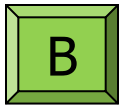
- Previous lecture:
  - Introduction to objects and classes
- Today's lecture:
  - Defining a class
    - Properties
    - Constructor and other methods
  - Objects are passed by reference to functions
- Announcements:
  - Test 2A due today 4:30pm EDT
    - Consulting hours, Piazza resume afterwards
  - Project 5 released, due next Thurs

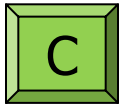
Quiz: Object-oriented vocabulary

Which of the following is incorrect?

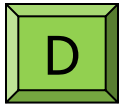
– **Methods** are functions that define a class's behavior



– Variables store **handles** to objects, so two different variables may reference the same object

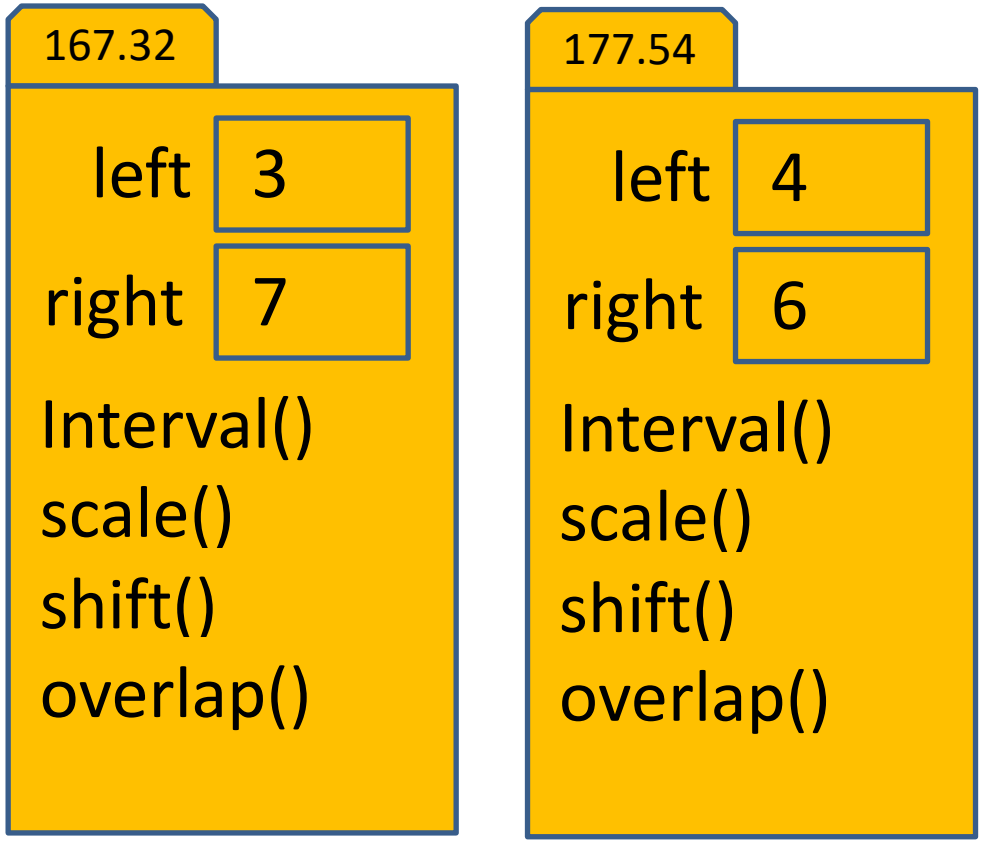


– **Classes** are instances of objects, each with their own copy of property values



