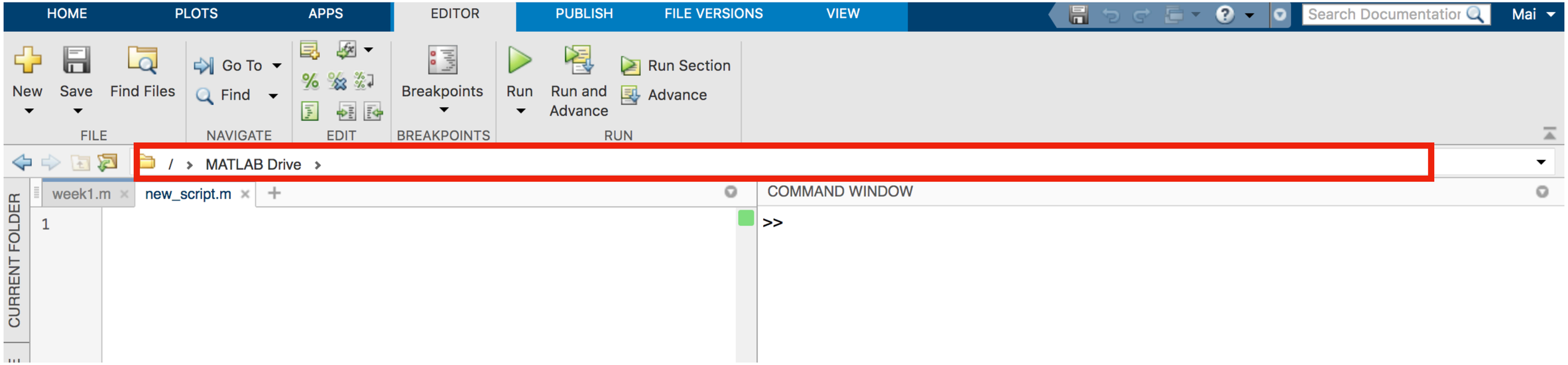
Data types, variables, and selection

Mai Nguyen

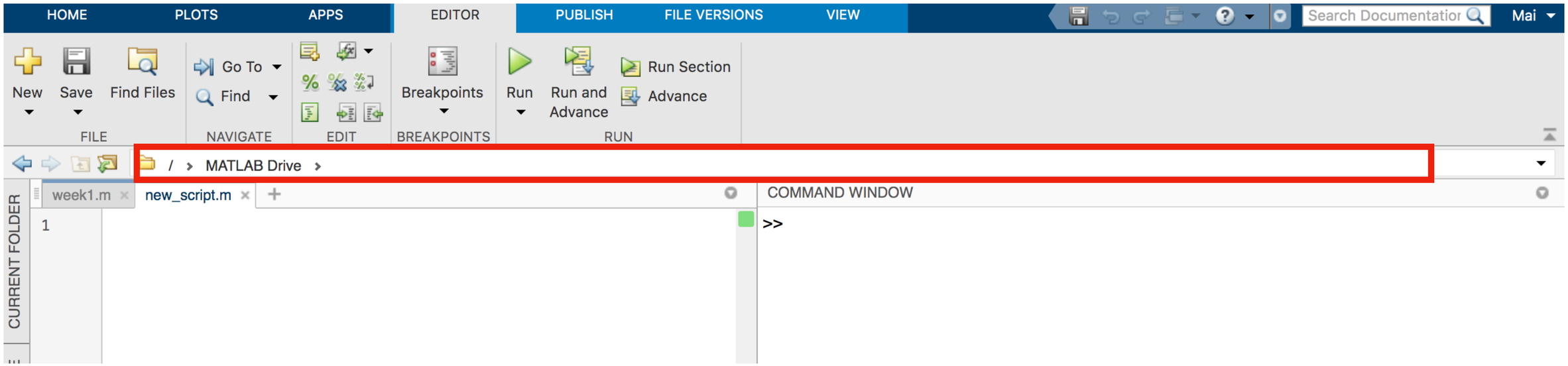
Navigating MATLAB part II

- MATLAB only has access to files in your **working directory** or **current directory**



Navigating MATLAB part II

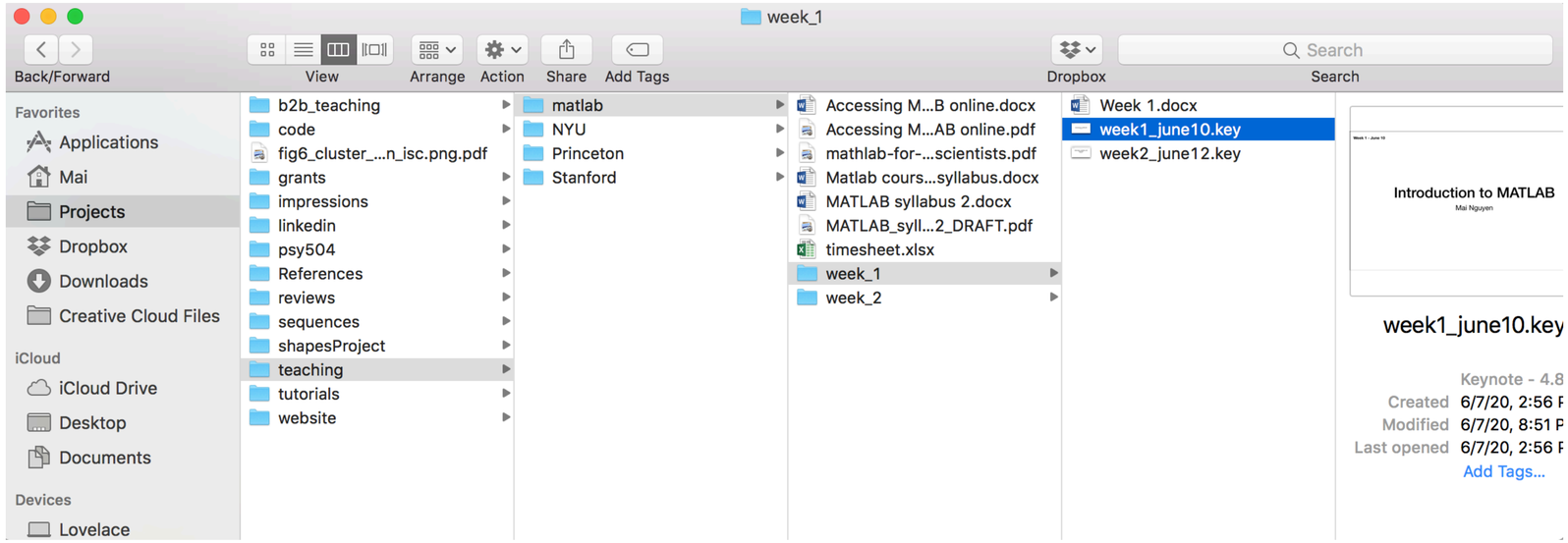
- MATLAB only has access to files in your **working directory** or **current directory**



