- 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)

- Announcements:
  - Project 5 due Thursday at 11pm
  - Section this week in the computer lab (UP B7)
  - Review session this Sunday. Location and time TBA.
  - Prelim 2 next Tuesday, Nov 10th

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

1	9	0	4

MCMIV

1	9	0	4

‘M’ concat. ‘CM’ concat. ‘’ concat. ‘IV’

‘’	‘’	‘’	‘’
M	C	X	I
MM	CC	XX	II
MMM	CCC	XXX	III
	CD	XL	IV
	D	L	V
	DC	LX	VI
	DCC	LXX	VII
	DCCC	LXXX	VIII
	CM	XC	IX

Concatenate entries from these cell arrays!

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
```

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

Now we can build the Roman numeral cell array for 1,…,3999

```
for a = 0:3
  for b = 0:9
    for c = 0:9
      for d = 0:9
        n = a*1000 + b*100 + c*10 + d;
        if n>0
          C{n} = [Thou2R(a) Hund2R(b)...
                                Tens2R(c) Ones2R(d)];
        end
      end
    end
  end
end
```

Now we can build the Roman numeral cell array for 1,…,3999

```
for a = 0:3  % possible values in thous place
  for b = 0:9  % values in hundreds place
    for c = 0:9  % values in tens place
      for d = 0:9  % values in ones place
        n = a*1000 + b*100 + c*10 + d;
        if n>0
          C{n} = [Thou2R(a) Hund2R(b)...
                           Tens2R(c) Ones2R(d)];
        end
      end
    end
  end
end
```

Four strings concatenated together

The n^{th} component of cell array C

The reverse conversion problem

Given a Roman Numeral, compute its value.

Assume cell array **`C(3999,1)`** available where:

```
C{1} = 'I'
C{2} = 'II'
    :
C{3999} = 'MMMCMXCIX'
```

See script **`RN2Int`**

RN2Int.m

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

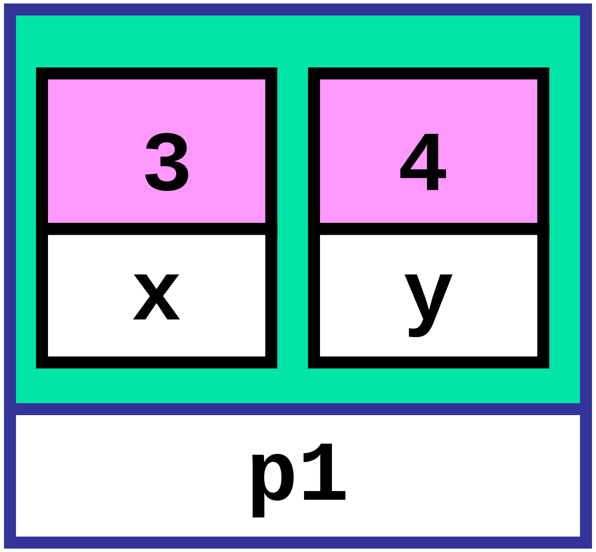
```
p1 = struct('x',3,'y',4);

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.

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.

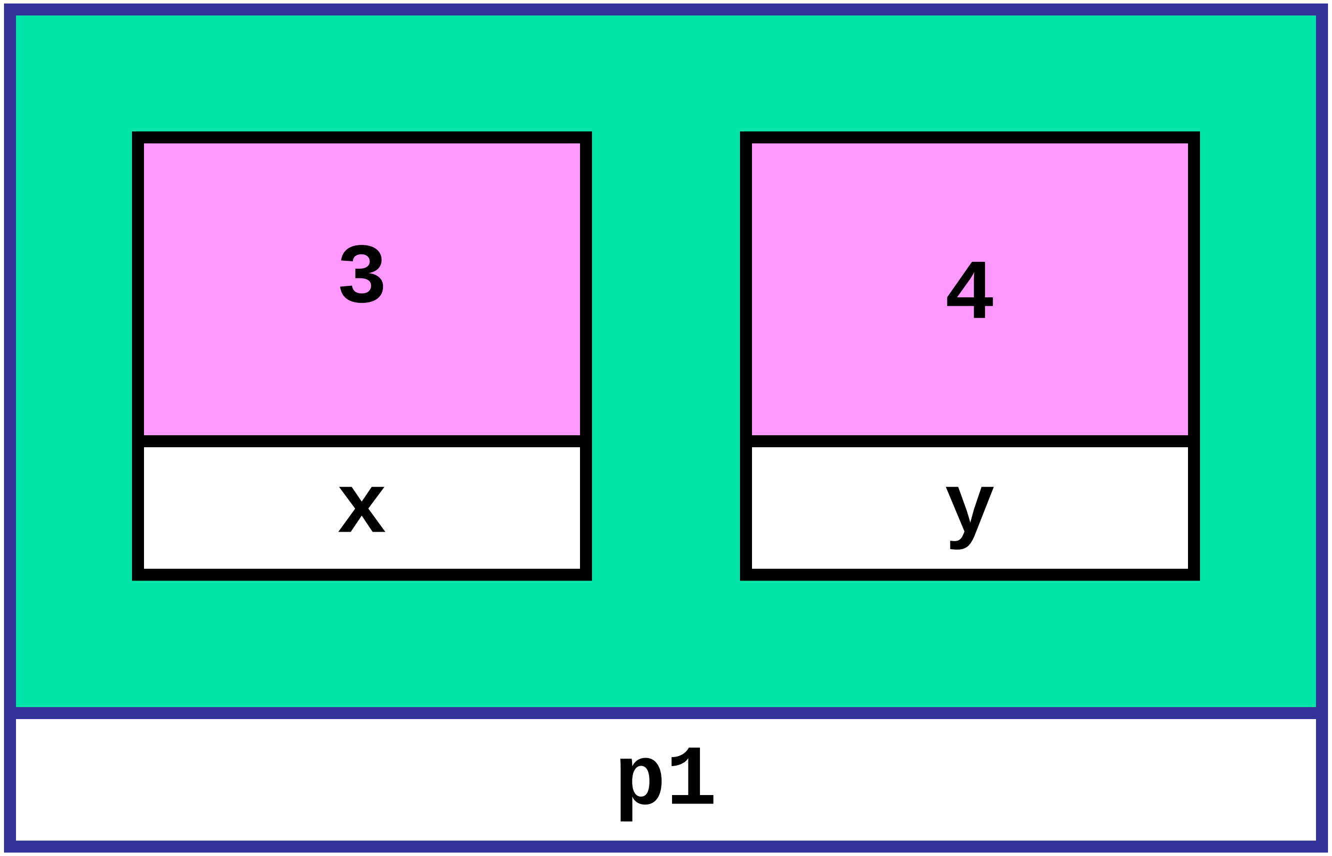
p1.x, p1.y, p2.x, p2.y participating as variables—because they are.

