CS 1112 Test 2B Review

What we'll do today

- **Review of these topics:**
  - Cell arrays
  - File input/output
  - Objects and classes
  - `sort()` and permutation indices
- **Some example problems**
- **Questions**

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

- **uint8:** integers ranging from 0 to 255
  Array of uint8 numbers: y = uint8(x);

- **char:** standard characters, including letters, digits, symbols. Multiple chars together form a *string*, but a string is *not* a type – it is just an array of characters.
  Array of characters: s = 'CS1112'; s = ['c', 's', '1', '1', '1', '2'];

- **logical:** also known as a boolean. Can be true/false or 0/1.
  Creating a logical: z = rand > 0.5

Matlab data types

A **type** is a way of representing data. You should be aware of these types:

- **double:** the default type for numbers in Matlab
  Array of doubles: x = [1, 2, 3];

- **uint8:** integers ranging from 0 to 255
  Array of uint8 numbers: y = uint8(x);

- **char:** standard characters, including letters, digits, symbols. Multiple chars together form a *string*, but a string is *not* a type – it is just an array of characters.
  Array of characters: s = 'CS1112'; s = ['c', 's', '1', '1', '1', '2'];

- **logical:** also known as a boolean. Can be true/false or 0/1.
  Creating a logical: z = rand > 0.5

An array can only hold values of one type. A *cell array* is a special kind of array that can hold data of different types. Yay!

Cell arrays

Arrays (e.g. vectors, matrices, 3-D arrays, etc.)

- Can hold *one scalar* value in each of its components,
  e.g. one double, one char, one uint8.
- Data of all components must be the same type

Cell arrays

Arrays (e.g. vectors, matrices, 3-D arrays, etc.)

- Can hold *one scalar* value in each of its components,
  e.g. one double, one char, one uint8.
- Data of all components must be the same type

Cell arrays

- Each cell can store something “larger” than a scalar (but doesn’t have to).
- Can store a vector, matrix, or string, etc. in a single component
- Each cell can store something of a different type

Cell arrays

Initialize a cell array with cell(…) function	`c = cell(1,3);` % Cell array with 1 row, 3 columns
Obtain number of rows and columns	`[nr, nc] = size(c);` % Same as for other arrays
Put items (strings, in this case) into the cell array	`c = {'matlab', 'is', 'fun'};` % Commas optional
Display first item (string in this example)	`disp(c{1})` % Note the use of curly braces
Display first two items (vectorized)	`disp(c(1:2))` % Note the use of parentheses
Display first three letters of first string	`disp(c{1}(1:3))` % Note the use of curly braces *and* parentheses
Concatenate the strings (produces ‘matlab is fun’)	`s = [c{1} ‘ ’ c{2} ‘ ’ c{3}]` % Note the use of square brackets to create a string

Fall 2016 Prelim: **Question 5b**

Question 5b: (25 points)

Assume that function `getIndices` of Question 5a has been correctly implemented; make effective use of it in implementing function `aveScores` below. Note the example at the bottom of the page.

```
function CA = aveScores(M)
% M is a 2-d array of characters. Each row of M stores the scores of one student:
%   a netID followed by one or more scores and these data items are separated by
%   commas.  There may be trailing spaces in a row of M.
% CA is an n-by-2 cell array where n is the number of students whose record includes
%   at least two scores. In each row of CA, the first cell stores the netID of a
%   student who has at least two scores and the second cell stores the average score
%   of that student.  If no student has at least two scores then CA is an empty cell
%   array.
% ONLY these built-in functions are allowed: length, size, str2double, sum, mean
% Recall that str2double can handle leading and trailing spaces, e.g.,
%   str2double('87    ') returns the type double scalar 87.
```

Example: Suppose `M` is

```
['vaf34,80,100,90';...
 'aaj91,100      ';...
 'rt2253,75,95   ']
```

Then `aveScores(M)` should return a 2×2 cell array `CA`:

- In row 1 column 1 is `'vaf34'` and in row 1 column 2 is the type double scalar 90.
- In row 2 column 1 is `'rt2253'` and in row 2 column 2 is the type double scalar 85.

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	
2		
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2		
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3		
4		
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there is at least two test scores in the row, extract the netID and store it in the first column of CA	
4		
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there is at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1);**
4		
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there is at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1);**
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there is at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1);**
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	Use another for-loop (nested inside the first) that iterates from **k = 2:length(comma_idx)**, and determine indices of substring
5		

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there is at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1);**
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	Use another for-loop (nested inside the first) that iterates from **k = 2:length(comma_idx)**, and determine indices of substring
5	Store as a running sum in the second column of CA, and take the average after all scores have been extracted.	

