- Previous lecture:
  - Introduction to objects and classes
- Today's lecture:
  - Defining a class
    - Properties
    - Constructor and other methods
  - Objects are passed by reference to functions
- Announcements:
  - 2nd optional review session: W 5-6:30pm in Phillips 101
  - Prelim 2: Thurs at 7:30pm

Object-Oriented Programming

- First design and define the classes (of the objects)
  - Define the properties (data) and actions (methods, i.e., functions) of each class in a "class definition file"

- 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
  - Add one interval to another
  - Check if one interval is in another
  - Check if one interval overlaps with another

Class Interval

- An interval has two prope
  - left, right
- Actions—methods—of an
  - Scale, i.e., expand
  - Shift
  - Add one interval to anothe
  - Check if one interval is in a
  - Check if one interval overla

To specify the properties and actions of an object is to define its class

```
classdef Interval < handle

  properties
    left
    right
  end

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

    function shift(self, s)
      ...
    end

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

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

    ...

  end

end
```

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)

% Assign another right end point to Interval A
A.right= 14

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

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

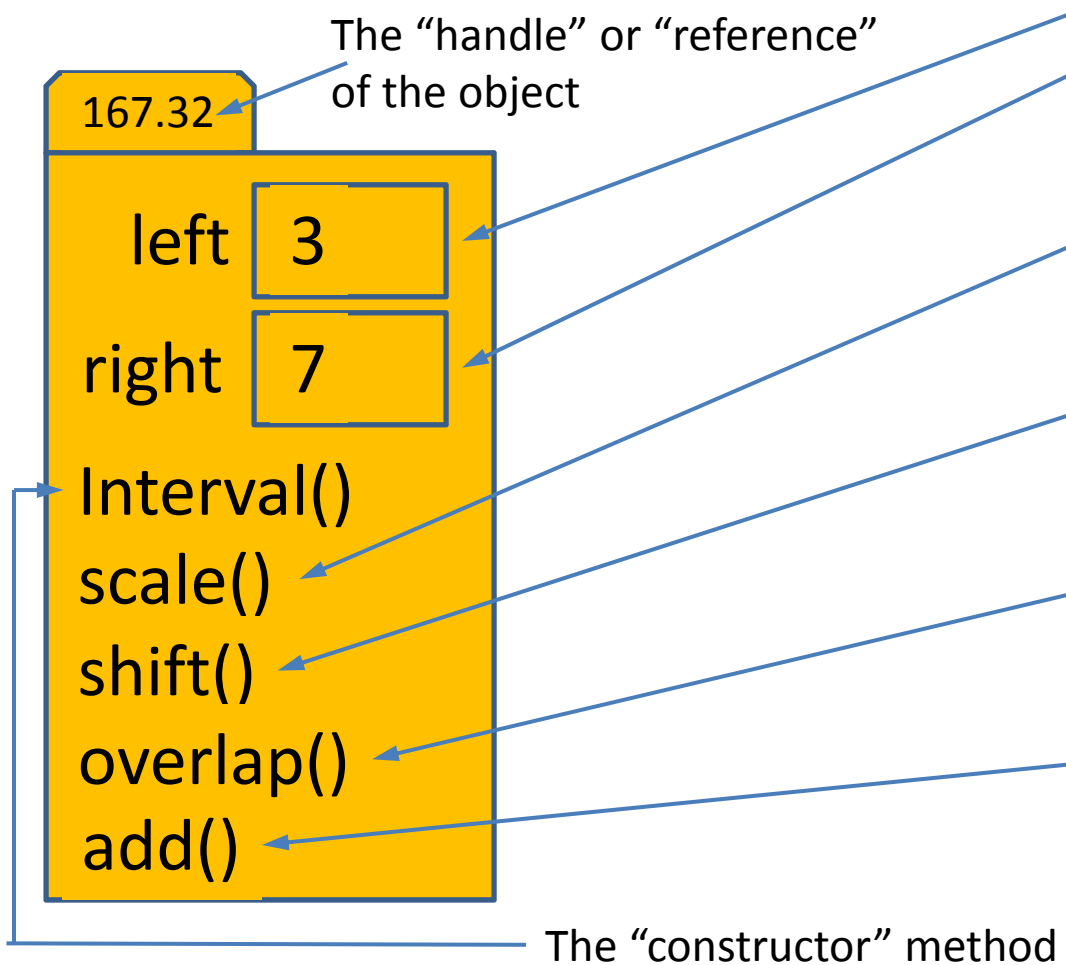
Given class Interval (file Interval.m) …