- This is where “you” are located in the file tree hierarchy. Files are saved here and MATLAB only “knows” what’s in the directory its currently in

Navigating MATLAB part II

- Probably most used to navigating in file window, like Finder



- Navigate using commands (not strictly necessary but easier + same as unix/bash)

Navigating MATLAB part II

1. Print working directory: pwd

```
>> pwd

ans =

    '/MATLAB Drive'
```

2. Print files in directory: ls

```
>> ls
new_script.m  Published  week1.m
```

3. Make a new directory: makedir()

```
>> mkdir('week_1')
>> ls
new_script.m  Published  week_1
```

4. Navigate to new directory: cd

cd directoryName to enter directory
cd .. to go up a directory

```
>> cd week_1/
>> pwd

ans =

    '/MATLAB Drive/week_1'
```

5. Make a new script: edit filename

(also use to open existing file)

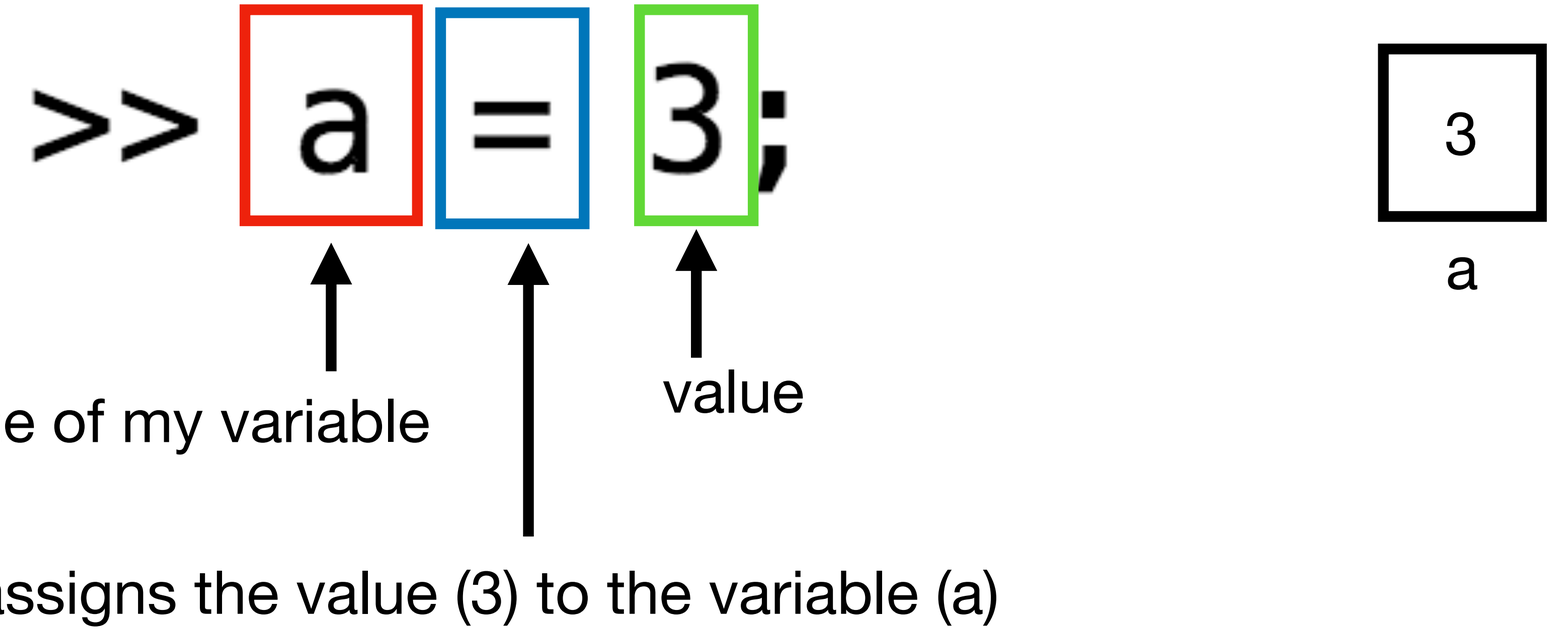
```
>> edit lecture2_script.m
>> ls
lecture2_script.m
```

Overview

- Navigating MATLAB, part II
- **More on variables**
- Data Types
  - Numbers: ints, singles, doubles
  - Characters: strings
  - Collections of numbers: arrays
- Manipulating arrays

More on variables

- Can use variables to hold values



More on variables: naming rules

- Cannot start with a number
  - Valid variables: day1, score_0
  - Invalid variables: 4elements, 100_years_ago
- Can only include letters, numbers, and underscores
  - Valid variables: sokka, f1re_nat10n, appa_the_flying_bison
  - Invalid variables: anng!, me@gmail.com,
- Can’t use reserved keywords: e.g. end, if, else, for
- Can’t use spaces

More on variables: naming rules

Exercise 1: Which are valid variable names?

i_write_lots_and_lots_of_comments

my variable

a10

apples&bananas

counter_1

1stBase

More on variables: naming guidelines

- **Descriptive!!**
    - Bad: a, var, x, y
    - Good: pretest_score, count, min_temp, v1_timeseries
- By convention, start with lowercase (because lazy)
    - Bad: NumPeanuts, Balloons_found
    - Good: numPeanuts, balloons_found
- Use capitalization or underscores for readability
    - Bad: numgoodvoxinroi
    - Good: numGoodVoxInRoi, or num_good_vox_in_roi