Fall 2016 Prelim: **Question 5b**

#	Thing we need to do	Programming concept needed to do this thing
1	Loop through the individual strings (rows) of M	A for-loop that iterates **for row_M = 1:size(M,1)**
2	Find the commas in a given string. Skip to the next row if there is less than 2 test scores	Use the getIndices function from the part 5a to find the indices of the commas **comma_idx**. Use an if statement to check if the string has at least 2 scores
3	If there is at least two test scores in the row, extract the netID and store it in the first column of CA	Since not all rows of M will be stored in output CA, set up a rowCA index which updates each step. Then **CA{row_CA, 1} = M(rowM , 1:comma_idx(1)-1);**
4	Knowing the indices of the commas, loop through the corresponding substrings to extract test scores	Use another for-loop (nested inside the first) that iterates from **k = 2:length(comma_idx)**, and determine indices of substring
5	Store as a running sum in the second column of CA, and take the average after all scores have been extracted.	Initialize the second column to zero outside the for-loop of step 4, then convert the substring to a double and add to the second column.

Fall 2016 Prelim: **Question 5b**

```
for rowM = 1:size(M,1)
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

```
end
```

Fall 2016 Prelim: **Question 5b**

```
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2




    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
rowCA = 0;
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2
        rowCA = rowCA+1;
        CA{ rowCA, 1 } = M( rowM, 1:comma_idx(1)-1 );
    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
rowCA = 0;
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2
        rowCA = rowCA+1;
        CA{ rowCA, 1 } = M( rowM, 1:comma_idx(1)-1 );

        for k = 1:length(comma_idx)
            left = comma_idx(k)+1;
            if k < length(comma_idx)
                right = comma_idx(k+1)-1;
            else
                right = size(M,2); % After last comma, take all remaining characters
            end

        end

    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

Fall 2016 Prelim: **Question 5b**

```
rowCA = 0; CA = {};
for rowM = 1:size(M,1)
    comma_idx = getIndices( M( rowM , : ), ‘,’ );
    if length(comma_idx) >= 2
        rowCA = rowCA+1;
        CA{ rowCA, 1 } = M( rowM, 1:comma_idx(1)-1 );
        CA{ rowCA, 2 } = 0;
        for k = 1:length(comma_idx)
            left = comma_idx(k)+1;
            if k < length(comma_idx)
                right = comma_idx(k+1)-1;
            else
                right = size(M,2); % After last comma, take all remaining characters
            end
            CA{rowCA,2} = CA{rowCA,2} + str2double( M( rowM, left:right ) );
        end
        CA{rowCA,2} = CA{rowCA,2}/length(comma_idx);
    end
end
```

Connection to previous slide:
Red: for-loop to look at each string in M
Orange: extract commas
Green: extract and store netID
Blue: extract test score indices
Black: compute average test score

File input/output

- Open a file: `fopen()`
- Read it line-by-line until end-of-file: `fgetl(), feof()`
- Write data into a file: `fprintf()`
- Close a file: `fclose()`

File input/output: **Open/close a file**

Syntax:
```
fid = fopen(filename, 'r');
fclose(fid);
```

- `fid` stores file ID of the opened file, used as input later
- Permission `'r'` indicates that we are reading the file
  - `'w'` when writing into file after discarding all existing content
  - `'a'` when appending to the end of the file
- `;` needed after file commands

File input/output: **Read line-by-line until end**

Syntax:

```
fid = fopen('statePop.txt', 'r');
while ~feof(fid)
    str = fgetl(fid);
end
fclose(fid);
```

- `~feof(fid)` returns false only if we reached the end-of-file
- `fgetl(fid)` gives next line (1 line) as string/char array
- Can read only part of file by replacing the `~feof(fid)` condition

File input/output: **Store data into vector/array**

Syntax:

```
fid = fopen('statePop.txt', 'r');
i = 1;
while ~feof(fid)
    str = fgetl(fid);
    pop(i) = str2double(str(3:7));
    i = i + 1;
end
fclose(fid);
```

- `str2double` converts a **string** representing a numeric value to a scalar numeric value of type **double**
  - E.g: `x = str2double('-3.24')` → `x = -3.24`

File input/output: **Write into file**

Syntax:

```
fid = fopen('popSm2Lg.txt', 'w');
for i = 1:length(Cnew)
    fprintf(fid, '%s\n', Cnew{i});
end
fclose(fid);
```

- `fprintf(fid, ...)` prints on the file with ID `fid`
- `'%s\n'` specifies to print `Cnew{i}` in string format