– **Constructors** return handles to newly allocated objects

Multiple Interval objects



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

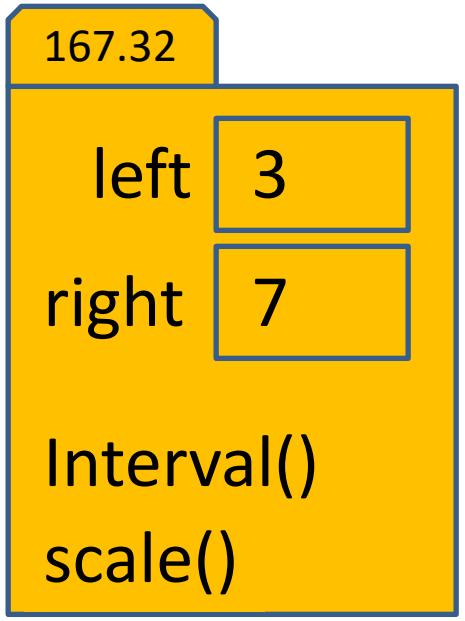
```
classdef Interval < handle

  properties
    left
    right
  end

  methods
    function scale(self, f)
      …
    end

    function shift(self, s)
      …
    end

    function Inter = overlap(self, other)
      …
    end

    …

  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



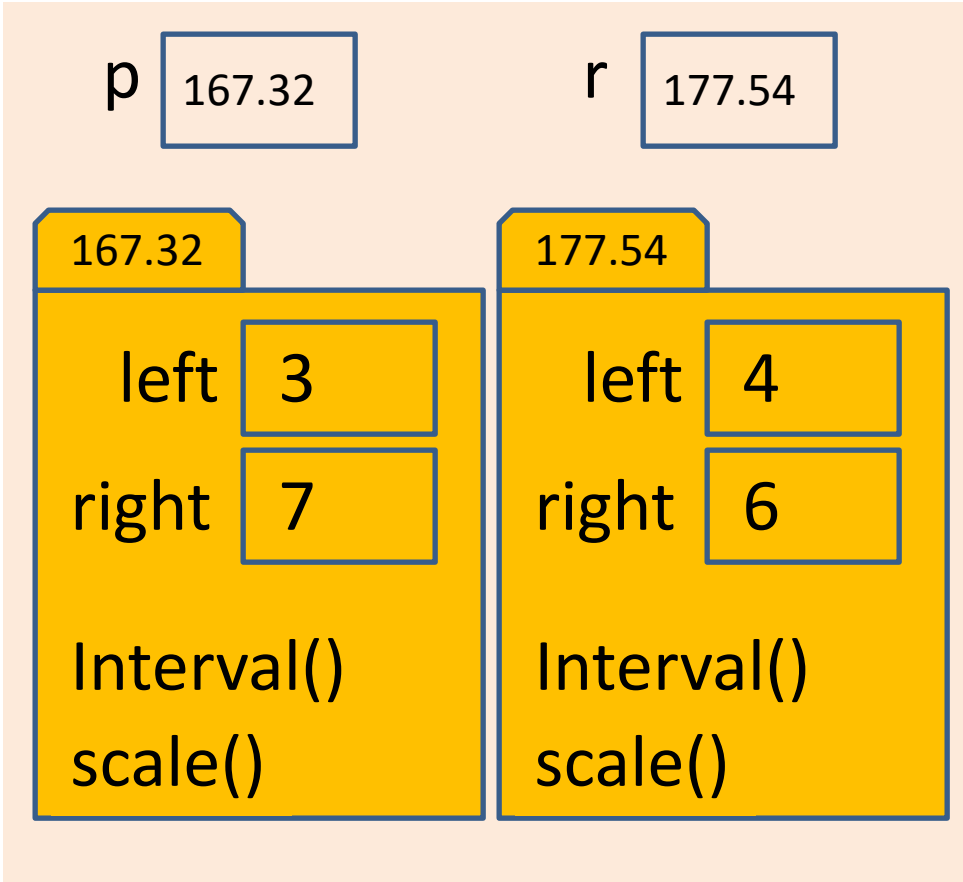
Constructor, a special method with these jobs:

- 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 referenced by its handle

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

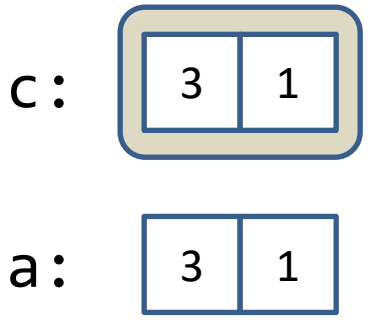


A handle, also called a reference, is like an address; it indicates the memory location where the object is stored.

Value vs. reference

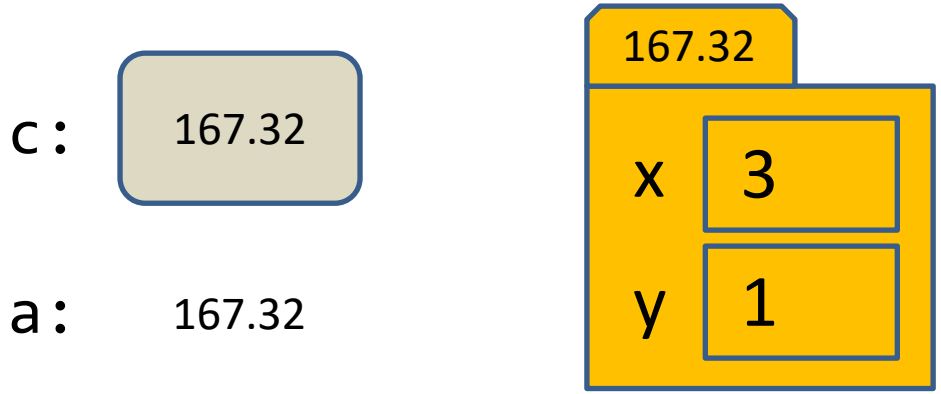
Arrays