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

% Assign another right end point to Interval A
A.right= 14

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

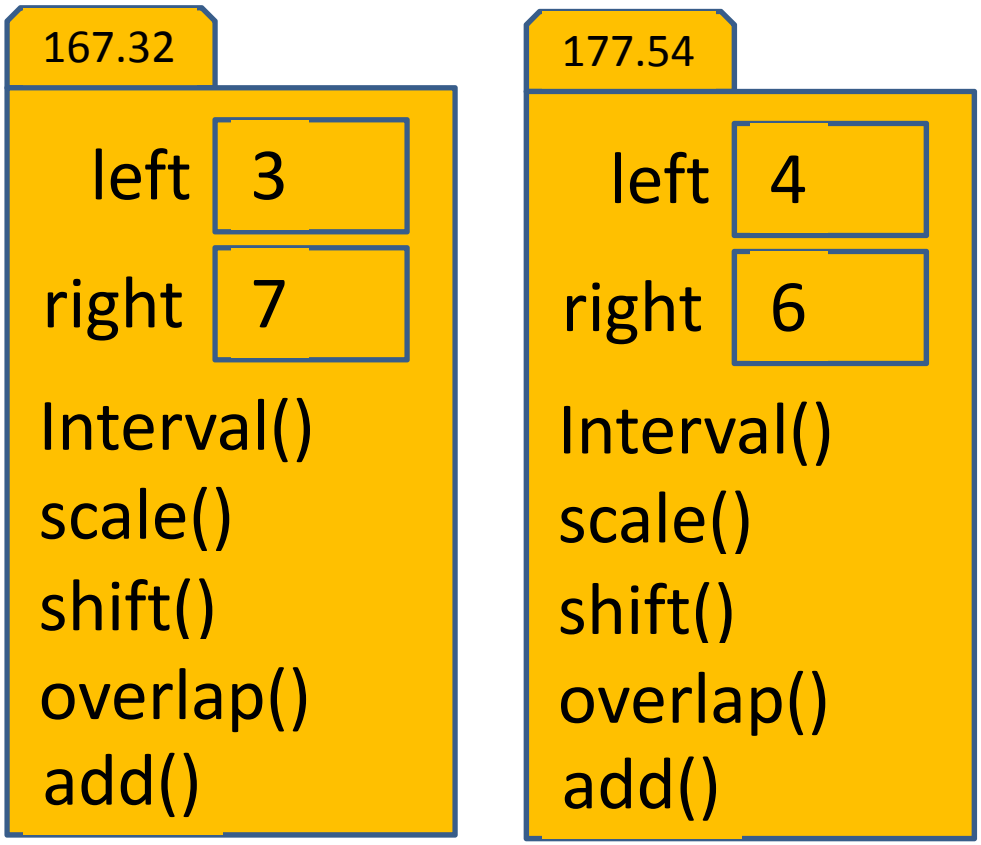
An Interval object




```
classdef Interval < handle

  properties
    left
    right
  end

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

    function shift(self, s)
      ...
    end

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

    function Inter = add(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

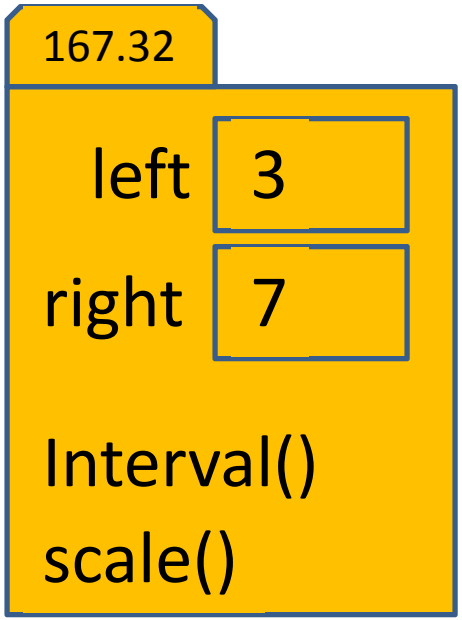


Every object (instance) contains every "instance variable" and every "instance method" defined in the class. Every object has its own handle.

