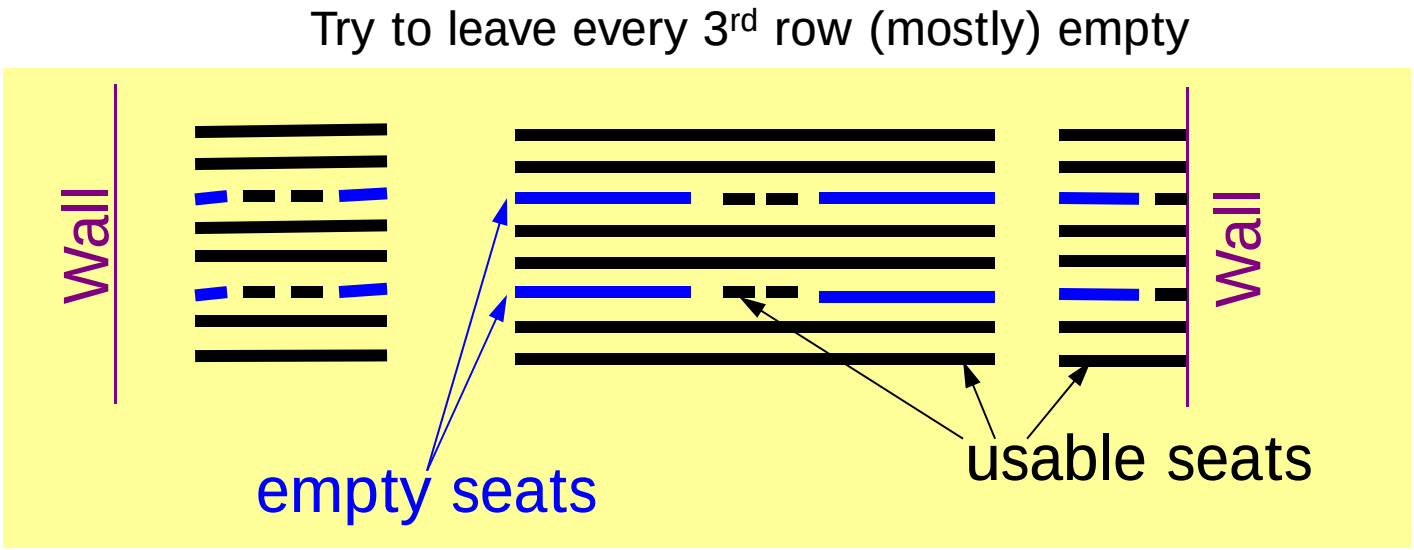


- Announcements:
  - Prelim 2 tonight 7:30pm
    - Last names A-J: Olin Hall 155
    - Last names K-Z: Uris Hall G01
  - Lab exercise problem 2 to be submitted on CMS by Monday 11/17, at 11pm.

- Previous lecture:
  - Objects are passed by reference to functions
  - Details on class definition (constructor, instance method)
- Today's lecture:
  - Overriding methods
  - Array of objects
  - Methods that handle variable numbers of arguments
- Announcements:
  - Prelim 2 tonight 7:30pm
    - Last names A-J: Olin Hall 155
    - Last names K-Z: Uris Hall G01
  - Lab exercise problem 2 to be submitted on CMS by Monday 11/17, 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

Constructor

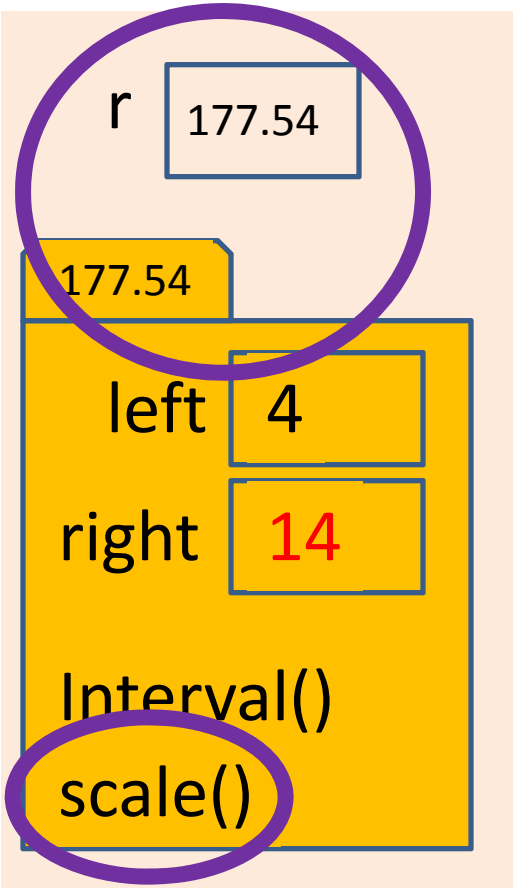
Instance methods (functions)