```
c= { [3, 1] };
a= c{1};
a(2)= 4;
disp(c{1}(2))
```



Object handles

```
c= { Pair(3, 1) };
a= c{1};
a.y= 4;
disp(c{1}.y)
```




```
classdef Pair < handle
   properties
      x
      y
   end
end
```

Syntax for calling an instance method

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

Method name

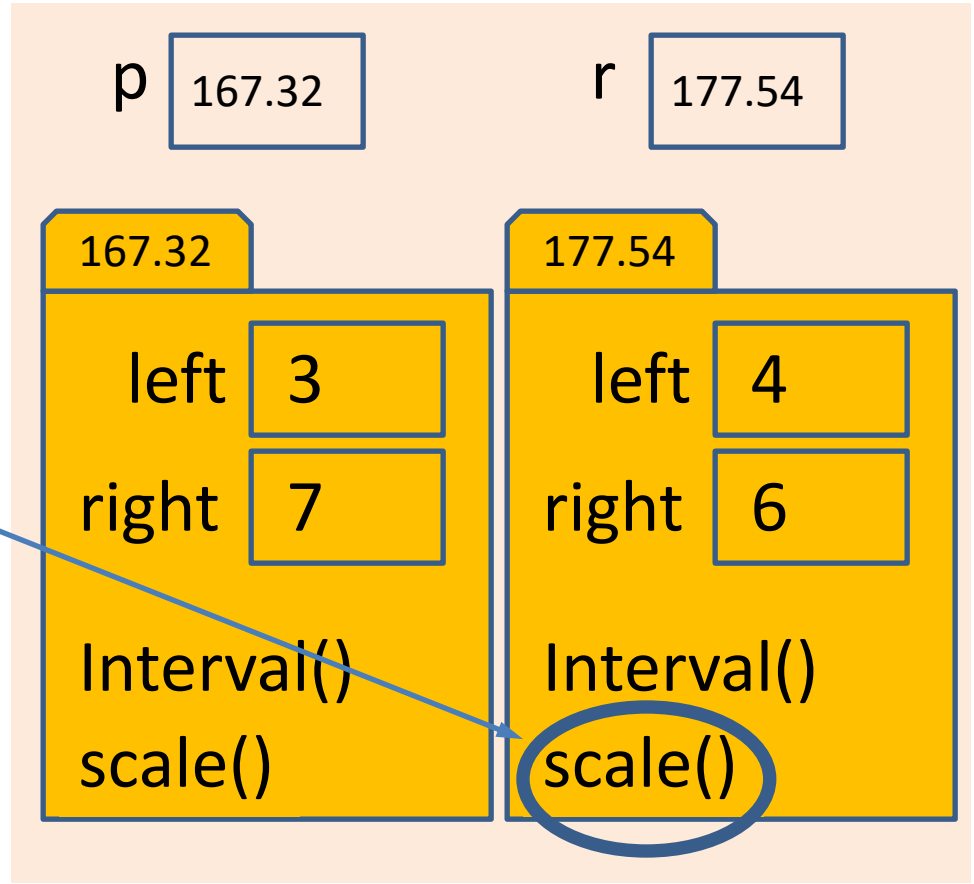
Reference of the object whose method is to be dispatched

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)
```



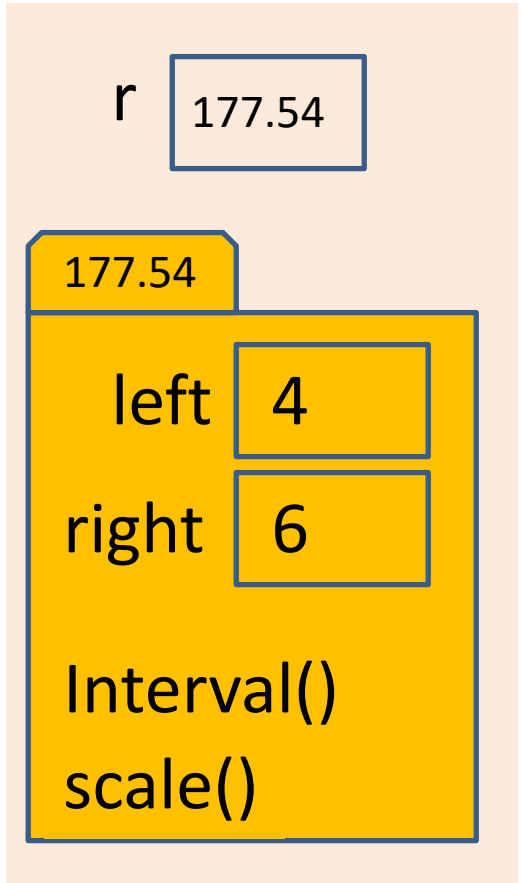
The owner of the method to be dispatched

