

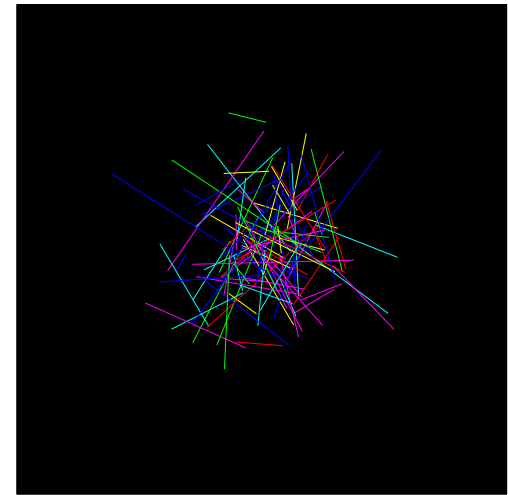
- Previous Lecture:
 - File I/O, use of cell array

- Today's Lecture:
 - Structures
 - Structure array (i.e., an array of structures)
 - A structure with array fields

- Announcements:
 - **Project 5** due **Thurs 11/5** at 11pm. Reduced late penalty of 5% applies to submission made up to 11/6 at 11pm
 - **Prelim 2** on **Thurs 11/12** at 7:30pm. Email TA Wayne (wtu4) now if you have an exam conflict (include the course and instructor info of the conflicting exam)

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.

Options for storing a point (-4, 3.1)

- Simple scalars `xdat` -4 `ydat` 3.1 *Ungrouped data*
-

- Simple vector `ptdat`

1	2
-4	3.1

Related data grouped into an array. X-coord implicitly labelled 1; y-coord implicitly labelled 2
 - Cell array `ptdatc` {

1
-4

2
3.1

 } *Related data grouped into an array. X-coord implicitly labelled 1; y-coord implicitly labelled 2*
-

- Struct `pt`

x	y
-4	3.1

Related data grouped into a struct variable. Explicit, clear labelling is possible via field names

Example: a Point structure

```
% p1 is a Point
```

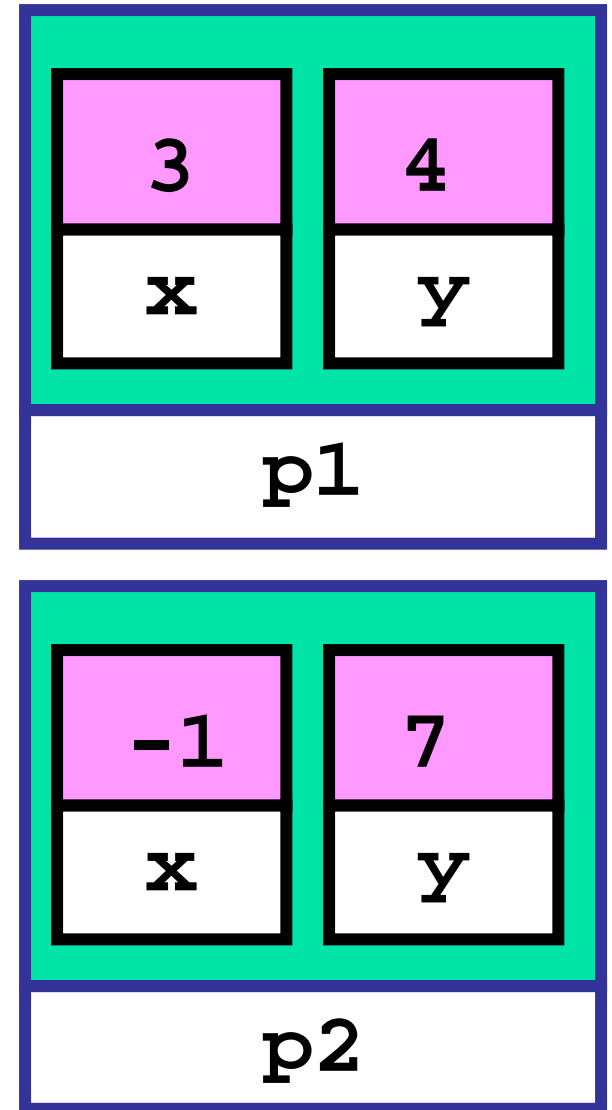
```
p1.x= 3;
```

```
p1.y= 4;
```

```
% p2 is another Point
```

```
p2.x= -1;
```

```
p2.y= 7;
```

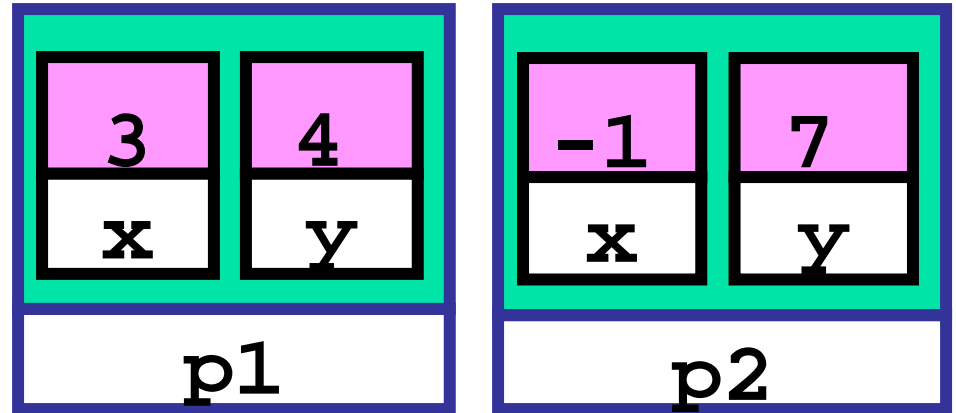


A Point has two properties—**fields**—x and y

Working with Point structures

```
p1.x=3; p1.y=4;
```

```
p2.x=-1; p2.y=7;
```



```
% Distance between points p1 and p2
```

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

Note that `p1.x`, `p1.y`, `p2.x`, `p2.y` participate in the calculation as variables—because they are.

