- Previous lecture:
  - File I/O, sort
- Today's lecture:
  - Introduction to objects and classes
- Announcements:
  - Exercise important for Thurs lecture
  - Prelim Thurs, 6:30pm
    - Don't look at review solutions until completing problems yourself
  - Project 4B solutions on CMS
  - Project 5 due next week
    - Start early! Use it to study for prelim

Different kinds of abstraction

- Packaging procedures (program instructions) into a function
  - A program is a set of functions executed in the specified order
  - Data is passed to (and from) each function
- Packaging data into an array or structure
  - Elevates thinking
  - Reduces the number of variables being passed to and from functions
- Packaging data, and the instructions that work on those data, into an object
  - A program is the interaction among objects
  - Object-oriented programming (OOP) focuses on the design of data-instructions groupings

A card game, developed in two ways

- Develop the algorithm—the logic—of the card game:
  - Set up a deck as an array of cards. (First, choose representation of cards.)
  - Shuffle the cards
  - Deal cards to players
  - Evaluate each player's hand to determine winner

Procedural programming: focus on the algorithm, i.e., the procedures, necessary for solving a problem

- Identify "objects" in the game and define each:
  - Card
    - Properties: suit, rank
    - Actions: compare, show
  - Deck
    - Property: array of Cards
    - Actions: shuffle, deal, get #cards left
  - Hand …
  - Player …
- Then write the game—the algorithm—using objects of the above "classes"

A card game, developed in two ways

- Develop the algorithm—the logic—of the card game:
  - Set up a deck as an array of cards. (First, choose representation of cards.)
  - Shuffle the cards
  - Deal cards to players
  - Evaluate each player's hand to determine winner

Procedural programming: focus on the algorithm, i.e., the procedures, necessary for solving a problem

- Identify "objects" in the game and define each:
  - Card
    - Properties: suit, rank
    - Actions: compare, show
  - Deck
    - Property: array of Cards
    - Actions: shuffle, deal, get #cards left
  - Hand …
  - Player …
- T
  al
  th

Object-oriented programming: focus on the design of the objects (data + actions) necessary for solving a problem

Notice the two steps involved in OOP?

- Define the classes (of the objects)
  - Identify the properties (data) and actions (methods, i.e., functions) of each class

- Create the objects (from the classes) that are then used—that interact with one another

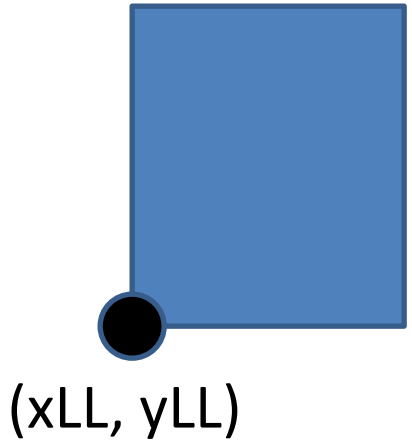
Defining a class ≠ creating an object

- A class is a specification/template
  - E.g., a cookie cutter specifies the shape of a cookie

- An object is a concrete instance of the class
  - Need to apply the cookie cutter to get a cookie (an instance, the object)
  - Many instances (cookies) can be made using the class (cookie cutter)
  - Instances do not interfere with one another. E.g., biting the head off one cookie doesn't remove the heads of the other cookies

Example class: Rectangle

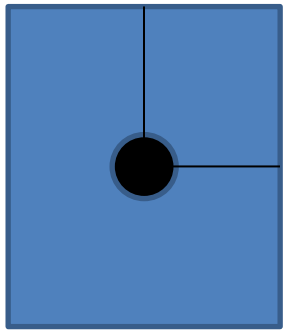
- Properties:
  - xLL, yLL, width, height
- Methods (actions):
  - Calculate area
  - Calculate perimeter
  - Draw
  - Intersect (the intersection between two rectangles is a rectangle!)



Poll: properties & methods

What if rectangles stored the following properties instead:

- xCenter, yCenter, halfWidth, halfHeight



Can they still provide these methods?