Fall 2019 Prelim: **Question 6.1**

We have a plain text file containing data on many cities in New York state. Each line of the file stores the data of one city, in the order name, latitude, longitude, and population. The data items on a line are separated by exactly two spaces. As an example, two of the many lines from the file are shown below with the symbol ␣ indicating a single space:

```
New␣York␣␣40°39′40″N␣␣73°56′38″W␣␣8175133
Ithaca␣␣42°26′36″N␣␣76°30′0″W␣␣30999
```

(6.1) Complete the following function as specified:

```
function D = parseData(cityData)
% Read text from the file named by cityData and return a 2D cell array.
% Each line of the file contains information on one city:  name, latitude,
%   longitude, and population.  Those 4 data items are separated by exactly 2
%   spaces. There are no leading or trailing spaces on each line.
% D is a n-by-4 cell array where n is the number of lines of text in the file,
%   and each cell in one row of D stores one data token for a city.  The city
%   name, latitude, and longitude are each stored in D as a char row vector; the
%   population is stored in D as a type double scalar.
% Example: suppose one line from the file is
%   'New␣York␣␣40°39′40″N␣␣73°56′38″W␣␣8175133'.  Then the row in cell array D
%   that corresponds to that line stores 'New York' in the 1st cell, '40°39′40″N'
%   in the 2nd cell, '73°56′38″W' in the 3rd cell, and 8175133 in the 4th cell.
%   There must not be any leading or trailing spaces in the char vectors.
```

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	
2		
3		
4		

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2		
3		
4		

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2	Given a line, find locations where the two spaces occur.	
3		
4		

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2	Given a line, find locations where the two spaces occur.	Set a for-loop that goes through a line from the file, and check where the two-space separators are at using **strcmp()** and an **if-block**.
3		
4		

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2	Given a line, find locations where the two spaces occur.	Set a for-loop that goes through a line from the file, and check where the two-space separators are at using **strcmp()** and an **if-block**.
3	Find and store the city name, latitude, and longitude into correct cells in cell array D.	
4		

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2	Given a line, find locations where the two spaces occur.	Set a for-loop that goes through a line from the file, and check where the two-space separators are at using **strcmp()** and an **if-block**.
3	Find and store the city name, latitude, and longitude into correct cells in cell array D.	Use the for-loop index to find where the data token starts/ends. Extract the char sub-array from the line. Initialize indices for D and use them with proper increments to assign data tokens to correct cells in D.
4		

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2	Given a line, find locations where the two spaces occur.	Set a for-loop that goes through a line from the file, and check where the two-space separators are at using **strcmp()** and an **if-block**.
3	Find and store the city name, latitude, and longitude into correct cells in cell array D.	Use the for-loop index to find where the data token starts/ends. Extract the char sub-array from the line. Initialize indices for D and use them with proper increments to assign data tokens to correct cells in D.
4	Get the population as a double and store in cell array D.	

Fall 2019 Prelim: **Question 6.1**

#	Thing we need to do	Programming concept needed to do this thing
1	Open the file and read each line of data.	Use **fopen()** to open the file, set a while-loop with **feof()** to traverse to the end of the file, and grab each line with **fgetl()**. After we're done, close the file with **fclose()**.
2	Given a line, find locations where the two spaces occur.	Set a for-loop that goes through a line from the file, and check where the two-space separators are at using **strcmp()** and an **if-block**.
3	Find and store the city name, latitude, and longitude into correct cells in cell array D.	Use the for-loop index to find where the data token starts/ends. Extract the char sub-array from the line. Initialize indices for D and use them with proper increments to assign data tokens to correct cells in D.
4	Get the population as a double and store in cell array D.	Use **str2double()** to change the last data token into a double, then store in the last column of D.

Fall 2019 Prelim: **Question 6.1**

```
function D = parseData(cityData)

    fid = fopen(cityData, 'r');

    while ~feof(fid)
        s = fgetl(fid);









    end
    fclose(fid);
```

Connection to previous slide:
Red: open file and read each line
Orange: go through line to find ' '
Green: find/store first 3 data tokens in D
Blue: store population as double in D

Fall 2019 Prelim: **Question 6.1**

```
function D = parseData(cityData)

    fid = fopen(cityData, 'r');

    while ~feof(fid)
        s = fgetl(fid);
        for k = 1:length(s)-1
            if strcmp(s(k:k+1), '  ')



            end
        end


    end
    fclose(fid);
```

Connection to previous slide:
Red: open file and read each line
Orange: go through line to find ' '
Green: find/store first 3 data tokens in D
Blue: store population as double in D