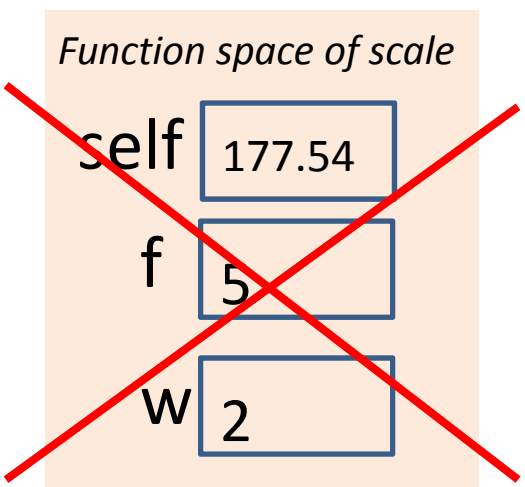
```
classdef  Interval < handle
% An Interval has a left end and a right end

   properties
      left
      right
   end

   methods
      function Inter = Interval(lt, rt)
      % Constructor:  construct an Interval obj
         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
   end
end
```

This file's name is Interval.m

Object is passed to a function by reference

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



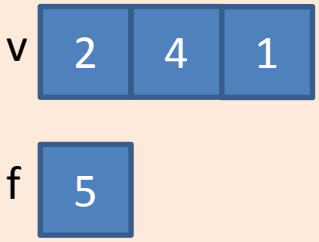



```
classdef  Interval < handle
% An Interval has a left end and a rig

   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
```

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.

Command Window workspace

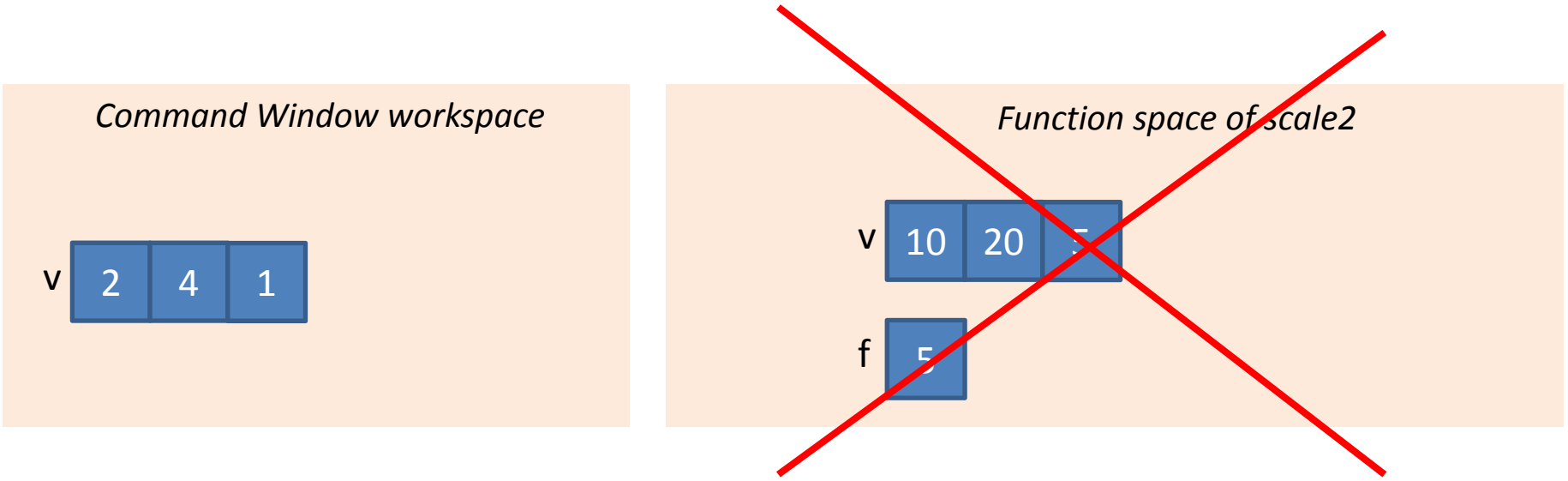
Function space of scale2




```
v= [2 4 1];
scale2(v,5)
disp(v) %???
```

```
function scale2(v,f)
% Scale v by a factor f
v= v*f;
```

