# While you are waiting

Unscramble the following letters to form a September word

**tuumalan**

# ENGR 101 – Lecture 3

Vectors and Matrices

# 00_Todays_Lecture

- I shall be turning the lecture into a giant lab today
- You may find a .pdf version of Today's Lecture on our class Google Drive (accessible from our CANVAS site)
  - Look for folder **00_Todays_Lecture**
- If you have a laptop, then you can follow links to some simple exercises
- If you don't, please partner with a neighbour(s)
- Solutions will be published after lecture, plus extra slides

# Lecture Goals

- Previous lecture: Intro to MATLAB: variables, expressions and operators
  - Introduction to MATLAB
  - Variables
  - Expressions
  - Operators
  - Suggested readings, Attaway, Chap 1.1-1.4 and 3.1-3.2

- Today's lecture: Vectors and Matrices
  - Working with *Lobster*
  - Vectors and indexing
  - Matrices and 2-types of matrix indexing
  - Suggested readings, Attaway, Chap 2.1 and 2.3

# MATLAB programming – precedence

- **Order of precedence**
  1. **(highest) parentheses**
  2. **exponentiation**
  3. **negation**
  4. **multiplication, division**
  5. **addition, subtraction**
  6. 
  7. 
  8. 
  9. 
  10. **(lowest) assignment (=)**