Data Types: things you can store in variables

- Text/characters
- Numbers
- Groups of numbers
- Data Types II: structs, cell arrays

Overview

- Navigating MATLAB, part II
- More on variables
- **Data Types**
  - **Numbers: ints, singles, doubles**
  - **Characters: strings**
  - **Collections of numbers: arrays**
- Manipulating arrays

Storing text

- Data type: strings, made up of chars
- Syntax: Surround text with single quotation marks
- Examples

```
>> my_name = 'Mai';
>> class_name = 'intro_matlab';
>> moby_dick = 'Call me Ishmael.';
```

Storing Numbers

- Data types: int, float (single, double). By default all numbers are stored as doubles in MATLAB

- Examples:

```
>> year = 2020;
>> annual_salary = 30000;
>> monthly_salary = annual_salary / 12;
```

Storing groups of numbers: arrays

- Vector: 1-dimensional array
    - Syntax 1: var_name = [val1 val2 val3 val4];
    - Syntax 2: var_name = [val1, val2, val3, val4];
- Examples:

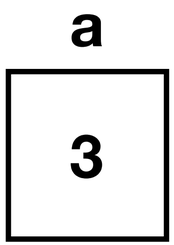
```
>> scores = [1 .86 .92 .71]

scores =

    1.0000    0.8600    0.9200    0.7100
```

```
>> responses = [1 1 3 2 4]

responses =

     1     1     3     2     4
```

Storing groups of numbers: arrays

- Previously: a variable is a box containing a value

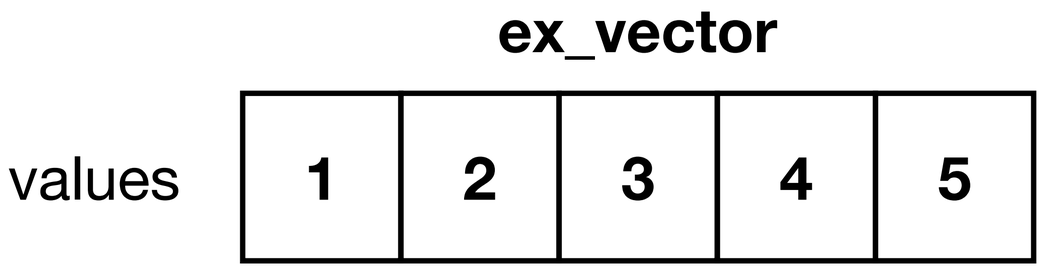
```
>> a = 3;
```



Storing groups of numbers: arrays

- Array: sequential series of boxes containing values

```
>> ex_vector = [1 2 3 4 5];
```



- Number of boxes (values) is the **length** of the array. Use length(var) fx to get length of the array.

```
>> length(ex_vector)

ans =

     5
```

Storing groups of numbers: matrices

- Matrix: 2-dimensional array
    - Syntax: var_name = [val1 val2 val3**;** val4 val5 val6];
- Examples

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

example_matrix =

     1     2     3     4
     5     6     7     8
```

```
>> another_example = [1 1; 2 2; 3 3]

another_example =

     1     1
     2     2
     3     3
```

Storing groups of numbers: matrices

- Same idea as vectors with with 2 dimensions now

ex_matrix

values

rows \ columns	1	2	3	4	5
1	**1**	**2**	**3**	**4**	**5**
2	**6**	**7**	**8**	**9**	**10**

- Matrices have length (longest dimension) and size (rows x cols)

```
>> length(ex_matrix)

ans =

     5
```

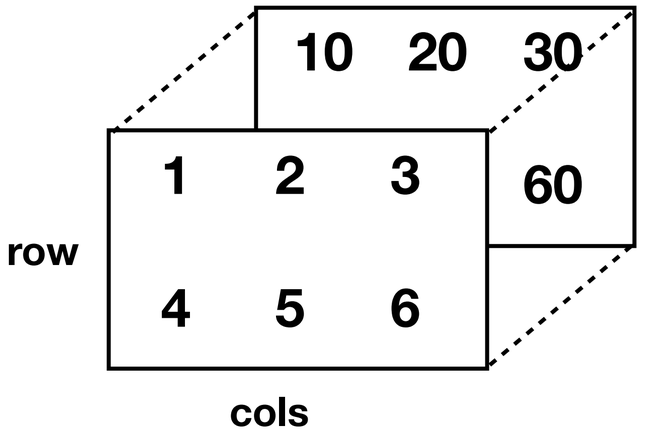
```
>> size(ex_matrix)

ans =

     2     5
```

Storing groups of numbers: matrices

- Matrices can have more than 2 dimensions (e.g. 3D brain data)

```
>> z1 = [1 2 3; 4 5 6];
z2 = [10 20 30; 40 50 60];
z1(:,:,2) = z2;

>> size(z1)


ans =


     2     3     2
```

rows (x) **cols (y)** **layers (z)**



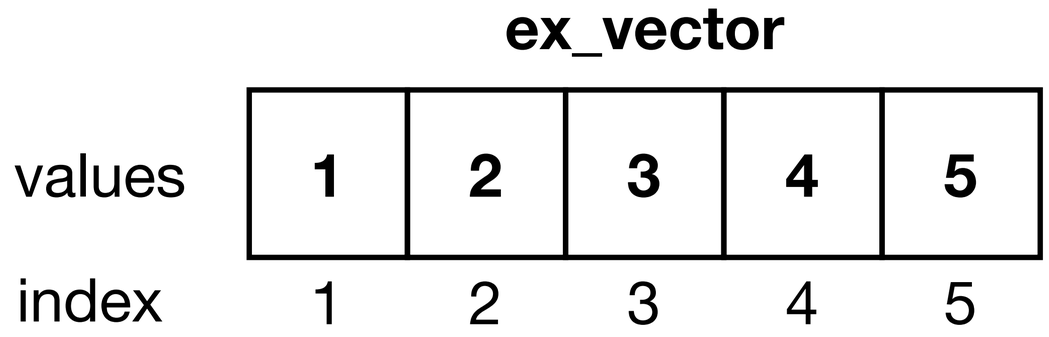
Overview

- Navigating MATLAB, part II
- More on variables
- Data Types
  - Numbers: ints, singles, doubles
  - Characters: strings
  - Collections of numbers: arrays
- **Manipulating arrays**

Vectors: selection by index

- Indexing is a way of accessing specific values in an array based on their position

