

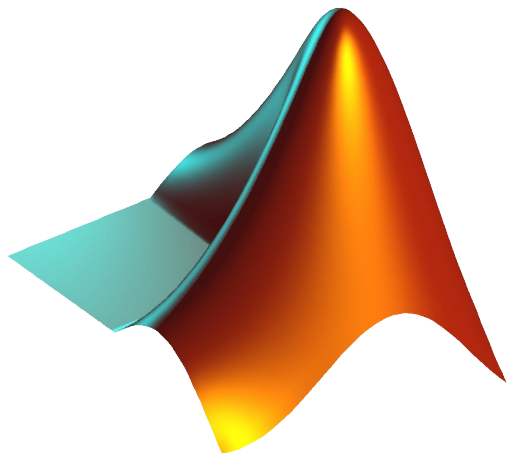
CS 1112 Introduction to Computing Using MATLAB

Instructor: Dominic Diaz

Website:

<https://www.cs.cornell.edu/courses/cs1112/2022fa/>

Today: object-oriented programming



Agenda and announcements

- Last time
 - Finished cell arrays, file input, file output
 - Finished prelim 2 material
- Today
 - Object-oriented programming
- Announcements
 - Project 5 due Monday 11/14
 - Prelim 2 next Thursday!
 - tutoring (sign up on CMS) Monday 11/7 - Wednesday 11/9
 - Review session 11/9 6:30 – 8pm in Thurston Hall room 203
 - Apply by November 14th if you would like to be a consultant for this class!

Prelim 2 topics

- 2-dimensional array (matrix)
- 3-dimensional array (e.g. color image data)
- Computing in type uint8
- Array (2-d, 3-d) algorithms/patterns: full array traverse, partial array traverse (e.g., rectangular subarray, triangular subarray)
- Characters and char arrays
- Linear search
- Cell array
- Vectorized code
- File input/output

Objects and classes

- A **class** is a data specification
 - Specify the properties of some class of things



- An **object** is a concrete instance of the class



When to use objects and Object-Oriented Programming (OOP)?

Think about rectangles...

Up til now we've drawn a bunch of rectangles using drawRect

hold on

```
DrawRect(1,2,1,2, 'g')
```

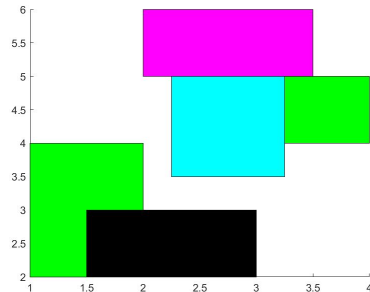
```
DrawRect(3,4,1,1, 'g')
```

```
DrawRect(1.5,2,1.5,1, 'k')
```

```
DrawRect(2,5,1.5,1, 'm')
```

```
DrawRect(2.25,3.5,1,1.5, 'c')
```

hold off



But if we have an application where we need lots of rectangles and to be able to apply different functions to these rectangles... use OOP!

```
rect1 = rectangle(1, 1, 4, 4)  
rect2 = rectangle(2, 2, 5, 6);  
A = rect1.getArea();  
P = rect1.getPerimeter();  
rect1.drawRect('g')  
rect3 = rect1.overlap(rect2);
```

When to use objects and Object-Oriented Programming (OOP)?

Think about rectangles...

Up til now we've drawn a
rectangles using drawR

hold on

```
DrawRect(1,2
```

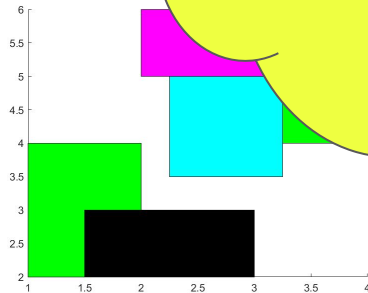
```
DrawRect(3,4
```

```
DrawRect(1.5,
```

```
DrawRect(2,5,
```

```
DrawRect(2.25
```

hold off



OOP focuses on
creating object
with methods
(functions) that act
on those object.

ation where we
nd to be able to
o these

```
1, 4, 4)
```

```
2, 5, 6);
```

```
eter();
```

```
('g')
```

```
overlap(rect2);
```

Class Interval

An interval has two **properties**

- left, right

Actions—**methods**— of an interval include

- Scale: make the interval larger or smaller
- Shift: move the interval
- Check if two intervals overlap
- ...

We're missing a few things. Let's talk about these class definitions in a little more detail!

```
classdef Interval < handle

    properties
        left
        right
    end

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

        function shift(self, f)
            ...
        end

        function Inter = overlap(self, f)
            ...
        end
    end
end
```

What's in a classdef?

```
classdef classname < handle
```

```
    properties
```

```
        prop1
```

```
        ...
```

```
    end
```

```
    methods
```

```
        function constructor
```

```
        ...
```

```
    end
```

```
        function method1
```

```
        ...
```

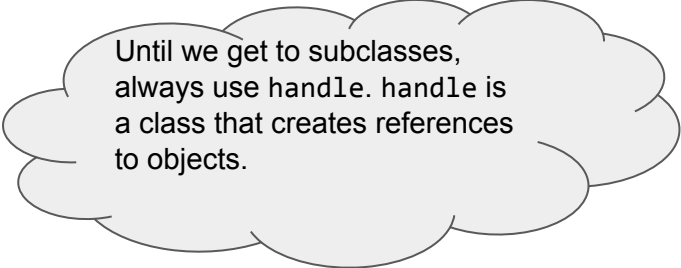
```
    end
```

```
    ...
```

```
end
```

```
end
```

begins with the keyword `classdef`:
`classdef classname < handle`



Until we get to subclasses, always use `handle`. `handle` is a class that creates references to objects.