***if two or more operations have the same precedence, the expression is executed from left to right***

# MATLAB programming – Your turn

- **Connect the left side with it's right side?**

A) 7+9/2

B) (7+9)/2

C) 27.^(1/3)+32.^0.2

D) 27.^1/3+32.^0.2

1) 8

2) 5

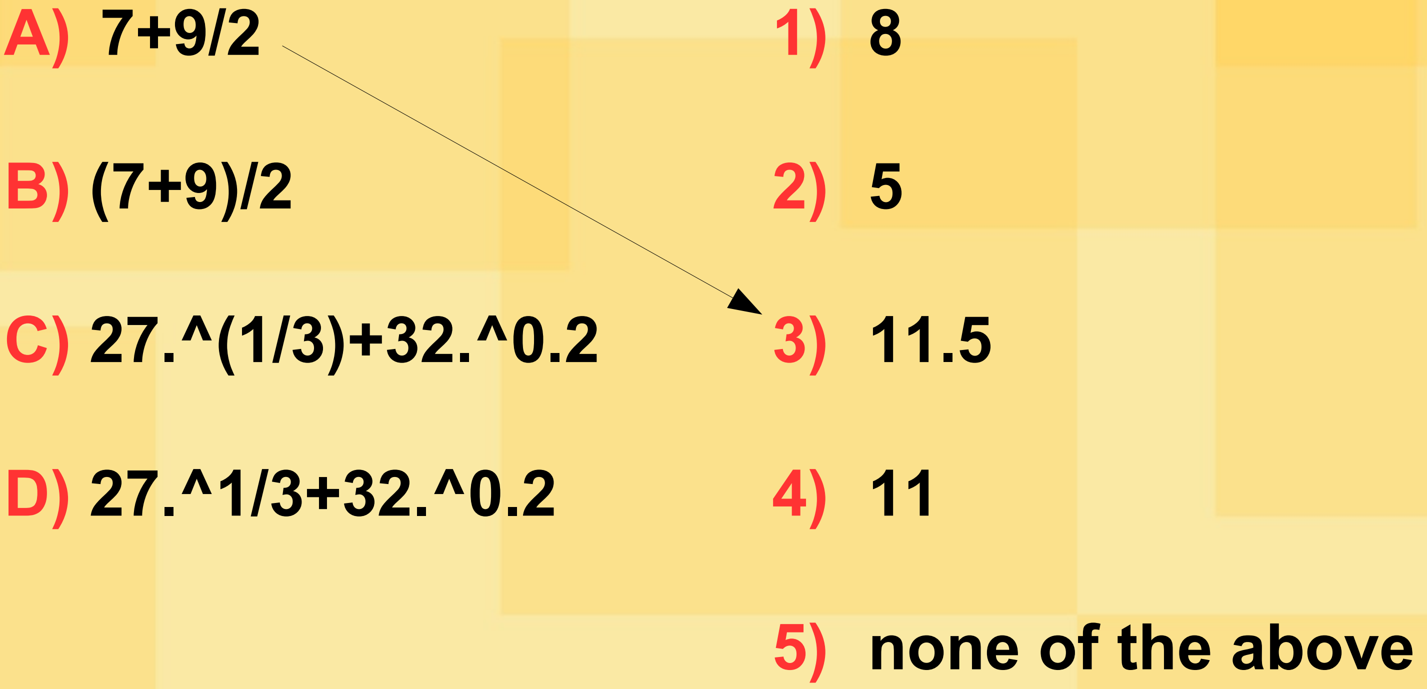
3) 11.5

4) 11

5) none of the above

# MATLAB programming – Your turn

- Connect the left side with it's right side?



A) 7+9/2

B) (7+9)/2

C) 27.^(1/3)+32.^0.2

D) 27.^1/3+32.^0.2

1) 8

2) 5

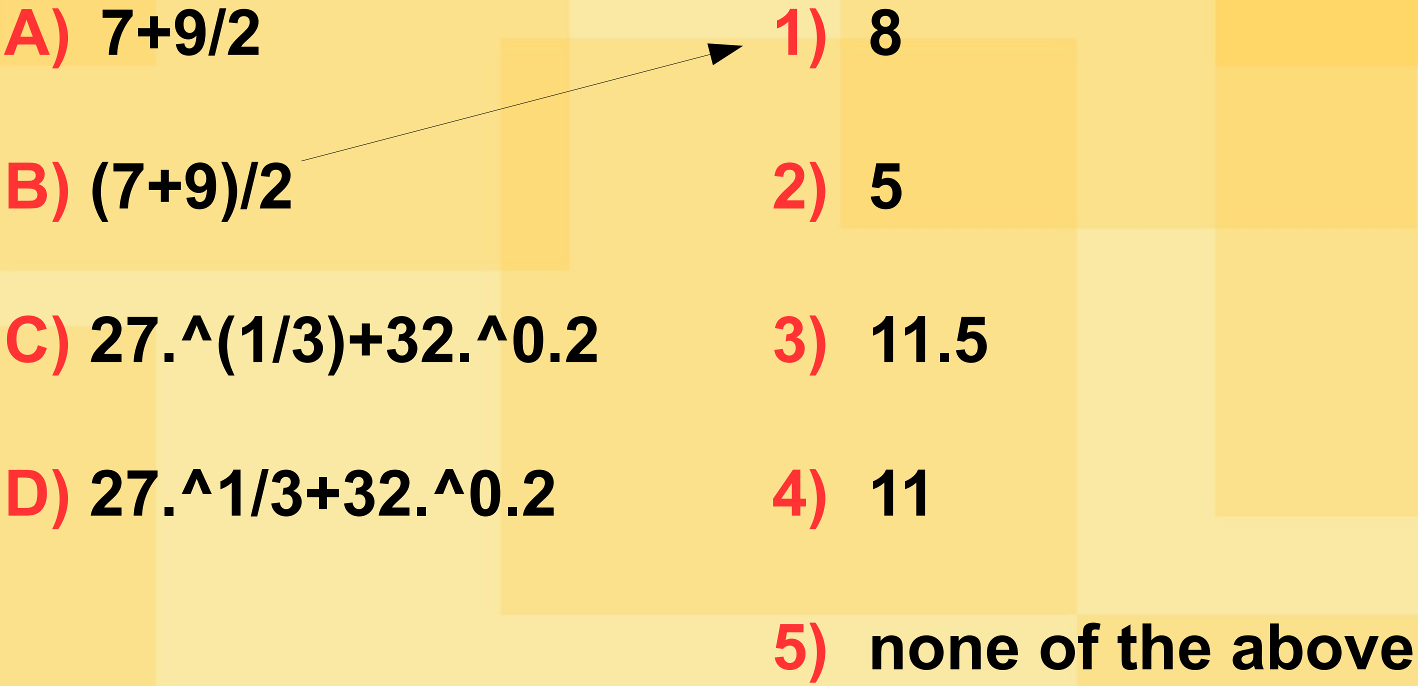
3) 11.5

4) 11

5) none of the above

# MATLAB programming – Your turn

- Connect the left side with it's right side?