```
>> ex_vector = [1 2 3 4 5];
```

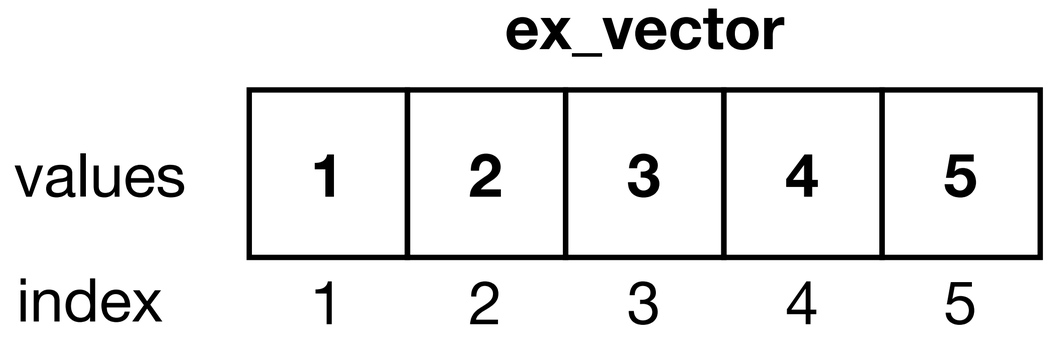


Vectors: selection by index

- Indexing is a way of accessing specific values in an array based on their position

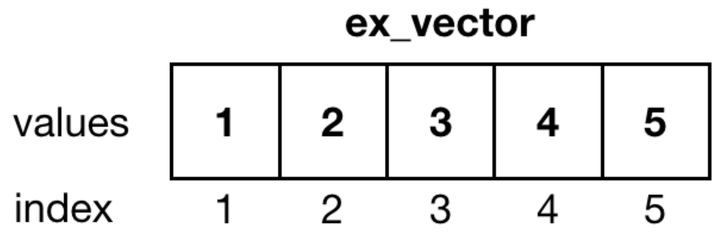
- Get value at an index using syntax: ex_vector(index)

```
>> ex_vector = [1 2 3 4 5];
```



Vectors: selection by index

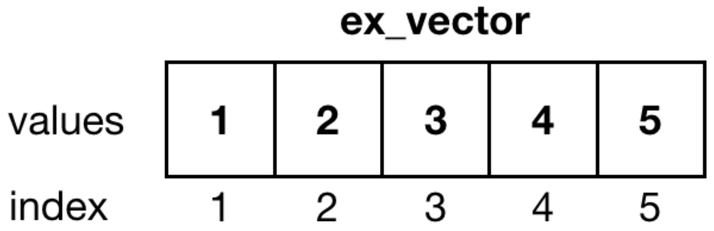
- **Ex 1:** Get single value
- Syntax: ex_vector(index)




```
>> ex_vector(3)

ans =

     3
```

Vectors: selection by index

- **Ex 2:** Get must values using colon operator
- Syntax: ex_vector(startInd:stepSize:stopInd)




```
>> ex_vector(1:3)

ans =

     1     2     3
```

```
>> ex_vector(1:2:5)

ans =

     1     3     5
```

```
>> ex_vector(4:-1:2)

ans =

     4     3     2
```

Exercises: vector indexing

1. Create a vector with values 1-10

2. Get the length of the vector

3. Return the value at index position 5

4. Return all the even numbers

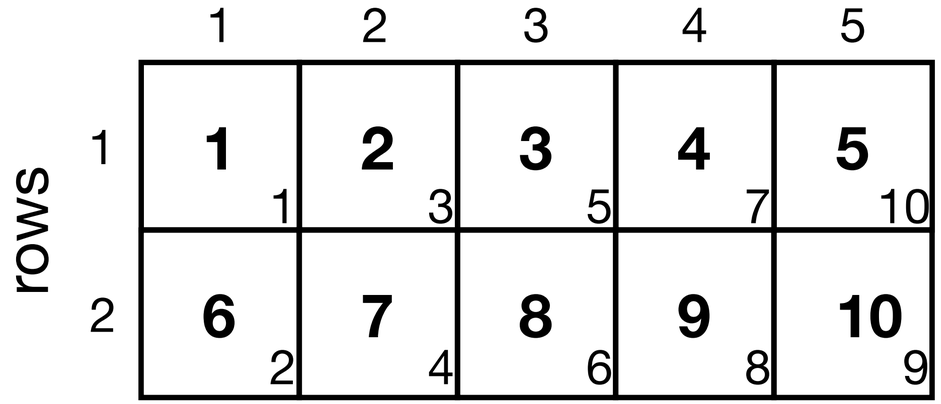
5. Return values in decreasing order from 9 to 4 inclusive

2D Matrices: selection

- Indexing 2D matrix gets a little more complicated

ex_matrix

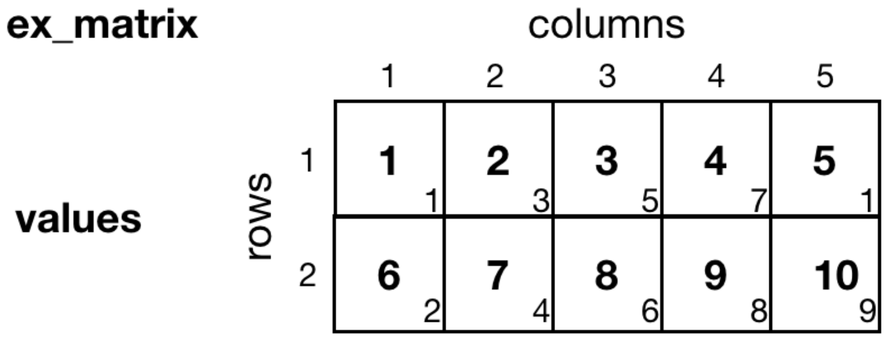
values



2D Matrices: selection by index

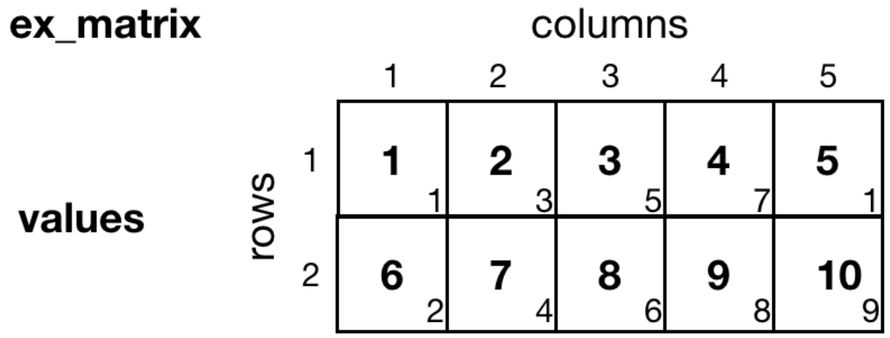
- **Option 1**: Select by index as before