Fall 2019 Prelim: **Question 6.1**

```
function D = parseData(cityData)

    fid = fopen(cityData, 'r');
    r = 1;
    while ~feof(fid)
        s = fgetl(fid); c = 1; tStart = 1;
        for k = 1:length(s)-1
            if strcmp(s(k:k+1), '  ')
                tEnd = k-1; D{r,c} = s(tStart:tEnd);
                c = c+1; tStart = k+2;
            end
        end


    end
    fclose(fid);
```

Connection to previous slide:
Red: open file and read each line
Orange: go through line to find ' '
Green: find/store first 3 data tokens in D
Blue: store population as double in D

Fall 2019 Prelim: **Question 6.1**

```
function D = parseData(cityData)

    fid = fopen(cityData, 'r');
    r = 1;
    while ~feof(fid)
        s = fgetl(fid); c = 1; tStart = 1;
        for k = 1:length(s)-1
            if strcmp(s(k:k+1), '  ')
                tEnd = k-1; D{r,c} = s(tStart:tEnd);
                c = c+1; tStart = k+2;
            end
        end
        D{r,4} = str2double( s(tStart:length(s)) );
        r = r+1;
    end
    fclose(fid);
```

Connection to previous slide:
Red: open file and read each line
Orange: go through line to find ' '
Green: find/store first 3 data tokens in D
Blue: store population as double in D

Objects and Classes

- **Class:** A file that specifies *properties* (variables) and *methods* (functions) associated with the item that the class represents
  - Contains a *constructor*, a special method that creates new objects

- **Object:** One *instance* of a class
  - Objects of the same class have the same properties and the same methods
  - The properties of objects of the same class can have *different values*

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Note that the `end` keyword is used to close the following:

1. The `classdef`
2. The `properties` section
3. The `methods` section
4. Each `function` inside the `methods` section

Objects and Classes: **Constructors**

Constructor: A method (function) that creates a new object

- Must have the same name as the class
- Can take in parameters to set property values
- Use `nargin` to ensure that constructor can be called without any arguments

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Implementation of this constructor:

```
function aml = Animal(n, s, a, hT)
    if (nargin == 4)
        aml.name = n;
        aml.species = s;
        aml.age = a;
        aml.hasTail = hT;
    end
end
```

If 4 arguments are not provided, the 4 properties will be set to default values.

Objects and Classes: **Create/reference objects**

Create new objects by calling the constructor, which returns a *reference to the new object* that should be stored in a variable.

Example:

```
a = Animal('Bobbert', 'pig', 2, 1);
%   An animal object with these properties is created:
%   name = 'Bobbert', species = 'pig', age = 2, hasTail = 1
%   a is the reference to this object.
```

Create an empty array of Animal objects using `.empty()`

Example: `b = Animal.empty()`

Check if an object/object array is empty using `isempty(<reference>)`

Example: `isempty(a)` returns 0, `isempty(b)` returns 1

Objects and Classes: **Calling methods**

Each method in a class takes in a minimum of one parameter (named `'self'`), which is *a reference to the object calling the method*

Syntax for calling a method:

`<reference>.<methodName>(2nd through last input variable)`

This is equivalent (but it is better to use the above way):

`<methodName>(self, 2nd through last input variable)`

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

How to use this method (from another script, function, etc.):

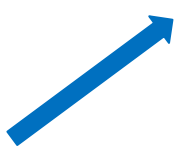
```
% Object reference should be
% created first
a = Animal('Bobbert', 'pig', 2, 1);

% Call method
a.birthday();     % or: birthday(a);

% See result of method call
disp(a.age)       % 3 will be displayed
```

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```



Implementation of this method:

```
function c = checkHasTail(self)
    if (self.hasTail == 1)
        c = 1;
    else
        c = 0;
    end
end
```

Objects and Classes Example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end
    methods
        function aml = Animal(n, s, a, hT)
            % set properties of aml
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Implementation of this method:

```
function c = isOlder(self, otherAnimal)
    if (self.age > otherAnimal.age)
        c = 1;
    else
        c = 0;
    end
end
```

How to use this method:

Age of a is 2

Age of b is 1

```
a = Animal('Bobbert', 'pig', 2, 1);
b = Animal('Robbert', 'frog', 1, 0);
disp(a.isOlder(b))    % will display 1
disp(b.isOlder(a))    % will display 0
```

Objects and Classes: **Arrays of objects**

Objects of the same class can be stored in a simple vector/array.

Objects of different classes (even classes which are related by inheritance) must be stored in a cell array.