A) 7+9/2

B) (7+9)/2

C) 27.^(1/3)+32.^0.2

D) 27.^1/3+32.^0.2

1) 8

2) 5

3) 11.5

4) 11

5) none of the above

# MATLAB programming – Your turn

- Connect the left side with it's right side?

| | |
|---|---|
| A) 7+9/2 | 1) 8 |
| B) (7+9)/2 | 2) 5 |
| C) 27.^(1/3)+32.^0.2 | 3) 11.5 |
| D) 27.^1/3+32.^0.2 | 4) 11 |
| | 5) none of the above |

# MATLAB programming – Your turn

- Connect the left side with it's right side?

A) 7+9/2    1) 8

B) (7+9)/2    2) 5

C) 27.^(1/3)+32.^0.2    3) 11.5

D) 27.^1/3+32.^0.2 → 4) 11

5) none of the above

# Vectors and Matrices

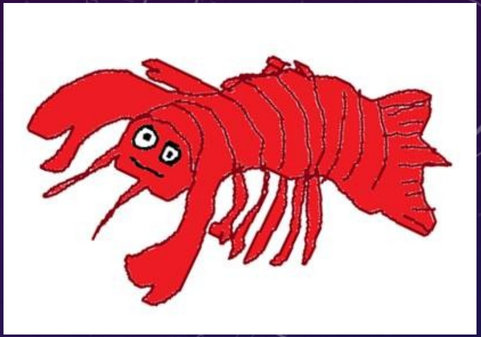
- MATLAB permits easy vector manipulation
  - For example, in the expression `C = A + B`, the variables may be vectors as easily as they may be scalars
- Scalar: A single number or element
- Vector: A one-dimensional sequence of numbers or elements

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

- Matrix: A two-dimensional grid of numbers or elements

| | | |
|---|---|---|
| 7 | 3 | 9 |
| 5 | 7 | 2 |

# Lobster



- We'll use a program visualization tool called "Lobster" throughout the course for examples and in-class exercises.
- Lobster supports interactive MATLAB coding (not functions).
- Go to **lobster.eecs.umich.edu/matlab**
- Type a line of code, hit enter, and Lobster will visualize it.

https://lobster.eecs.umich.edu/matlab

# Creating Vectors

- Use square brackets [ ] to create a vector.
- Elements may be separated by spaces or commas.
- IMPORTANT: An "element" might also be another vector, which is essentially pasted into the new one.