```
>> ex_matrix(1:5)

ans =

     1     6     2     7     3

>> ex_matrix(10:-2:1)

ans =

    10     9     8     7     6
```



2D Matrices: selection by row/col

- **Option 2**: Select by row and column
- Syntax: ex_matrix(rows, cols)




```
>> ex_matrix(1,1)

ans =

     1
```

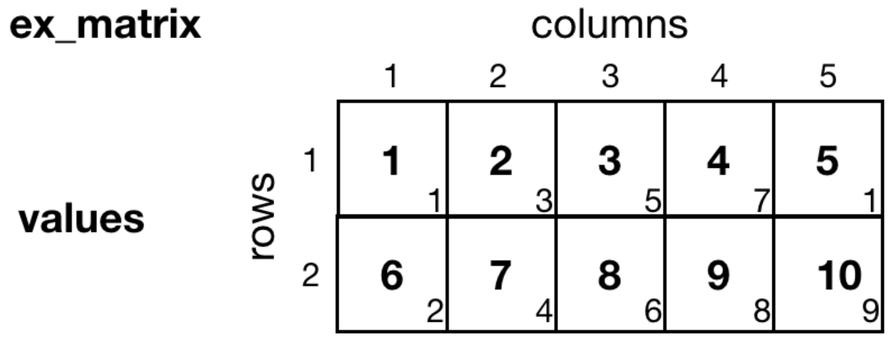
```
>> ex_matrix(2,4)

ans =

     9
```

```
>> ex_matrix(1,1:3)

ans =

     1     2     3
```

```
>> ex_matrix(1:2, 1:3)

ans =

     1     2     3
     6     7     8
```

2D Matrices: selection by row/col

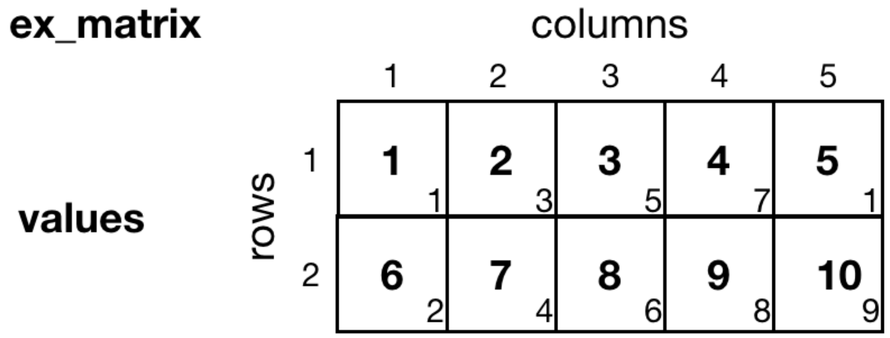
- **Option 3**: Select entire row with colon
- Syntax: ex_matrix(row,:)




```
>> ex_matrix(1,:)

ans =

     1     2     3     4     5

>> ex_matrix(2,:)

ans =

     6     7     8     9    10
```

2D Matrices: selection by row/col

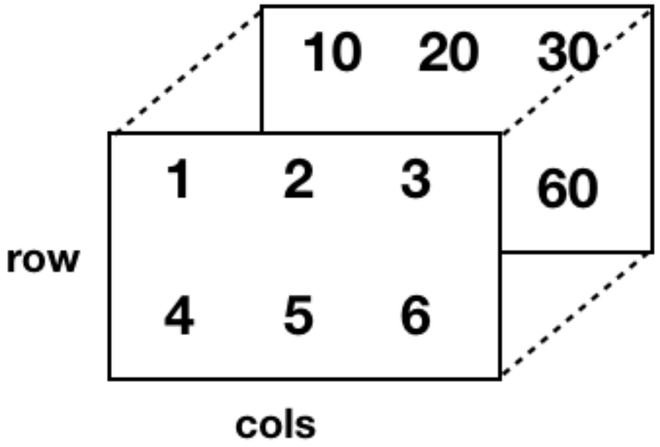
- **Option 4**: Select entire col with colon
- Syntax: ex_matrix(:, col)




```
>> ex_matrix(:,1)

ans =

     1
     6
```

```
>> ex_matrix(:,1:3)

ans =

     1     2     3
     6     7     8
```

ND matrices



- Mostly same as 2D matrices

Selection by index