Example: Write a function that takes in a vector z of Animal objects and returns a vector of the indices from z which contain objects whose species is ‘pig’:

```
function idx = FindPigs(z)
idx = []; k = 1;
for i = 1:length(z)
    if (strcmp(z(i).species, ‘pig’))
        idx(k) = i;
        k = k+1;
    end
end
```

Review Question #7

```
function idxs = greatestOverlap(iArray)
% Find the biggest pairwise overlap between Intervals in iArray.
% iArray is an array (length > 1) of Interval references.
% idxs is a vector of length 2 storing indices of the two Intervals
%   in iArray that overlap the most. If there is not a pair of overlapping
%   Intervals in iArray, idxs is an empty vector.
% Write efficient code: avoid unnecessary iteration
```

Potentially useful methods in the Interval class:

- `getWidth(self)` returns the difference between the left and right endpoints (i.e. the width) of the Interval object referenced by `self`.
- `overlap(self, other)` returns an Interval object whose endpoints are the points between which the two Interval objects, `self` and `other`, overlap. If they do not overlap, this method returns an empty Interval object.

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	
2		
3		
4		

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2		
3		
4		

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	
3		
4		

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	Use a **`maxWidthSoFar`** variable to keep track of the *width* of the maximum overlap we've found so far
3		
4		

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	Use a **`maxWidthSoFar`** variable to keep track of the *width* of the maximum overlap we've found so far
3	Store the indices from `iArray` of the Intervals which overlap the most	
4		

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	Use a **`maxWidthSoFar`** variable to keep track of the *width* of the maximum overlap we've found so far
3	Store the indices from `iArray` of the Intervals which overlap the most	Update `idxs` when `maxWidthSoFar` changes
4		

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	Use a **`maxWidthSoFar`** variable to keep track of the *width* of the maximum overlap we've found so far
3	Store the indices from `iArray` of the Intervals which overlap the most	Update `idxs` when `maxWidthSoFar` changes
4	**If no Intervals overlap**, `idxs` is an empty vector	

Review Question #7

#	Thing we need to do	Programming concept needed to do this thing
1	Find overlap between **all possible combinations** of two Interval objects (efficiently)	Use a **nested for-loop** to check all possible combinations in `iArray`
2	Determine the **maximum** overlap	Use a **`maxWidthSoFar`** variable to keep track of the *width* of the maximum overlap we've found so far
3	Store the indices from `iArray` of the Intervals which overlap the most	Update `idxs` when `maxWidthSoFar` changes
4	**If no Intervals overlap**, `idxs` is an empty vector	`idxs` should be **initialized as empty**, and only filled if the width of the overlap between any two Intervals is greater than 0

Review Question #7: **Solution**

```
function idxs = greatestOverlap(iArray)


    n = length(iArray);
    for i = 1:n-1          % Notice this loop ends at n-1
        for j = i+1:n      % Notice this loop started at i+1



        end
    end
```

Connection to previous slide:
Red: for-loop to check combinations
Orange: find maximum overlap
Green: update `idxs` when max changes
Blue: `idxs` empty when no overlap

Review Question #7: **Solution**

```
function idxs = greatestOverlap(iArray)


    maxWidth = 0;
    n = length(iArray);
    for i = 1:n-1          % Notice this loop ends at n-1
        for j = i+1:n      % Notice this loop started at i+1
            olap = iArray(i).overlap(iArray(j));
            if ~isempty(olap) && olap.getWidth() > maxWidth
                maxWidth = olap.getWidth();

            end
        end
    end
```

Connection to previous slide:
Red: for-loop to check combinations
Orange: find maximum overlap
Green: update `idxs` when max changes
Blue: `idxs` empty when no overlap

Review Question #7: **Solution**

```
function idxs = greatestOverlap(iArray)


    maxWidth = 0;
    n = length(iArray);
    for i = 1:n-1          % Notice this loop ends at n-1
        for j = i+1:n      % Notice this loop started at i+1
            olap = iArray(i).overlap(iArray(j));
            if ~isempty(olap) && olap.getWidth() > maxWidth
                maxWidth = olap.getWidth();
                idxs = [i j];
            end
        end
    end
```

Connection to previous slide:
Red: for-loop to check combinations
Orange: find maximum overlap
Green: update `idxs` when max changes
Blue: `idxs` empty when no overlap

Review Question #7: **Solution**

```
function idxs = greatestOverlap(iArray)

    idxs = [];
    maxWidth = 0;
    n = length(iArray);
    for i = 1:n-1          % Notice this loop ends at n-1
        for j = i+1:n      % Notice this loop started at i+1
            olap = iArray(i).overlap(iArray(j));
            if ~isempty(olap) && olap.getWidth() > maxWidth
                maxWidth = olap.getWidth();
                idxs = [i j];
            end
        end
    end
```