Different ways to create a structure

`% Create a struct by assigning field values`

```
p1.x= 3;
```

```
p1.y= 4;
```

`% Create a struct with built-in function`

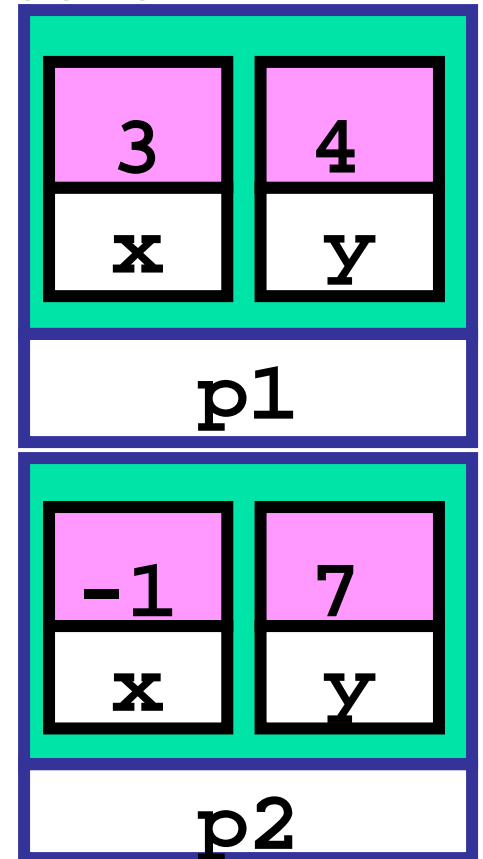
```
p2 = struct('x',-1, 'y',7);
```

`p2` is a structure.

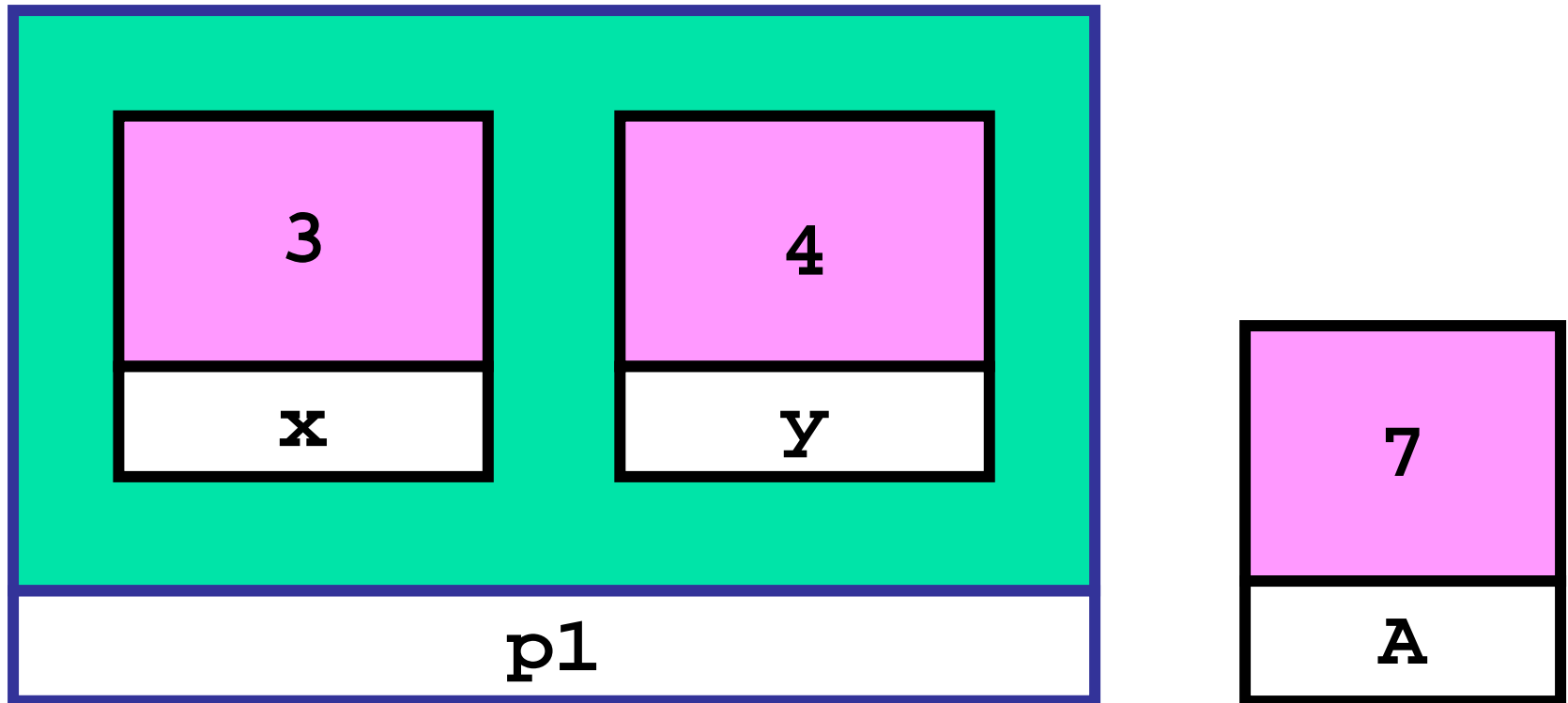
The structure has two fields.

Their names are `x` and `y`.

They are assigned the values -1 and 7.