Second is the properties that define the class

Third is the methods that act on objects of this class.

First method must be the *constructor* (the function used to construct objects)

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

What's in a classdef?

```
classdef classname < handle
```

```
    properties
```

```
        prop1
```

```
        ...
```

```
    end
```

```
    methods
```

```
        function constructor
```

```
            ...
```

```
        end
```

```
        function method1
```

```
            ...
```

```
        end
```

```
        ...
```

```
    end
```

```
end
```

```
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 factor f
```

```
        w = self.right - self.left;
```

```
        self.right = self.left + w*f;
```

```
    end
```

```
end
```

```
end
```

What's in a classdef?

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

Properties
that define
an interval
object

properties
left
right
end

constructor returns handle to
a newly allocated class object

Constructor
used to create
an interval
object

methods

```
function Inter = Interval(lt, rt)
% constructor: construct an Interval
% object
    Inter.left = lt;
    Inter.right = rt;
end
```

Other than the constructor,
each method's first parameter
must be the handle of the
object itself.
Call it `self`.

Method
(function) that
acts on interval
objects

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

The file on the right must be
called `Interval.m`

end
end

The constructor method

To create an Interval object, use its class name as a function call:

```
I = Interval(3,7);
```

Constructor, the method with the same name as the class, has the following jobs:

- Automatically compute the handle of the new object; handle is returned as output param
- Execute the function code (to assign values to the properties)

```
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 factor f
            w = self.right - self.left;
            self.right = self.left + w*f;
        end
    end
end
```

Given class Interval (in file Interval.m)

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

% Assign a new right end point
A.right = 14;

% Halve the width of A (scale by 0.5)
A.scale(0.5);

% See the result
disp(A.right)    % display 8
disp(A)         % display interval(2,8)
disp(B)         % display interval(-3,1)
```

Important observations:

- Each object is referenced by a name
- Two objects of the same class have the same properties
 - The values in those properties may be different though
- To access a property value, use the dot notation
- To access a method, use the dot notation
- Changing property values of one object doesn't affect property values of a distinct object

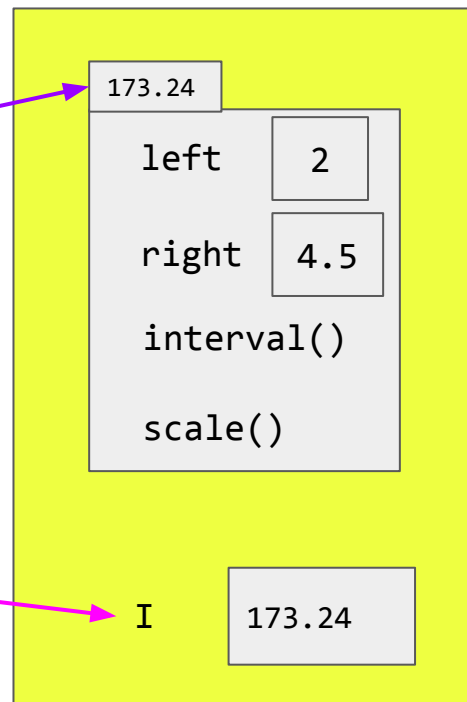
handles

A **handle** is a reference to a variable. In other words, when a variable holds a handle, it actually holds a reference to the object.

```
I = Interval(2, 4.5);
```

When I call this constructor function, MATLAB allocates space in memory for this object

I stores a handle to the object, not the object itself.



handles

A **handle** is a reference to a variable. In other words, when a variable holds a handle, it actually holds a reference to the object.

Why do we care? Because the object and a reference to that object act differently.

```
% a and b will store numbers.  
a = 10;  
b = a;  
b = 15;  
disp(b)    % will display 15  
disp(a)    % will display 10
```

```
% I1 and I2 will store handles  
% pointing to the same object  
I1 = Interval(1,5);  
I2 = I1;  
I2.right = 6;  
disp(I2)    % interval with  
             left=1  
             right=6  
  
disp(I1)    % interval with  
             left=1  
             right=6
```

Methods other than the constructor

Use the scale function with the object we are acting on before the function (and use dot notation):

```
I = Interval(0,5);  
I.scale(2);  
disp(I)           % Interval with properties  
                 left = 0  
                 right = 10
```

```
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 factor f  
            w = self.right - self.left;  
            self.right = self.left + w*f;  
        end  
    end  
end
```

Methods other than the constructor

objHandle.methodName(inputParam2, inputParam3, ..., inputParamN)

```
I = Interval(0,5);  
I.scale(2);  
disp(I)
```

Inputs other than the first input show up in parenthesis after the function name

The object handle goes before the function call.

```
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 factor f  
        w = self.right - self.left;  
        self.right = self.left + w*f;  
    end  
end  
end
```


More sample code

```
I1 = Interval(0, 2);
```

```
I2 = Interval(0, 2);
```

```
I3 = I1;
```

```
I3.scale(2)
```

```
disp(I2.left)      % displays 0
```

```
disp(I2.right)    % displays 2
```

```
disp(I1.left)     % displays 0
```

```
disp(I1.right)    % displays 4
```

```
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 factor f
            w = self.right - self.left;
            self.right = self.left + w*f;
        end
    end
end
```

OOP Vocab review

- **Class**: the template that specifies a custom type; includes the list of properties and methods of the type
- **Object**: an instance of the class
- **Property**: A variable defined in a class
- **Method**: a function defined in a class; it has access to the properties of the class
 - **Constructor**: special method that returns the handle to the newly allocated object
- **Handle**: unique identifier of an object generated by MATLAB; also called reference or address
- **Object-oriented programming**: a type of programming that focuses on creating object and writing methods that act on those objects