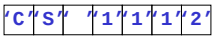
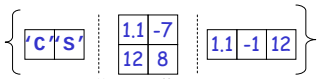
- Previous Lecture:
  - Cell arrays
- Today's Lecture:
  - More on cell arrays
  - Structures
  - Structure array (i.e., an array of structures)
  - A structure with array fields (next lecture)
- Announcement:
  - Discussion this week in the computer lab (UP B7)

Array vs. Cell Array

- Simple array

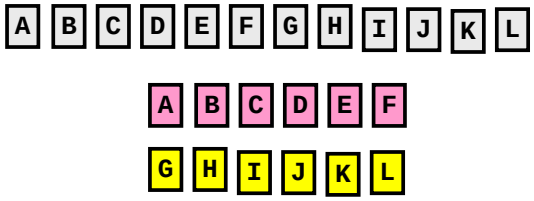


  - Each component stores one value (e.g., char, double, uint8)
  - All components have the same type
- Cell array

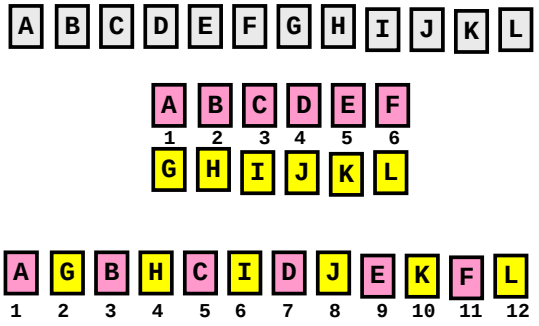


  - Each cell can store something "bigger" than one value, e.g., a vector, a matrix, a string
  - The cells may store items of different types

Perfect Shuffle, Step 1: cut the deck



Perfect Shuffle, Step 2: Alternate



Example: Build a cell array of Roman numerals for 1 to 3999

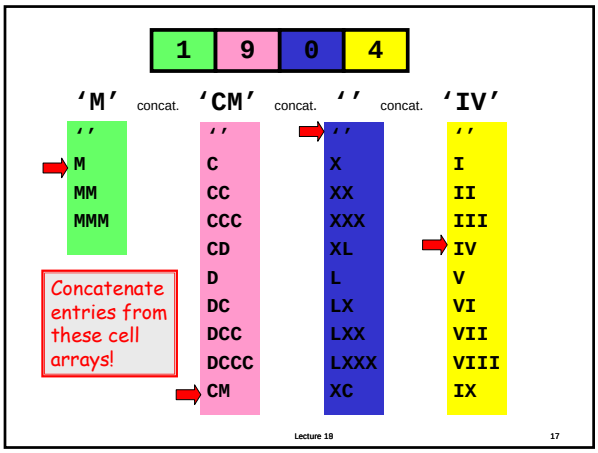
```
C{1} = 'I'
C{2} = 'II'
C{3} = 'III'
   :
C{2007} = 'MMVII'
   :
C{3999} = 'MMMXMXCIX'
```


Example

1904 = 1*1000 + 9*100 + 0*10 + 4*1

= M CM IV

= MCMIV



Ones-Place Conversion

```
function r = Ones2R(x)
% x is an integer that satisfies
%      0 <= x <= 9
% r is the Roman numeral with value x.

Ones = {'I', 'II', 'III', 'IV', ...
        'V', 'VI','VII', 'VIII', 'IX'};

if x==0
     r = '';
else
     r = Ones{x};
end
```


Similarly, we can implement these functions:

```
function r = Tens2R(x)
% x is an integer that satisfies
%      0 <= x <= 9
% r is the Roman numeral with value 10*x.
```

```
function r = Hund2R(x)
% x is an integer that satisfies
%      0 <= x <= 9
% r is the Roman numeral with value 100*x
```

```
function r = Thou2R(x)
% x is an integer that satisfies
%      0 <= x <=3
% r is the Roman numeral with value 1000*x
```


We want all the Roman Numerals from 1 to 3999. We have the functions Ones2R, Tens2R, Hund2R, Thou2R.

The code to generate all the Roman Numerals will include loops—nested loops. How many are needed?