Creating a structure (by direct assignment)

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

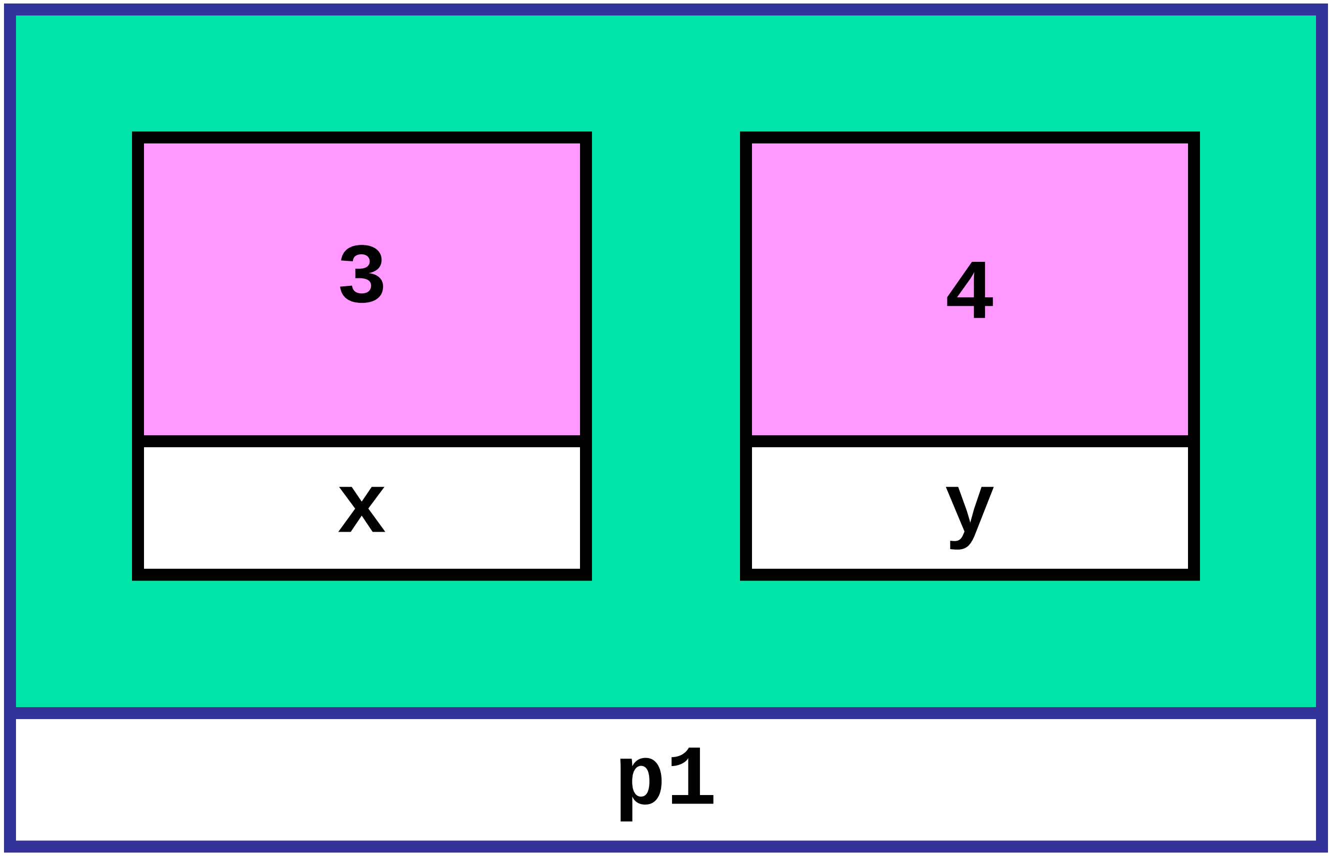
`p1` is a structure.
The structure has two fields.
Their names are **`x`** and **`y`**.
They are assigned the values 3 and 4.