```
>> z1(2)

ans =

     4

>> z1(10)

ans =

    50
```

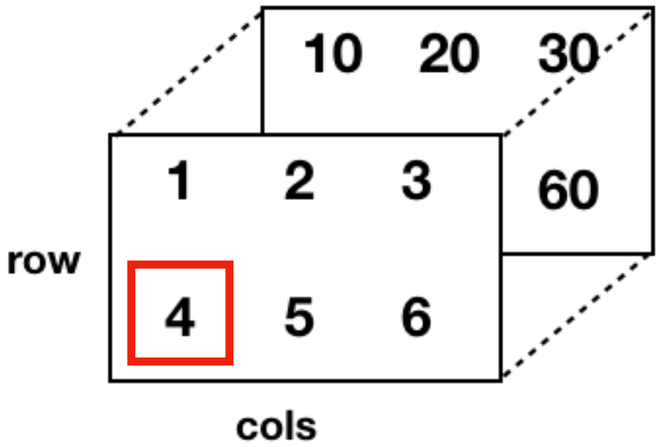
Selection by x,y,z

```
>> z1(1,2,1)

ans =

     2

>> z1(1,2,2)

ans =

    20
```

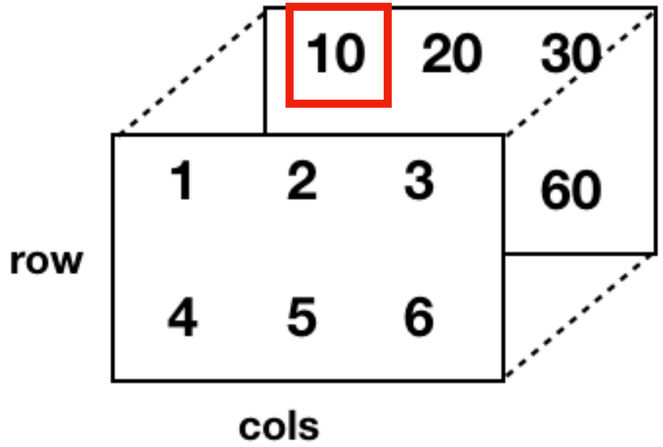
ND matrices

- Mostly same as 2D matrices: selection by index

```
>> z1(2)

ans =

     4
```



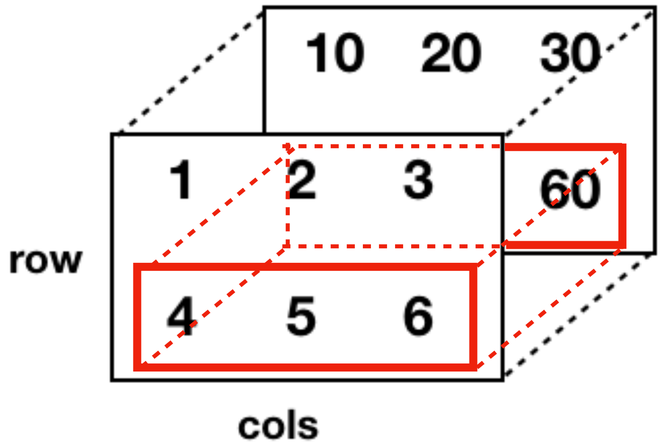

```
>> z1(7)

ans =

    10
```



ND matrices

- Mostly same as 2D matrices: selection by rows/cols/slices (dims)

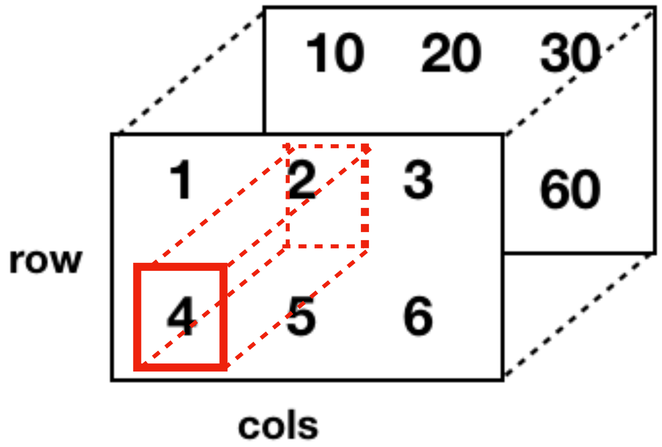
```
>> z1(2,1,1)

ans =

     4
```



ND matrices

- Mostly same as 2D matrices: selection by rows/cols/slices (dims)

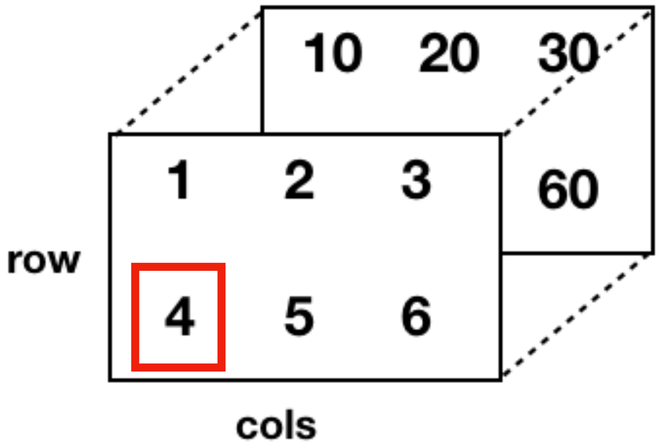
```
>> z1(2,1,1)

ans =

     4
```



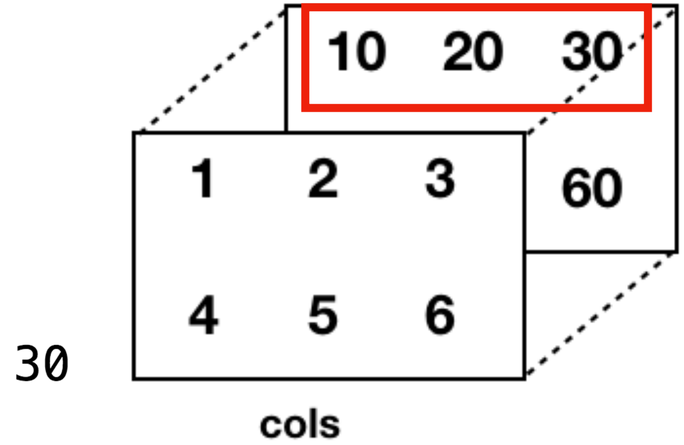
ND matrices

- Mostly same as 2D matrices: selection by rows/cols/slices (dims)