```
classdef Interval < handle

  properties
    left
    right
  end

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

    function shift(self, s)
      ...
    end

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

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

    ...

  end

end
```

Simplified Interval class

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

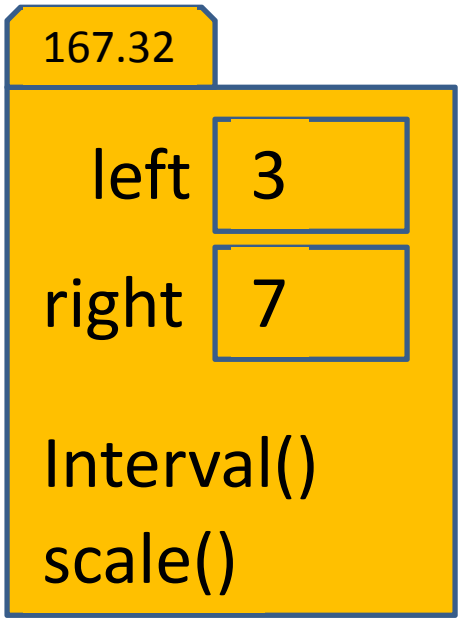



```
classdef  Interval < handle
% An Interval has a left end and a right end

   properties
     left
     right
   end

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

     function scale(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
```



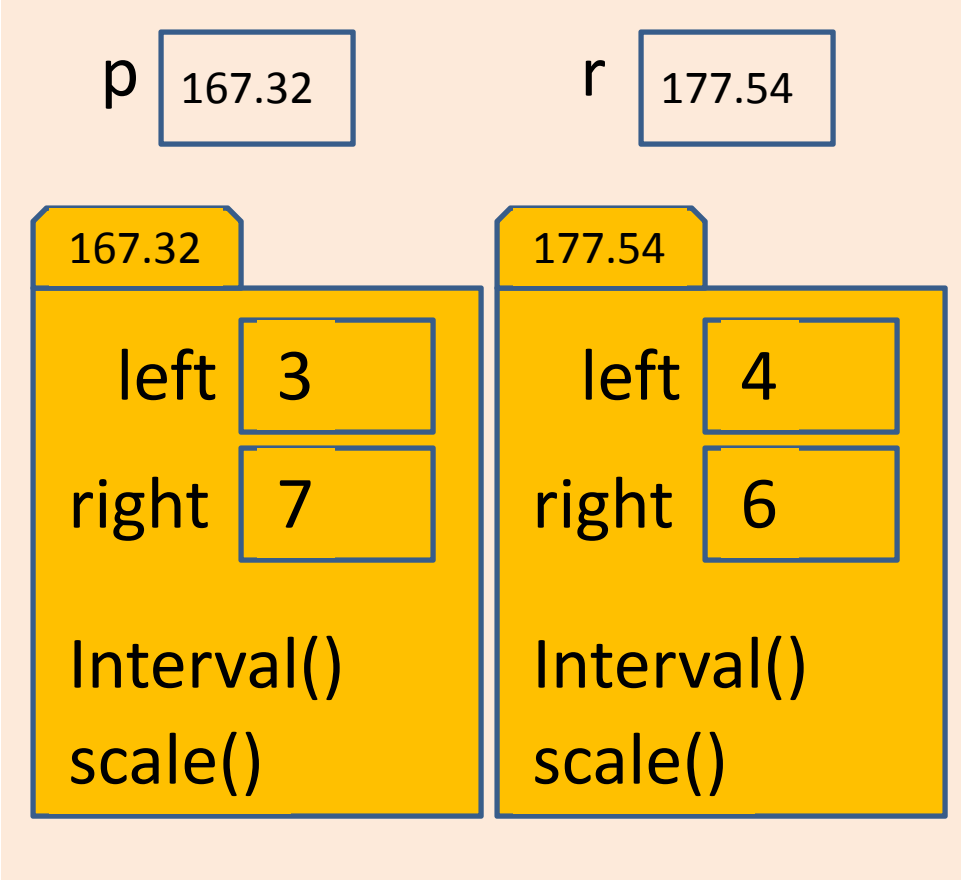
Constructor, a special method with these jobs:

- compute the handle of the new object,