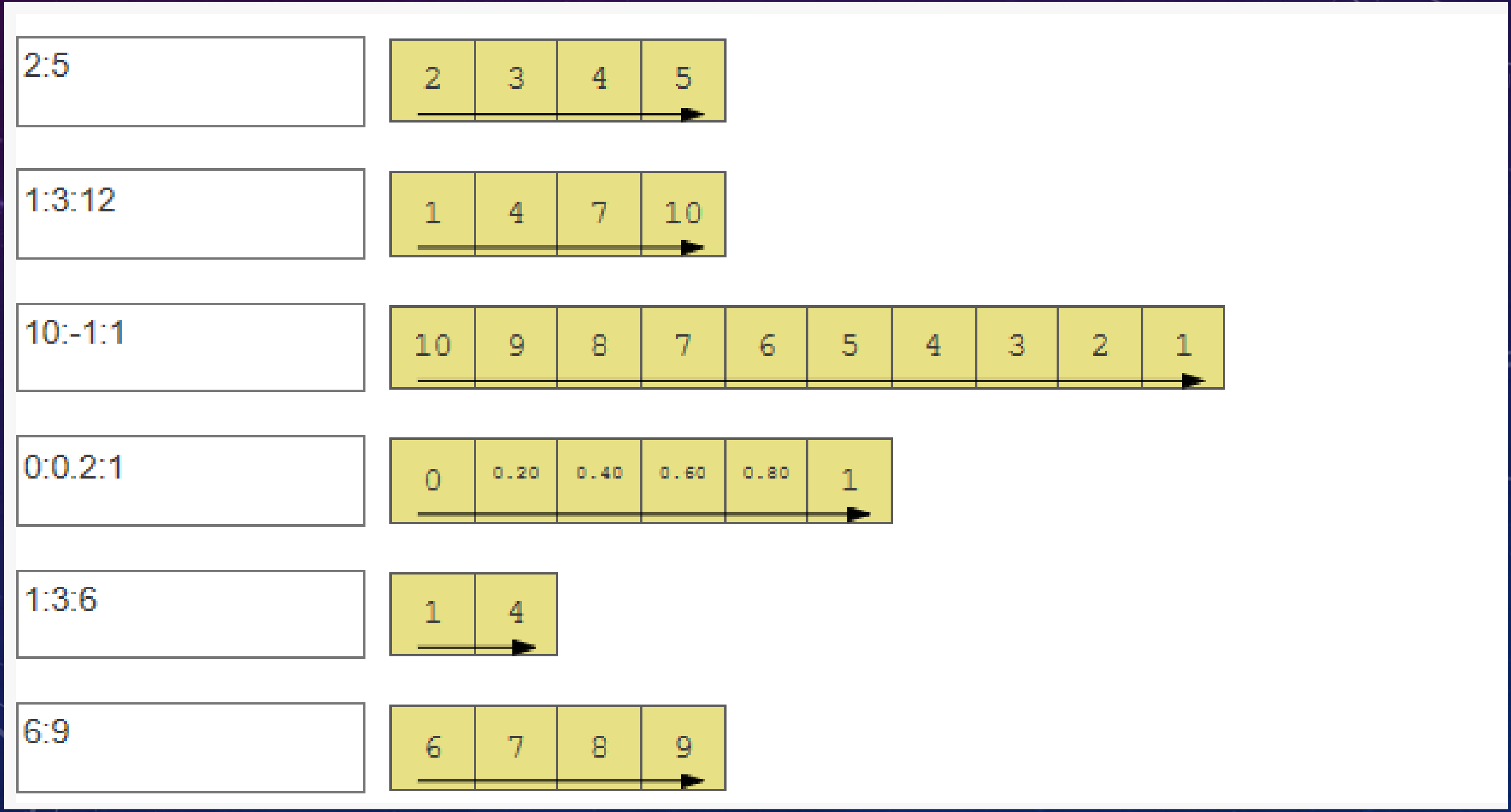
https://goo.gl/8yiHzx

# Creating Ranges of Values

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

  `first:step:last`

- `If` `step` is omitted, it defaults to 1



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 2:5 | 2 | 3 | 4 | 5 | | | | | | |
| 1:3:12 | 1 | 4 | 7 | 10 | | | | | | |
| 10:-1:1 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 0:0.2:1 | 0 | 0.20 | 0.40 | 0.60 | 0.80 | 1 | | | | |
| 1:3:6 | 1 | 4 | | | | | | | | |
| 6:9 | 6 | 7 | 8 | 9 | | | | | | |

# Creating Matrices

- A matrix is also created with [ ]
  - Rows are separated with a semicolon ";" (or a newline)
  - You can use smaller matrices as components.
  - If things don't line up (e.g. different row sizes), you'll get an error.
  -
  - lobster requires using ; to separate rows and , to separate columns

https://goo.gl/XLBa1D

4
min

# Your turn: Creating Matrices

- Use either MATLAB or lobster.eecs.umich.edu/matlab
- Create these matrices:

| | |
|---|---|
| 3 | 6 |
| 9 | 8 |

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

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |

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

# 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]
```

# Transpose, `flipud`, `fliplr`

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

d

- The transpose operation ( ' ) takes a matrix and produces a copy with the rows turned into columns.
- The `flipud` ("flip up/down") and `fliplr` ("flip left/right") functions allow us to reverse all columns or rows.

|   |   |
|---|---|
| 1 | 4 |
| 2 | 5 |
| 3 | 6 |

`d'`

|   |   |   |
|---|---|---|
| 4 | 5 | 6 |
| 1 | 2 | 3 |

`flipup(d)`

|   |   |   |
|---|---|---|
| 3 | 2 | 1 |
| 6 | 5 | 4 |

`fliplr(d)`

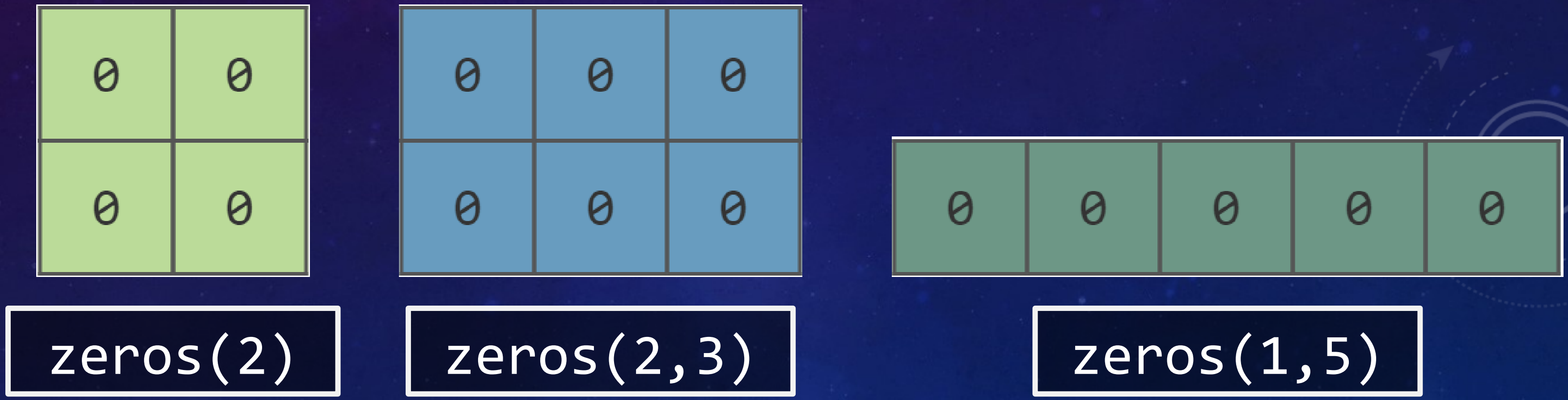
# Everything in MATLAB is an **Array**

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

| | Scalar | Vector | Matrix | … |
|---|---|---|---|---|
| Number of Dimensions | 0 | 1 | 2 | … |
| (row x col) | 1x1 | *m* x 1 or 1 x *n* | *m* x *n* | … |
| Examples | 1x1<br>7 | 3x1<br>3<br>7<br>6<br>"column vector"<br>1x3<br>8 2 9<br>"row vector" | 3x3<br>3 8 3<br>5 7 8<br>3 2 1 | … |

# Functions for Creating Matrices: `zeros`

- `zeros(m,n)`
  - Creates an `m` x `n` matrix of zeros.
- `zeros(m)`
  - Same, but uses `m` for both rows and columns.



| | |
|---|---|
| 0 | 0 |
| 0 | 0 |

`zeros(2)`

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |

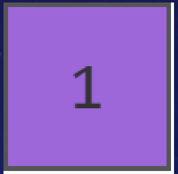
`zeros(2,3)`

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |

`zeros(1,5)`

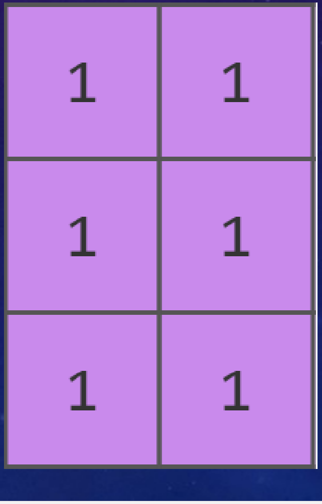
# Functions for Creating Matrices: ones

- `ones(m,n)`
  - Creates an `m` x `n` matrix of zeros.
- `ones(m)`
  - Same, but uses `m` for both rows and columns.



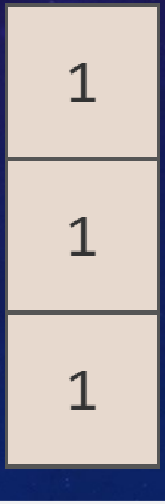
| |
|---|
| 1 |

`ones(1)`



| | |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |

`ones(3,2)`



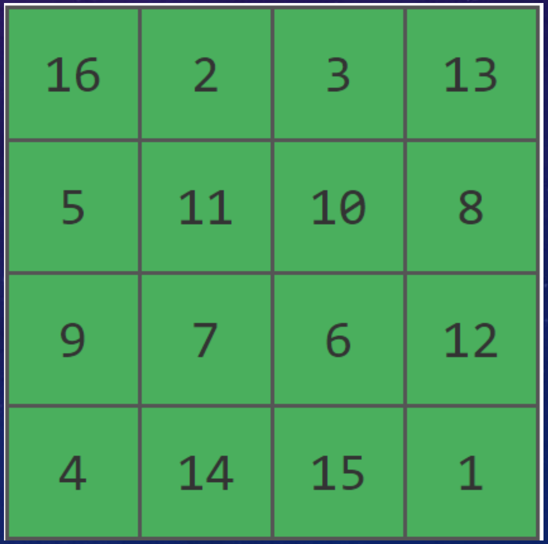
| |
|---|
| 1 |
| 1 |
| 1 |

`ones(3,1)`

# Functions for Creating Matrices: `magic`

- `magic(s)`
  - Creates an *s* x *s* "magic" matrix.
  - Magic because each column sum = each row sum = each diagonal sum.

- We'll use this sometimes to create example matrices.



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