- Calculate area
- Calculate perimeter
- Draw
- Intersect

A: yes

B: no

Example class: TimeOfDay

- Properties:
  - Hour, minute, second
- Methods (actions):
  - Show (e.g., display in hh:mm:ss format)
  - Advance (e.g., advance current time by some amount)

Matlab supports procedural and object-oriented programming

- We have been writing procedural programs—focusing on the algorithm, implemented as a set of functions
- We have used objects in Matlab as well, e.g., graphics
- A plot is a "*handle graphics*" object
  - Can produce plots without knowing about objects
  - Knowing about objects gives more possibilities

Objects of the same class have the same properties

```
x= 1:10;
% Two separate graphics objects:
plot(x, sin(x), 'k-')
plot(x(1:5), 2.^x(1:5), 'm-*')
```

- Both objects have some x-data, some y-data, some line style, and some marker style. These are the properties of one kind, or class, of the objects (plots)
- The values of the properties are different for the individual objects

Optional reading: Script **`demoPlotObj.m`** shows some properties of graphics objects. Can also see MATLAB documentation for further detail.

Object-Oriented Programming

- First design and define the classes (of the objects)
  - Identify the properties (data) and actions (methods, i.e., functions) of each class

- Then create the objects (from the classes) that are then used, that interact with one another

Class Interval

- An interval has two properties:
  - left, right
- Actions—methods—of an interval include
  - Scale, i.e., expand
  - Shift
  - Check if one interval is in another
  - Check if one interval overlaps with another

See `demoInterval0.m`

Class Interval

- An interval has two properties:
  - left, right
- Actions—methods—of an interval include
  - Scale, i.e., expand
  - Shift
  - Check if one interval is in another
  - Check if one interval overlaps with another

```
classdef Interval < handle

  properties
    left
    right
  end

  methods
    function scaleRIght(self, f)
      …
    end

    function shift(self, s)
      …
    end

    function Inter = overlap(self, other)
      …
    end

    …

  end

end
```

To specify the properties and actions of an object is to define its class. This files is Interval.m