Connection to previous slide:
Red: for-loop to check combinations
Orange: find maximum overlap
Green: update `idxs` when max changes
Blue: `idxs` empty when no overlap

Built-in `sort()` and permutation indices

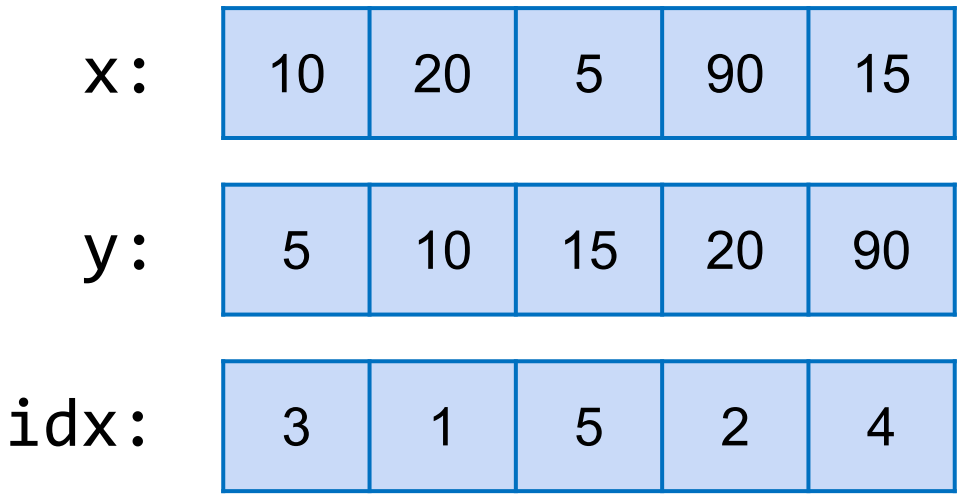
Syntax: `[y, idx] = sort(x)`

- Both outputs have same size as `x`.
- `y` stores entries of x in *ascending* order.
- `idx` stores the indices of the sorted arrangement: `y = x(idx)`

- To sort in descending order: `[y, idx] = sort(x, 'descend')`
- If input is a 2d-array, we can pick the dimension along which to sort:
  - `sort(x, 1)` sorts each column (default)
  - `sort(x, 2)` sorts each row

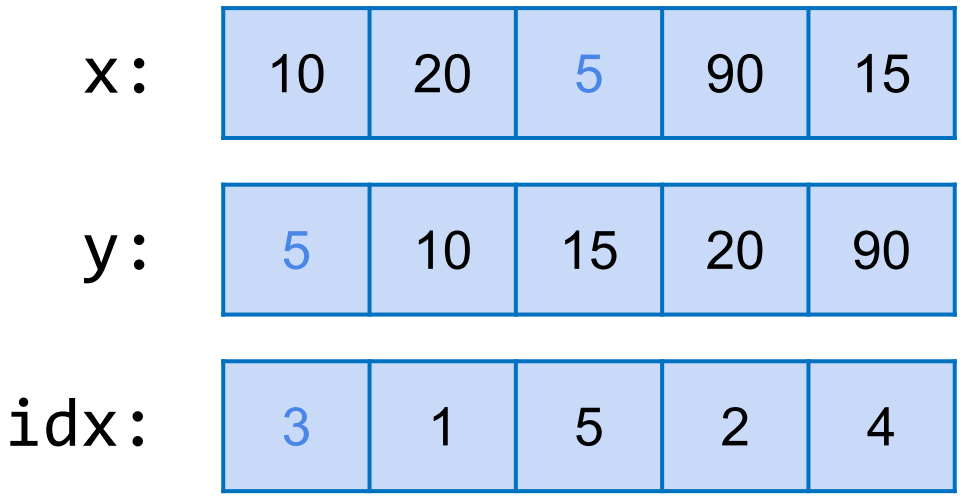
Built-in `sort()` and permutation indices

```
[y, idx] = sort(x)
```