```
magic(4)
```

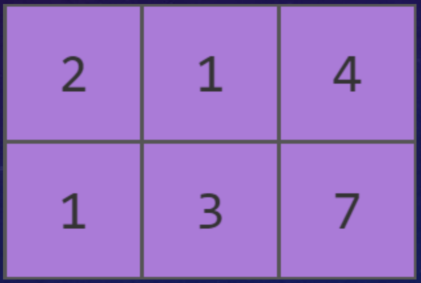
# Functions for Creating Matrices: `repmat` and `reshape`

- `repmat(A,xTimes,yTimes)`
  - Creates a new matrix based on repeating matrix A a certain number of times in the x direction and y direction.

- `reshape(A,numRows,numCols)`
  - Creates a new matrix with the same elements as A, but the requested number of rows and columns. The dimensions must match up with the original number of elements in A.

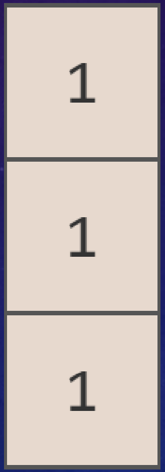
# size, length, and numel functions

- 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 matrices, this is a vector containing: [# of rows, # of cols]



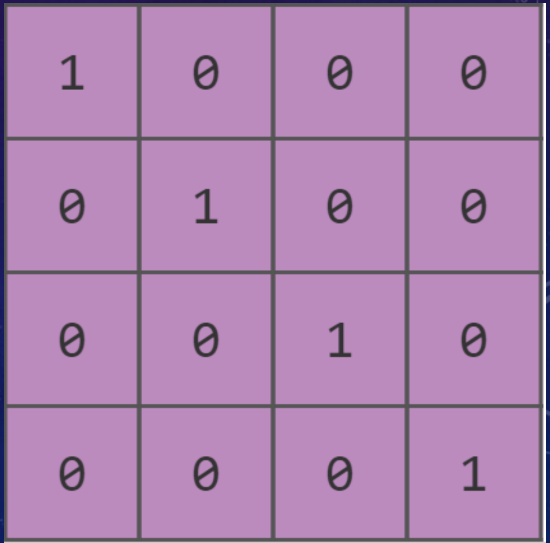
| | | |
|---|---|---|
| 2 | 1 | 4 |
| 1 | 3 | 7 |