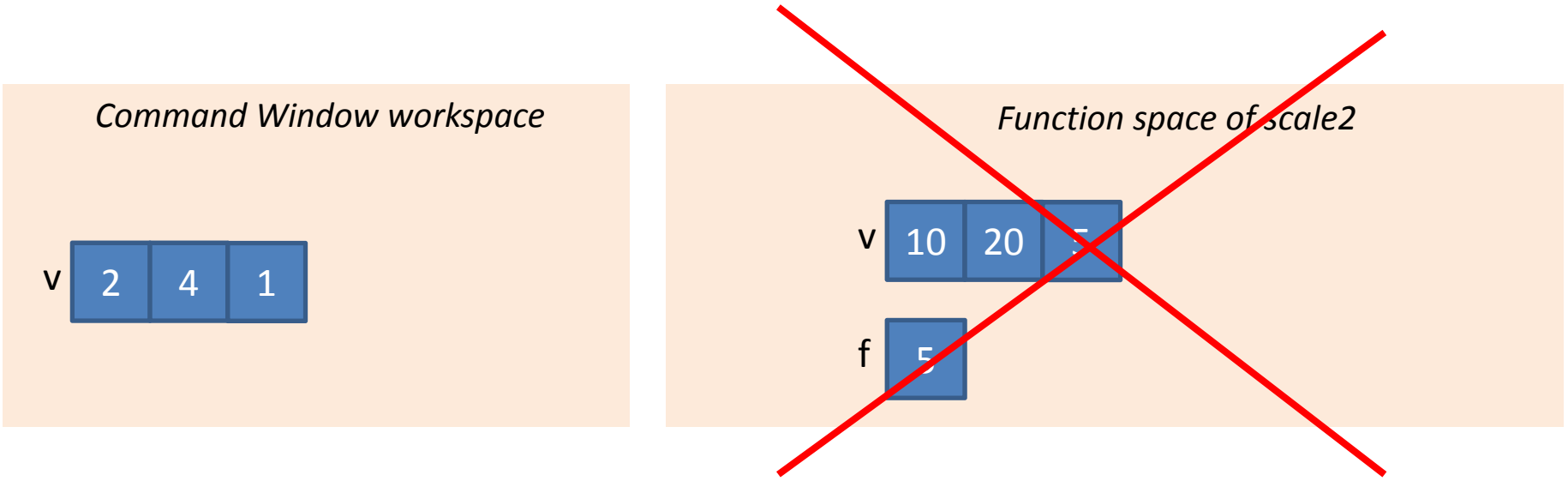
Non-objects are passed to a function by value




```
v= [2 4 1];
scale2(v,5)
disp(v) %???
```

```
function scale2(v,f)
% Scale v by a factor f
v= v*f;
```

Non-objects are passed to a function by value




```
v= [2 4 1];
scale2(v,5)
disp(v) %NO CHANGE
```

```
function scale2(v,f)
% Scale v by a factor f
v= v*f;
```

Non-objects are passed to a function by value

Objects are passed to a function by reference

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

```
classdef Interval < handle
  :
  methods
    :
    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
```

```
v= [2 4 1];
scale2(v,5)
disp(v) %NO CHANGE
```

```
function scale2(v,f)
% Scale v by a factor f
v= v*f;
```