How to visualize structure **p1**




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

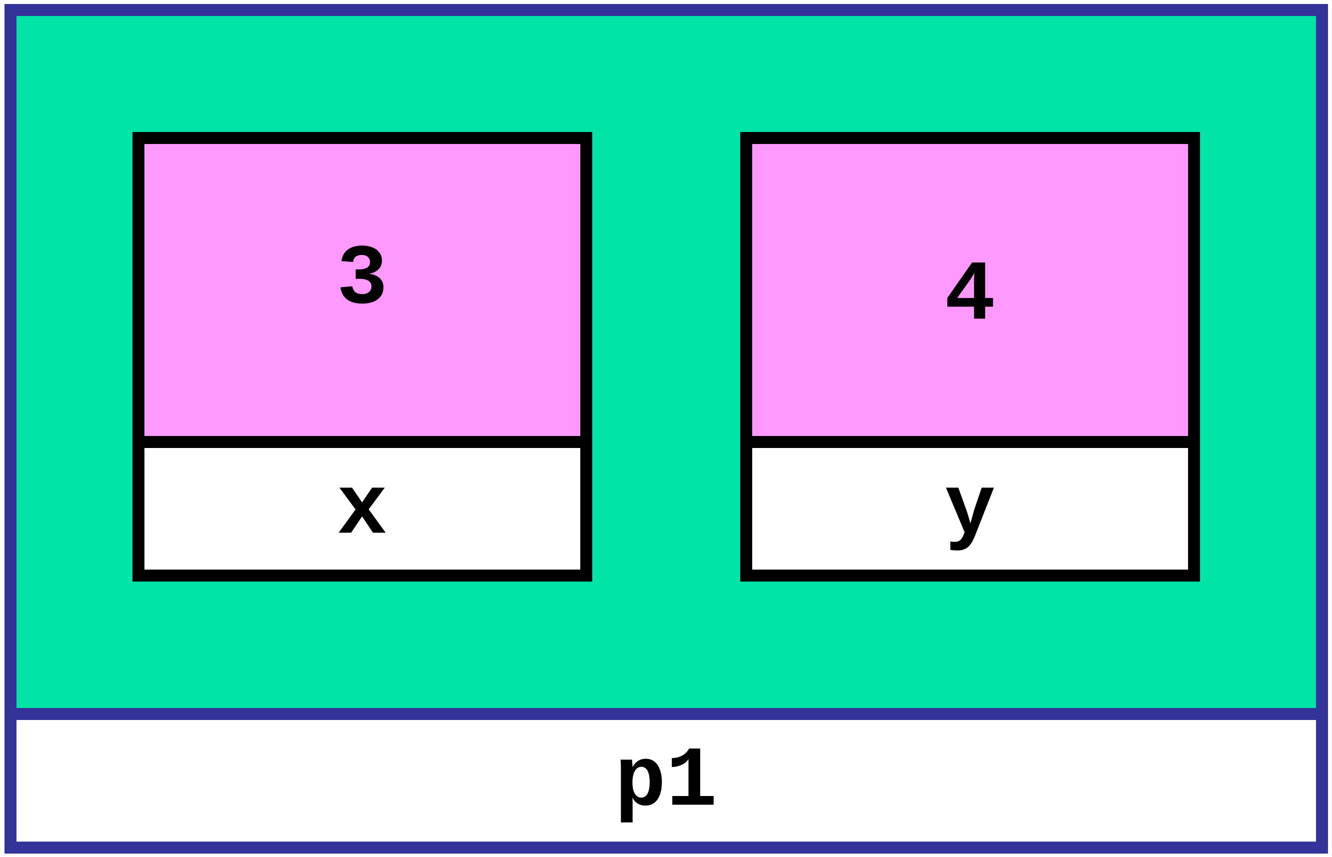
Accessing the fields in a structure




```
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);
```