```
>> z1(2,1,1)

ans =

     4
```




```
>> z1(1,:,2)

ans =

    10    20    30
```

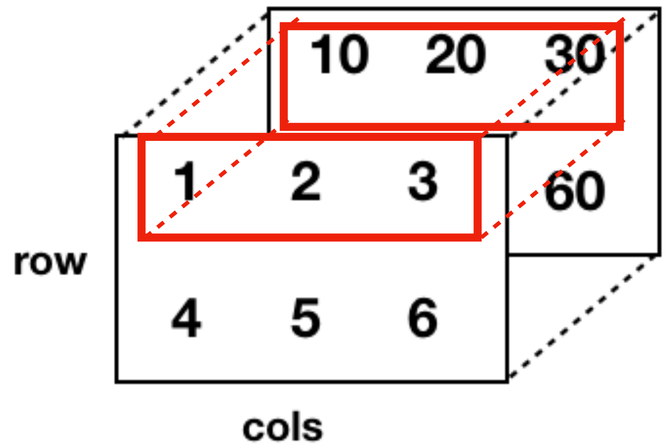


ND matrices

- Tricky: selecting “slices” from the ND matrix

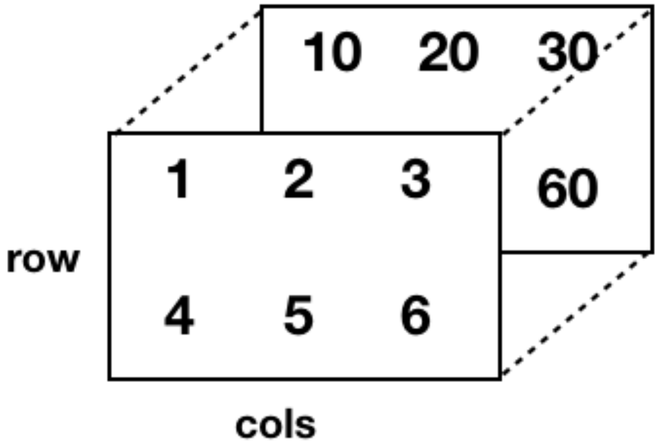
```
>> z1(1,:,:)

ans(:,:,1) =

     1     2     3


ans(:,:,2) =

    10    20    30

>> size(z1(1,:,:))

ans =

     1     3     2
```

```
>> squeeze(z1(1,:,:))

ans =

     1    10
     2    20
     3    30

>> size(squeeze(z1(1,:,:)))

ans =

     3     2
```



ND matrices

- When selecting across



Exercises: matrix selection

1. Create a 3x4 matrix (3 rows, 4 columns). The first row should contain values 1-4, the second 5-9, etc. Print the matrix

2. Get the length and size of the matrix

3. Select the number 4 using indexing. Repeat but use row/column selection

4. Print all the numbers in order (1-12)

5. Print all the numbers in row 2. Print all the numbers in col 3.

6. Print all the numbers in row 1 and row 2

Creating matrices

1. Create manually (as before)

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

example_matrix =

     1     2     3     4
     5     6     7     8
```

Creating matrices

2. Colon operator

-Syntax: matrix = [startNum : stepSize : stopNum]

-Can omit stepSize, default = 1

Example 1: vector from 1-4

```
>> seq_vector = [1:4]

seq_vector =

     1     2     3     4
```

Example 2: Every other number from 1-10

```
>> every_other_vector = [1:2:10]

every_other_vector =

     1     3     5     7     9
```

Creating matrices

2. Colon operator

-Syntax: matrix = [startNum : stepSize : stopNum]

-Can omit stepSize, default = 1

Example 3: decreasing from 6 to 2

```
>> dec_vector = [6:-1:2]

dec_vector =

     6     5     4     3     2
```

Creating matrices

3. **Built-in functions (can do any ndim matrix)**
   - ones(row, col)
   - zeros(row, col)
   - nans(row, col)
   - diag(row, col)
   - rand(row, col)
   - randi(row, col)

Changing values in a matrix

- **Option 1:** Change a specific position by index or row/col

```
>> a = ones(2,3)

a =

     1     1     1
     1     1     1

>> a(2,3) = 100

a =

     1     1     1
     1     1   100
```

```
>> b = zeros(2,5)

b =

     0     0     0     0     0
     0     0     0     0     0

>> b(1,:) = -1

b =

    -1    -1    -1    -1    -1
     0     0     0     0     0
```

```
>> c = [1:3; 4:6; 7:9]

c =

     1     2     3
     4     5     6
     7     8     9

>> c([1 1], [2 3]) = 99

c =

     1    99    99
     4     5     6
     7     8     9
```

Changing values in a matrix

- **Option 2:** Sclar addition, subtraction, multiplication, division

```
>> a = ones(2,2)

a =

     1     1
     1     1

>> a = a + .5

a =

    1.5000    1.5000
    1.5000    1.5000
```

```
>> b = zeros(2,2)

b =

     0     0
     0     0

>> b = b - 1

b =

    -1    -1
    -1    -1
```

```
>> c = ones(2,3)

c =

     1     1     1
     1     1     1

>> c = c * 20

c =

    20    20    20
    20    20    20
```

Exercises

1. Use the colon operator to create a 2x5 matrix with numbers 1-5 in row1 and 6-10 in row 2

2. Replace the value at index 3 with 100.

3. Replace all the values in row 2 with -10.

4. Multiple the resulting vector by 0.5

Matrix operations

- **Addition, subtraction**: add or subtract values at same index

Make two matrices

```
>> a = [1:3; 4:6]
```

```
a =

     1     2     3
     4     5     6
```

```
>> b = [1:3; 10:12]
```

```
b =

     1     2     3
    10    11    12
```

Add

```
>> a + b
```

```
ans =

     2     4     6
    14    16    18
```

Matrix operations

- **Scalar multiplication**: multiply everything by same value

```
>> a = [1:3; 4:6]

a =

     1     2     3
     4     5     6
```

```
>> a * 2

ans =

     2     4     6
     8    10    12
```

Matrix operations

- **Matrix multiplication**: multiply matrices (#cols a must match #rows in b)

```
>> a = [1:3; 4:6]

a =

     1     2     3
     4     5     6

>> b = [1 1; 2 2; 3 3]

b =

     1     1
     2     2
     3     3
```

```
>> a * b

ans =

    14    14
    32    32
```