Non-objects are passed to a function by value

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.
```

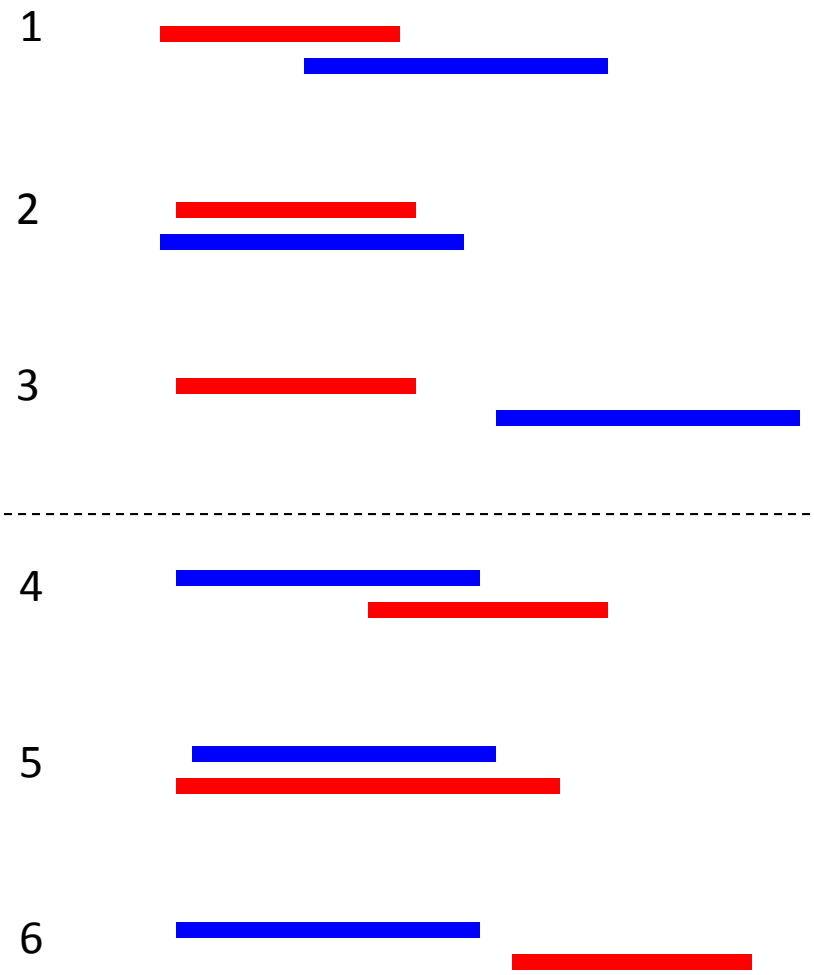
Better!

```
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

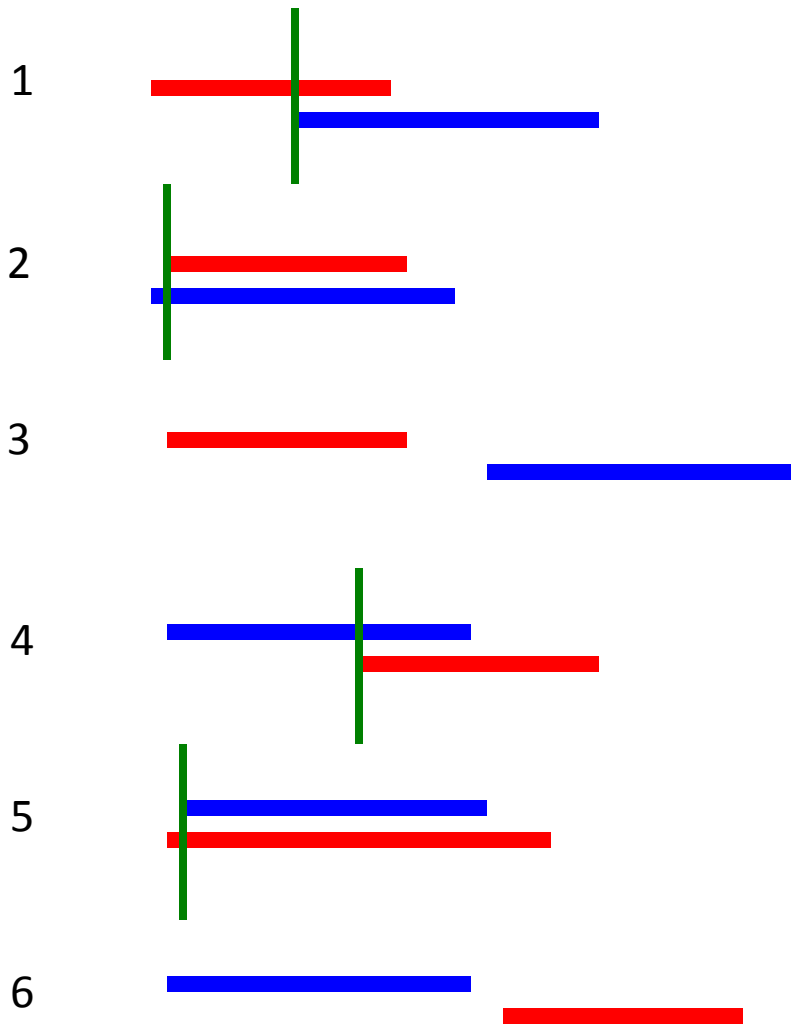
```
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% self is empty Interval.
```