- execute the function code (to assign values to properties), and
- return the handle of the object.

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

A handle object is referenced by its handle

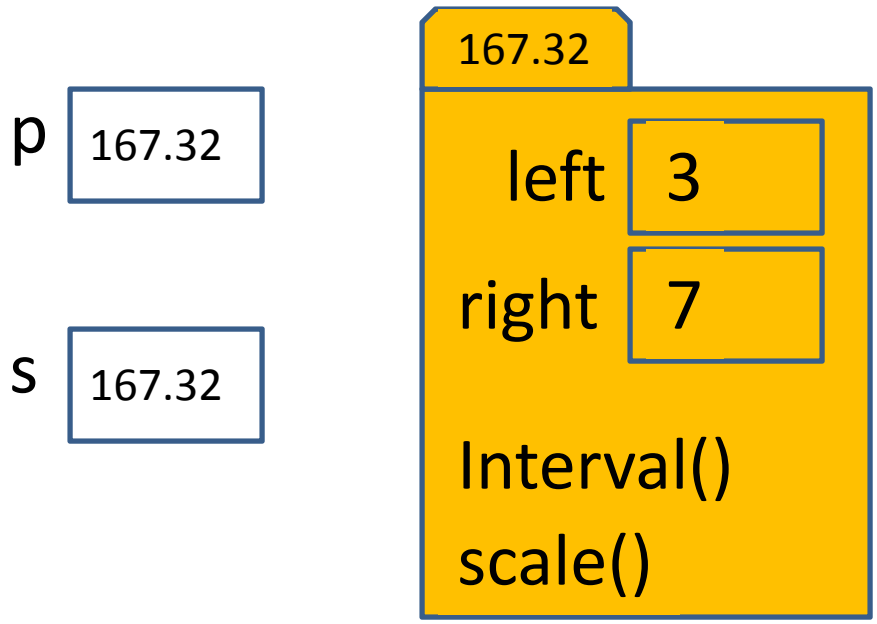
```
p = Interval(3,7);
r = Interval(4,6);
```



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

What is the effect of referencing?

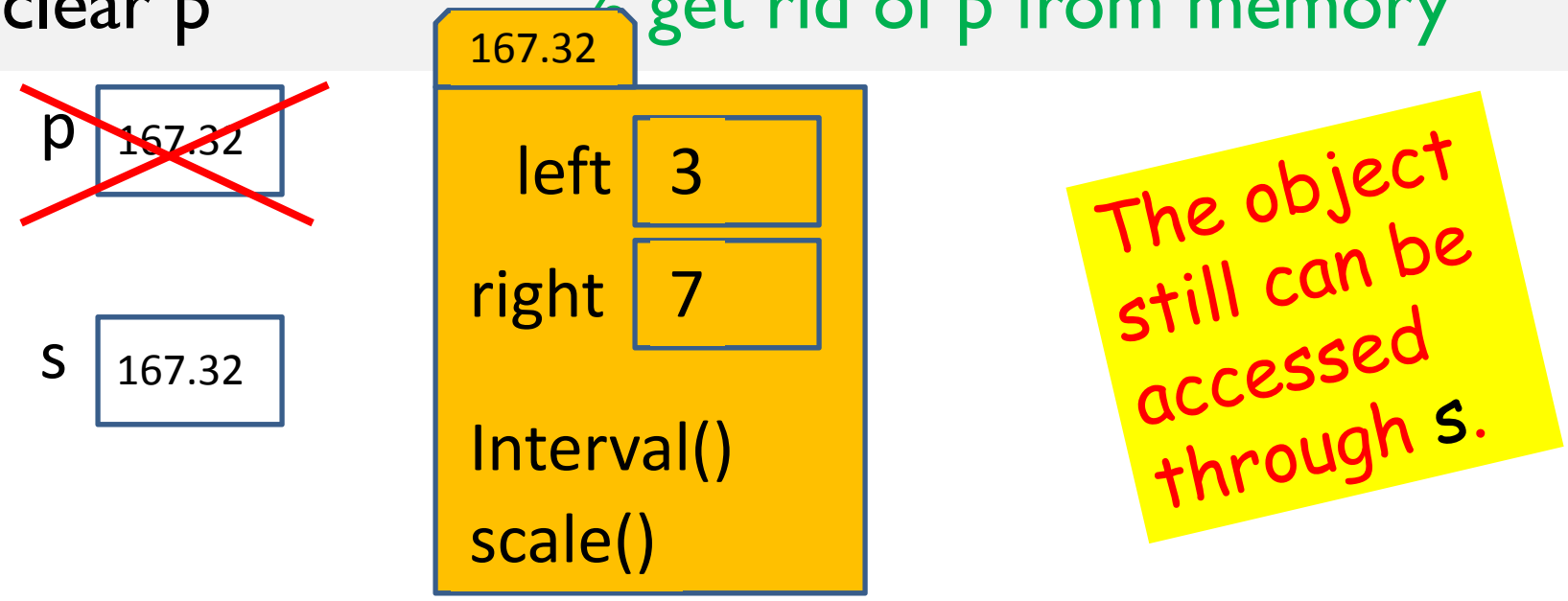
```
p = Interval(3,7);   % p references an Interval object
s = p;               % s stores the same reference as p
s.left = 2;          % change value inside object
disp(p.left)         % 2 is displayed
```



The object is not copied—no new object is created!
s and p both reference the same object.

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
```