Built-in `sort()` and permutation indices

```
[y, idx] = sort(x)
```

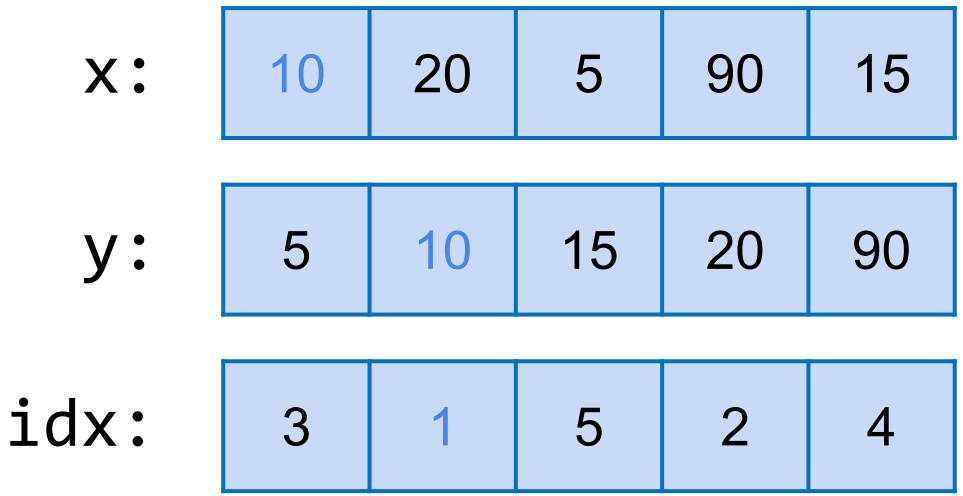


`idx(1) = 3` ⇔ `x(3)` is the smallest

`y(1) = x(idx(1)) = x(3)`

Built-in `sort()` and permutation indices

```
[y, idx] = sort(x)
```

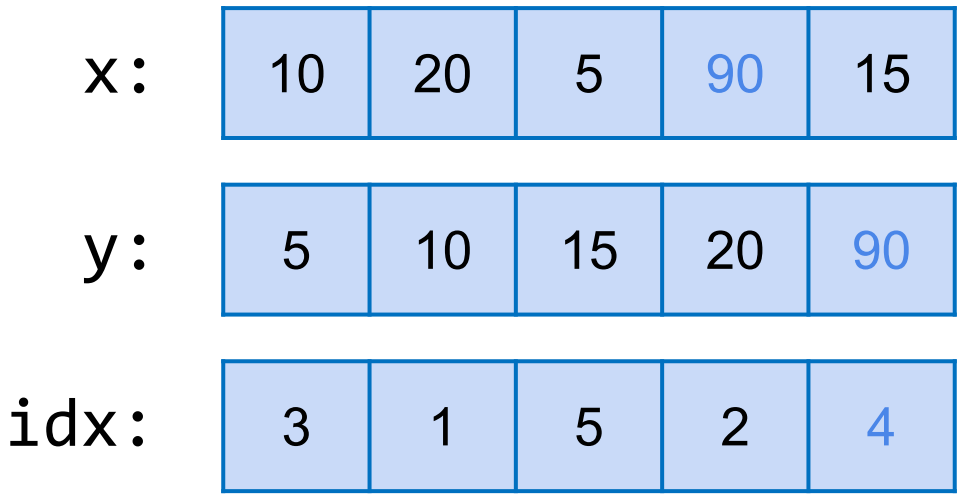


`idx(1) = 3` ⇔ `x(3)` is the smallest
`y(1) = x(idx(1)) = x(3)`

`idx(2) = 1` ⇔ `x(1)` is the 2nd-smallest
`y(2) = x(idx(2)) = x(1)`

Built-in `sort()` and permutation indices

```
[y, idx] = sort(x)
```



`idx(1) = 3` ⇔ `x(3)` is the smallest
`y(1) = x(idx(1)) = x(3)`

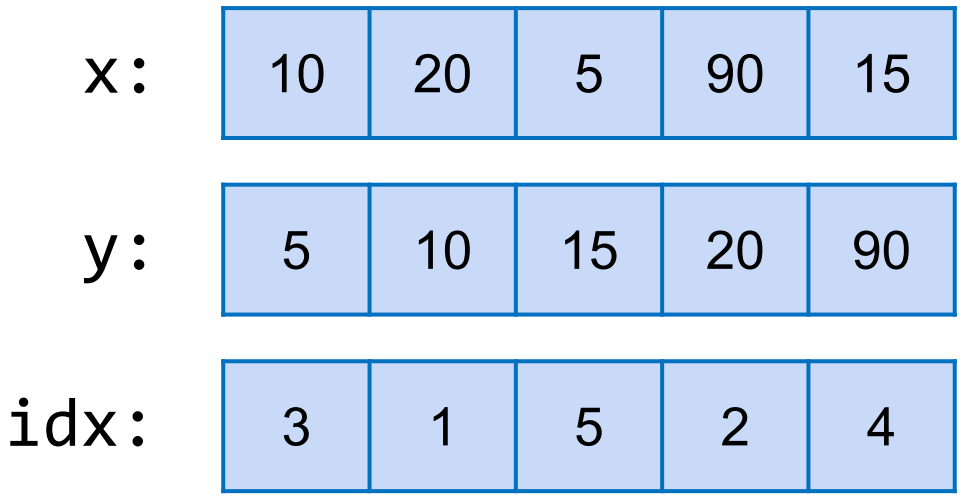
`idx(2) = 1` ⇔ `x(1)` is the 2nd-smallest
`y(2) = x(idx(2)) = x(1)`

