

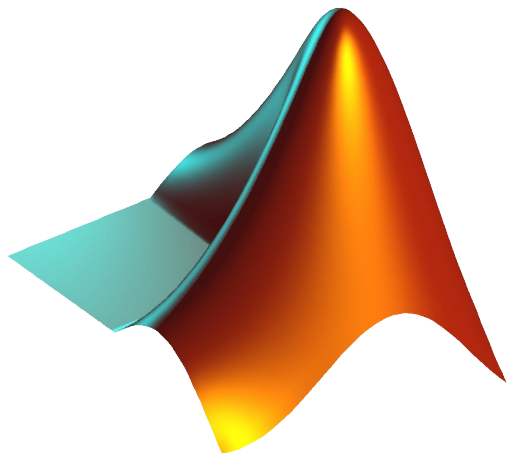
CS 1112 Introduction to Computing Using MATLAB

Instructor: Dominic Diaz

Website:

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

Today: Cell arrays and object-oriented programming



Agenda and announcements

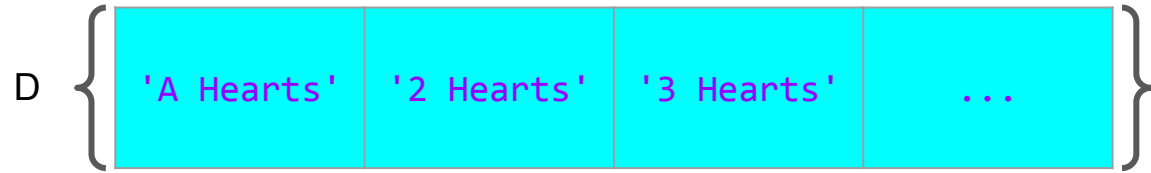
- Last time
 - Data types in MATLAB
 - Cell arrays
- Today
 - Finish cell arrays
 - File input and output
 - Object-oriented programming
- Announcements
 - Project 5 due Monday 11/14
 - 1 long problem
 - If you would like us to suggest a partner for you, fill out the P5 partner survey by the end of today!
 - Prelim 2 next Thursday!
 - Submit regrade request to prelim 2 time and location assignment on CMS if you have a conflict
 - tutoring (sign up on CMS) Monday 11/7 - Wednesday 11/9
 - Review session 11/9 6:30 – 8pm in Thurston Hall room 203

New example

```
suit = {'Hearts', 'Clubs', 'Spades', 'Diamonds'};
rank = {'A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K'};
i = 1;
D = {};
for k = 1:length(suit)
    for j = 1:length(rank)
        D{i} = [ rank{j} ' ' suit{k} ];
        i = i + 1;
    end
end
```



We have a deck of cards. How can we draw n random unique cards from the deck?



```
cards = cell(1,n);           % Store cards that have been picked
for i = 1:n
    % generate random integer until a new card is picked

    % store index of picked card and picked card itself
end
```

We have a deck of cards. How can we draw n random unique cards from the deck?



```
cards = cell(1,n);           % Store cards that have been picked
for i = 1:n
    % generate random integer until a new card is picked
    randInt = randi(length(D));
    % store index of picked card and picked card itself
end
```

randi(n) generates random integer from 1 to n

We have a deck of cards. How can we draw n random unique cards from the deck?



```
cards = cell(1,n);           % Store cards that have been picked
for i = 1:n
    % generate random integer until a new card is picked
    randInt = randi(length(D));

    % store index of picked card and picked card itself

    cards{i} = D{randInt};
end
```

We have a deck of cards. How can we draw n random unique cards from the deck?



```
alreadyPicked = zeros(1,n); % Store indices of already picked cards
cards = cell(1,n); % Store cards that have been picked
for i = 1:n
    % generate random integer until a new card is picked
    randInt = randi(length(D));

    % store index of picked card and picked card itself

    cards{i} = D{randInt};
end
```

