

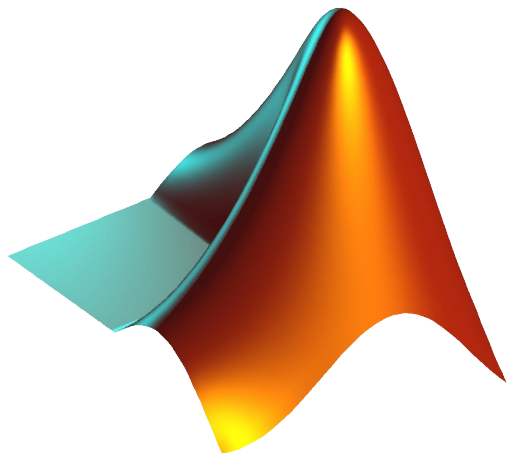
CS 1112 Introduction to Computing Using MATLAB

Instructor: Dominic Diaz

Website:

<https://www.cs.cornell.edu/courses/cs1112/2022fa/>

Today: object-oriented programming



Agenda and announcements

- Last time
 - Object-oriented programming
- Today
 - One last image thing
 - Object-oriented programming
 - Methods other than the constructor
 - Overloading/overriding functions/methods
- Announcements
 - Exercise tomorrow will be half OOP and the other half will be an image question...
 - Project 5 due Monday 11/14
 - Prelim 2 this Thursday!
 - tutoring (sign up on CMS) Tuesday 11/8 - Wednesday 11/9
 - Review session Wednesday 11/9 6:30 – 8pm in Thurston Hall room 203
 - Apply by November 14th if you would like to be a consultant for this class!
 - For assignments and exams, only use function and MATLAB keywords that you have been taught in this course
 - Especially do not use `break`, `continue`, `try`, `catch`, `switch`, `return`

Mistake on lec 17 slides

```
img = imread('ManTakingPhoto.png');
imgBlur = zeros(size(img));
imgBlur = uint8(imgBlur);
% assuming we are finding the average pixel value around
% the pixel in the row 252 and column 283
row = 252; col = 283;
block = img(row-2:row+2, col-2:col+2);

avgPixel = 0;
for i = 1:size(block,1)
    for j = 1:size(block,2)
        avgPixel = avgPixel + block(i,j);
    end
end

numPixels = size(block,1)*size(block,2);
imgBlur(row,col) = avgPixel/numPixels;
```

Block →



Matrix storing pixel values of block ↙

56	75	39	96	94
32	65	46	74	34
25	109	158	245	237
224	235	224	226	234
254	255	251	242	247

Error in lecture 17 slides (fixed now):
avgPixel starts off as double but becomes uint8 after this line. uint8 only stores integers between 0 and 255.

Could have overflow (we could need avg

How to deal with this problem? - solution 1

```
img = imread('ManTakingPhoto.png');
imgBlur = zeros(size(img));
imgBlur = uint8(imgBlur);
% assuming we are finding the average pixel value around
% the pixel in the row 252 and column 283
row = 252; col = 283;
block = img(row-2:row+2, col-2:col+2);
numPixels = size(block,1)*size(block,2);

avgPixel = 0;
for i = 1:size(block,1)
    for j = 1:size(block,2)
        avgPixel = avgPixel + block(i,j)/numPixels;
    end
end

imgBlur(row,col) = avgPixel;
```

Divide by numPixels before
you add

How to deal with this problem? - solution 2

```
img = imread('ManTakingPhoto.png');
imgBlur = zeros(size(img));
imgBlur = uint8(imgBlur);
% assuming we are finding the average pixel value around
% the pixel in the row 252 and column 283
row = 252; col = 283;
block = img(row-2:row+2, col-2:col+2);
```

```
avgPixel = 0;
for i = 1:size(block,1)
    for j = 1:size(block,2)
        avgPixel = avgPixel + double(block(i,j));
    end
end
```

```
numPixels = size(block,1)*size(block,2);
imgBlur(row,col) = avgPixel/numPixels;
```

Turn the uint8 number into a double
so avgPixel stays a double and you
won't have to worry about overflow

Review of OOP

Class: template that specifies a custom type

Object: an instance of a class

Constructor: Special method that returns the handle of an object

```
% construct an object of class interval  
I1 = Interval(1,5);
```

```
% access a property of the object  
r = I1.right;
```

```
% Apply a method on the object  
I1.scale(3);    % scale object by 3
```

```
classdef Interval < handle  
% An interval has a left end and a right end  
  
    properties  
        left  
        right  
    end  
  
    methods  
        function Inter = Interval(lt, rt)  
            % constructor: construct an Interval  
            % object  
            Inter.left = lt;  
            Inter.right = rt;  
        end  
  
        function scale(self,f)  
            % scale the interval by factor f  
            w = self.right - self.left;  
            self.right = self.left + w*f;  
        end  
    end  
end
```

Methods other than the constructor

In the function def, should always have an an object handle as first input*

Call it self.

The other inputs can be anything (handles for other object, doubles, arrays, ...).