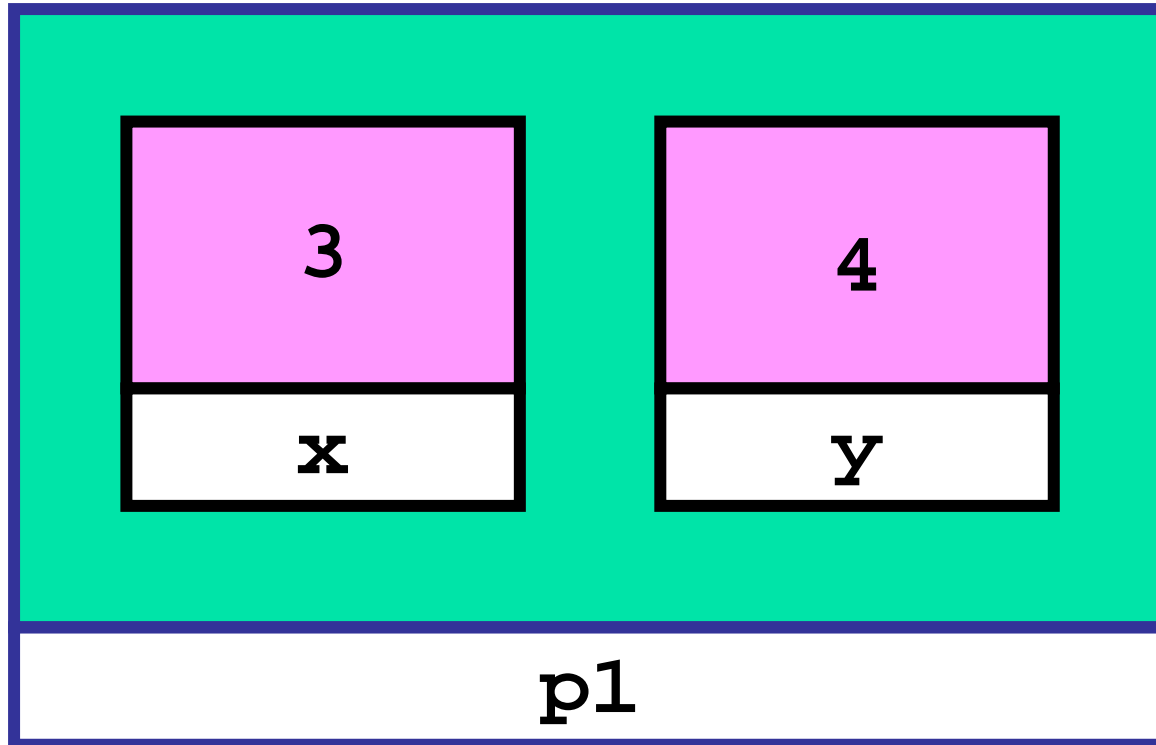
Accessing the fields in a structure



```
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( 'sname', '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;
Pt= MakePoint(a,b); % create a point struct
                    % according to definition
                    % in MakePoint function
```

