• Previous lecture:
  – Structure & structure array

• Today’s lecture:
  – Introduction to objects and classes
  – Value vs. reference
  – Instantiating an object; accessing its properties and methods

• Announcements:
  – Discussion this week in classrooms, not UP B7
  – Prelim 2 at 7:30pm tonight
    • Lastnames A-H Kimball B11
    • Lastnames I-Q Upson B17
    • Lastnames R-Z Hollister B14
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 a **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
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
The `plot` handle graphics object in Matlab

```
x=...; y=...;
plot(x,y)  creates a graphics object
```

- In the past we focused on the visual produced by that command. If we want the visual to look different we make another plot.
- We can actually “hold on” to the graphics object—store its “handle”—so that we can later make changes to that object.

See `demoPlotObj.m`
Objects of the same class have the same properties

```matlab
x = 1:10;
% Two separate graphics objects:
plot(x, sin(x), 'k-')
plot(x(1:5), 2.^x, '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 objects (plots)
- The values of the properties are different for the individual objects
To specify the properties & methods of an object is to define its class

- An interval has two endpoints
- We may want to perform these actions:
  - scale and shift individual intervals
  - Determine whether two intervals overlap
  - Add and subtract two intervals

```plaintext
classdef Interval < handle

  properties
    left
    right
  end

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

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

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

    ...
  end
end
```
Defining a class ≠ creating an object

- A class is a specification
  - 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
Simplified Interval class

To create an Interval object, use its class name as a function call: $p = \text{Interval}(3, 7)$
The constructor method

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

The constructor, a specialized method whose main jobs are to
- compute the handle of the new object
- execute the function code (to assign values to properties)
- return the handle of the object
A handle object is referenced by its handle

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

### Interval Class Definition

```matlab
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 object
            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
```
A handle object is referenced by its handle

\[
p = \text{Interval}(3,7); \\
r = \text{Interval}(4,6); \\
\]

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?

\[ p = \text{Interval}(3,7); \quad \% \text{ } p \text{ } \text{references an Interval object} \]

\[ s = p; \quad \% \text{ } s \text{ } \text{stores the same reference as } p \]

\[ s.\text{left} = 2; \quad \% \text{ } \text{change value inside object} \]

\[ \text{disp}(p.\text{left}) \quad \% \text{ } 2 \text{ } \text{is displayed} \]
By contrast, structs are stored by value …

P.x=5; P.y=0;  % A point struct P
Q=P;           % Q gets a copy of P--copy
               %   all the values in the fields
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
Calling an object’s method (instance method)

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

Syntax: `<referencecname>..<methodname>(<arguments>)`
Syntax for calling an instance method

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

```matlab
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

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

```
classdef Interval < handle
    % An Interval has a left end and a right
    properties
        left
        right
    end
    methods
        function Inter = Interval(lt, rt)
            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
```

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

```
r 177.54
```
Executing an instance method

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

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 disappears when the function ends.
A class file begins with keyword `classdef`:

```
classdef classname < handle

The class specifies handle objects
```

Constructor returns a reference to the class object

```
% Constructor: construct an Interval object
Inter.left= lt;
Inter.right= rt;
end
```

Each instance method’s first parameter must be a reference to the instance (object) itself

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

Use keyword `end` for keywords `classdef`, `properties`, `methods`, `function`.

```
end
```