• Previous lecture:
  – Structure & structure array
• Today’s lecture:
  – More on structs and cell array
  – Introduction to Matlab objects and classes
• Announcements:
  – Project 5 due Friday 4/22 at 11pm
  – Prelim 2 on Tues, 4/26, at 7:30pm
  – Prelim 2 topics: end with Project 5 and non-OOP part of this lecture, i.e., will NOT include OOP

---

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

---

All possible (i,j,k) combinations but avoid duplicates. Loop index values have this relationship: $i < j < k$

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

---

Still get the same result if all three loop indices end with $n$?

A: Yes
B: No

---

Structures with array fields

Let’s develop a structure that can be used to represent a colored disk. It has four fields:

- **xc**: x-coordinate of center
- **yc**: y-coordinate of center
- **r**: radius
- **c**: rgb color vector

Examples:

```matlab
D1 = struct('xc',1,'yc',2,'r',3,... 'c',[1 0 1]);
D2 = struct('xc',4,'yc',0,'r',1,... 'c',[.2 .5 .3]);
```

---

Example: Averaging two disks

```matlab
D = (D1 + D2) / 2;
```
Example: compute “average” of two disks
% D1 and D2 are disk structures.
% Average is:
r  = (D1.r  + D2.r) /2;
xc = (D1.xc + D2.xc)/2;
yc = (D1.yc + D2.yc)/2;
c  = (D1.c  + D2.c) /2;
% The average is also a disk
D = struct('xc',xc,'yc',yc,'r',r,'c',c)

---

How do you assign to \( g \) the green-color component of disk \( D \)?

\[
D = \text{struct}(‘xc’, 3.5, ‘yc’, 2, ‘r’, 1.0, ‘c’, [0.4, 0.1, 0.5])
\]

A: \( g = D.g; \)
B: \( g = D.c.g; \)
C: \( g = D.c.2; \)
D: \( g = D.c(2); \)
E: other

---

Example: subset of clicker IDs

\[\text{IDs} = ['d091314'; ... 'h134d83'; ... 'h4567s2'; ... 'fr83209']\]

Find subset that begins with ‘h’

\[L = ['h134d83', ... 'h4567s2']\]

---

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

---

End of coverage for Prelim 2
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

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
    • Property: array of Cards
  – Player

• Then write the game—the algorithm—using objects of the above “classes”

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

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

Example class: Rectangle

• Properties:
  – xLL, yLL, width, height

• Methods (actions):
  – Calculate area
  – Calculate perimeter
  – Draw
  – Intersect (the intersection between two rectangles is a rectangle!)

Example class: Time

• Properties:
  – Hour, minute, second

• Methods (actions):
  – Show (e.g., display in hh:mm:ss format)
  – Advance (e.g., advance current time by some amount)
Example class: Window (e.g., dialog box)

- Properties:
  - Title, option buttons, input dialog …
- Methods (actions):
  - Show
  - Resize
  - …

Many such useful classes have been predefined!

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

```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 hang on to the graphics object—store its “handle”—so that we can later make changes to that object.

A “movie” of a ball tracing a curve—3 frames

**Without considering objects:** draw the three “frames” independently, each time drawing both line and ball.

**Make use of objects:** make the line and ball two independent objects that we can hold on, i.e., we hold on to their individual handles. Leave the line alone, but at each iteration give the ball new \(x\)-, \(y\)-values

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 the objects (plots)
- The values of the properties are different for the individual objects

**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

See `demoPlotObj.m`
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

Given class Interval (file Interval.m) ...

% Create 2 Intervals, call them A, B
A= Interval(2,4.5) % Assignment another right end
A.right= 14 % Half the width of A (scale by 0.5)
A.scale(.5) % See the result
A       % show all property values in A
B       % show all property values in B

Multiple Interval objects

167.32
left 3
right 7
Interval()
scale()
shift()
overlap()
add() % See demoInterval0.m

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

Given class Interval (file Interval.m) ...

% Create 2 Intervals, call them A, B
A= Interval(2,4.5) % Assignment another right end
A.right= 14 % Half the width of A (scale by 0.5)
A.scale(.5) % See the result
A       % show all property values in A
B       % show all property values in B

Multiple Interval objects

167.32
left 3
right 7
Interval()
scale()
shift()
overlap()
add() % See demoInterval0.m

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

Observations:
- Each object is referenced by a name.
- Two objects of same class has 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

The "handle" or "reference" of the object

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

Simplified Interval class

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

167.32
left 3
right 7
Interval()
scale()

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

Simple Interval class
% An Interval has a left end and a right end
classdef Interval < handle
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*(1+f); % simplify
end
end

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