A: 2 B: 4 C: 6 D: 8

Example: subset of clicker IDs

IDs

```
['d091314'; ...
 'h134d83'; ...
 'h4567s2'; ...
 'fr83209']
```

Find subset that begins with 'h'

L

```
{'h134d83', ...
 'h4567s2'}
```

```
k= 0;
for r=1:size(IDs,1)
  % check row r
  if IDs(r,1)=='h'
    k= k+1;
    L{k }= IDs(r,:);
  end
end
```

Directly assign into a particular cell—good!

```
L= {};

for r=1:size(ID,1)
  if IDs(r,1)=='h'

    L= [L, IDs(r,:)];
  end
end
```

Concatenate cells or cell arrays—prone to problems!

Data are often related

- A point in the plane has an x coordinate and a y coordinate.
- If a program manipulates lots of points, there will be lots of x's and y's.
- Anticipate clutter. Is there a way to "package" the two coordinate values?

Packaging affects thinking

Our Reasoning Level:

P and Q are points. Compute the midpoint M of the connecting line segment.

Behind the scenes we do this:

$M_x = (P_x + Q_x)/2$
$M_y = (P_y + Q_y)/2$

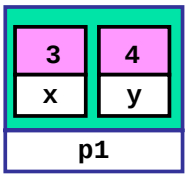
We've seen this before: functions are used to "package" calculations.

This packaging (a type of abstraction) elevates the level of our reasoning and is critical for problem solving.

Simple example

```
p1 = struct('x',3,'y',4);

p2 = struct('x',-1,'y',7);
```




```
D = sqrt((p1.x-p2.x)^2 + (p1.y-p2.y)^2);
```

D is distance between two points.

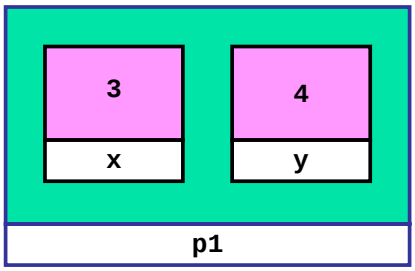
p1.x, p1.y, p2.x, p2.y participating as variables—because they are.

Creating a structure (by direct assignment)

```
p1 = struct('x',3,'y',4);
```

p1 is a structure.
The structure has two fields.
Their names are **x** and **y**.
They are assigned the values 3 and 4.

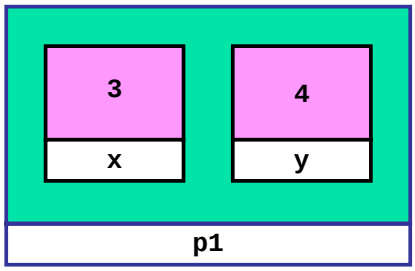
Accessing the fields in a structure




```
A = p1.x + p1.y;
```

Assigns the value 7 to A

Assigning to a field in a structure




```
p1.x = p1.y^2;
```

Assigns the value 16 to p1.x

A structure can have fields of different types

```
A = struct('name', 'New York',…
           'capital', 'Albany',…
           'Pop', 15.5)
```

- Can have combinations of string fields and numeric fields
- Arguments are given in pairs: a field name, followed by the value

Legal/Illegal maneuvers

```
Q = struct('x',5,'y',6)

R = Q        % Legal. R is a copy of Q

S = (Q+R)/2  % Illegal. Must access the
             % fields to do calculations

P = struct('x',3,'y') % Illegal. Args must be
                      % in pairs (field name
                      % followed by field
                      % value)

P = struct('x',3,'y',[]) % Legal. Use [] as
P.y = 4                  % place holder
```


Structures in functions

```
function d = dist(P,Q)
% P and Q are points (structure).
% d is the distance between them.

d = sqrt((P.x-Q.x)^2 + ...
         (P.y-Q.y)^2);
```


Example "Make" Function

Good style: use a "make" function to highlight a structure's definition

```
function P = MakePoint(x,y)
% P is a point with P.x and P.y
% assigned the values x and y.

P = struct('x',x,'y',y);
```

Then in a script or some other function…

```
a= 10;  b= rand(1);
Pt= MakePoint(a,b); % create a point struct
                    % according to definition
                    % in MakePoint function
```


Another function that has structure parameters

```
function DrawLine(P,Q,c)
% P and Q are points (structure).
% Draws a line segment connecting
% P and Q. Color is specified by c.

plot([P.x Q.x],[P.y Q.y],c)
```