In contrast, structs are stored by value …

```
P.x=5; P.y=0;   % A point struct P
Q=P;            % Q gets a copy of P--Q is ANOTHER
                %   struct with same field values
Q.y=9;          % Changes Q's copy only, not P's
disp(P.y)       % What is displayed?
```

A: 0

B: 9

B: Something else

In contrast, structs are stored by value …

```
P.x=5; P.y=0;  % A point struct P
Q=P;           % Q gets a copy of P--Q is ANOTHER
               %   struct with same field values
Q.y=9;         % Changes Q's copy only, not P's
disp(P.y)      % 0 is display
```

In fact, storing-by-value is true of all non-handle-object variables. You already know this from before …

```
a=5;
b=a+1;    % b stores the value 6, not
          %   the "definition" a+1
a=8;      % Changing a does not change b
disp(b)   % 6 is displayed
```

Syntax for calling an instance method

```
r = Interval(4,6);
r.scale(5)
```

Method name

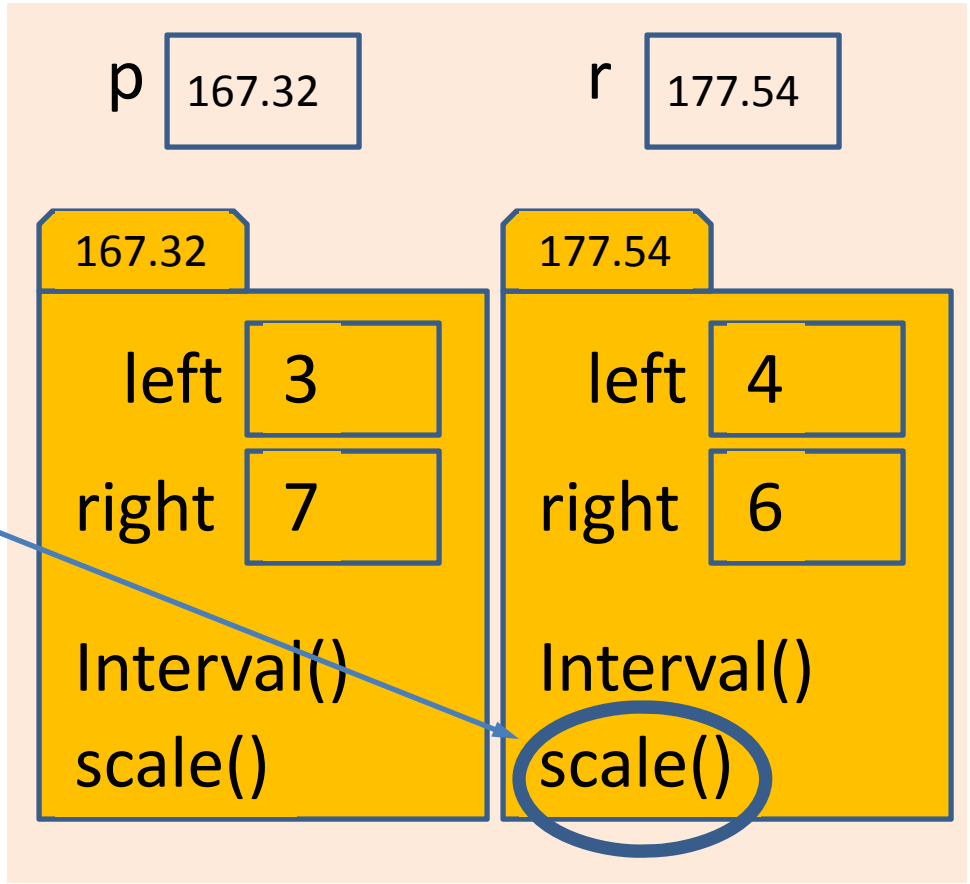
Reference of the object whose method is to be dispatched