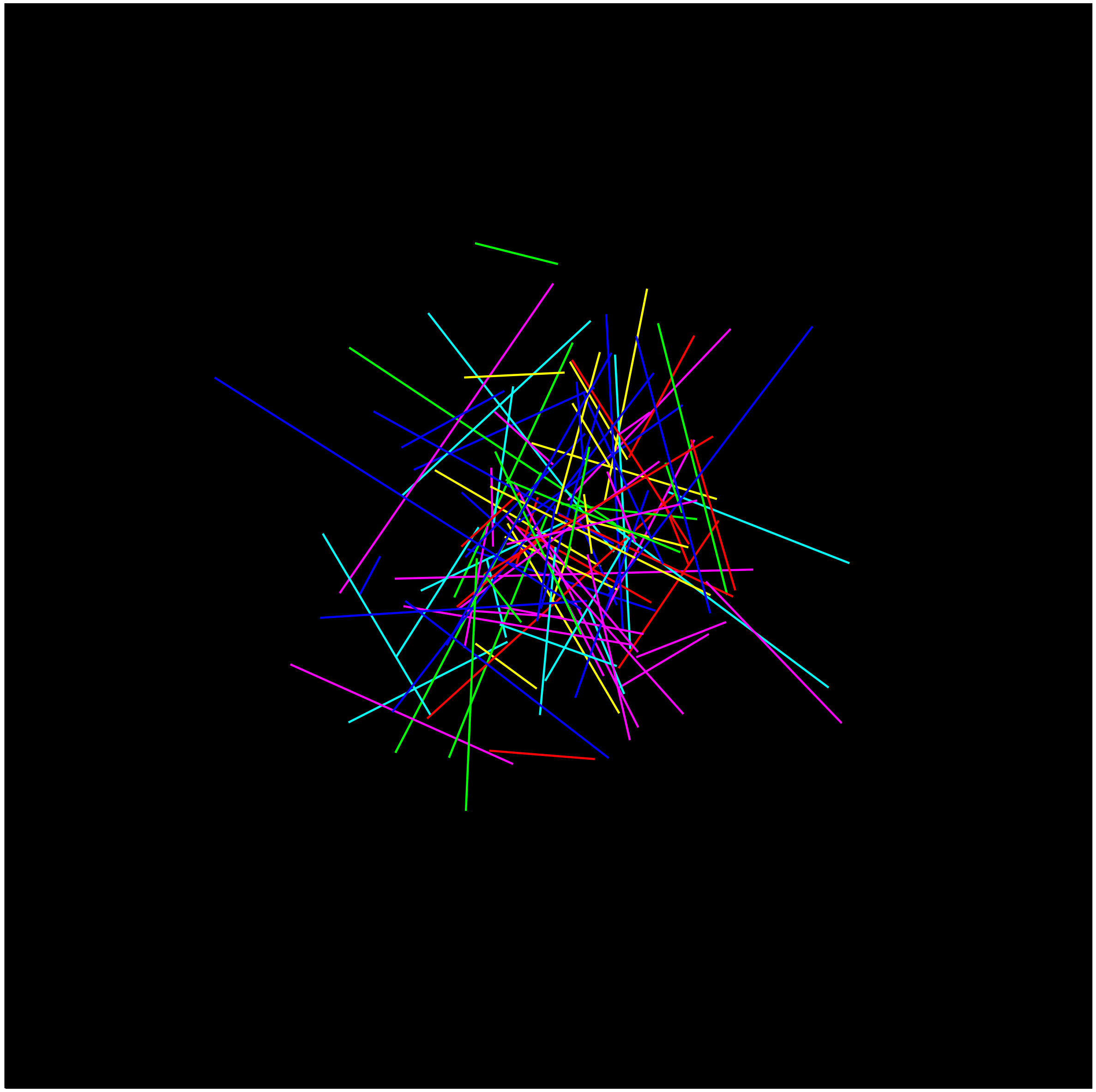
Sample “Make” Function

```
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);
```

Good Style—highlights the structure's definition

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)
```



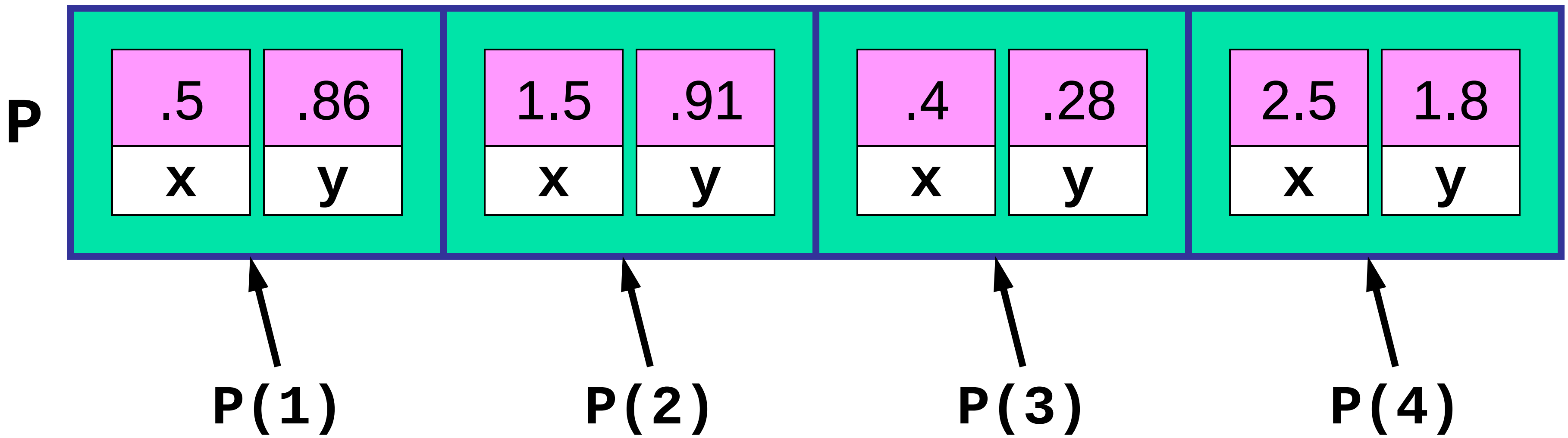
Pick Up Sticks