```
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]
```

# Please remember the terminology

I have quietly used the following terminology

Operators – in MATLAB, these will be included in many of the statements in your coding (arithmetic operations)

Functions – in MATLAB, these are groups of statements that often use operators to perform a task (trig, logic functions)

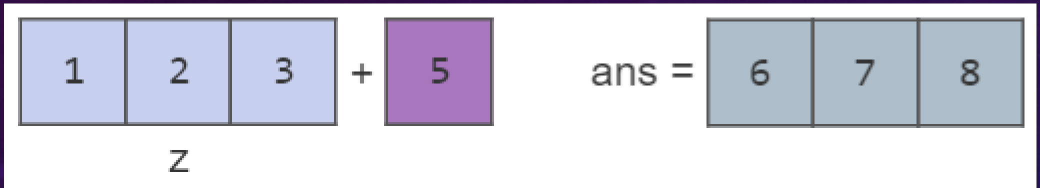
# Operating on Matrices and Vectors

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

https://goo.gl/YqScDE

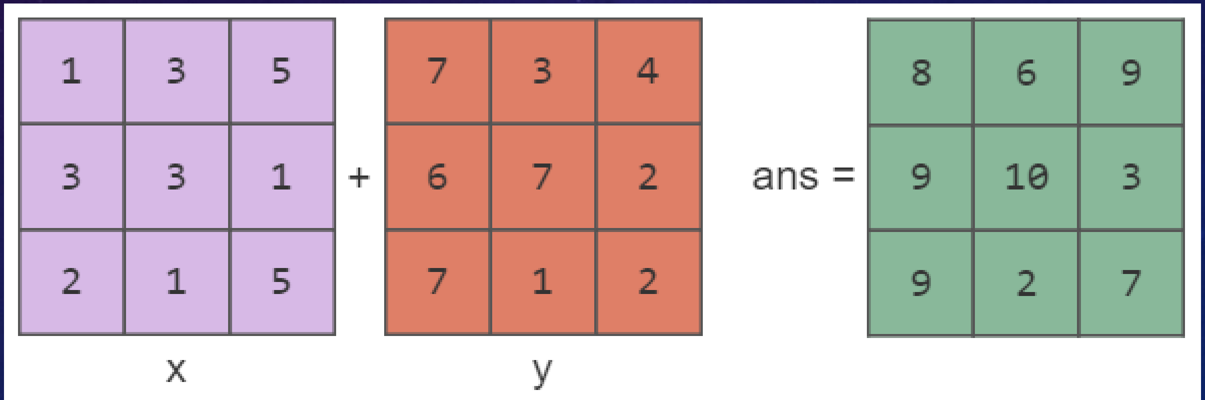
# Addition with Arrays and Scalars

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



$$\underset{z}{\begin{bmatrix}1 & 2 & 3\end{bmatrix}} + \begin{bmatrix}5\end{bmatrix} \qquad \text{ans} = \begin{bmatrix}6 & 7 & 8\end{bmatrix}$$

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



$$\underset{x}{\begin{bmatrix}1 & 3 & 5\\ 3 & 3 & 1\\ 2 & 1 & 5\end{bmatrix}} + \underset{y}{\begin{bmatrix}7 & 3 & 4\\ 6 & 7 & 2\\ 7 & 1 & 2\end{bmatrix}} \qquad \text{ans} = \begin{bmatrix}8 & 6 & 9\\ 9 & 10 & 3\\ 9 & 2 & 7\end{bmatrix}$$

- 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 | .^ | ^ |

MATLAB also supports "matrix operations", which are used for linear algebra operations on 2D matrices. We won't use these in 101.

The "array" and "matrix" versions of + and - do the same thing.

For the other array operations, **don't forget the dot**! You don't want the matrix version.

# Your turn: Arithmetic Array Operations

5
min

- 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 (find the 2 errors !!!):

```
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

Don't forget the dot in .*!

| | | |
|---|---|---|
| [illegible] | ~~4~~ | [illegible] |
| [illegible] | ~~15~~ | [illegible] |

b - c * 2

| | | |
|---|---|---|
| 0 | -1 | 2 |
| -1 | 1 | 5 |

d / c

| | |
|---|---|
| 1 | 7 |
| 2 | 5 |

Using ./ here would be better style.

a + d

Error: Dimension mismatch!

10 – d

| | |
|---|---|
| 9 | 3 |
| 8 | 5 |

a .* b

| | | |
|---|---|---|
| 6 | 4 | 8 |
| 6 | 15 | 7 |

Lobster: goo.gl/mTFWYs

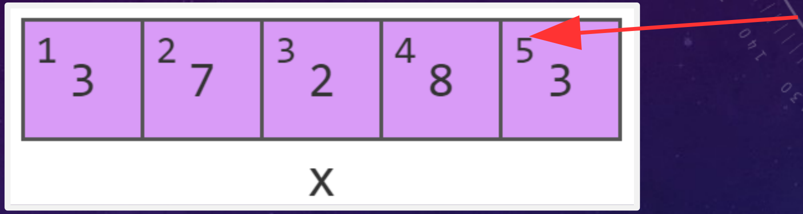
# Break Time: work on project 0

We'll start again in 5 minutes.