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?
```



A: 5

B: 6

C: 14

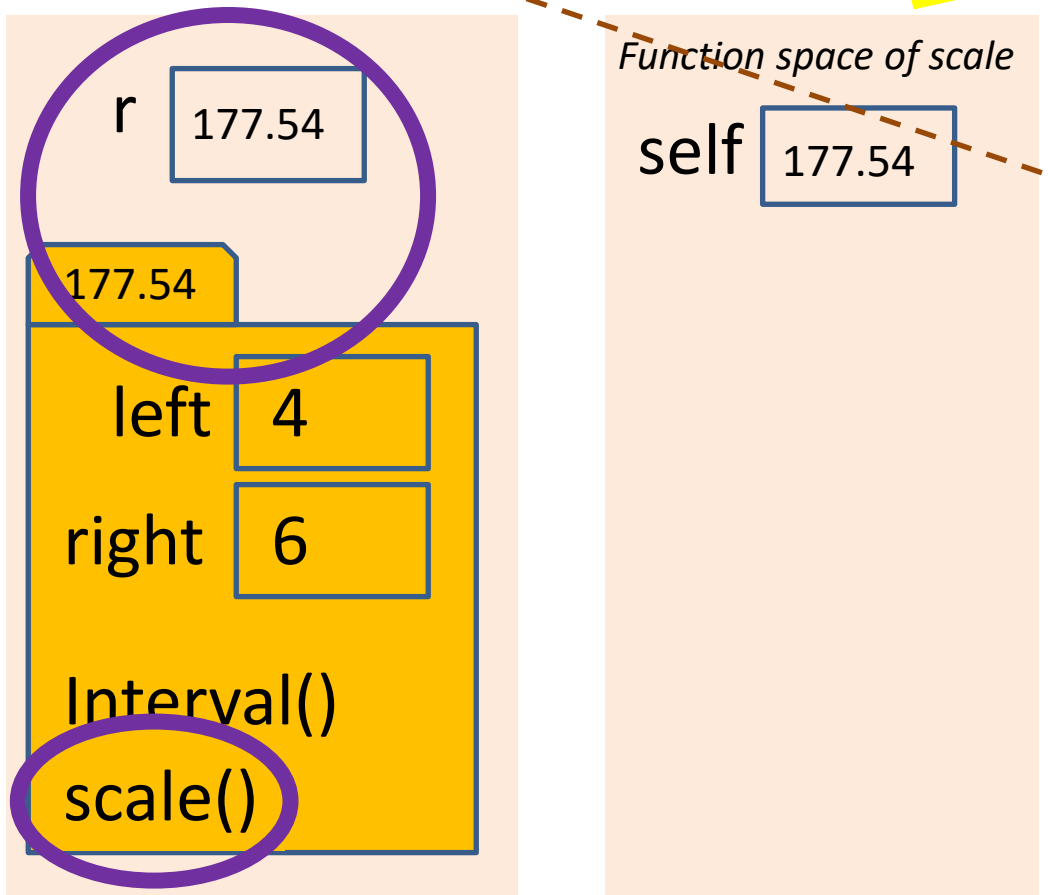
D: 30

```
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
```

Executing an instance method

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

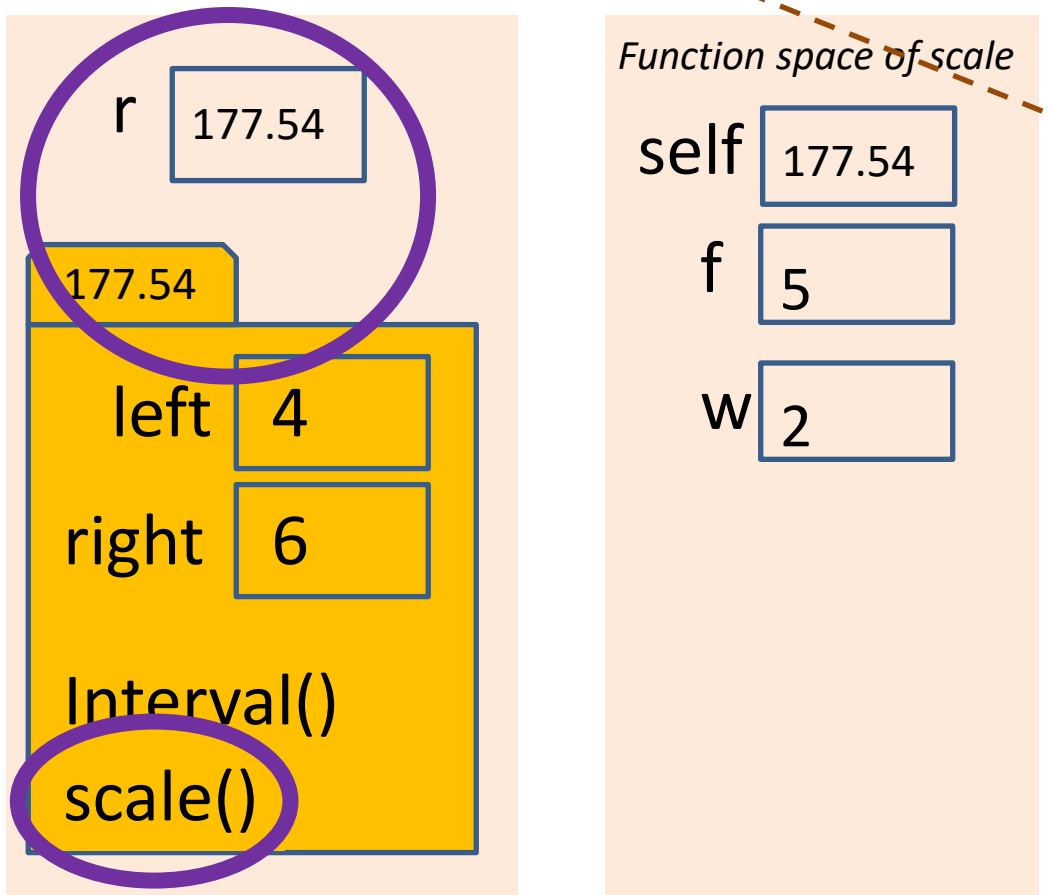
1st parameter (self) references itself, i.e., its own handle. It gets what's in r




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

Executing an instance method

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

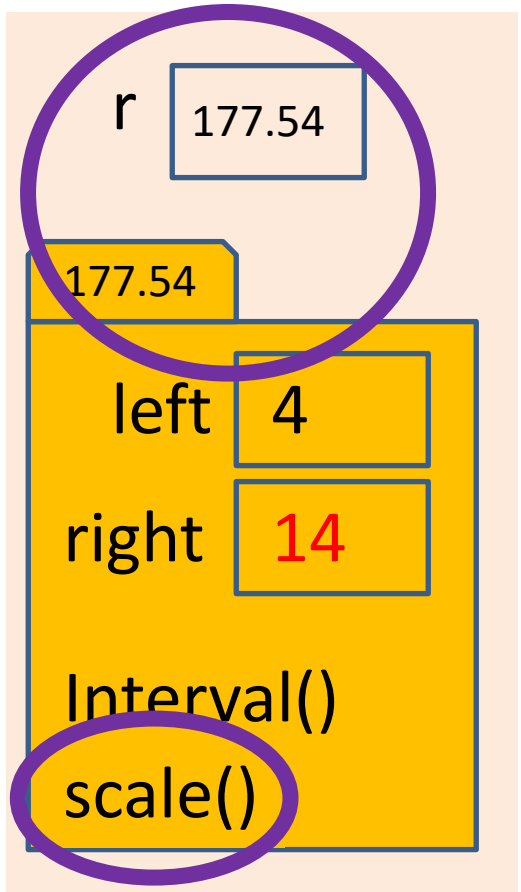


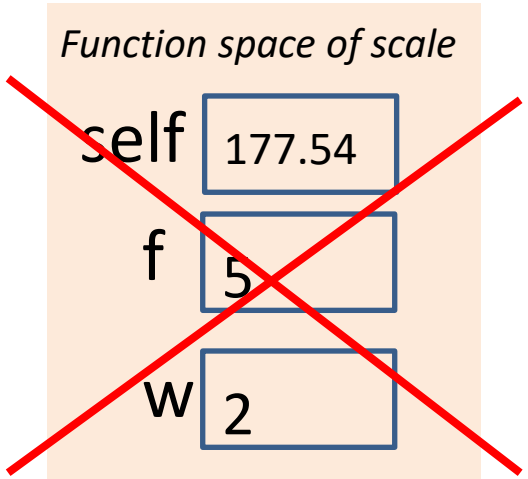
Function space of scale