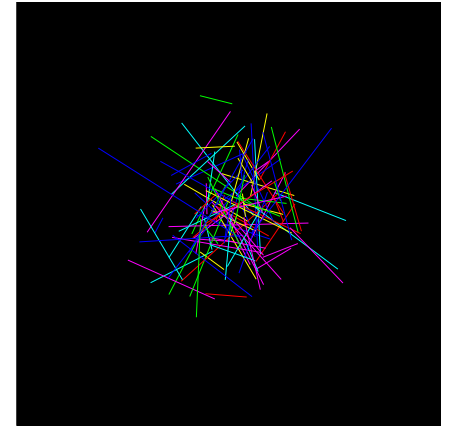
Another function that has structure parameters

```
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

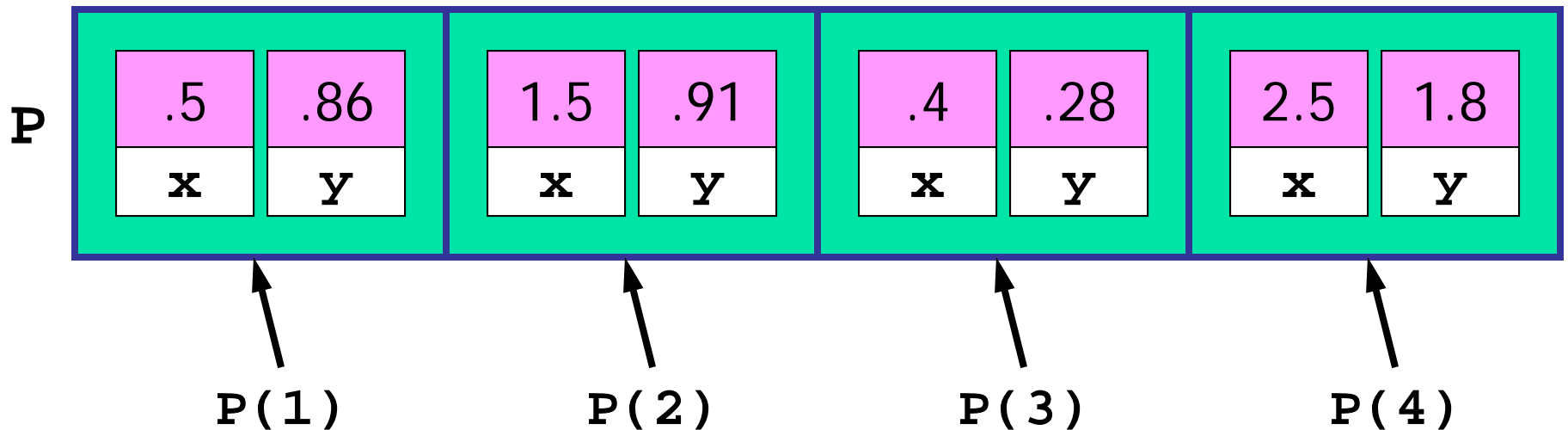
```
s = 'rgbmcy';  
for k=1:100  
    P = MakePoint(randn,randn);  
    Q = MakePoint(randn,randn);  
    c = s(ceil(6*rand));  
    DrawLine(P,Q,c)  
end
```



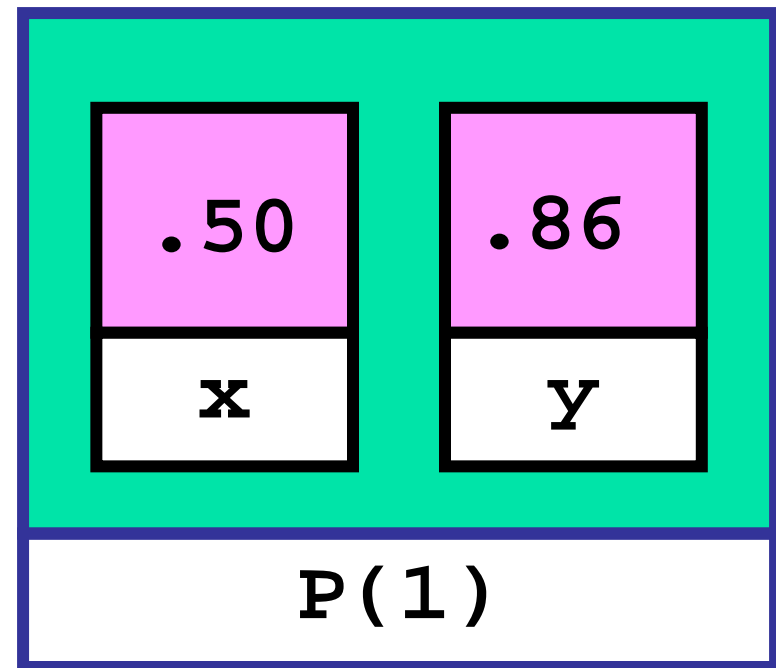
