

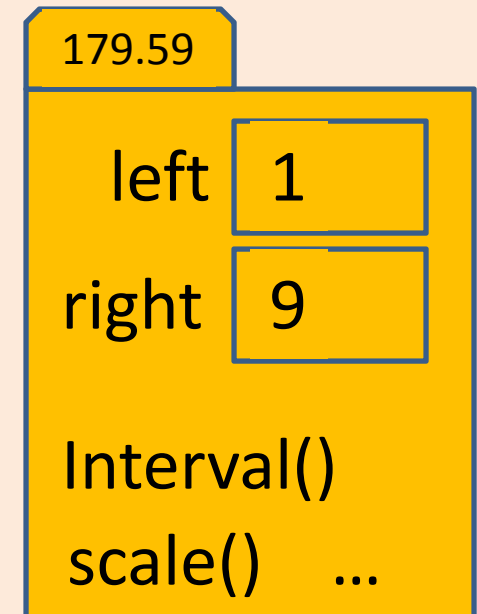
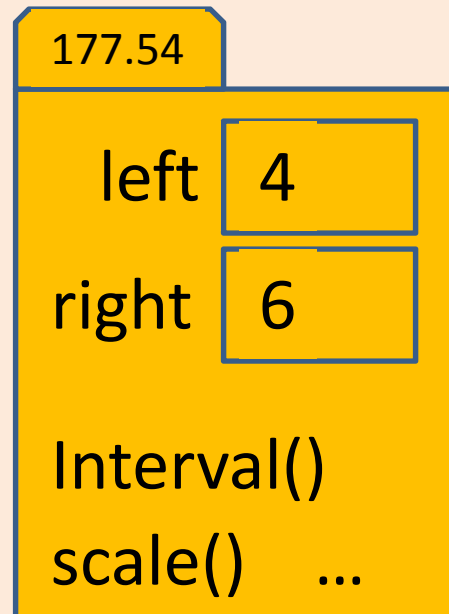
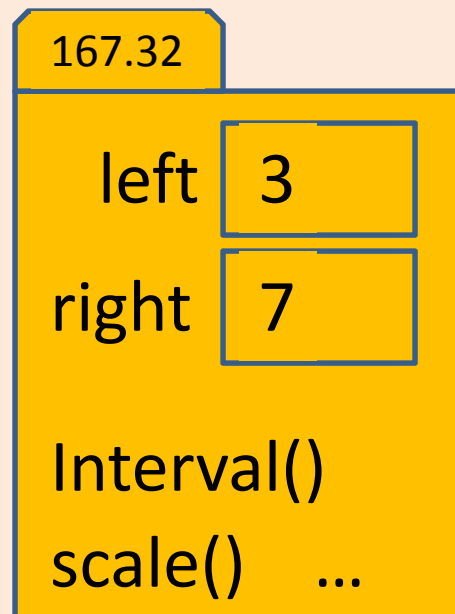
- Previous lecture:
 - Objects are passed by reference to functions
 - Details on class definition
 - Constructor
 - Methods
 - Overloading methods
- Today's lecture:
 - Array of objects
 - Methods that handle variable numbers of arguments
 - Attributes for properties and methods
- Announcements:
 - Discussion this week in Upson B7 lab
 - Prelim 2 to be returned at end of lecture. Unclaimed papers (and those on which student didn't indicate the lecture time) can be picked up after 5pm today during consulting hours (Su-R 5-10p) at ACCEL Green Rm (Carpenter Hall)

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
---	--------	--------	--------



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
- An overloaded **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
```

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
```

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).getWidth() > inter.getWidth()
        inter= A(k);
    end
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

```

```
classdef LocalWeather < handle
  properties
    city=""; temps=Interval.empty();
    precip=0;
  end
  methods
    function lw = LocalWeather(fname)
      ...
    end

    function showCityName(self)
      ...
    end
  end %methods
end %classdef
```

```
classdef LocalWeather < handle
  properties
    city=""; temps=Interval.empty();
    precip=0;
  end
  methods
    function lw = LocalWeather(fname)
      ...
    end

    function showCityName(self)
      disp(self.city)
    end
    ...
  end %methods
end %classdef
```

Function to show data of a month of **LocalWeather**

```
function showMonthData(self, m)
% Show data for month m, 1<=m<=12.

end
```

Should display which month, the high and low temperatures, and precipitation

Function to show data of a month of **LocalWeather**

```
function showMonthData(self, m)
% Show data for month m, 1<=m<=12.

mo= { 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', ...
      'July', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec' };
fprintf('%s Data\n', mo{m})
fprintf('Temperature range: ')
disp(self.temps(m))
fprintf('Average precipitation: %.2f\n', ...
      self.precip(m))
end
```

See `LocalWeather.m`

Observations about our class `Interval`

- We can use it (create `Interval` objects) anywhere
 - Within the `Interval` class, e.g., in method `overlap`
 - “on the fly” in the Command Window
 - In other function/script files – not class definition files
 - In another class definition
- Designing a class well means that it can be used in many different applications and situations