```
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
```

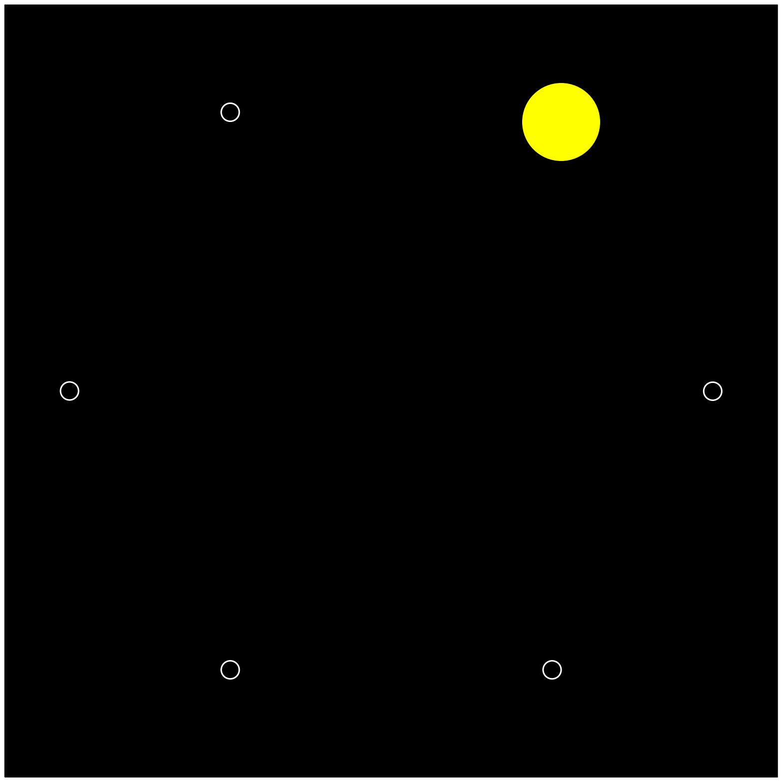
Generates two random points and chooses one of six colors randomly.

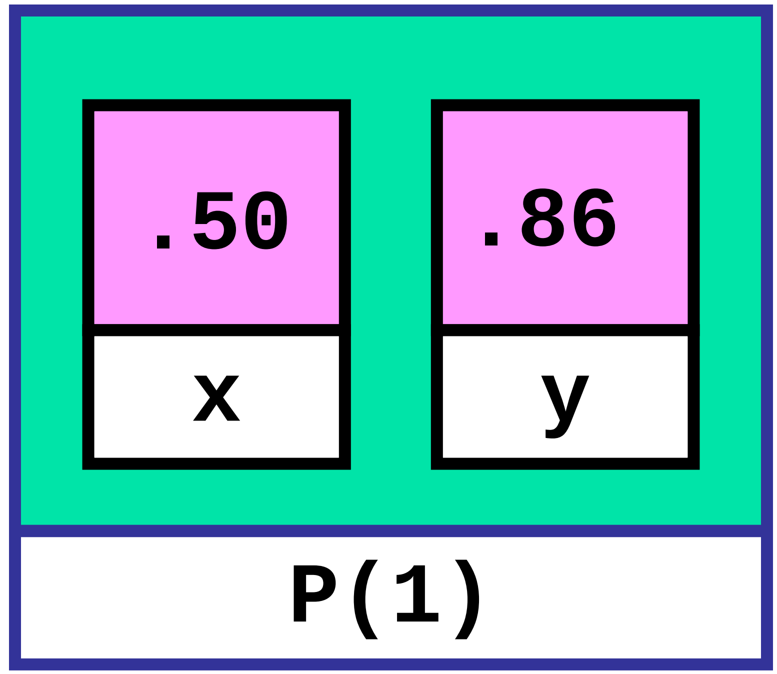
Structure Arrays

- 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)



