [1,4])

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

x

4	3
4	1

ans =

4	3
4	1

ans

3	4	4
---	---	---

x

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

x(2, [4, 3, 3])

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

x

3	4	4
---	---	---

ans =

3	4	4
---	---	---

# Matrices and Row/Column Indexing

□ Specify separate row/column indices.

□ Rows first, then columns, separated by a comma.

□ Again, we can use either a **single number**, a **matrix of several**, or a **:**.

The **:** is useful for selecting full rows and/or columns.

ans

4

6

8

1

x

4

2

6

3

7

2

4

3

4

6

8

1

x(3,:)

4

2

6

3

7

2

4

3

4

6

8

1

x

ans =

4

6

8

1

ans

2

6

2

4

6

8

x

4

2

6

3

7

2

4

3

4

6

8

1

x(:,[2 3])

4

2

6

3

7

2

4

3

4

6

8

1

x

2

6

2

4

6

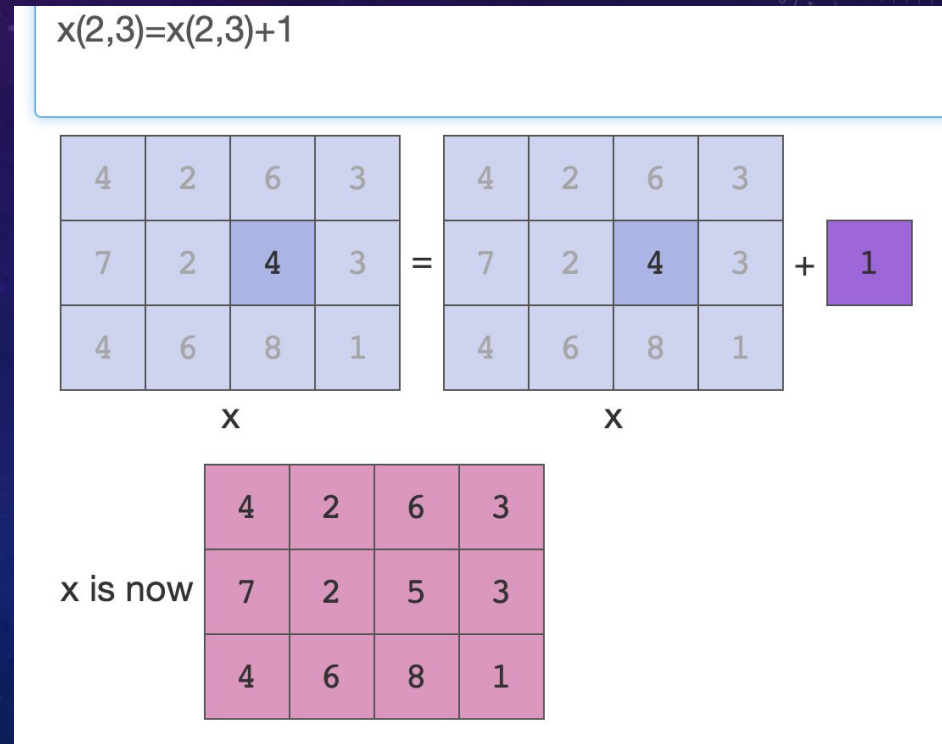
8

ans =

# Updating Elements with Indexing

- Recall that in an update assignment, the same variable appears on the LHS and RHS (e.g.  $x = x + 1$ );
- To **update element(s)** with indexing...
  - The indexing expression for those elements appears on the LHS and RHS.

$$x(2,3) = x(2,3) + 1$$





# Examples: Row/Column Indexing

□ Here are several examples of row/column indexing tasks:

□ Set all elements in the first row to 10.      $x(1,:) = 10;$

$x(1,:) = 10$

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

= 

10
----

x

x is now

10	10	10	10
7	2	4	3
4	6	8	1

# Examples: Row/Column Indexing

□ Here are several examples of row/column indexing tasks:

□ Increase the bottom-right element by 1.  $x(\text{end}, \text{end}) = x(\text{end}, \text{end}) + 1;$

ans

4

x

4	2	6	3
7	2	4	3
4	6	8	2

$x(\text{end}, \text{end}) = x(\text{end}, \text{end}) + 1;$

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

x

=

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

x

+

1
---

x is now

4	2	6	3
7	2	4	3
4	6	8	2

# Examples: Row/Column Indexing

□ Here are several examples of row/column indexing tasks:

□ Swap the first and last column.

`x(:,[1,end]) = x(:,[end,1]);`

`x(:,[1,end]) = x(:,[end,1])`

10	10	10	10		10	10	10	10
7	2	4	3	=	7	2	4	3
4	6	8	1		4	6	8	1
x					x			

x is now	10	10	10	10
	3	2	4	7
	1	6	8	4

# Rearranging Using Indexing

Just as with regular indexing, the order in which you specify rows/columns matters. You can use this to rearrange rows/columns.

`x(:,[3,2,4])`

1. 2. 3. 4

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



ans =

6	2	3
4	2	3
8	6	1

# Removing Rows/Columns with "Delete Syntax"

- To remove rows or columns from a matrix, assign `[]` to them.

1	2	3
4	5	6
7	8	9

**X**

$x(2,:) = [];$

1	2	3
7	8	9

**X**

$x(:, [1,3]) = [];$

2
8

**X**

- Caution! Generally you only want to do this with whole rows/columns (i.e. use the `:` somewhere).
- An alternate way to do this is just select all rows/columns you want to keep.



# Creating Similarly-Sized Matrices

- Use the `size` function to get the size of one matrix, and then create another matrix of the same size.

2	1	4
1	3	7

A

0	0	0
0	0	0

`zeros(size(A))`



## Exercise: Row/Column Indexing

- Use row/column indexing to perform the following operations. Each one should work regardless of the size of the matrix.
- Double the value of each element in the first row of a matrix.
- Create a new matrix from only the odd numbered columns in an original.
- Set the corner elements of a matrix to all zeros.

x

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

# Solution: Row/Column Indexing

- Use row/column indexing to perform the following operations. Each one should work regardless of the size of the matrix.

- Double the value of each element in the first row of a matrix.

$$x(1,:) = 2 * x(1,:)$$

- Create a new matrix from only the odd numbered columns in an original.

$$y = x(:,1:2:end)$$

- Set the corner elements of a matrix to all zeros.

$$x([1,end],[1,end]) = 0$$

# FYI: Matrices and Linear Indexing

- Each element also has a sequential index used in linear indexing (as opposed to the row/column indexing we have been using).
- Ordered first by columns, then by rows.
  - This is called "column-major" order.
  - Warning! It's easy to get this backward!
- We *highly* recommend using row/column indexing because it generally is clearer and leads to fewer headaches, but some MATLAB help files use linear indexing, so now you know what it is. 😊

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

X

x([1,2,4])

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

X

ans =	2	0	4
-------	---	---	---





4 min

## Challenge: Indexing (This is tricky!)

Write a single line of code to shift all columns one to the left and then move the first column to be the last column.



# Challenge Solution: Indexing

Write a single line of code to shift all columns one to the left and then move the first column to be the last column.

```
x = x(:, [2:end,1])
```