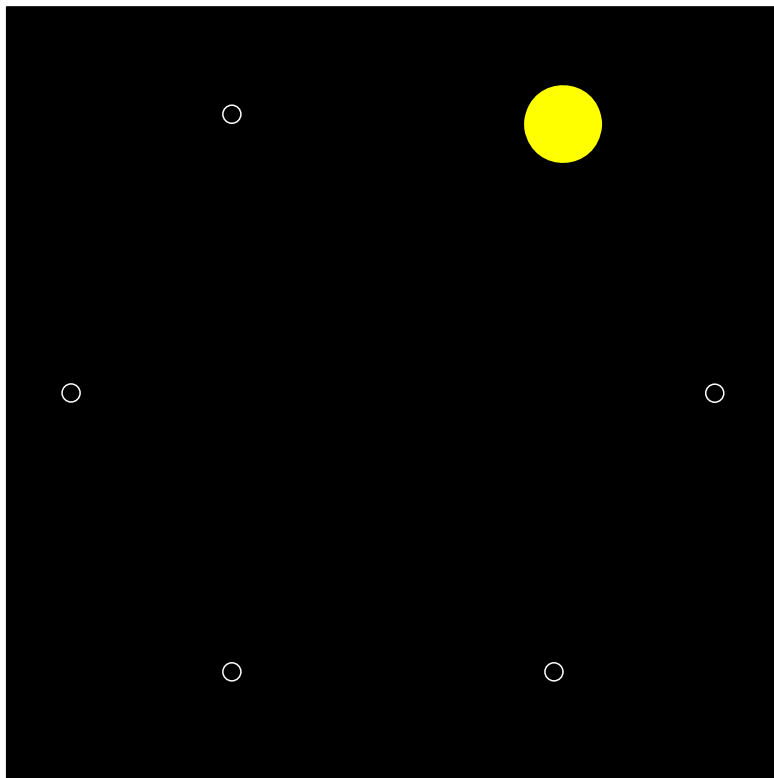
Generates two random points and connect them using one of six colors chosen randomly.

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)

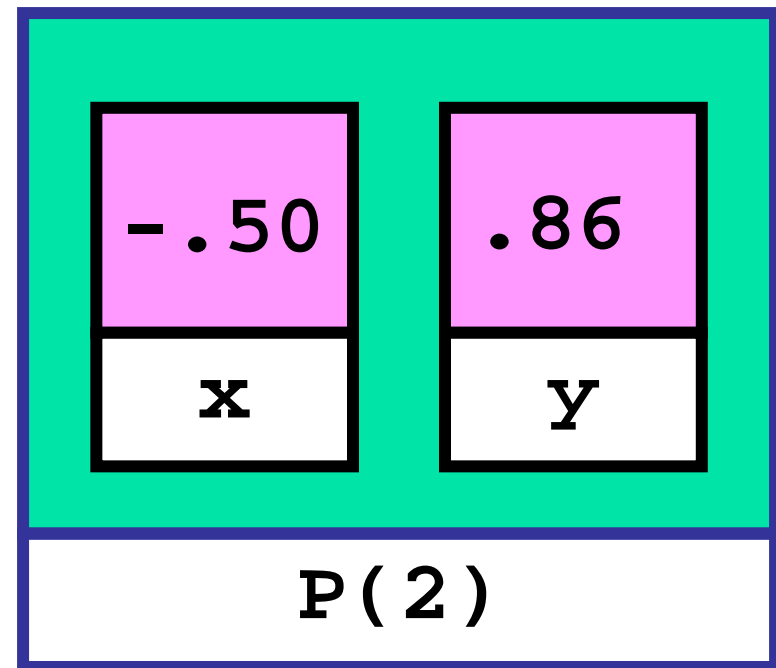
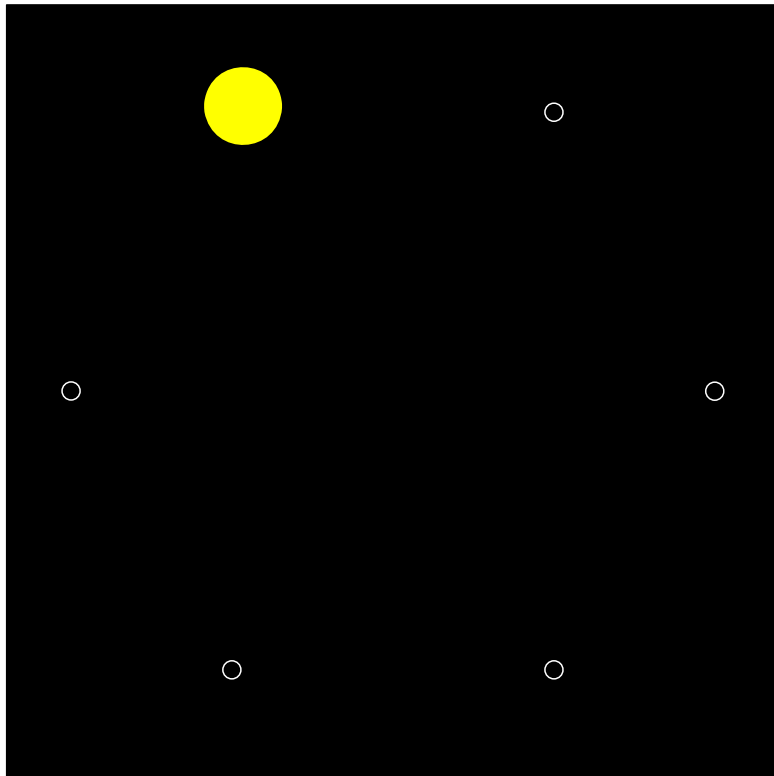


An Array of Points



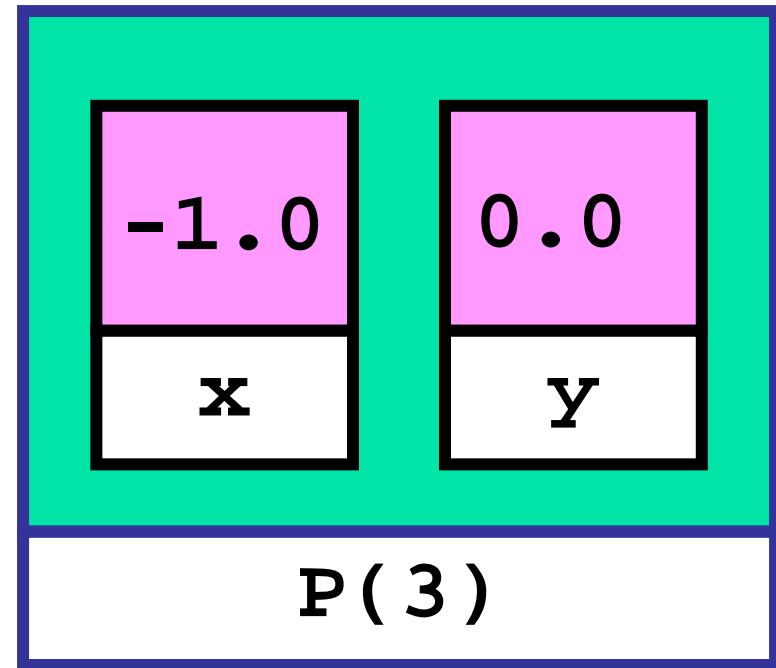
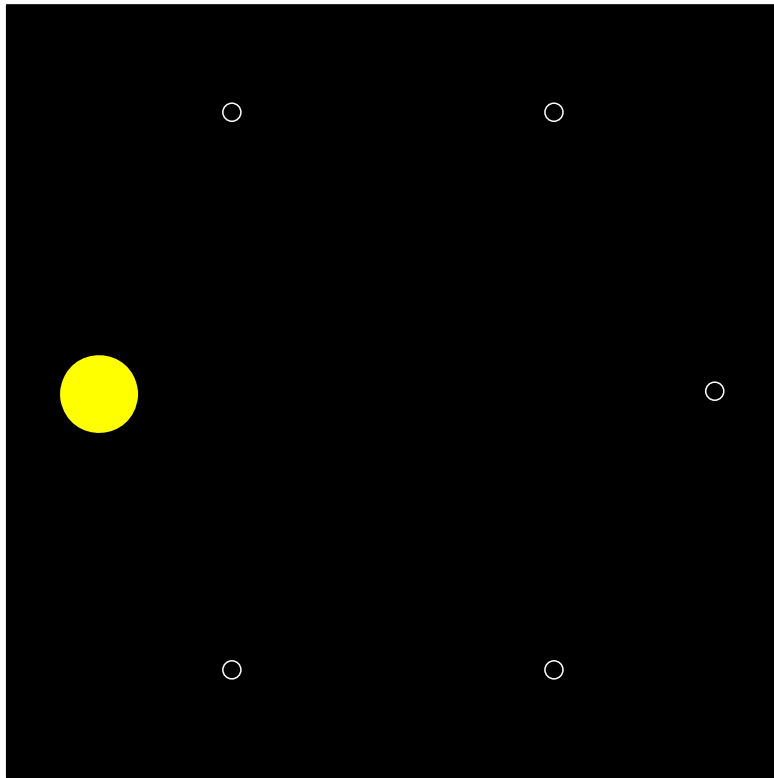