```
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
```

Object is passed to a function by reference

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



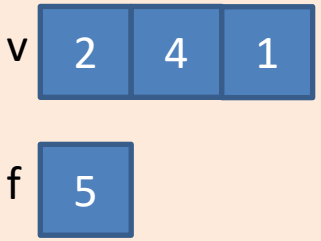



```
classdef  Interval < handle
% An Interval has a left end and a right end

   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
```

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

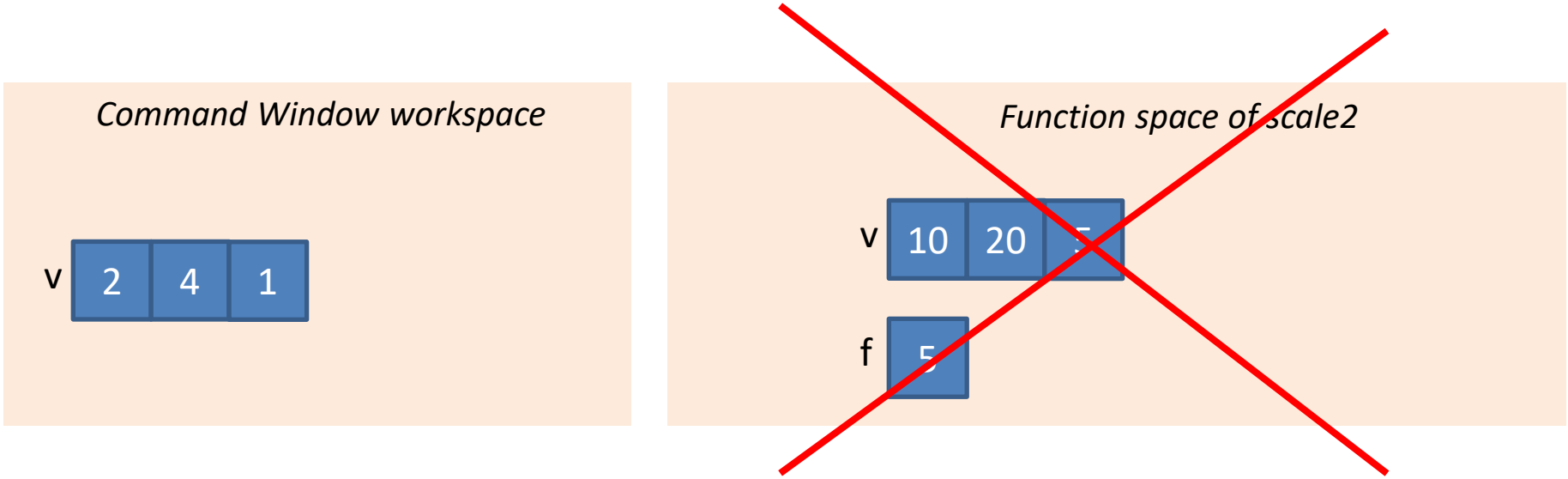
Function space of scale2




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

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

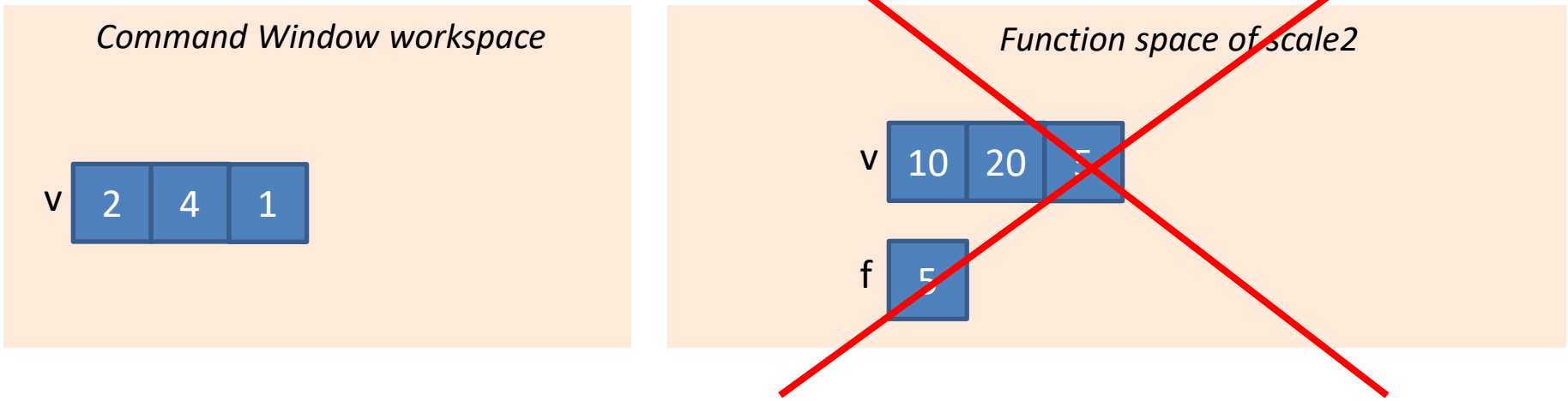
Non-objects are passed to a function by value