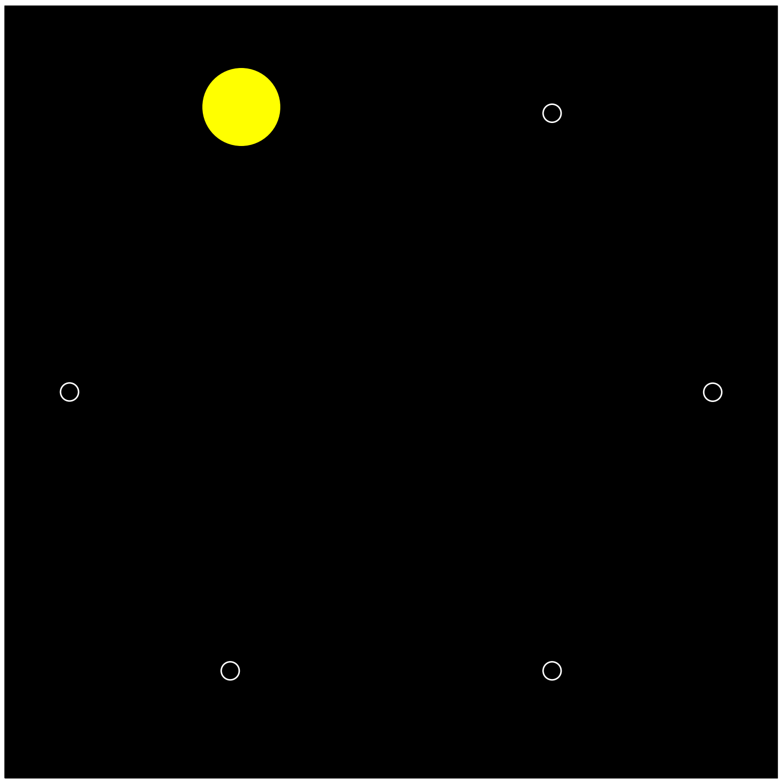
An Array of Points

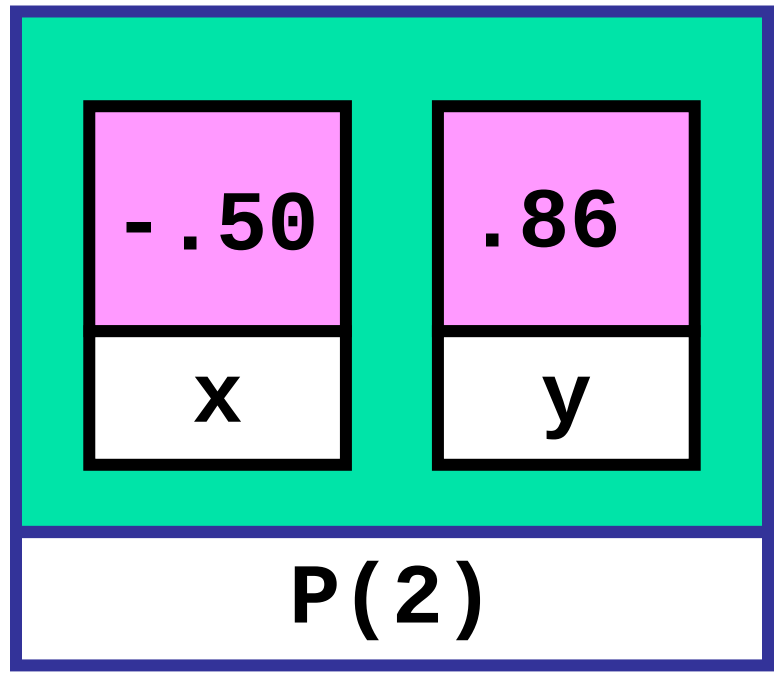




`P(1) = MakePoint(.50,.86)`

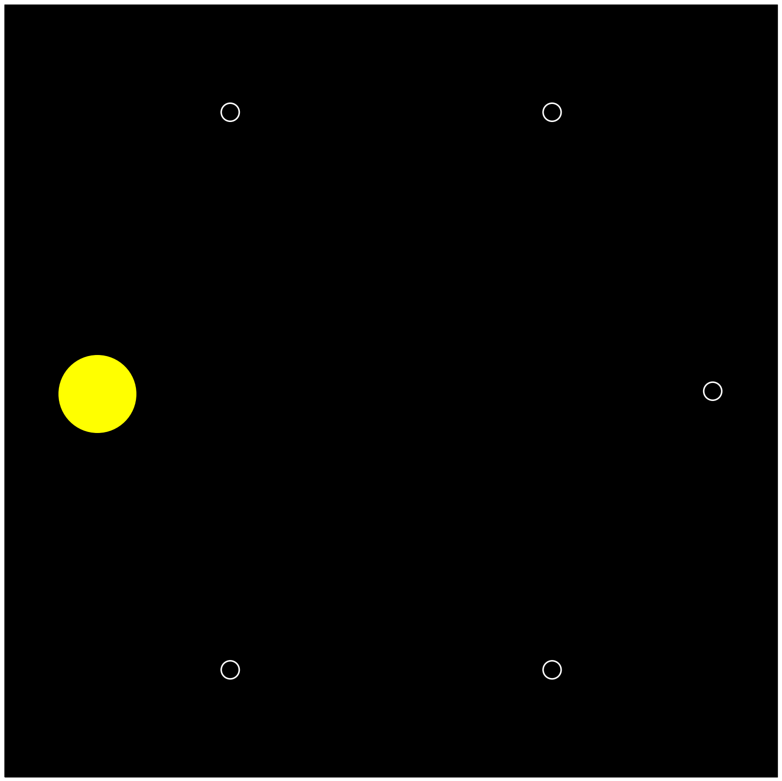
An Array of Points

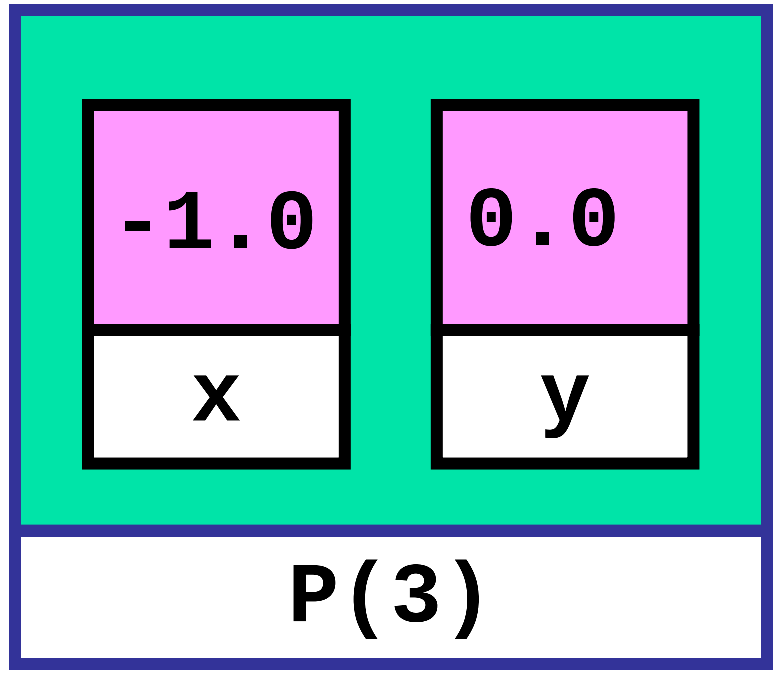