These methods (functions) are inside the classdef

Given class Interval (file Interval.m) …

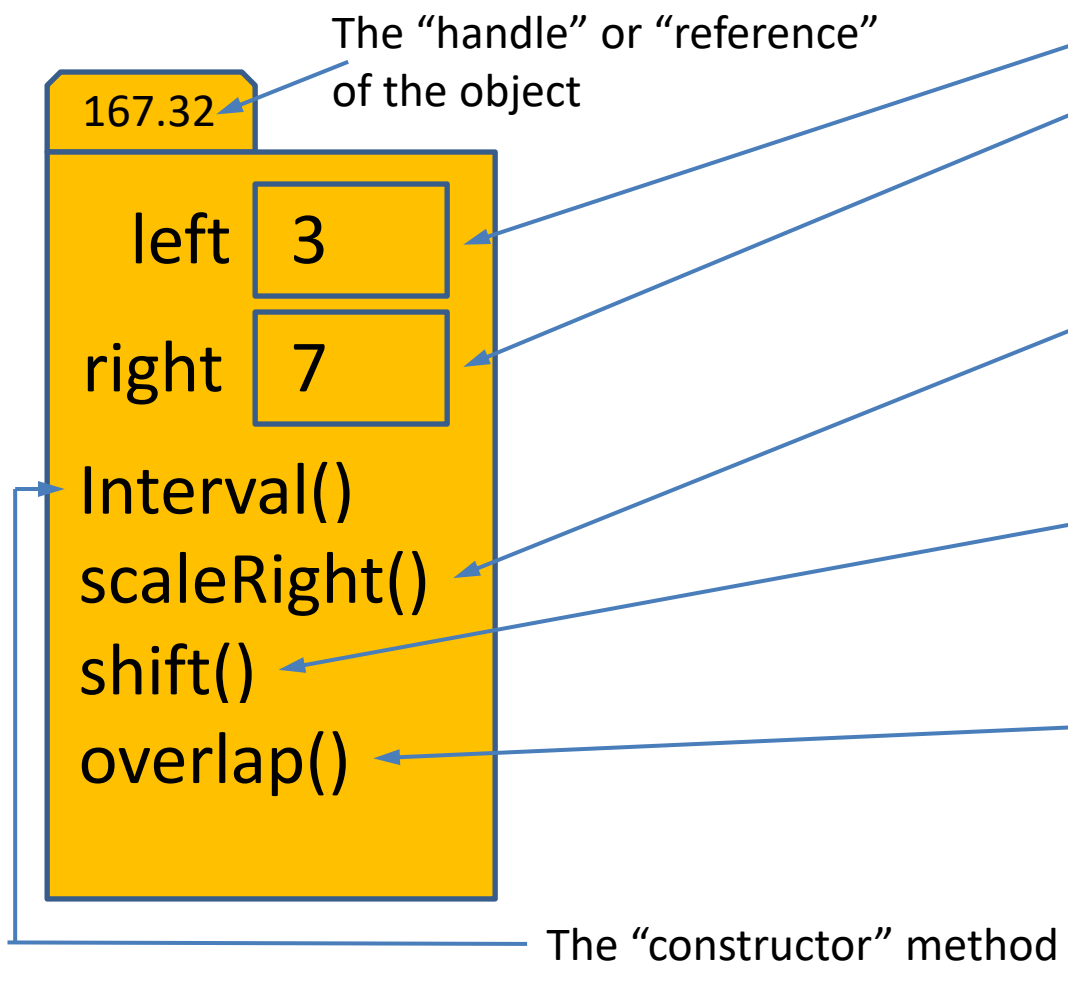
```
% Create 2 Intervals, call them A, B
A= Interval(2,4.5)
B= Interval(-3,1)

% Assignment another right end point
A.right= 14

% Half the width of A (scale by 0.5)
A.scaleRight(.5)

% See the result
disp(A.right) % show value in right property in A
disp(A)       % show all property values in A
disp(B)
```

Observations:

- Each object is referenced by a name.
- Two objects of same class has the same properties (and methods).
- To access a property value, you have to specify whose property (which object's property) using the dot notation.
- Changing the property values of one object doesn't affect the property values of another object.

See `demoInterval0.m`

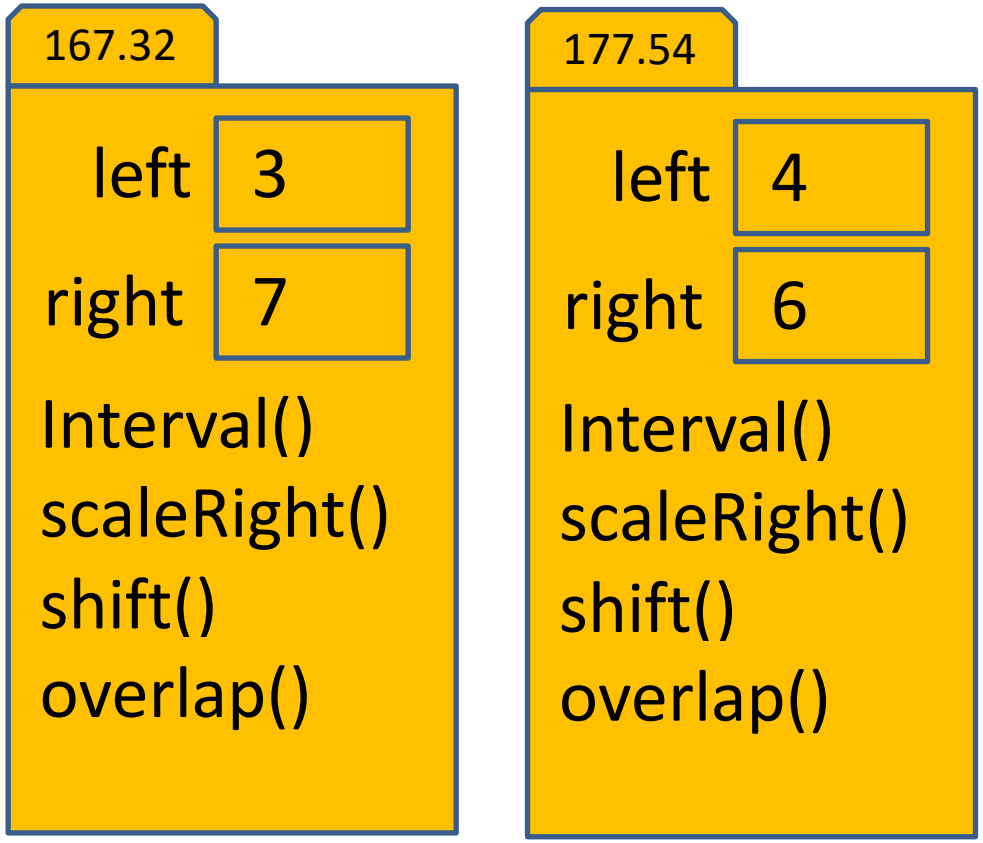
An Interval object




```
classdef Interval < handle

  properties
    left
    right
  end

  methods
    function scaleRIght(self, f)
      ...
    end

    function shift(self, s)
      ...
    end

    function Inter = overlap(self, other)
      ...
    end

    ...

  end

end
```