```
P(1) = MakePoint(.50, .86)
```

An Array of Points



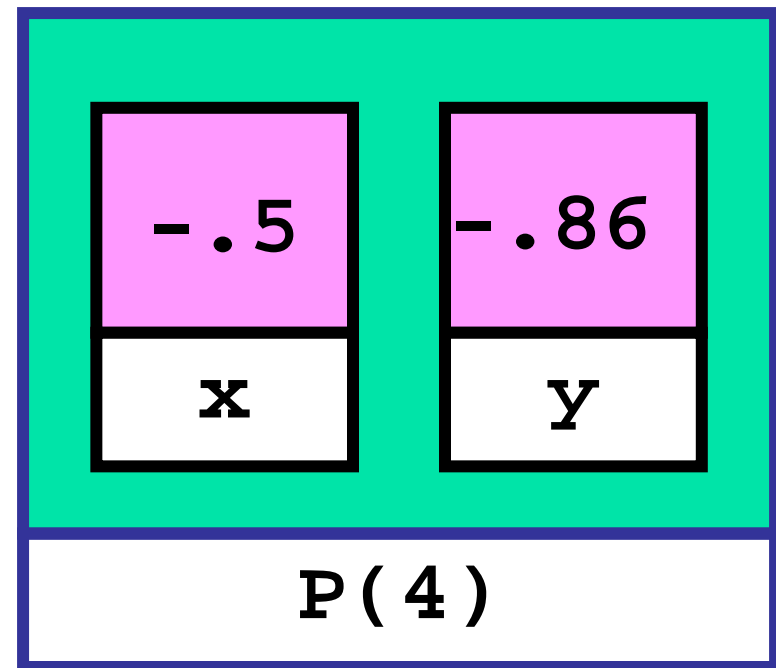
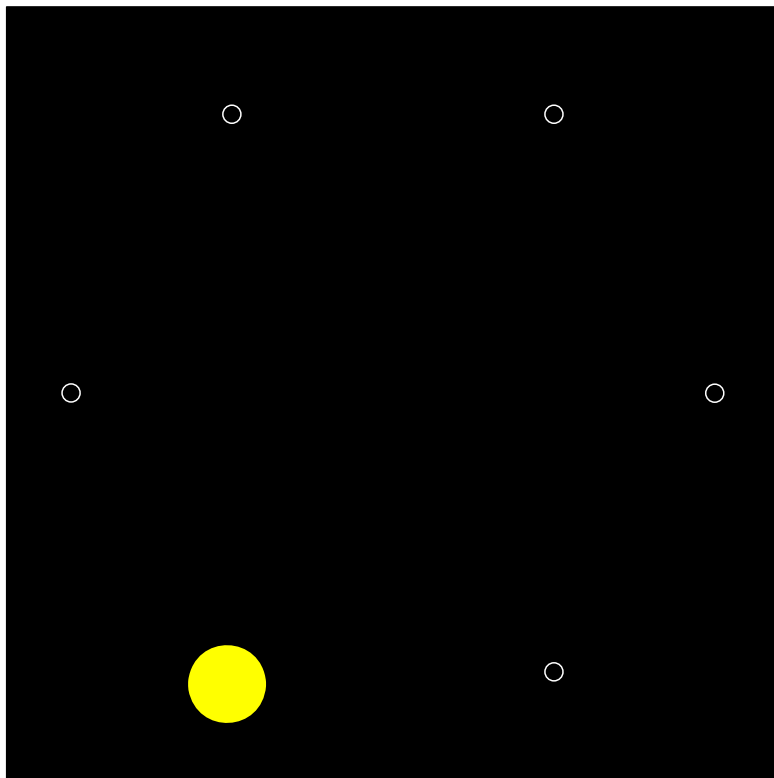
`P(2) = MakePoint(-.50, .86)`

An Array of Points



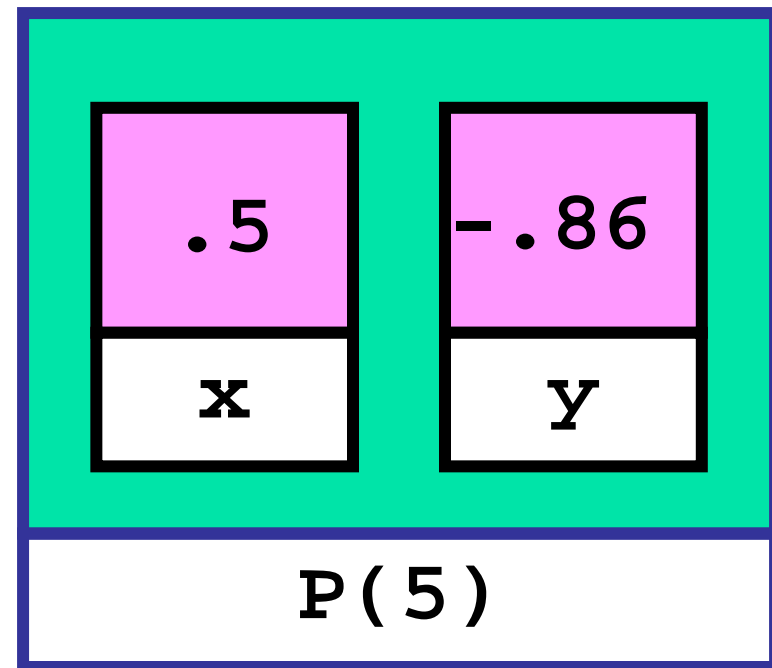
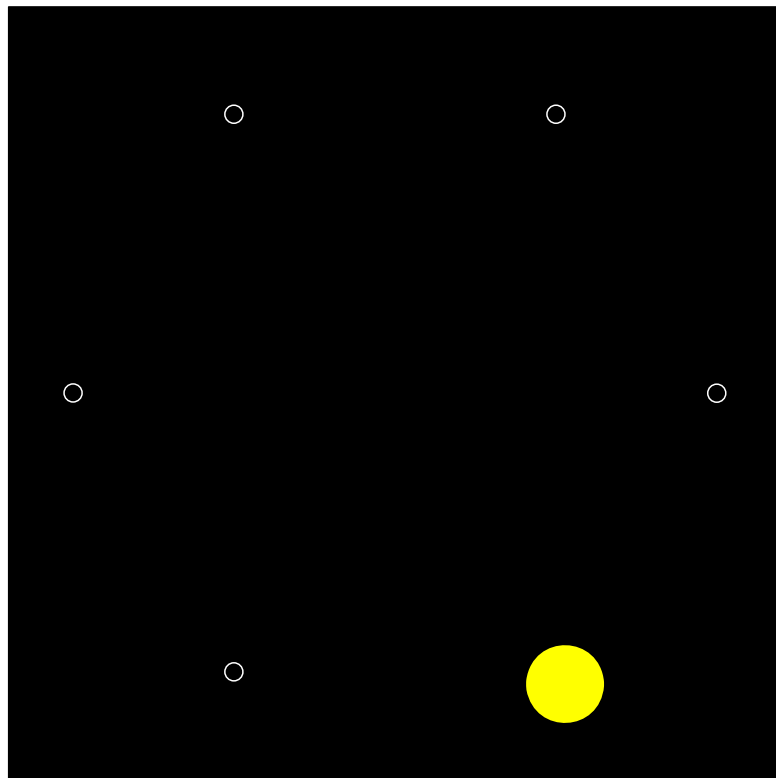
P(3) = MakePoint(-1.0, 0.0)

An Array of Points



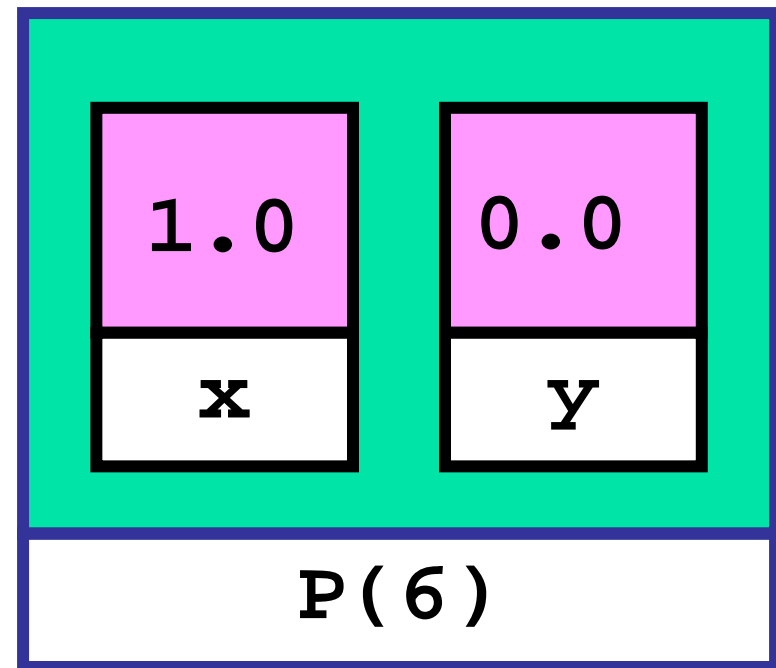