```
P(2) = MakePoint(-.50,.86)
```

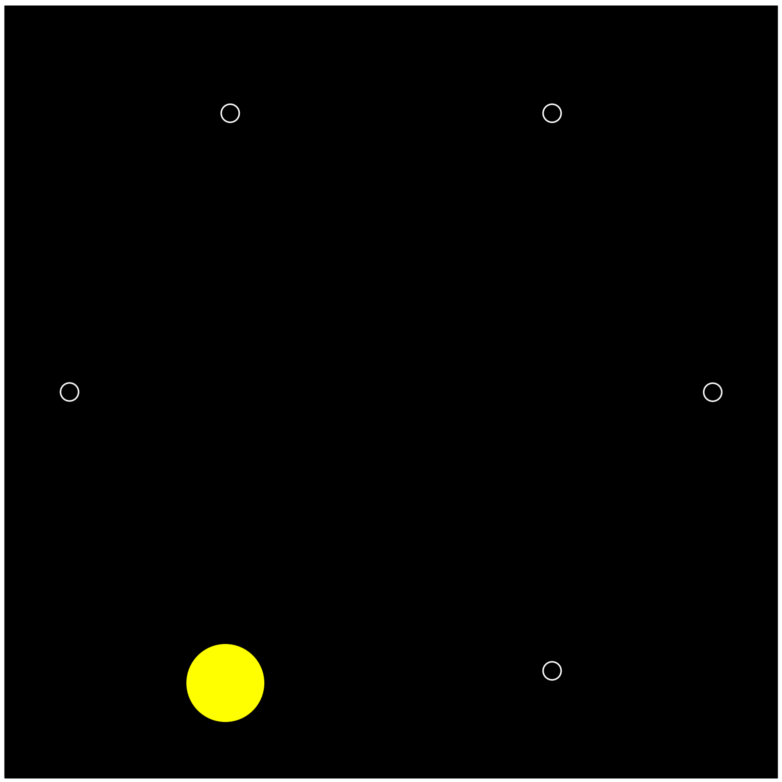
An Array of Points

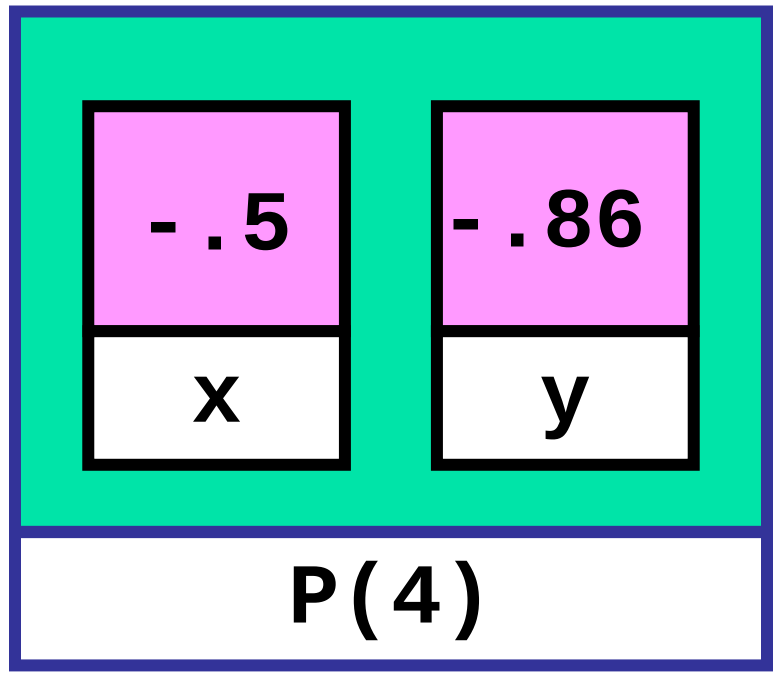




P(3) = MakePoint(-1.0,0.0)

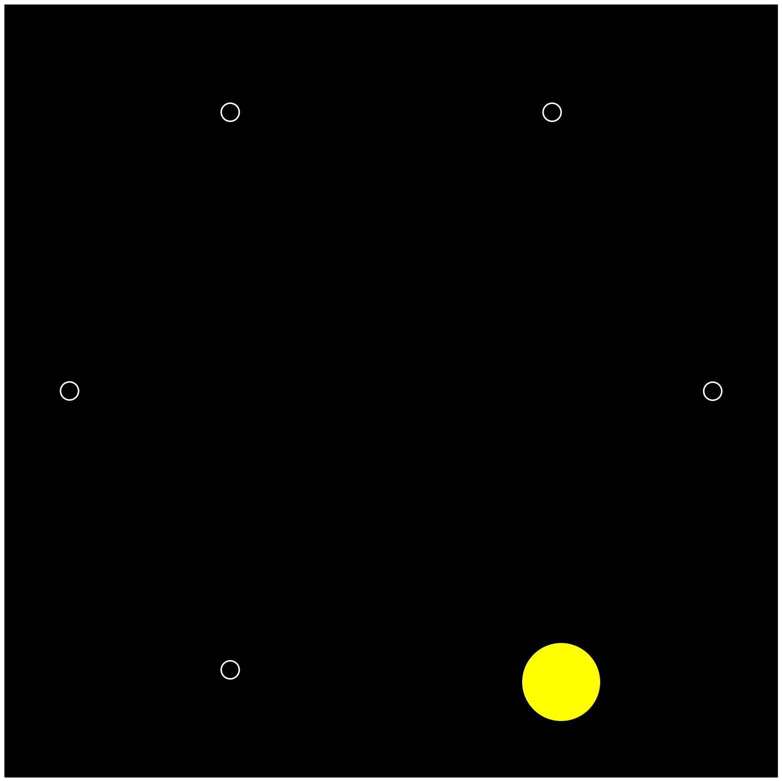
An Array of Points

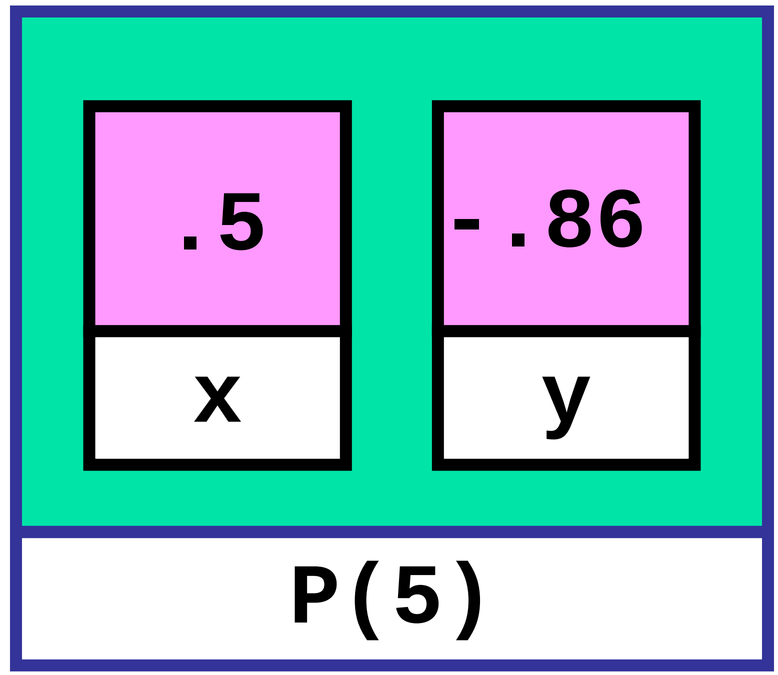




`P(4) = MakePoint(-.50,-.86)`