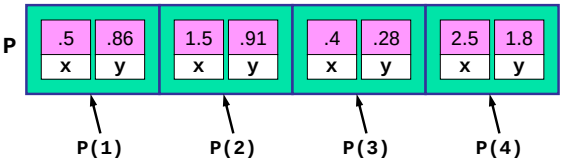

Pick Up Sticks

```
s = 'rgbmcy';
for k=1:100
   P = MakePoint(randn(1),randn(1));
   Q = MakePoint(randn(1),randn(1));
   c = s(ceil(6*rand(1)));
   DrawLine(P,Q,c)
end
```

Generates two random points and chooses one of six colors randomly.

Structure Arrays

- An array whose components are structures
- All the structures must be the same (have the same fields) in the array
- Example: an array of points (point structures)



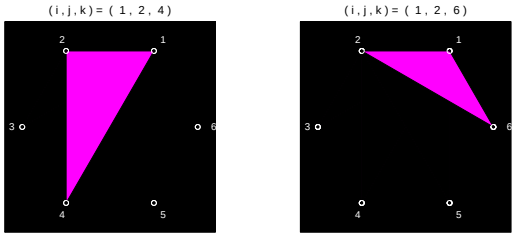
Function returning an array of points (point structures)

```
function P = CirclePoints(n)

theta = 2*pi/n;
for k=1:n
   c = cos(theta*k);
   s = sin(theta*k);
   P(k) = MakePoint(c,s);
end
```

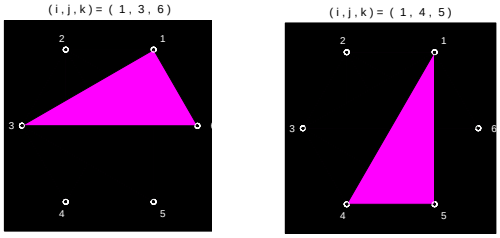

Example: all possible triangles

- Place n points uniformly around the unit circle.
- Draw all possible unique triangles obtained by connecting these points 3-at-a-time.




```
function DrawTriangle(P,Q,R,c)
% Draw c-colored triangle;
% triangle vertices are points P,
% Q, and R.

fill([P.x Q.x R.x], ...
     [P.y Q.y R.y], c)
```

The following triangles are the same: (1,3,6), (1,6,3), (3,1,6), (3,6,1), (6,1,3), (6,3,1)

Bad! i, j, and k should be different, and there should be no duplicates

```
for i=1:n
  for j=1:n
    for k=1:n
       % Draw a triangle with vertices
       %   P(i), P(j), and P(k)
    end
   end
end
```


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

i j k

i = 1	i = 2	i = 3	i = 4
1 2 3	2 3 4	3 4 5	4 5 6
1 2 4	2 3 5	3 4 6	
1 2 5	2 3 6	3 5 6	
1 2 6	2 4 5		
1 3 4	2 4 6		
1 3 5	2 5 6		
1 3 6			
1 4 5			
1 4 6			
1 5 6			

```
for i=1:n-2
  for j=i+1:n-1
    for k=j+1:n
      disp([i j k])
    end
  end
end
```

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

```
for i=1:n-2
  for j=i+1:n-1
    for k=j+1:n
      % Draw triangle with
      % vertices P(i),P(j),P(k)
    end
  end
end
```


All possible triangles

```
% Drawing on a black background
for i=1:n-2
  for j=i+1:n-1
    for k=j+1:n
      DrawTriangle( P(i),P(j),P(k),'m')
      DrawPoints(P)
      pause
      DrawTriangle(P(i),P(j),P(k),'k')
    end
  end
end
```


Still get the same result if all three loop indices end with n? A: Yes B: No

i j k

1 2 3
1 2 4
1 2 5
1 2 6
1 3 4
1 3 5
1 3 6
1 4 5
1 4 6
1 5 6
i = 1

2 3 4
2 3 5
2 3 6
2 4 5
2 4 6
2 5 6
i = 2

3 4 5
3 4 6
3 5 6
i = 3

4 5 6
i = 4

```
for i=1:n
  for j=i+1:n
    for k=j+1:n
      disp([i j k])
    end
  end
end
```