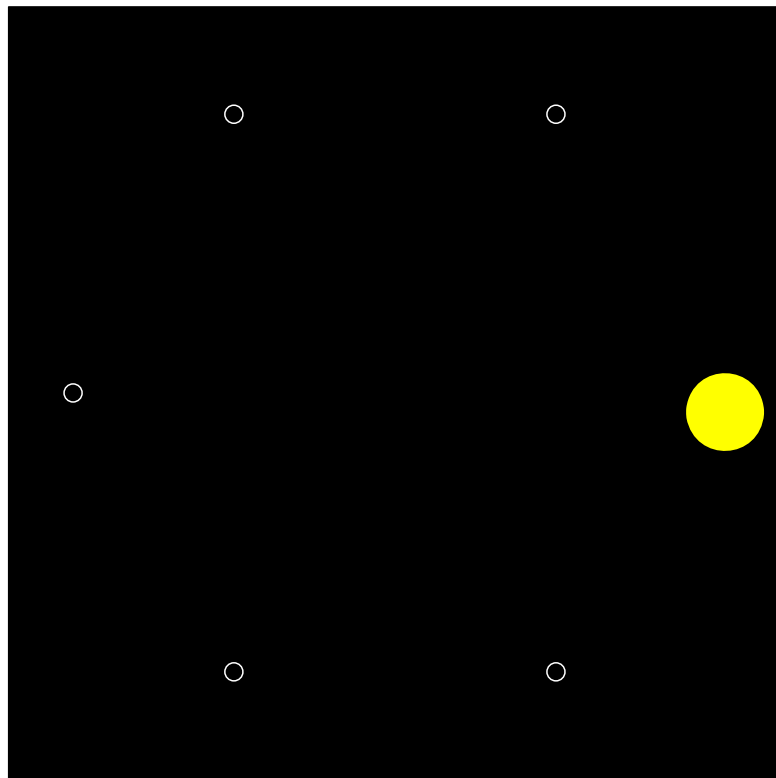
$P(4) = \text{MakePoint}(-.50, -.86)$

An Array of Points



`P(5) = MakePoint(.50, -.86)`

An Array of Points



P(6) = MakePoint(1.0, 0.0)

Function returning an array of **points** (point structures)

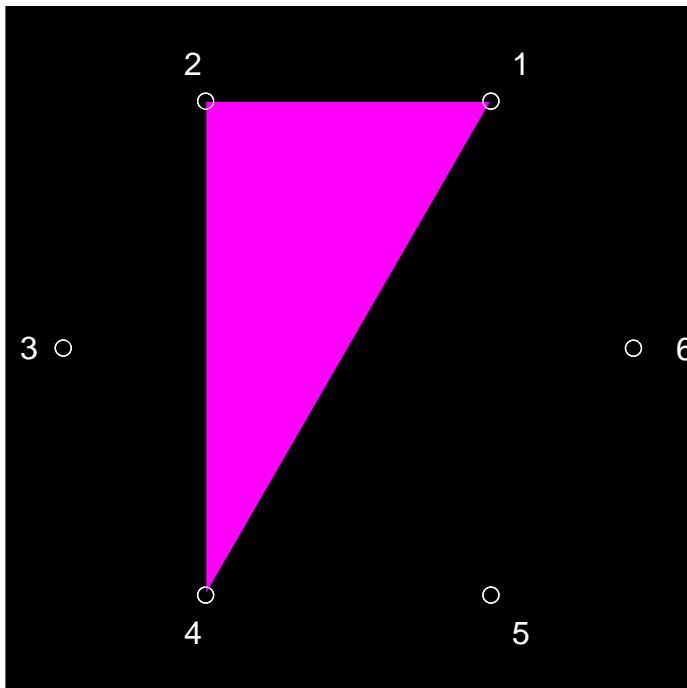
```
function P = CirclePoints(n)
%P is array of n point structs; the
%points are evenly spaced on unit circle