To call the method:

```
I2 = Interval(0,1);  
I2.scale(5)
```

*methods without object handle as first input would be called a static method.

```
< handle  
; a left end and a right end
```

```
properties  
    left  
    right  
end  
  
methods  
    function Inter = Interval(lt, rt)  
        % constructor: construct an Interval  
        % object  
        Inter.left = lt;  
        Inter.right = rt;  
    end  
  
    function scale(self,f)  
        % scale the interval by factor f  
        w = self.right - self.left;  
        self.right = self.left + w*f;  
    end  
end  
end
```

Methods other than the constructor

```
objHandle.methodName(inputParam2, inputParam3, ..., inputParamN)
```

To call the method:

```
I2 = Interval(0,1);  
I2.scale(5)
```

What happens when we call this function?

Self stores handle to I2

f stores the value 5

$w = 1 - 0 = 1$

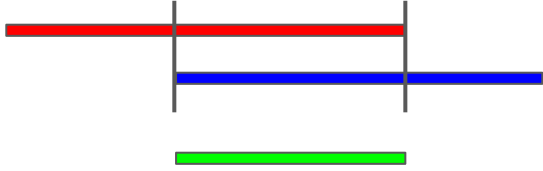
right prop of object becomes $0 + 5*1$

```
disp(I2)  
% interval with properties:  
left: 0  
right: 5
```

```
< handle  
a left end and a right end
```

```
left  
right  
end  
  
methods  
function Inter = Interval(lt, rt)  
% constructor: construct an Interval  
% object  
Inter.left = lt;  
Inter.right = rt;  
end  
  
function scale(self,f)  
% scale the interval by factor f  
w = self.right - self.left;  
self.right = self.left + w*f;  
end  
  
end  
end
```


Method to find overlap between two intervals



Method to find overlap between two intervals



In each of the overlapping cases, I can set

```
left = max(red.left, blue.left);  
right = min(red.right, blue.right);  
intervalOverlap = Interval(left, right);
```

Method to find overlap between two Intervals

```
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% Inter is empty array of class Interval.
```

```
end
```

Method to find overlap between two Intervals

```
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% Inter is empty array of class Interval.
Inter = Interval.empty();
left = max(self.left, other.left);
right = min(self.right, other.right);
if right-left > 0
    Inter = Interval(left, right);
end
end
```

Method to find overlap between two Intervals

```
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% Inter is empty array of class Interval.
Inter = Interval.empty();
left = max(self.left, other.left);
right = min(self.right, other.right);
if right-left > 0
    Inter = Interval(left, right);
end
end
```


Built-in function to create an empty array of the specified class

```
% Example use of the overlap function
I1 = Interval(3,7);
I2 = Interval(4, 4+rand*5);
X = I1.overlap(I2);
if ~isempty(X)
    fprintf('(%f,%f)\n', X.left,X.right)
end
```

Method to find overlap between two Intervals

```
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% Inter is empty array of class Interval.
Inter = Interval.empty();
left = max(self.left, other.left);
right = min(self.right, other.right);
if right-left > 0
    Inter = Interval(left, right);
end
end
```

Let's look up the
documentation for isempty



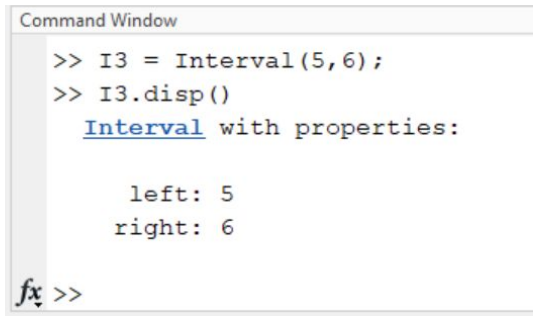
```
% Example use of the overlap function
I1 = Interval(3,7);
I2 = Interval(4, 4+rand*5);
X = I1.overlap(I2);
if ~isempty(X)
    fprintf('(%f,%f)\n', X.left,X.right)
end
```

Overriding built-in functions

- You can change the behavior of a built-in function for an object of a class by implementing a function of the same name in the class definition
 - Called **overriding** (called overloading in MATLAB documentation)
- A typical built-in function to override is disp

% Without overriding the disp function

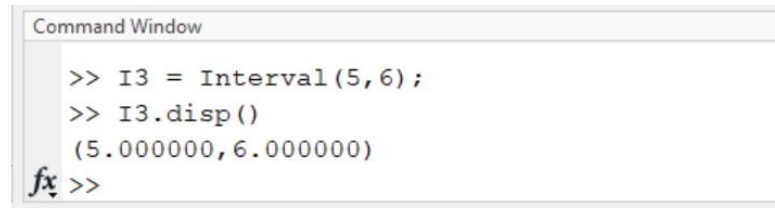
```
I3 = Interval(5,6);  
disp(I3)
```



```
Command Window  
  
>> I3 = Interval(5,6);  
>> I3.disp()  
Interval with properties:  
  
    left: 5  
    right: 6  
  
fx >>
```