Argument for the second parameter specified in function header (f). Argument for first parameter (self) is absent because it is the same as r, the owner of the method

```
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
```

Calling an object's method (instance method)

```
p = Interval(3,7);
r = Interval(4,6);
r.scale(5)
```



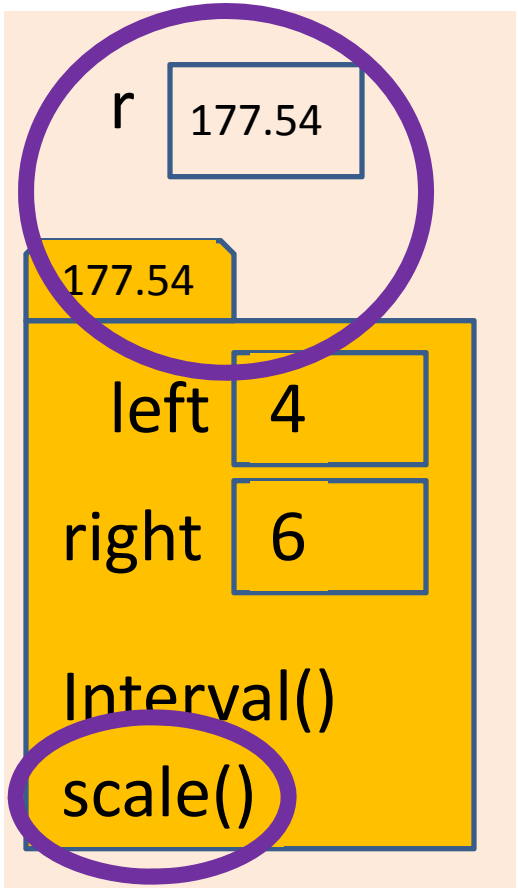
The owner of the method to be dispatched

Syntax:

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

Executing an instance method

```
r = Interval(4,6);
r.scale(5)
disp(r.right) %What will it be?
```

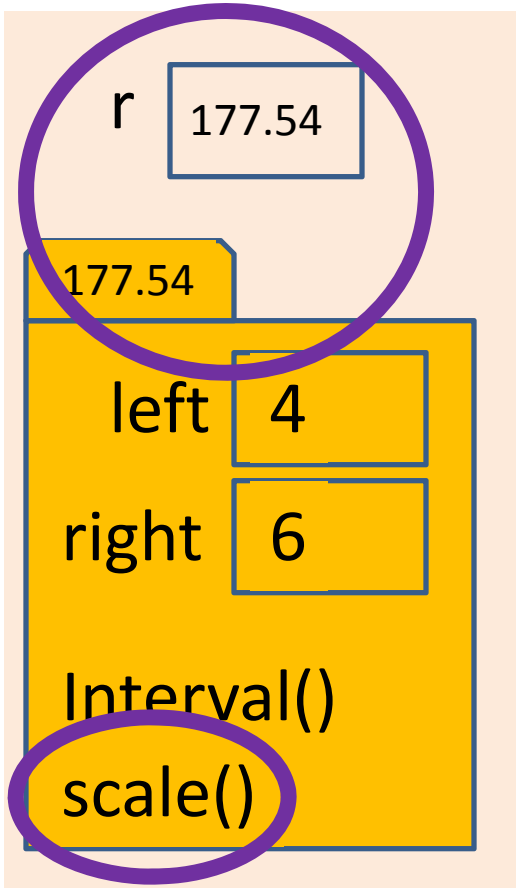


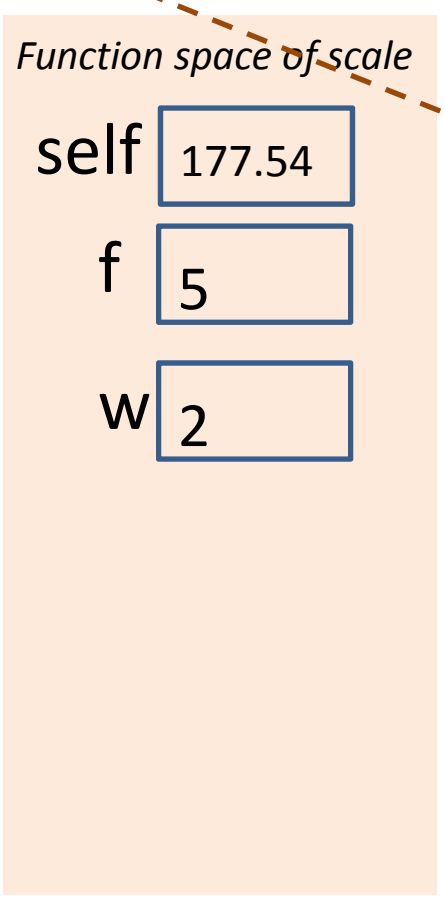
1st parameter (self) reference itself, i.e., its own handle. It ge what's in r