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

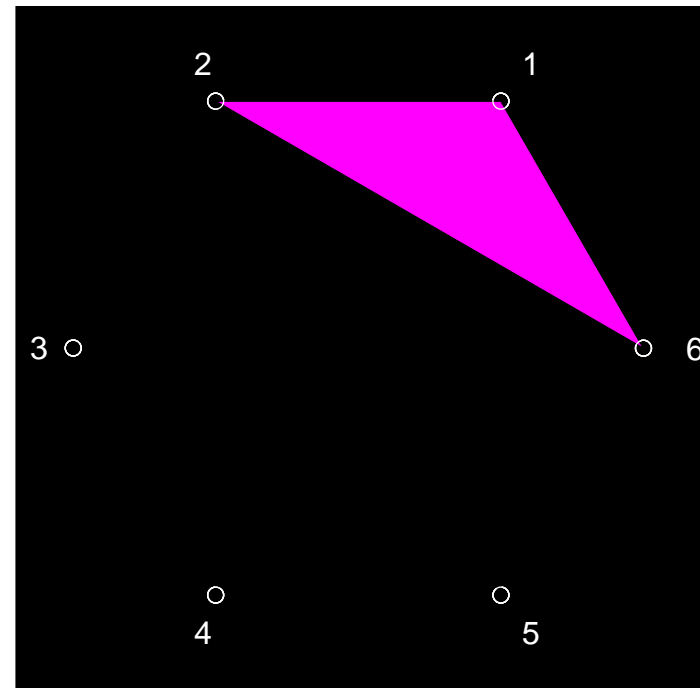
Example: all possible triangles

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

$(i, j, k) = (1, 2, 4)$



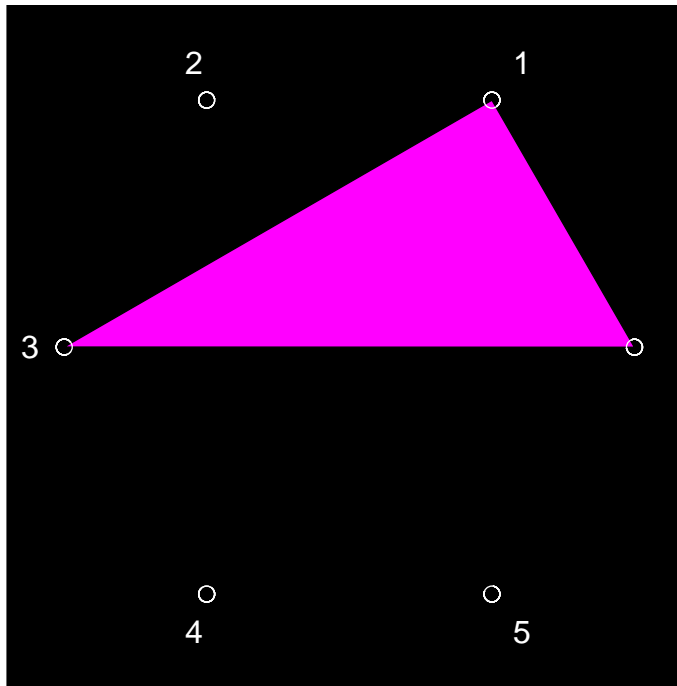
$(i, j, k) = (1, 2, 6)$



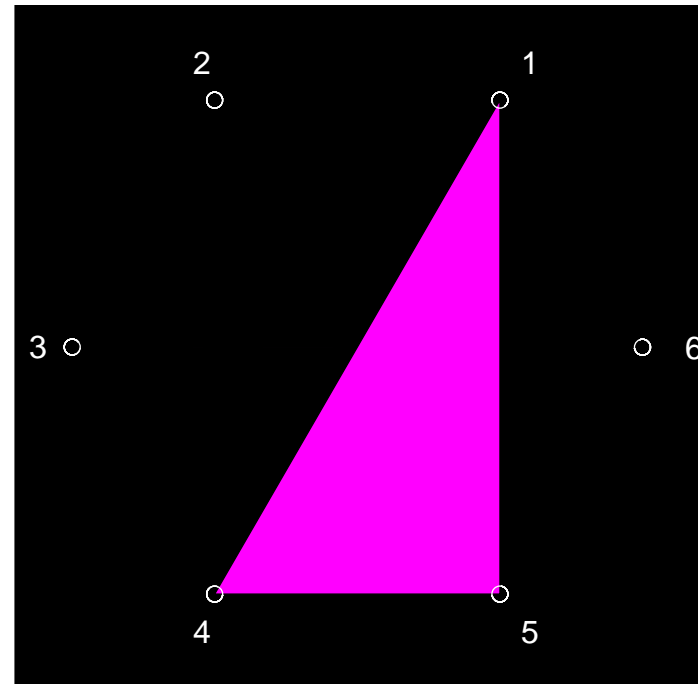
```
function DrawTriangle(U,V,W,c)
% Draw c-colored triangle;
% triangle vertices are points U,
% V, and W.

fill([U.x V.x W.x], ...
     [U.y V.y W.y], c)
```

$(i, j, k) = (1, 3, 6)$



$(i, j, k) = (1, 4, 5)$



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

```
% Given P, an array of point structures
for i=1:n
    for j=1:n
        for k=1:n
            DrawTriangle(P(i),P(j),P(k),'m')
            pause
            DrawTriangle(P(i),P(j),P(k),'k')
        end
    end
end
end
```

All possible (i,j,k) combinations but avoid duplicates.

Loop index values have this relationship $i < j < k$

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-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
            disp([i j k])
        end
    end
end
```

```
for i=1:n
    for j=1:n
        for k=1:n
            if i<j && j<k
                disp([i j k])
            end
        end
    end
end
```

Both versions print all possible, unique combinations of (i,j,k), but the left fragment is far more efficient

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
end
```

All possible unique 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
end
```

See `LotsaTriangles.m`