We have a deck of cards. How can we draw n random unique cards from the deck?



```
alreadyPicked = zeros(1,n); % Store indices of already picked cards
cards = cell(1,n); % Store cards that have been picked
for i = 1:n
    % generate random integer until a new card is picked
    randInt = randi(length(D));
    while sum(randInt == alreadyPicked) > 0
        randInt = randi(length(D));
        disp(randInt)
    end
    % store index of picked card and picked card itself

    cards{i} = D{randInt};
end
```

Checks if randInt has already been picked and re-generates a random integer until a new integer (that has not been used) is generated

We have a deck of cards. How can we draw n random unique cards from the deck?



```
alreadyPicked = zeros(1,n); % Store indices of already picked cards
cards = cell(1,n); % Store cards that have been picked
for i = 1:n
    % generate random integer until a new card is picked
    randInt = randi(length(D));
    while sum(randInt == alreadyPicked) > 0
        randInt = randi(length(D));
        disp(randInt)
    end
    % store index of picked card and picked card itself
    alreadyPicked(i) = randInt;
    cards{i} = D{randInt};
end
```

File input and output

File input

Given some file 'data.txt' that we need to process, we can use the functions fopen, feof, fgetl, fclose

```
fid = fopen('data.txt', 'r');
```

```
k = 0;
```

```
D = {};
```

```
while ~feof(fid)
```

```
    k = k + 1;
```

```
    D{k} = fgetl(fid);
```

```
end
```

```
fclose(fid);
```

File output

Given some cell array C that we want to write to a new file, we can use the functions fopen, fprintf, fclose

% Assume C stores a bunch of char arrays

```
fid = fopen('outData.txt', 'w');
```

```
for i = 1:length(C)
```

```
    fprintf(fid, '%s\n', C{i});
```

```
end
```

```
fclose(fid);
```

File input and output

This command opens the file outData.txt (or creates the file if it does not already exist). The 'w' indicates that we want to write to the file.

fprintf allows us to print to a file. The first input is the fileID, the second is the text you want to print

- Last two inputs work the same as regular fprintf('%s\n', C{i})

Always close a file that you open

File output

Given some cell array C that we want to write to a new file, we can use the functions fopen, fprintf, fclose

% Assume C stores a bunch of char arrays

```
fid = fopen('outData.txt', 'w');
```

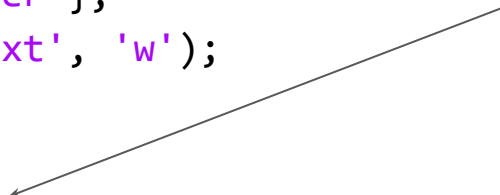
```
for i = 1:length(C)  
    fprintf(fid, '%s\n', C{i});  
end
```

```
fclose(fid);
```

Example: Saving data from a cell array to a text file

```
C = {1978, 'Michael Myers';  
     1979, 'The Tall Man';  
     1992, 'Candyman';  
     2014, 'The Babadook';  
     1990, 'Pennywise';  
     1931, 'Frankenstein';  
     1988, 'Chucky';  
     1984, 'Freddy Krueger'};  
fid = fopen('Halloween.txt', 'w');  
[nr, nc] = size(C);  
for i = 1:nr  
    fprintf(fid, '%d %s\n', C{i,1}, C{i,2});  
end  
fclose(fid);
```

Take care to ensure that there are not extra lines at the end of this file. How the code is currently written, there will be an extra empty line at the end of the file. How can we prevent this?



A note on vectorized code

```
% vectorized code to add
```

```
% two vectors
```

```
a = rand(1,4);
```

```
b = rand(1,4);
```

```
c = a + b;
```

```
% non-vectorized version
```

```
a = rand(1,4);
```

```
b = rand(1,4);
```

```
c = [];
```

```
for k = 1:length(a)
```

```
    c(k) = a(k) + b(k);
```

```
end
```

Vectorized code refers to operations that are performed on multiple components of a vector at the same time (in one statement).