# Vector Indexing

- All elements in a vector are indexed, starting with index = 1

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

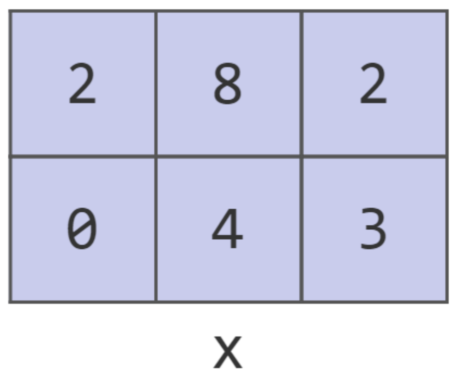


- To select elements, use the **indexing** operator, ( )
  - Either specify the **index** of the element, use a matrix to specify multiple indices, or use : for a range

https://goo.gl/b7QRtT

# Matrix Indexing

- Each element in a matrix also has a sequential index.
- Ordered first by columns, then by rows.
  - This is termed “column-major” order
  - Warning! It's easy to get this wrong!

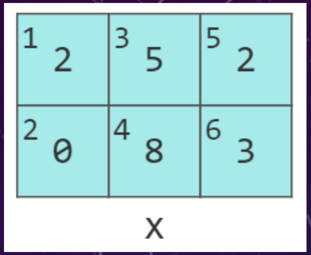


| | | |
|---|---|---|
| 2 | 8 | 2 |
| 0 | 4 | 3 |

x

https://goo.gl/KDfFWp

# Indexing: Reading from a Matrix



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

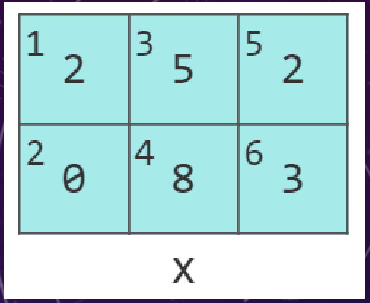
x

- It's like ordering elements of the matrix from a menu:
  - "I'll take a number 2, a number 4, a number 6...."

https://goo.gl/vbJ2hD

Skip !!

# Indexing: Writing into a Matrix



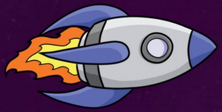
|  |  |  |
|---|---|---|
| 1 2 | 3 5 | 5 2 |
| 2 0 | 4 8 | 6 3 |

x

- Use assignment:
  - The LHS uses indexing to specify which elements to change.
  - The RHS specifies the new values.
  - eg., x([2, 4, 6]) = -1  or  x([2 4 6]) = [4 7 0]

https://goo.gl/vbJ2hD

Skip !!



2 min

# Your turn: Indexing (partner if w/o PC)

- Replace all elements in the 1st row of matrix x with 0. (***follow the link to the matrix***)

https://goo.gl/6KBtjg

# Solution: Indexing

- There are lots of ways to do it. A few examples:
  - `x(1) = 0; x(4) = 0; x(7) = 0`
  - `x([1,4,7]) = 0`
  - `x(1:3:7) = 0`
  - `x(1:3:end) = 0`

The end keyword gives the last index of the array. This is equivalent to `x(1:3:9)` in our specific case.

- However, there's a much more intuitive way to do this…

# Row/Column Indexing

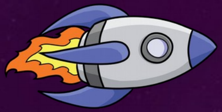
- Instead of selecting elements by specifying an index, we specify BOTH row(s) and column(s)
  - Rows first, then columns, separated by a comma
  - Again, we can use either a single number, a smaller matrix, or a : where the : refers to an entire row or column.

https://goo.gl/QFWxsj

# Row/Column Indexing

- Just as with regular indexing, the order in which you specify rows/columns matters.
  - You can also "order" more than one of each!

https://goo.gl/W2f3HH



6 min

# Your turn: 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 the original.
  - Set the corner elements of the new matrix to 0

- TIP: use the end keyword (***follow the link to the matrix)***

https://goo.gl/GVAS2f

# 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 the new matrix to 0

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

# 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.

6
min

# Challenge: Indexing (These are hard!)

1. Write a single line of code to reverse the columns in a matrix.
2. Write a single line of code to shift all columns one to the left.

   – TIP: starting column = 2, ending column = 1, now try different ways that you might fill in the intervening columns (***follow the link to the matrix***)

https://goo.gl/GVAS2f

# Solution: Fancy Indexing

1. Write a single line of code to reverse the columns in a matrix.

```
x(:, 1:end) = x(:, end:-1:1)
```

2. Write a single line of code to shift all columns one to the left.

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

See you on Monday !!!