% Overriding the disp function with your own display method

```
I3 = Interval(5,6);  
disp(I3)
```



```
Command Window  
  
>> I3 = Interval(5,6);  
>> I3.disp()  
(5.000000,6.000000)  
fx >>
```

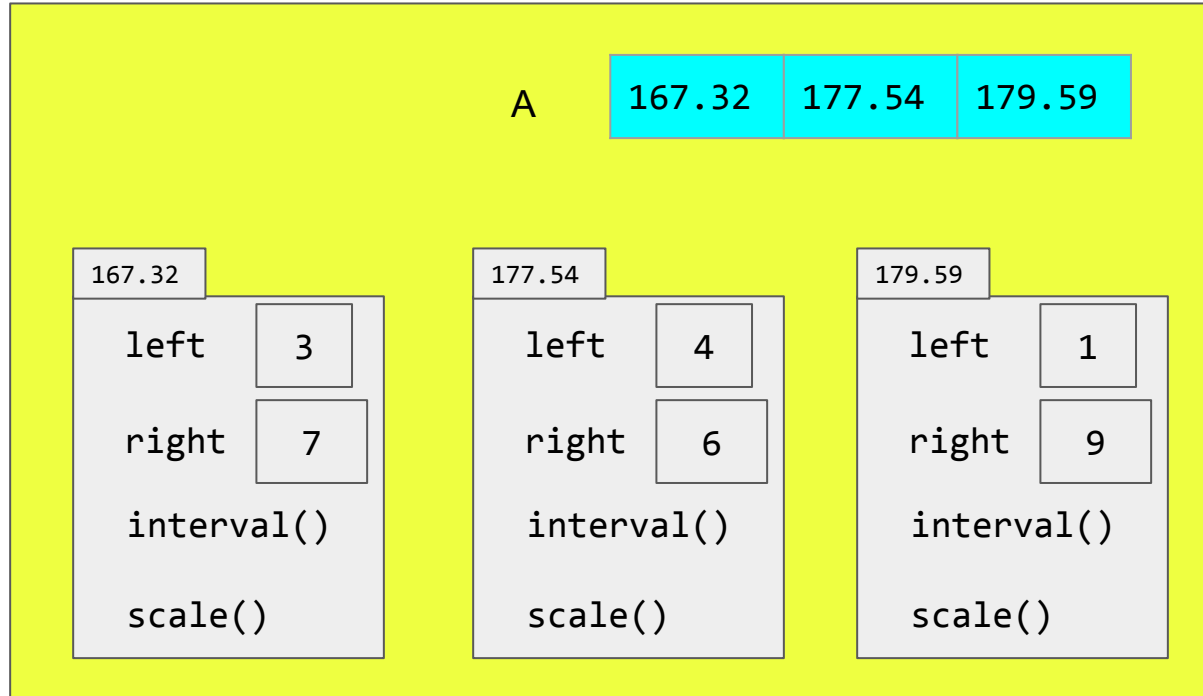
You will write your own disp method in the Interval class def tomorrow!

Let's talk about arrays of object (actually, array of references to objects)

```
A = Interval(3,7);
```

```
A(2) = Interval(4,6);
```

```
A(3) = Interval(1,9);
```



Constructor should be able to handle call with no inputs

```
A = Interval(3,7);  
A(2) = Interval(4,6);  
A(3) = Interval(1,9);  
A(5) = Interval(2,5);
```

Error!

Why is there an error? The Interval constructor requires two input parameters

The user specified 2 inputs for A(5), but...

In order to assign A(4), MATLAB has to make an interval—call the constructor—but with no values for the arguments → **Error!**

```
properties  
    left  
    right  
end  
  
methods  
    function Inter = Interval(lt, rt)  
        % constructor: construct an Interval  
        % object  
        Inter.left = lt;  
        Inter.right = rt;  
    end  
  
    function scale(self,f)  
        % scale the interval by factor f  
        w = self.right - self.left;  
        self.right = self.left + w*f;  
    end  
end  
end
```

Function overloading

Problem: the empty constructor passes 0 inputs but our constructor requires 2 input arguments.

Solution: need a new keyword that allows us to write the constructor to work with variable numbers of input arguments

MATLAB's solution: `nargin`

```
classdef Interval < handle
% An interval has a left end and a right end

    properties
        left
        right
    end

    methods
        function Inter = Interval(lt, rt)
% constructor: construct an Interval
% object
            Inter.left = lt;
            Inter.right = rt;
        end

        function scale(self,f)
% scale the interval by factor f
            w = self.right - self.left;
            self.right = self.left + w*f;
        end
    end
end
```

Examples of function overloading

- `plot(x,y)`, `plot(x,y, 'm-*')`

Plot works with 2 inputs or 3 inputs (or even more)

We need our constructors to work for variable numbers of inputs. We can do this using the `nargin` keyword

- `rand()`, `rand(2)`, `rand(1,3)`

rand works with 0 inputs, 1 input, 2 inputs, ...