Left: addition happens for all components of a and b at the same time (in one statement)

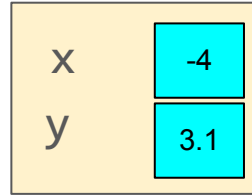
Right: addition happens on one component of a and one component of b at a time

If we say no vectorized code, you want to perform an operation on some array element-by-element

End of prelim 2
material!

Packaging data: options for storing a point (-4, 3.1)

- Simple scalars



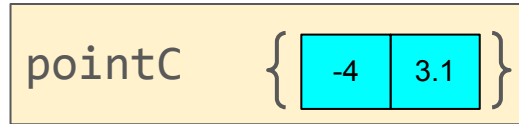
Ungrouped data

- Simple vector

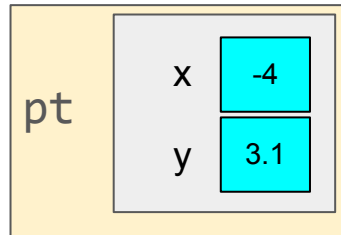


Related data grouped into an array. Can be homogeneous data (like in a simple vector) or heterogeneous data (like in a cell array)

- Cell Array



- Object



Related data grouped according to a class definition. Allows us to group data and have names for each property of this data.

A card game developed in two ways

- Develop the algorithm of the card game
 - Set up a deck as an array of cards
 - Shuffle the cards
 - Deal card to the players
 - Evaluate each player's hand to determine winner
- Identify the “**objects**” in the game and define each
 - Card
 - Properties: suit, rank
 - Actions: compare, show
 - Deck
 - Properties: array of cards
 - Actions: shuffle, deal, get #cards
 - Hand ...
 - Then write the game—the algorithm—using objects of the above “**class**”

A card game developed in two ways

- Develop the algorithm of the card game
 - Set up a deck as an array of cards
 - Shuffle the cards
 - Deal card to the players
 - Evaluate each player's hand to determine winner
- Identify the “objects” in the game and define each
 - Card
 - Properties: suit, rank
 - Actions: compare, show
 - Deck
 - Properties: array of cards
 - Actions: shuffle, deal, get #cards
 - Hand ...

Procedural Programming:

Focus on the algorithm (the procedures) necessary for solving the problem

Object-Oriented Programming:

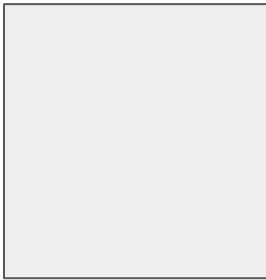
Focus on the design of the objects (data + actions on that data) necessary for solving a problem

Two steps of object-oriented programming

- Define the classes (of the objects)
 - Identify the properties (data) and actions (functions) of each class
- Create the objects (from the classes) that are then used—that interact with one another

Example class: rectangle

- Properties that define a rectangle
 - xLL, yLL, width, height
- Methods (actions we want to be able to do with a rectangle)
 - Calculate area
 - Calculate perimeter
 - Draw the rectangle
 - Intersect (the intersection between two rectangles is a rectangle!)



(xLL, yLL)

In OOP, we'll be able to create an **object** of **class** rectangle and easily apply all of these methods on this object

Objects and classes

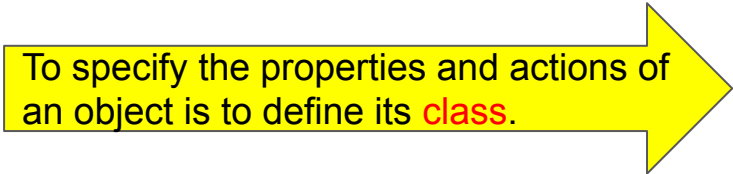
- A **class** is a data specification
 - Basically like how a cookie cutter specifies the shape of a cookie
- An **object** is a concrete instance of the class
 - Need to apply the cookie cutter to get a cookie
 - Many instances can be made using the class
 - Instances do not interfere with one another. For example, biting the head off another cookie doesn't remove the heads of the other cookies



First look at a class

Class Interval

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



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

```
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
```