Compare two intervals

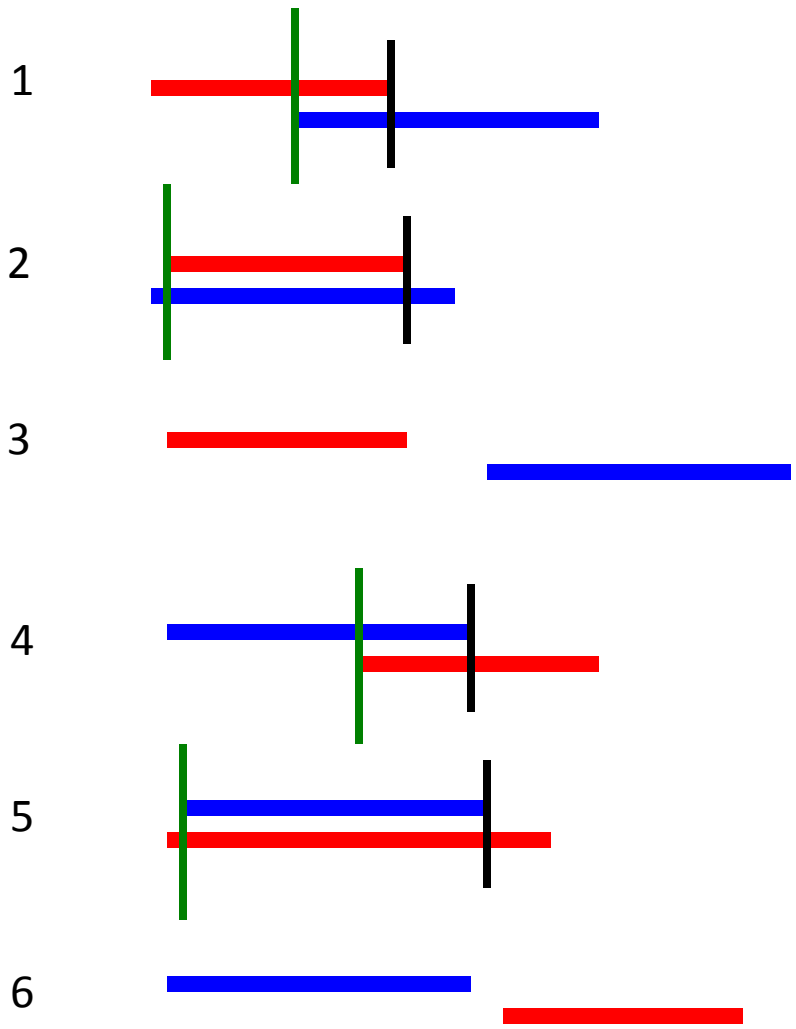


redRight < blueRight

blueRight < redRight

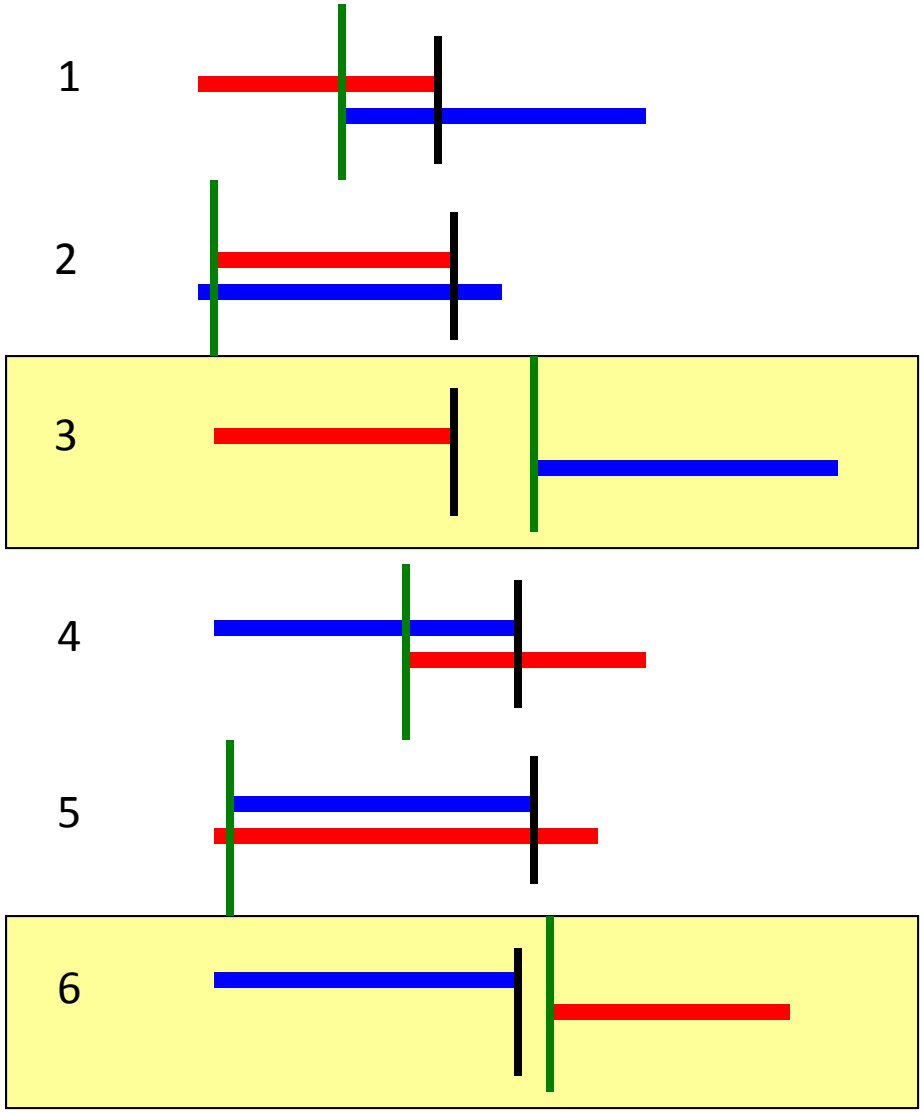


The overlap's left (OLeft) is the rightmost of the two original lefts



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 to create an empty array of the specified class

Built-in function isempty

```
% 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

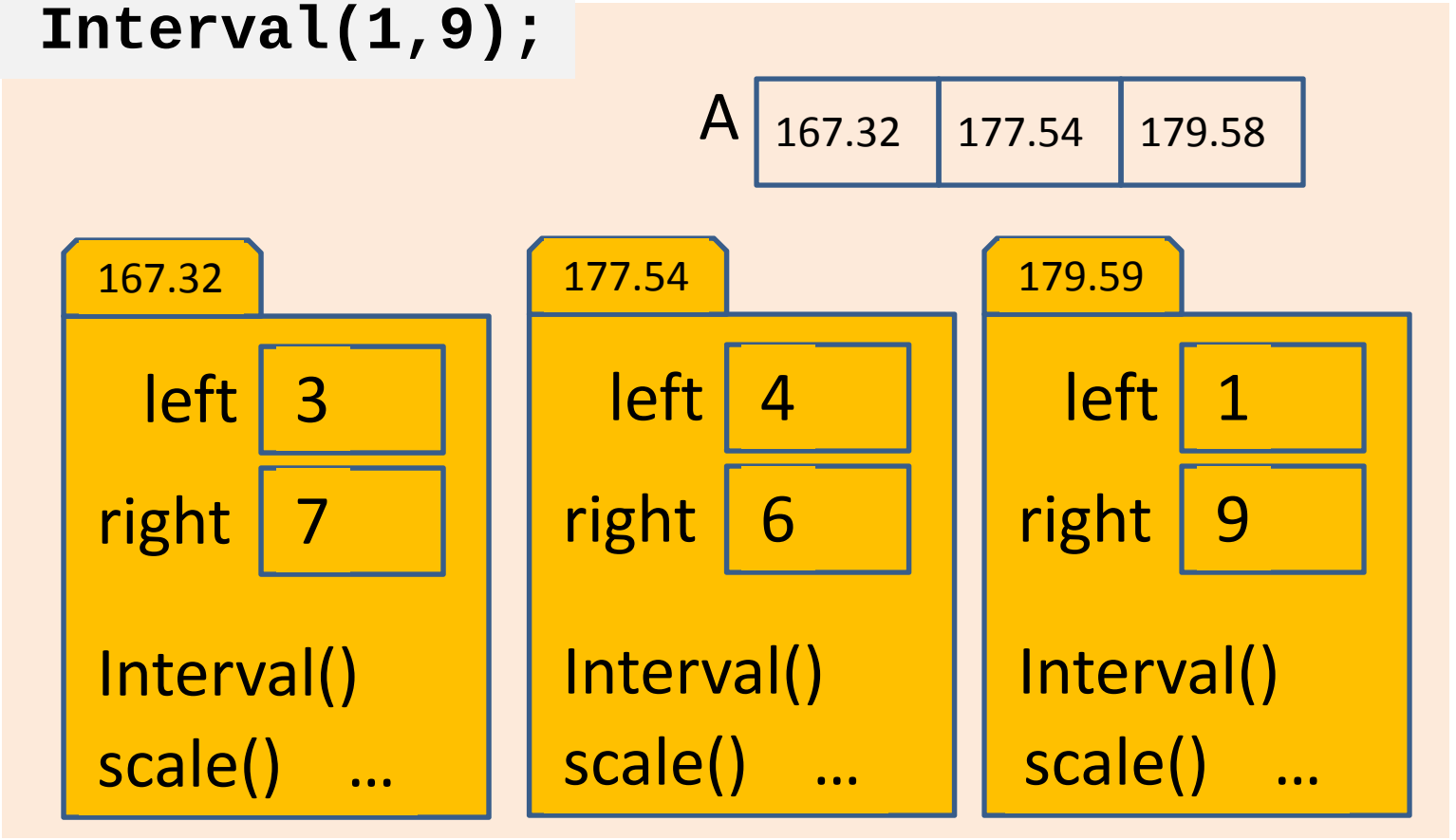
- You can change the behavior of a built-in function for an object of a class by implementing a function of the same name in the class definition
- Called "overriding" (called "overloading" in Matlab documentation)
- A typical built-in function to override is **`disp`**
  - Specify which properties to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



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.

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
```