An object is also called an "instance" of a class. It contains every property, "instance variable," and every "instance method" defined in the class.

Multiple Interval objects

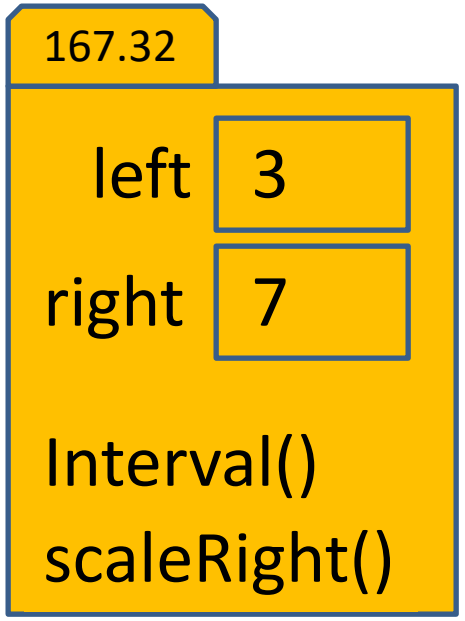



```
classdef Interval < handle

  properties
    left
    right
  end

  methods
    function scaleRight(self, f)
      …
    end

    function shift(self, s)
      …
    end

    function Inter = overlap(self, other)
      …
    end

    …

  end

end
```

Every object (instance) contains every "instance variable" and every "instance method" defined in the class. Every object has a unique handle.

Simplified Interval class

To create an Interval object, use its class name as a function call: p = Interval(3,7)



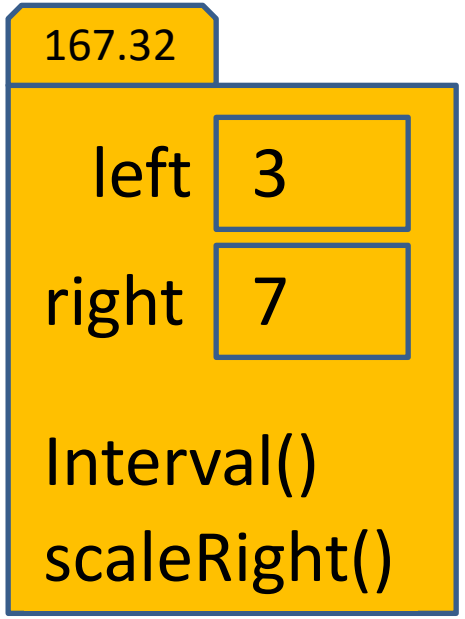

```
classdef  Interval < handle
% An Interval has a left end and a right end

   properties
     left
     right
   end

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

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

The *constructor* method

To create an Interval object, use its class name as a function call: p = Interval(3,7)