```
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);
```

Better!

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

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

```
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
% Inter is empty array of class Interval.
```

Compare two intervals

1

2

redRight < blueRight

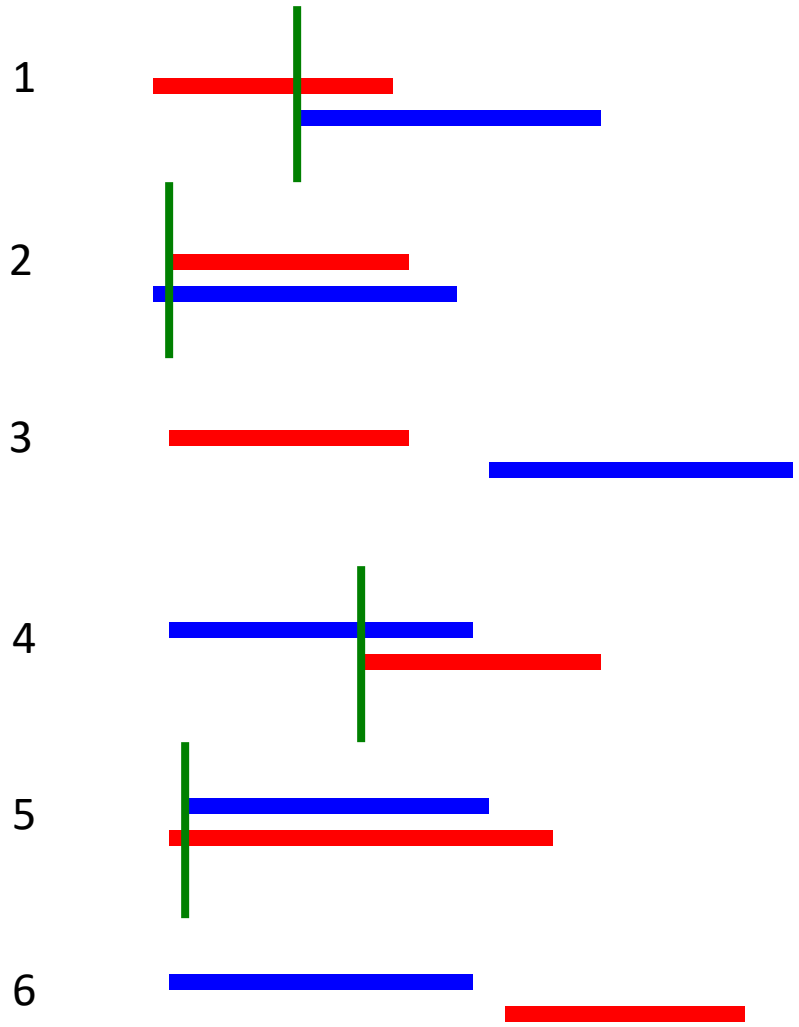
3

4