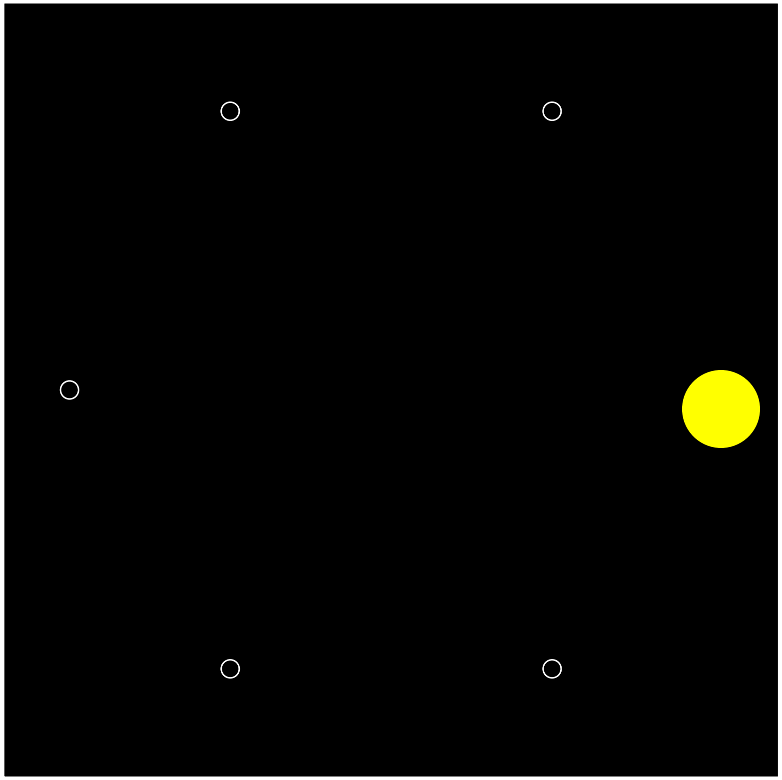
An Array of Points

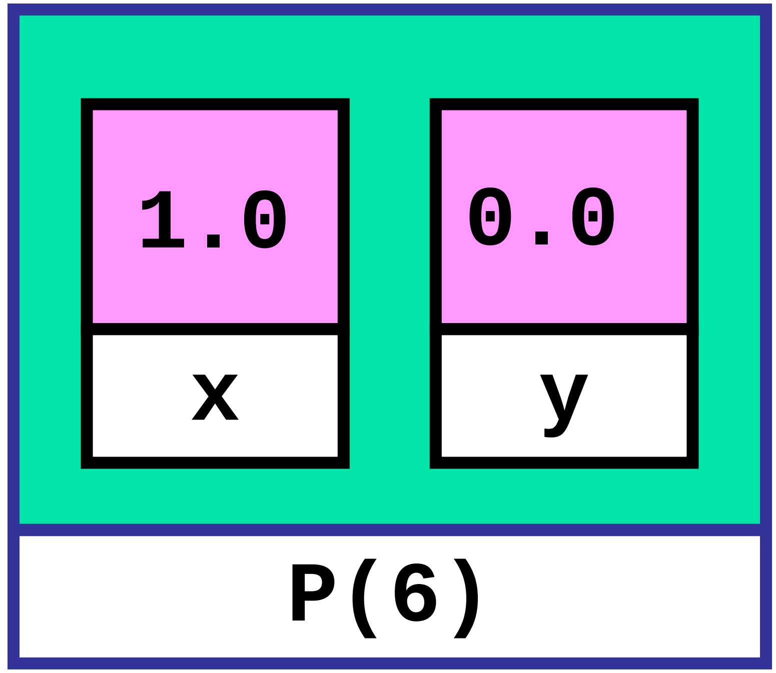




P(5) = MakePoint(.50,-.86)

An Array of Points






```
P(6) = MakePoint(1.0,0.0)
```

Function returning an array of points (point structures)

```
function P = CirclePoints(n)

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