⋮

`idx(5) = 4` ⇔ `x(4)` is the 5th-smallest
`y(5) = x(idx(5)) = x(4)`

Built-in `sort()` and permutation indices

```
[y, idx] = sort(x)
```



`idx(1) = 3` ⇔ `x(3)` is the smallest
`y(1) = x(idx(1)) = x(3)`

`idx(2) = 1` ⇔ `x(1)` is the 2nd-smallest
`y(2) = x(idx(2)) = x(1)`

⋮

`idx(5) = 4` ⇔ `x(4)` is the 5th-smallest
`y(5) = x(idx(5)) = x(4)`

In vector notation, **`y = x(idx)`**

Review Question #4

```
function Pts = sortPoints(Pts)
% Given an array of Point objects Pts where each object has two properties,
% x and y, sort Pts so that the objects are in the order of
% increasing distance from (0,0)
```

Review Question #4

#	Thing we need to do	Programming concept needed to do this thing
1	Find distances from `(0,0)` to all `Point` objects	
2		
3		

Review Question #4

#	Thing we need to do	Programming concept needed to do this thing
1	Find distances from `(0,0)` to all `Point` objects	Set a **for-loop** to use each `Point` object in `Pts`
2		
3		

Review Question #4

#	Thing we need to do	Programming concept needed to do this thing
1	Find distances from `(0,0)` to all `Point` objects	Set a **for-loop** to use each `Point` object in `Pts`
2	Compute and store distances for each point	
3		

Review Question #4

#	Thing we need to do	Programming concept needed to do this thing
1	Find distances from `(0,0)` to all `Point` objects	Set a **for-loop** to use each `Point` object in `Pts`
2	Compute and store distances for each point	For each `Point` object, compute the distance with the `x` and `y` properties. Then append/store in the corresponding spot of a vector.
3		

Review Question #4

#	Thing we need to do	Programming concept needed to do this thing
1	Find distances from `(0,0)` to all `Point` objects	Set a **for-loop** to use each `Point` object in `Pts`
2	Compute and store distances for each point	For each `Point` object, compute the distance with the `x` and `y` properties. Then append/store in the corresponding spot of a vector.
3	Sort points in order of increasing distance	

Review Question #4

#	Thing we need to do	Programming concept needed to do this thing
1	Find distances from `(0,0)` to all `Point` objects	Set a **for-loop** to use each `Point` object in `Pts`
2	Compute and store distances for each point	For each `Point` object, compute the distance with the `x` and `y` properties. Then append/store in the corresponding spot of a vector.
3	Sort points in order of increasing distance	Use `sort()` on the distance vector, then use the **permutation indices** to re-order objects in `Pts`.

Review Question #4: **Solution**

```
function Pts = sortPoints(Pts)

    for i = 1:length(Pts)


    end
```

Connection to previous slide:
Red: for-loop to traverse `Pts`
Green: compute/store distances
Blue: sort and re-order `Pts`

Review Question #4: **Solution**

```
function Pts = sortPoints(Pts)

    for i = 1:length(Pts)
        pt = Pts(i);
        d(i) = sqrt(pt.x^2 + pt.y^2);
    end
```

Connection to previous slide:
Red: for-loop to traverse `Pts`
Green: compute/store distances
Blue: sort and re-order `Pts`

Review Question #4: **Solution**

```
function Pts = sortPoints(Pts)

    for i = 1:length(Pts)
        pt = Pts(i);
        d(i) = sqrt(pt.x^2 + pt.y^2);
    end

    [~, idx] = sort(d);
    Pts = Pts(idx);
```

Connection to previous slide:
Red: for-loop to traverse `Pts`
Green: compute/store distances
Blue: sort and re-order `Pts`

Common Student Errors

- Getting the size of an array/Initializing arrays

```
size(A) = [nr, nc];
```
❌

vs

```
[nr, nc] = size(A);
```
✓

- For loops based on array size

```
for k = 1:length(nr)
```
❌

vs

```
for k = 1:nr
```
✓

- 2D Cell Array vs. Arrays in Cells

```
A{1, 2}
```

```
{1, 2, 3;
 4, 5, 6}
```

has 6 cells

```
A{1}(2)
```

```
{[1, 2, 3],
 [4, 6]}
```

has 2 cells