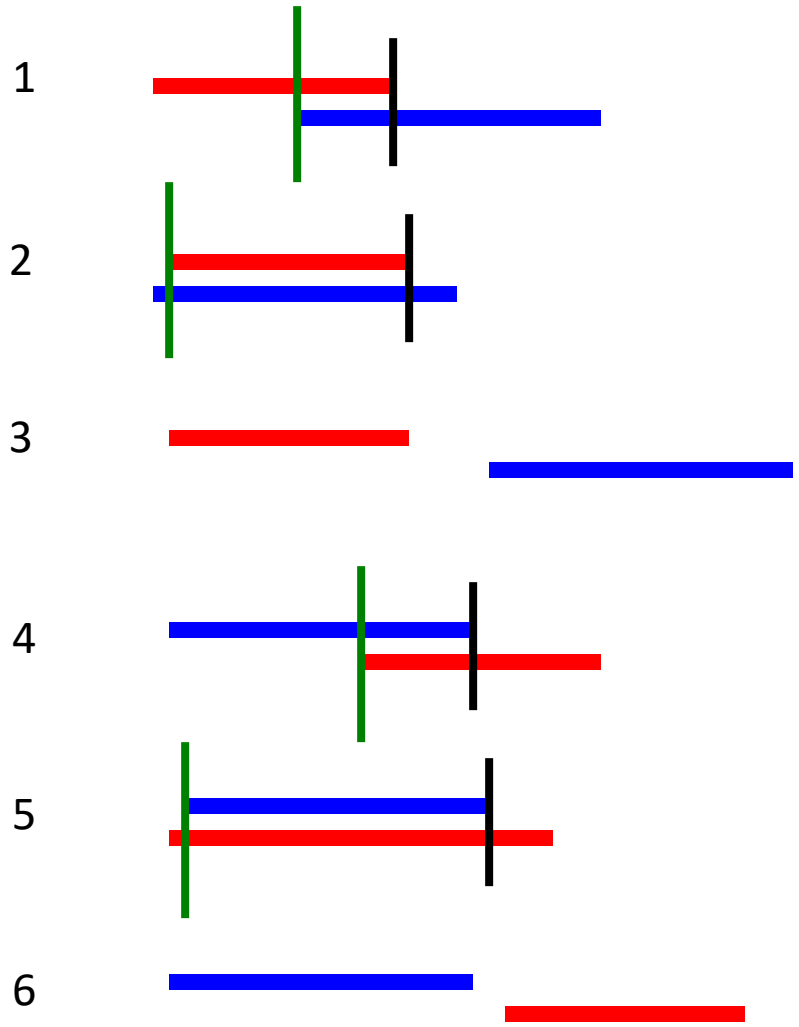
5

blueRight < redRight

6

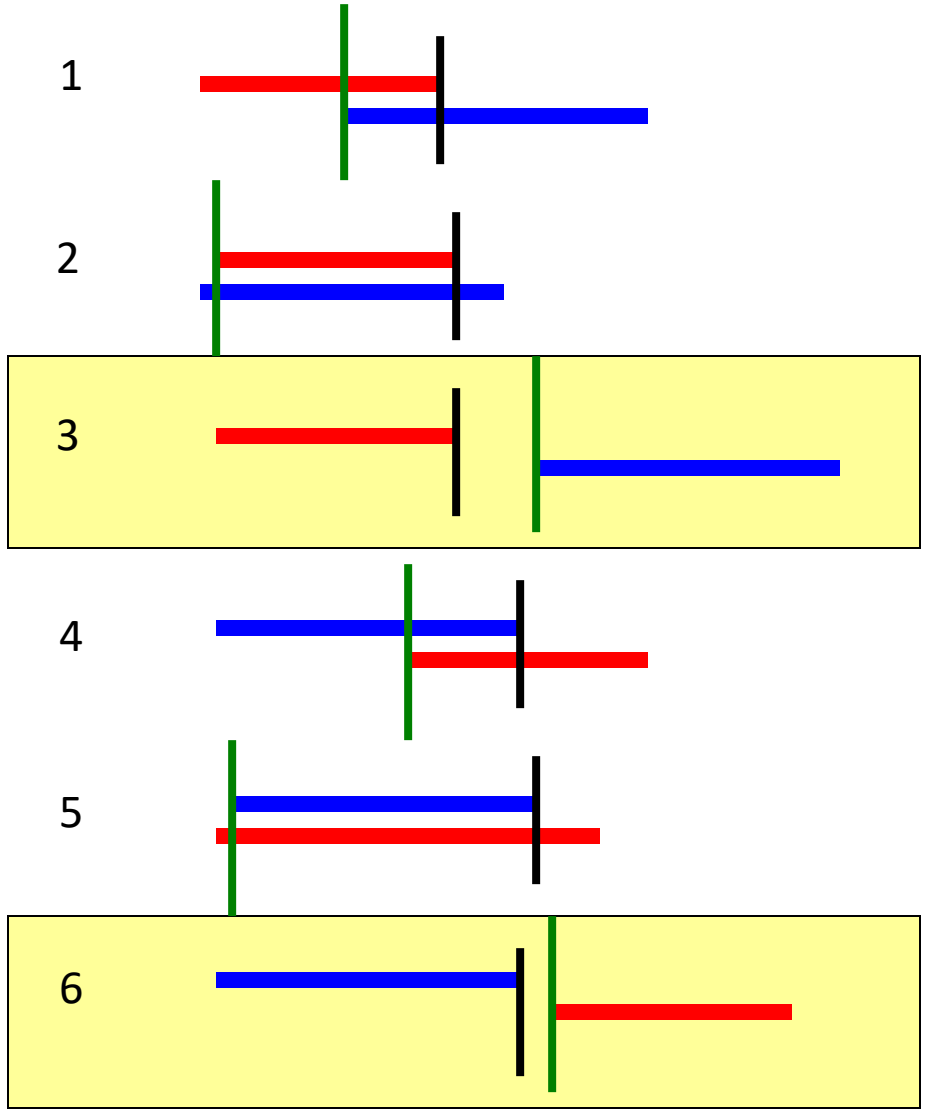


The overlap's left (OLeft) is the rightmost of the two original lefts



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

Implement overlap method

DEMO

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

  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
```

