

- **Previous lecture:**
 - Objects are passed by reference to functions
 - Details on class definition (constructor, instance method)
- **Today's lecture:**
 - More practice with instance methods
 - Overriding methods
 - Array of objects
 - Methods that handle variable numbers of arguments
- **Announcements:**
 - **Prelim 2** tonight 7:30pm
 - CALS Auditorium in Kennedy Hall (Rm 116)
 - **Lab exercise problem 2** to be submitted on CMS by Monday 11/16, at 11pm.

classdef syntax summary

A class file has the name of the class and begins with keyword `classdef`:

```
classdef classname < handle
```

The class specifies
handle objects

Constructor returns a reference to the class object

Each instance method's first parameter must be a reference to the instance (object) itself

Use keyword `end` for `classdef`, `properties`, `methods`, `function`.

Properties

properties

```
left  
right  
end
```

Constructor

methods

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

Instance
methods
(functions)

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

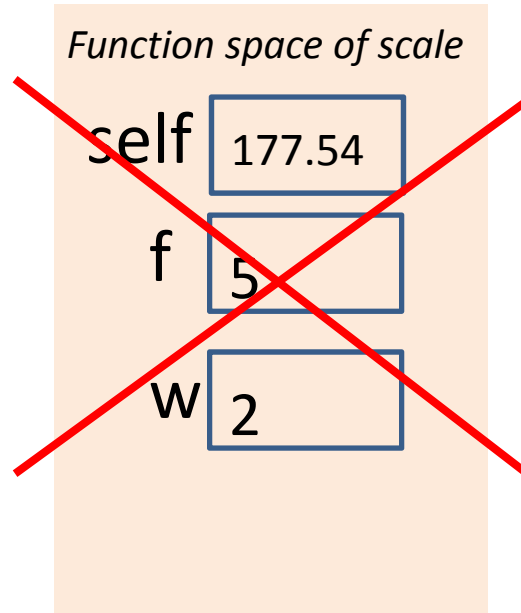
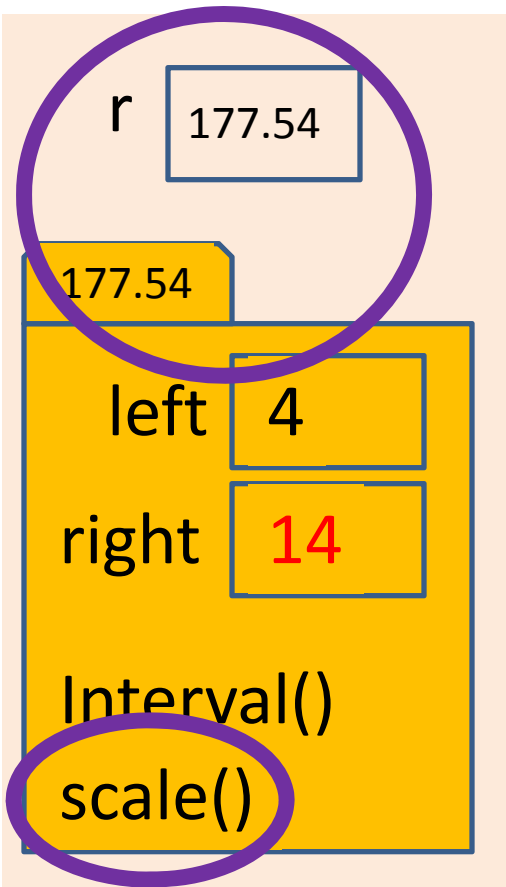
```
end  
end
```

This file's name is Interval.m

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

Object is passed to a function by reference

```
r = Interval(4,6);  
r.scale(5)  
disp(r.right) % updated value
```



Objects are passed to functions by reference. Changes to an object's property values made through the local reference (self) stays in the object even after the local reference is deleted when the function ends.

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

properties

left

right

end

methods

```
function Inter = Interval(lt, rt)
```

```
% Constructor: construct an Inte
```

```
Inter.left= lt;
```

```
Inter.right= rt;
```

```
end
```

```
function scale(self, f)
```

```
% Scale the interval by a factor f
```

```
w= self.right - self.left;
```

```
self.right= self.left + w*f;
```

```
end
```

Syntax for calling an instance method:

<reference>.<method>(<arguments for 2nd thru last parameters>)

```
p = Interval(3,7);  
r = Interval(4,6);
```

```
yesno= p.isIn(r);  
% Explicitly call  
% p's isIn method
```

```
yesno= isIn(p,r);  
% Matlab chooses the  
% isIn method of one  
% of the parameters.
```

Better!

```
classdef Interval < handle  
:  
methods  
:  
function scale(self, f)  
% Scale self by a factor f  
w= self.right - self.left;  
self.right= self.left + w*f;  
end  
  
function tf = isIn(self, other)  
% tf is true if self is in other interval  
tf= self.left>=other.left && ...  
self.right<=other.right;  
end  
  
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
% self is empty Interval.
```

Compare two intervals

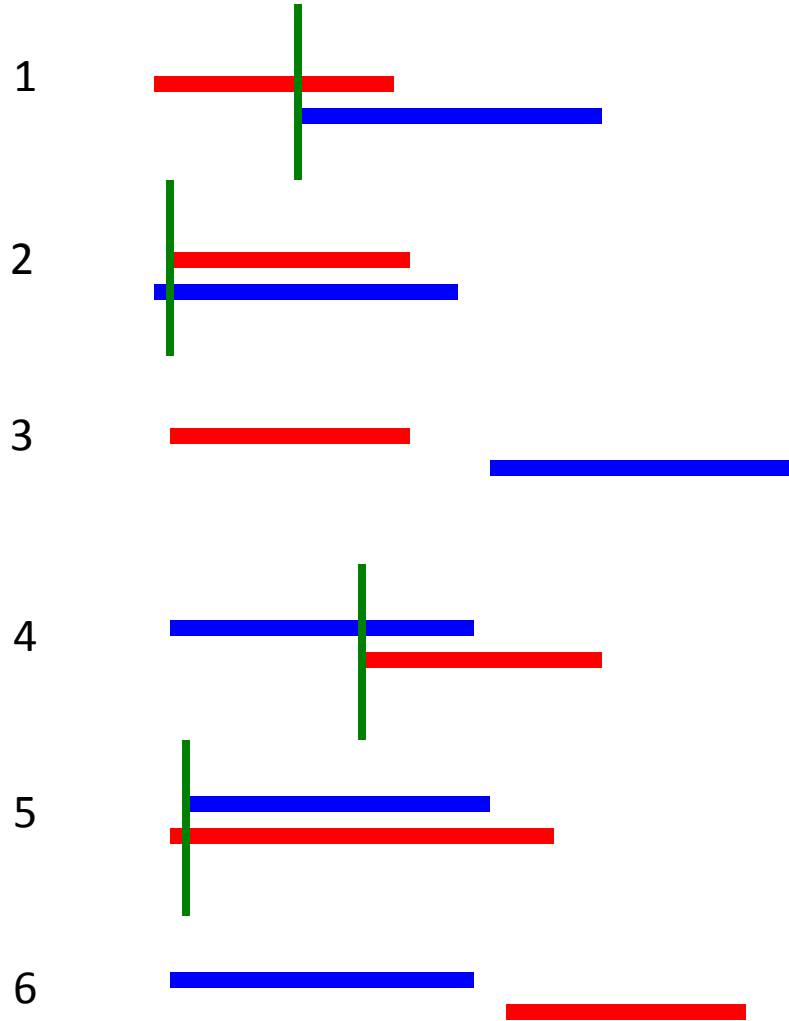


redRight < blueRight

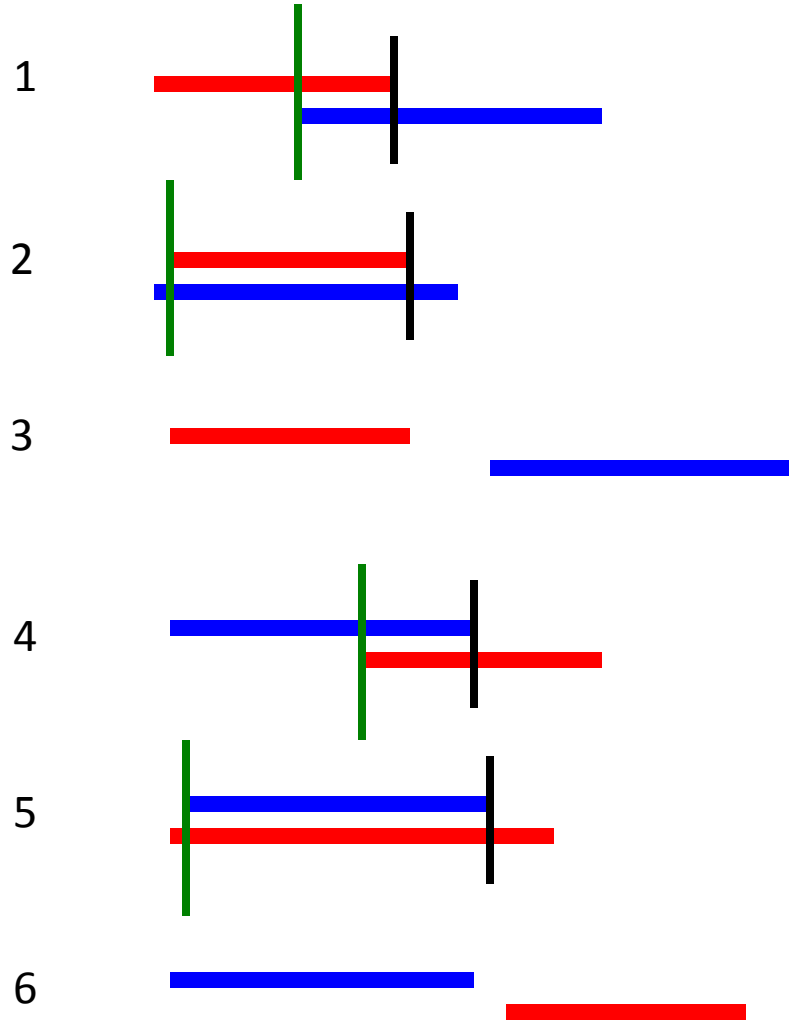


blueRight < redRight



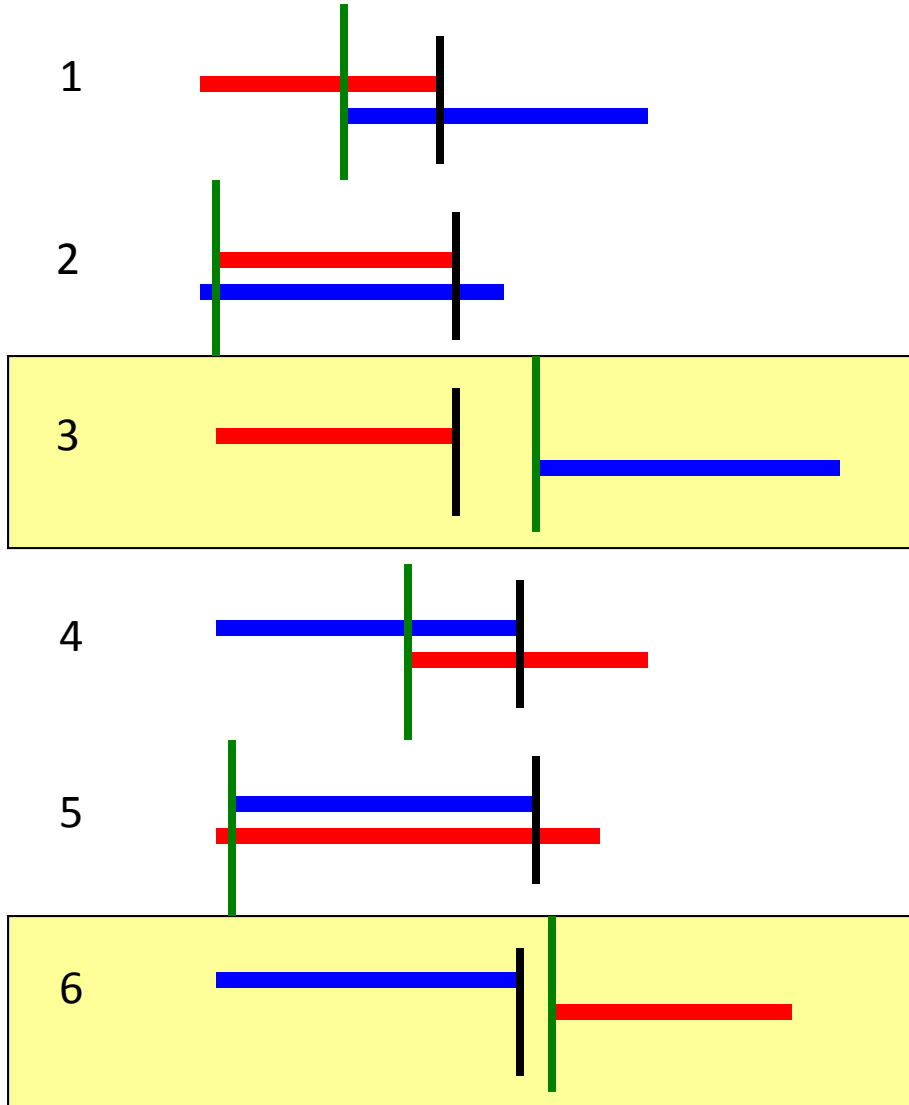


The overlap's left (OLeft) is the rightmost of the two original lefts



The overlap's left (OLeft) is the rightmost of the two original lefts

The overlap's right (ORight) is the leftmost of the two original rights



The overlap's left (OLeft) is the rightmost of the two original lefts

The overlap's right (ORight) is the leftmost of the two original rights

No overlap if $OLeft > ORight$

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

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

```
% Example use of overlap function
A= Interval(3,7);
B= Interval(4,4+rand*5);
X= A.overlap(B);
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**
 - Specify which properties to display, and how, when the argument to **disp** is (a reference to) an object
 - Matlab calls **disp** when there’s no semi-colon at the end of an assignment statement

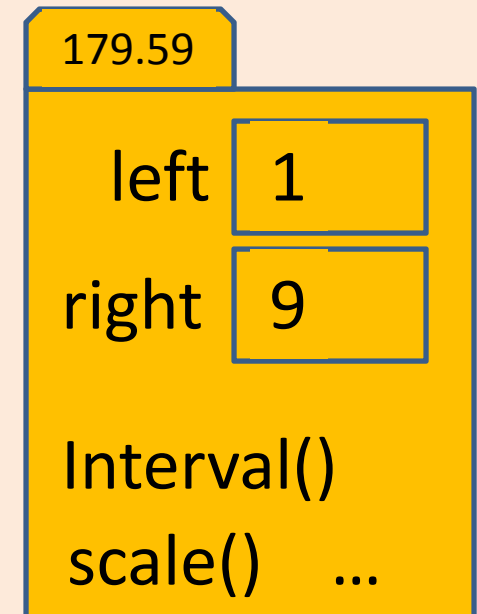
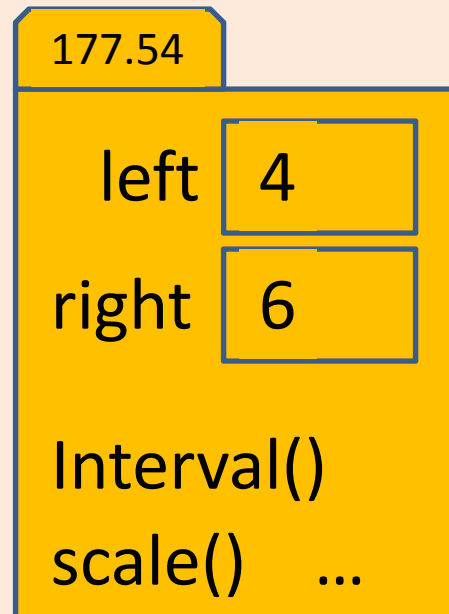
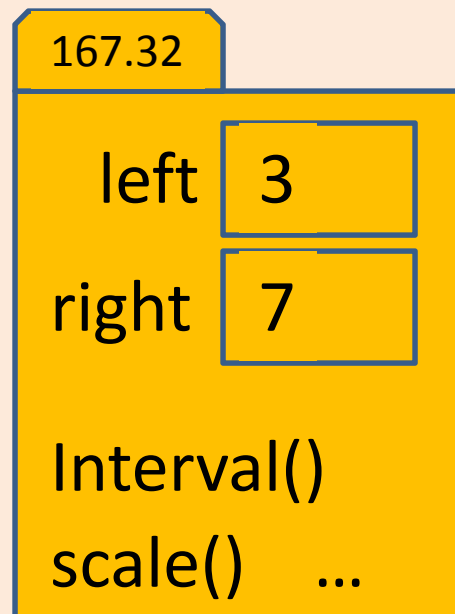
See `Interval.m`

An “array of objects” is really an ...

array of **references** to objects

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

A	167.32	177.54	179.58
---	--------	--------	--------



MATLAB allows an array to be appended

```
v = [3 1 5 9]
```

```
v(7) = 4
```

- What happens to $v(5)$ and $v(6)$?

3	1	5	9	0	0	4
---	---	---	---	---	---	---

- MATLAB assigns some “default value” to the skipped over components for simple, cell, and struct arrays
- For arrays of objects, you must implement the constructor to handle such a situation

Constructor needs to be able to handle a call with no arguments

```
>> A= Interval(3,7);      % Array of length 1
>> A(2)= Interval(4,6); % Array of length 2
>> A(3)= Interval(1,9); % Array of length 3
>> A(5)= Interval(2,5); % Array of length 5
```

Error!

- Interval constructor we have so far requires two parameters:
`function Inter = Interval(lt, rt)`
- User specified two arguments as required for A(5), but...
- Matlab has to assign A(4) "on its own" by calling the constructor, but no arguments get passed → **Error!**

Constructor that handles variable number of args

- When used inside a function, **nargin** returns the number of arguments that were passed

```
classdef Interval < handle

    properties
        left
        right
    end

    methods
        function Inter = Interval(lt, rt)

            Inter.left= lt;
            Inter.right= rt;

        end

        ...

    end

end
```

Constructor that handles variable number of args

- When used inside a function, **nargin** returns the number of arguments that were passed
- If **nargin**≠2, constructor ends without executing the assignment statements. Then **Inter.left** and **Inter.right** get any default values defined under properties. In this case the default property values are **[]** (type **double**)

```
classdef Interval < handle

    properties
        left
        right
    end

    methods
        function Inter = Interval(lt, rt)
            if nargin==2
                Inter.left= lt;
                Inter.right= rt;
            end
        end

        ...

    end

end
```

If a class defines an object that may be used in an array...

- **Constructor must be able handle a call that does not specify any arguments**
 - Use built-in command `nargin`, which returns the number of function input arguments passed
- The overridden `disp` method, if implemented, should check for an input argument that is an array and handle that case explicitly. Details will be discussed next lecture.

A function to create an array of **Intervals**

```
function inters = intervalArray(n)
% Generate n random Intervals.  The left and
% right ends of each interval is in (0,1)

for k = 1:n
    randVals= rand(1,2);
    if randVals(1) > randVals(2)
        tmp= randVals(1);
        randVals(1)= randVals(2);
        randVals(2)= tmp;
    end
    inters(k)= Interval(randVals(1),randVals(2));
end
```

An independent function, not an instance method. See `intervalArray.m`

A function to find the widest **Interval** in an array

```
function inter = widestInterval(A)
% inter is the widest Interval (by width) in
% A, an array of Intervals
```

An independent function, not an instance method. See `widestInterval.m`

A function to find the widest Interval in an array

```
function inter = widestInterval(A)
% inter is the widest Interval (by width) in
% A, an array of Intervals

inter= A(1); % widest Interval so far
for k= 2:length(A)
    if A(k).right - A(k).left > ...
        inter.right - inter.left
        inter= A(k);
    end
end
```

An independent function, not an instance method. See `widestInterval.m`

A function to find the widest **Interval** in an array

```
function inter = widestInterval(A)
% inter is the widest Interval (by width) in
% A, an array of Intervals

inter= A(1); % widest Interval so far
for k= 2:length(A)
    if A(k).getWidth() > inter.getWidth()
        inter= A(k);
    end
end
end
```

An independent function, not an instance method. See `widestInterval.m`