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

  properties
    left
    right
  end

  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
```

Executing an instance method

```
r = Interval(4,6);
r.scale(5)
disp(r.right) %What will it be?
```

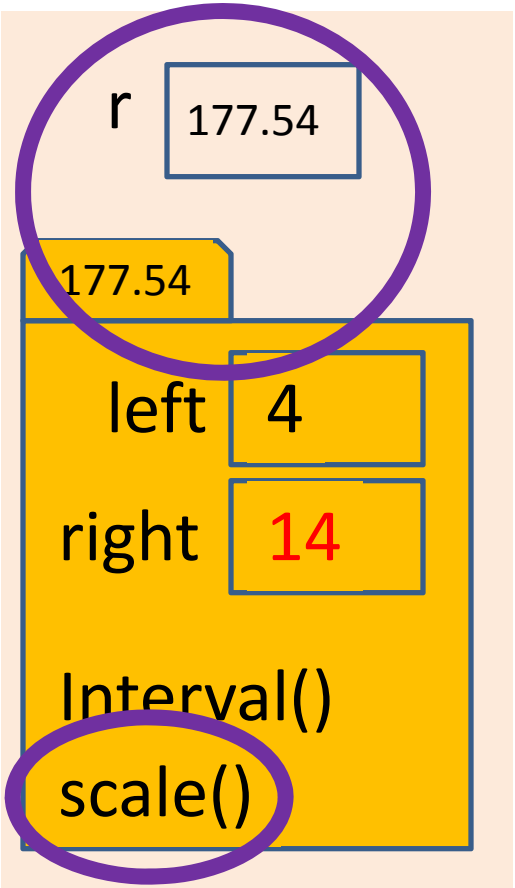


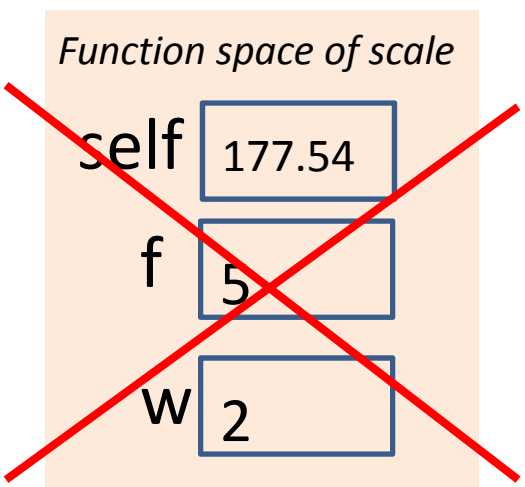



```
classdef  Interval < handle
% An Interval has a left end and a rig

   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
   end
end
```

Object is passed to a function by reference

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



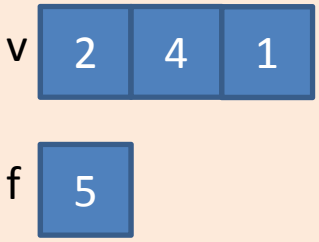