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

Overriding built-in functions

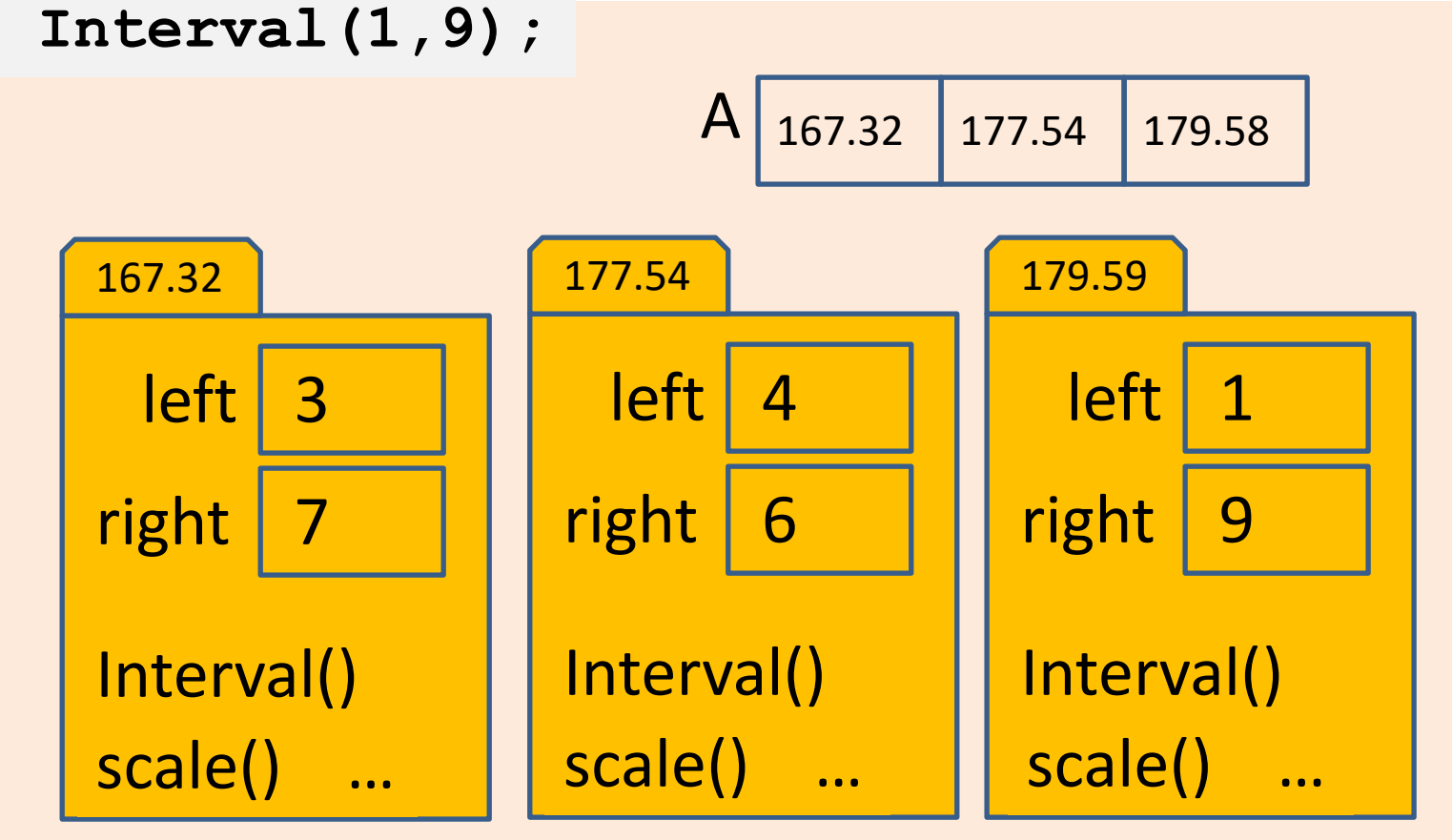
- You can change the behavior of a built-in function for an object of a class by implementing a function of the same name in the class definition
- Called "overriding" (called "overloading" in Matlab documentation)
- A typical built-in function to override is **`disp`**
  - Specify which properties to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



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