Matrix operations

- **Element-wise matrix multiplication**: multiply values at the same index. Matrices must be same size

```
>> a = [1 2; 3 4]

a =

     1     2
     3     4

>> b = [.5 1; 2 10]

b =

    0.5000    1.0000
    2.0000   10.0000
```

```
>> a .* b

ans =

    0.5000    2.0000
    6.0000   40.0000
```

Matrix operations

- **Transpose**: flip dimensions

```
>> a = [1:3; 4:6]

a =

     1     2     3
     4     5     6
```

```
>> a'

ans =

     1     4
     2     5
     3     6
```

Split up matrices

- You often might need to split up matrices into smaller matrices in sensible ways. You can do this by indexing and storing the output to a variable

```
>> a = [1 1 1 1; 2 2 2 2];

>> a_row1 = a(1,:)

a_row1 =

     1     1     1     1

>> a_row2 = a(2,:)

a_row2 =

     2     2     2     2
```

Concatenating matrices

- You might also want to do the opposite and combine multiple matrices. There are several ways to to this:

- **Horizontal concatenation:** [ ] with , or horzcat

```
a =                             b =

     1     1     1                 3     3     3
     2     2     2                 4     4     4


>> c = [a, b]

c =

     1     1     1     3     3     3
     2     2     2     4     4     4

>> c = horzcat(a,b)

c =

     1     1     1     3     3     3
     2     2     2     4     4     4
```

Concatenating matrices

- You might also want to do the opposite and combine multiple matrices. There are several ways to to this:

- Horizontal concatenation: [ ] with comma or horzcat

- **Vertical concatenation:** [ ] with semicolon or vertcat

```
a =

     1     1     1
     2     2     2
```

```
b =

     3     3     3
     4     4     4
```

```
>> c = [a; b]

c =

     1     1     1
     2     2     2
     3     3     3
     4     4     4
```

```
>> c = vertcat(a,b)

c =

     1     1     1
     2     2     2
     3     3     3
     4     4     4
```

Concatenating matrices

- You might also want to do the opposite and combine multiple matrices. There are several ways to to this:
- Horizontal concatenation: [ ] with comma or horzcat
- Vertical concatenation: [ ] with semicolon or vertcat
- Concatenate along new dimension: with [ ] or cat

```
a =

     1     1     1
     2     2     2
```

```
b =

     3     3     3
     4     4     4
```

```
>> c = cat(3, a, b)

c(:,:,1) =

     1     1     1
     2     2     2

c(:,:,2) =

     3     3     3
     4     4     4
```

```
>> c = a;
>> c(:,:,2) = b;
>> c

c(:,:,1) =

     1     1     1
     2     2     2

c(:,:,2) =

     3     3     3
     4     4     4
```

Exercises

1. Create a 3x2 matrix A of 3s. Create a second 3x2 matrix B with the numbers 1-6. Add the two matrices together. Multiple the two matrices by elements.

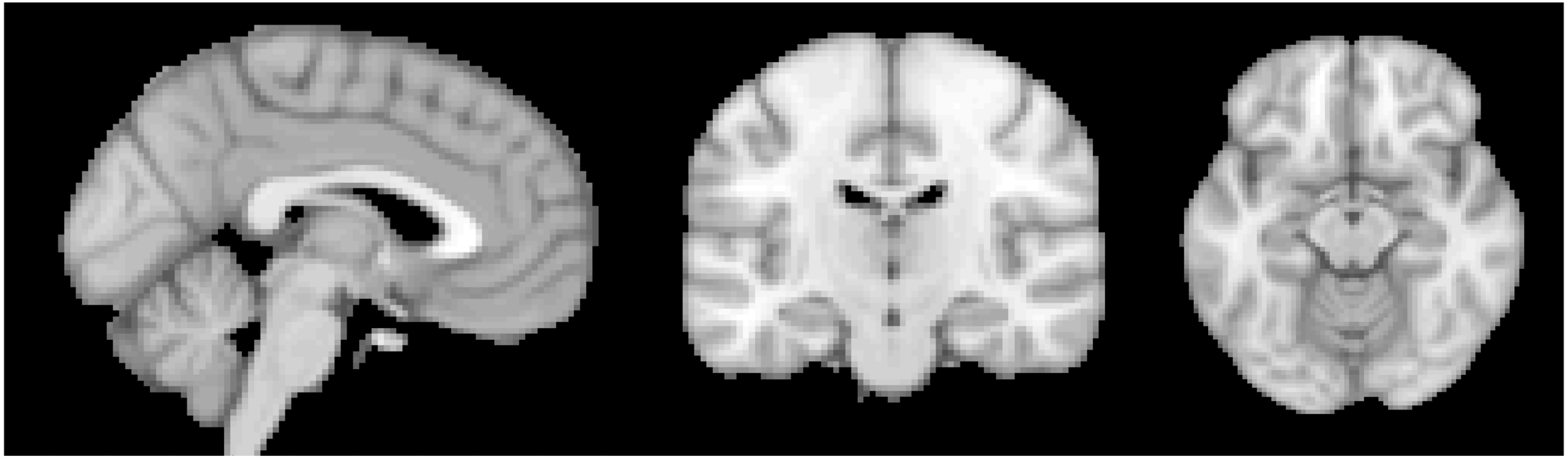
2. Try multiplying matrix A by matrix B. Why didn't it work? Use transpose to modify one of the matrices and try again.

3. Create a 2x2 matrix of ones, a 2x2 matrix of twos, and a 2x2 matrix of threes. Concatenate the three matrices in order vertically and then horizontally using square brackets and commas/semicolons. Repeat using the horzcat and vertcat functions.

4. Concatenate the matrices from (3) in a 2x2x3 matrix

HW #1

- Posted Friday afternoon on GitHub, due Tuesday by midnight
- Not graded, but will receive feedback



Review

current/working directory
valid variable naming
good variable naming practices
strings, vectors, matrices

Making a matrix
Manually
Colon operator
built-in functions
Modifying values

Selecting from array
By index
By row/col
Colon operator

Manipulating matrices
Matrix operations
concatenating
splitting
reshaping

Functions
Navigation
- pwd
- cd
- ls
- makedir()

Matrices
- length()
- size()
- ones()
- zeros()
- nans()
- diag()
- rand()
- randi()

Concatenation
- vertcat()
- horzcat()