```
classdef  Interval < handle
% An Interval has a left end and a rig

   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

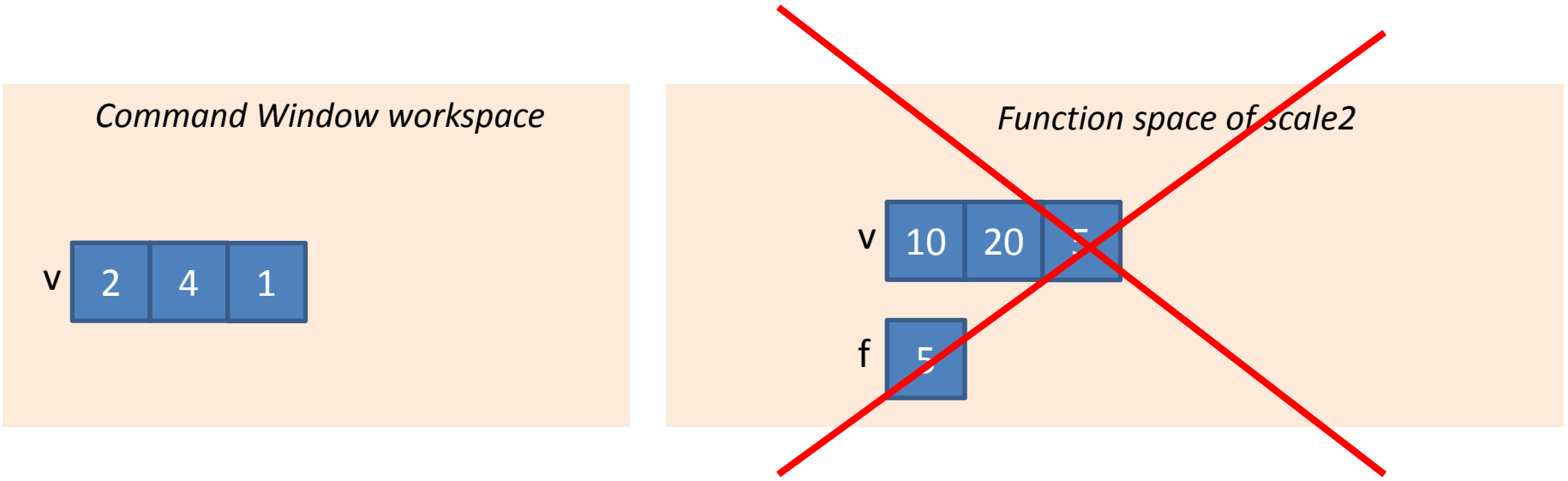
Function space of scale2




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

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

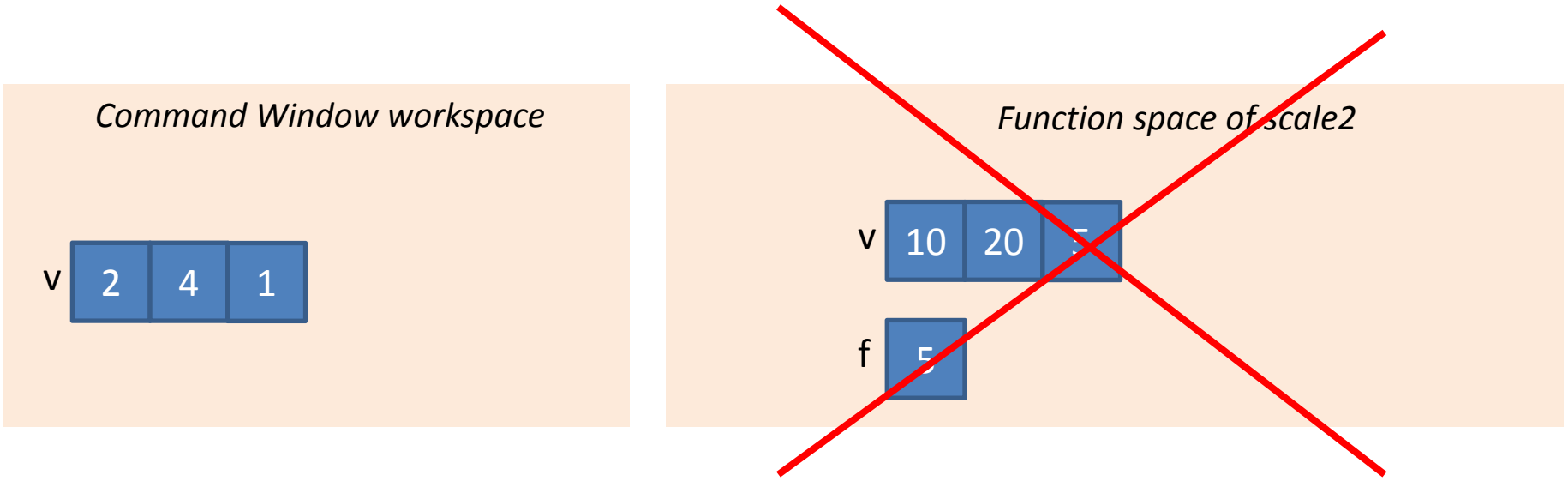
Non-objects are passed to a function by value




```
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

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