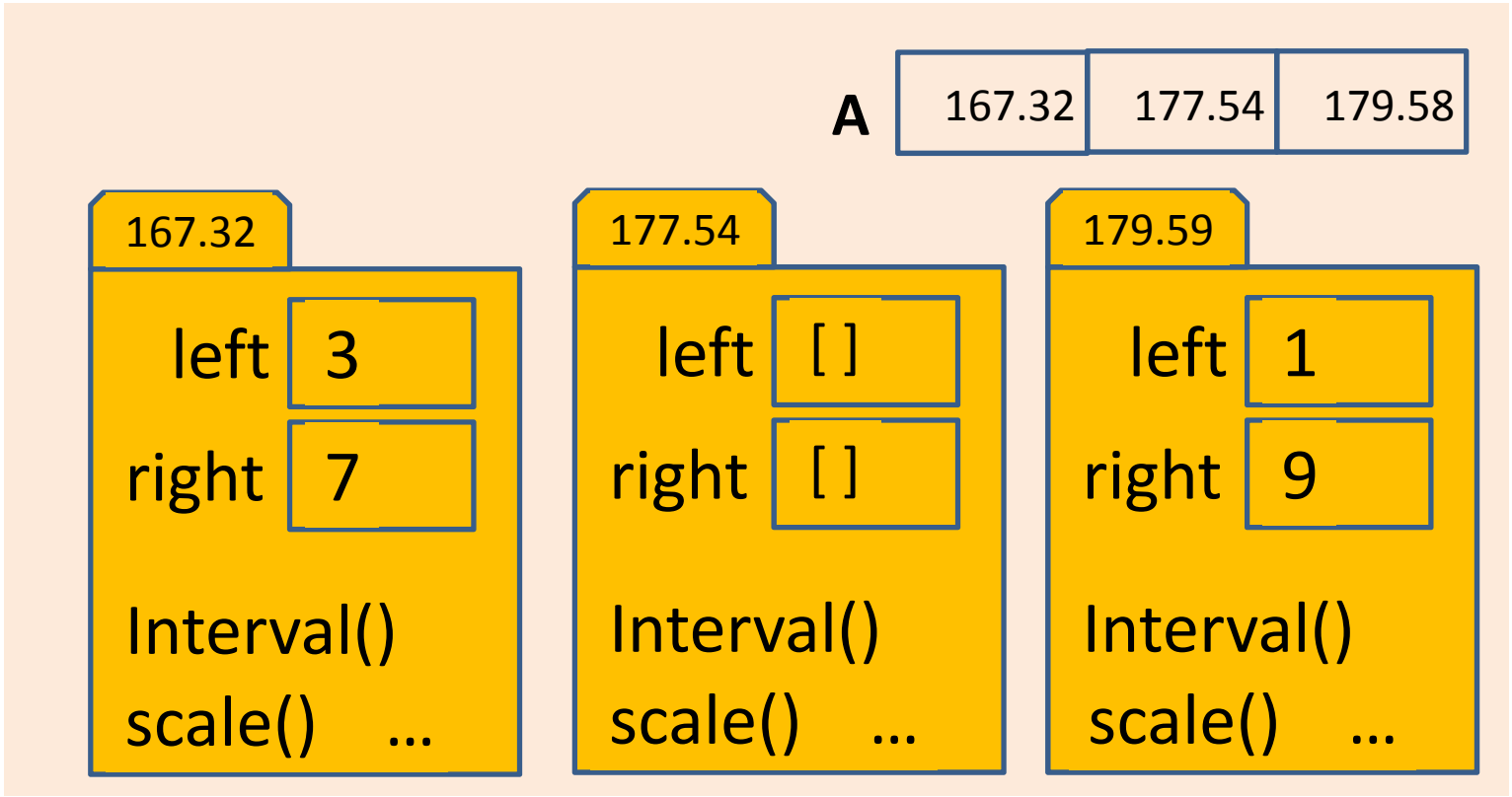
Constructor that handles variable number of args

- 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
```

Constructor should handle variable number of args

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



Matlab has to create an Interval object to put in **A(2)** and it calls the constructor without any arguments.

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 biggest Interval in an array

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

A function to find the biggest Interval in an array

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

inter= A(1);  % biggest 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 **biggestInterval.m**

A function to find the biggest Interval in an array

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

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

A weather object can make use of Intervals …

- Define a class LocalWeather to store the weather data of a city, including monthly high and low temperatures and precipitation
  - Temperature: low and high → an Interval
    - For a year → length 12 array of Intervals
  - Precipitation: a scalar value
    - For a year → length 12 numeric vector
  - Include the city name: a string

```
classdef LocalWeather < handle

  properties
    city     % string
    temps    % array of Intervals
    precip   % numeric vector
  end

  methods
    ...
  end

end
```

Weather data file

```
//Ithaca
//Monthly temperature and precipitation
//Lows (cols 4-8), Highs (col 12-16), precip (cols 20-24)
//Units:  English
      15       31    2.08
      17       34    2.06
      23       42    2.64
      34       56    3.29
      44       67    3.19
      53       76    3.99
      58       80    3.83
      56       79    3.63
      49       71    3.69
     NaN       59     NaN
      32       48    3.16
      22       36    2.40
```

Class **LocalWeather** should be able to construct an object from such data files, given the known file format.

See **ithacaWeather.txt**, **LocalWeather.m**

```
classdef  LocalWeather < handle

  properties
    city= ”;
    temps= Interval.empty();
    precip
  end

  methods
    function lw = LocalWeather(fname)
      …
    end

    …
  end

end
```

Set property variable that will store an array of objects to the correct type, either under properties or in the constructor

```
classdef LocalWeather < handle
 properties
  city=''; temps=Interval.empty(); precip=0;
 end
 methods
  function lw = LocalWeather(fname)
   fid= fopen(fname,'r');
   s= fgetl(fid);
   lw.city= s(3:length(s));
   for k= 1:3
      s= fgetl(fid);
   end
   for k=1:12
      s= fgetl(fid);
      lw.temps(k)= Interval(str2double(s(4:8), ...
                            str2double(s(12:16)));
      lw.precip(k)= str2double(s(20:24));
   end
   fclose(fid);
  end
  ...
 end  %methods
end  %classdef
```

```
//Ithaca
//Monthly temperature and p
//Lows (cols 4-8), Highs (c
//Units:  English
      15      31    2.08
      17      34    2.06
      23      42    2.64
      34      56    3.29
      44      67    3.19
      53      76    3.99
      58      80    3.83
      56      79    3.63
      49      71    3.69
     NaN      59     NaN
      32      48    3.16
      22      36    2.40
```