```
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
```

Stores the handle of the object being created



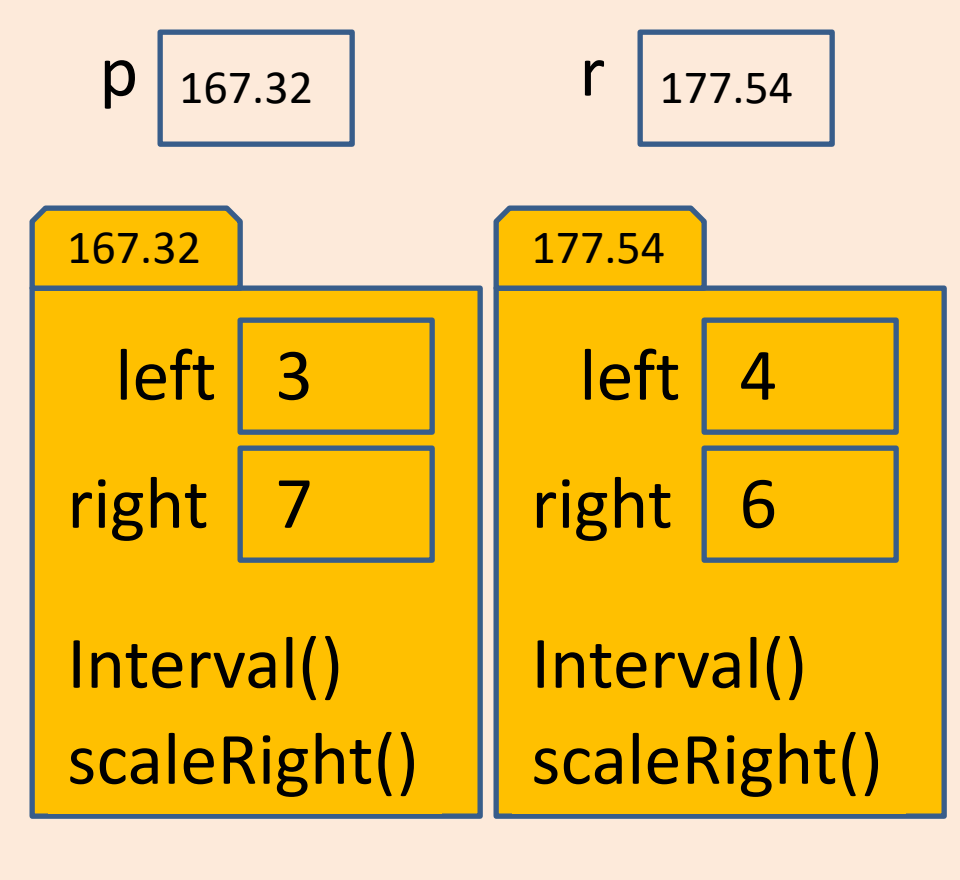
Constructor, a special method with these jobs:

- Automatically compute the handle of the new object; the handle must be returned.
- Execute the function code (to assign values to properties)

Constructor is the only method that has the name of the class.

A handle object is <u>referenced</u> by its handle

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

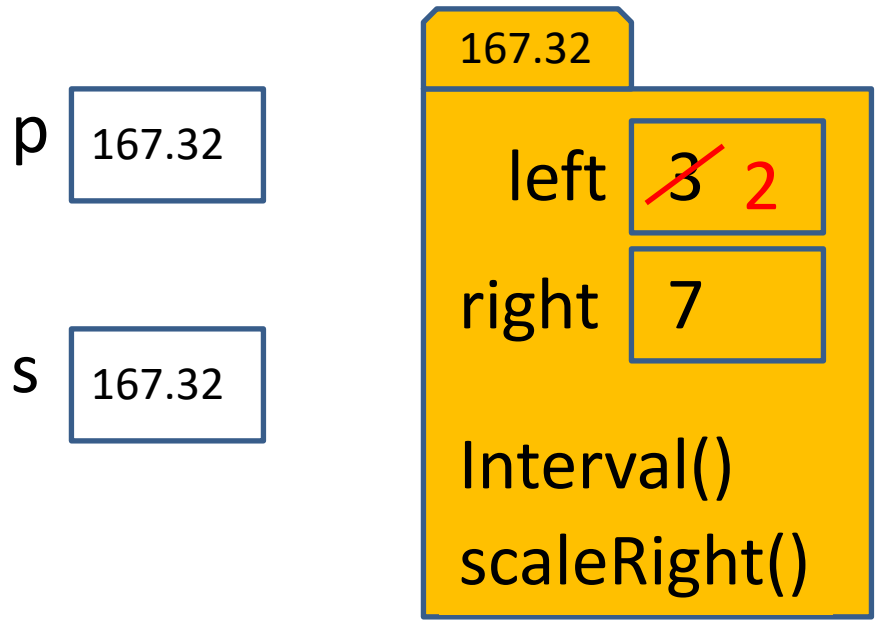


A handle, also called a reference, is like an address; it indicates the memory location where the object is stored.

What's the effect of storing data "by reference"?

What is the effect of referencing?

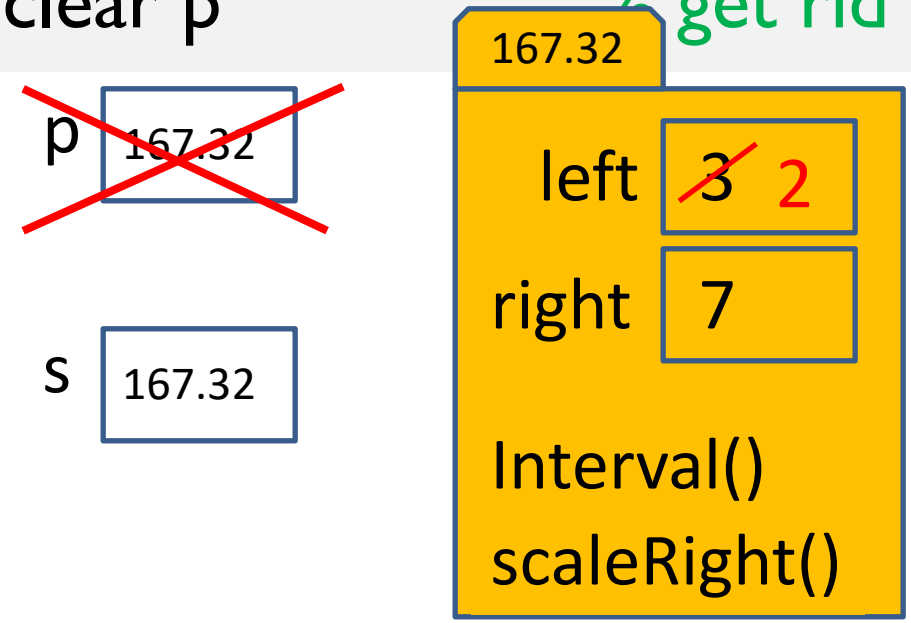
```
p = Interval(3,7);   % p references an Interval object
s = p;               % s stores the same reference as p
s.left = 2;          % change value inside object
disp(p.left)         % 2 is displayed
```



The object is not copied—no new object is created! s and p both reference the same object; s is an **alias** of p.

What is the effect of referencing?

```
p = Interval(3,7);   % p references an Interval object
s = p;               % s stores the same reference as p
s.left = 2;          % change value inside object
disp(p.left)         % 2 is displayed
clear p              % get rid of p from memory
```



The object still can be accessed through s.

In contrast, arrays are stored by value …

```
p= [3, 7];   % A vector with two elements
s= p;        % s gets a copy of p--s is ANOTHER
             %   vector with same element values
s(1)= 2;     % Changes s's copy only, not p's
disp(p(1))   % What is displayed?
```

A: 2

B: 3

B: Something else

Draw the memory!

p: [3 7]

s: