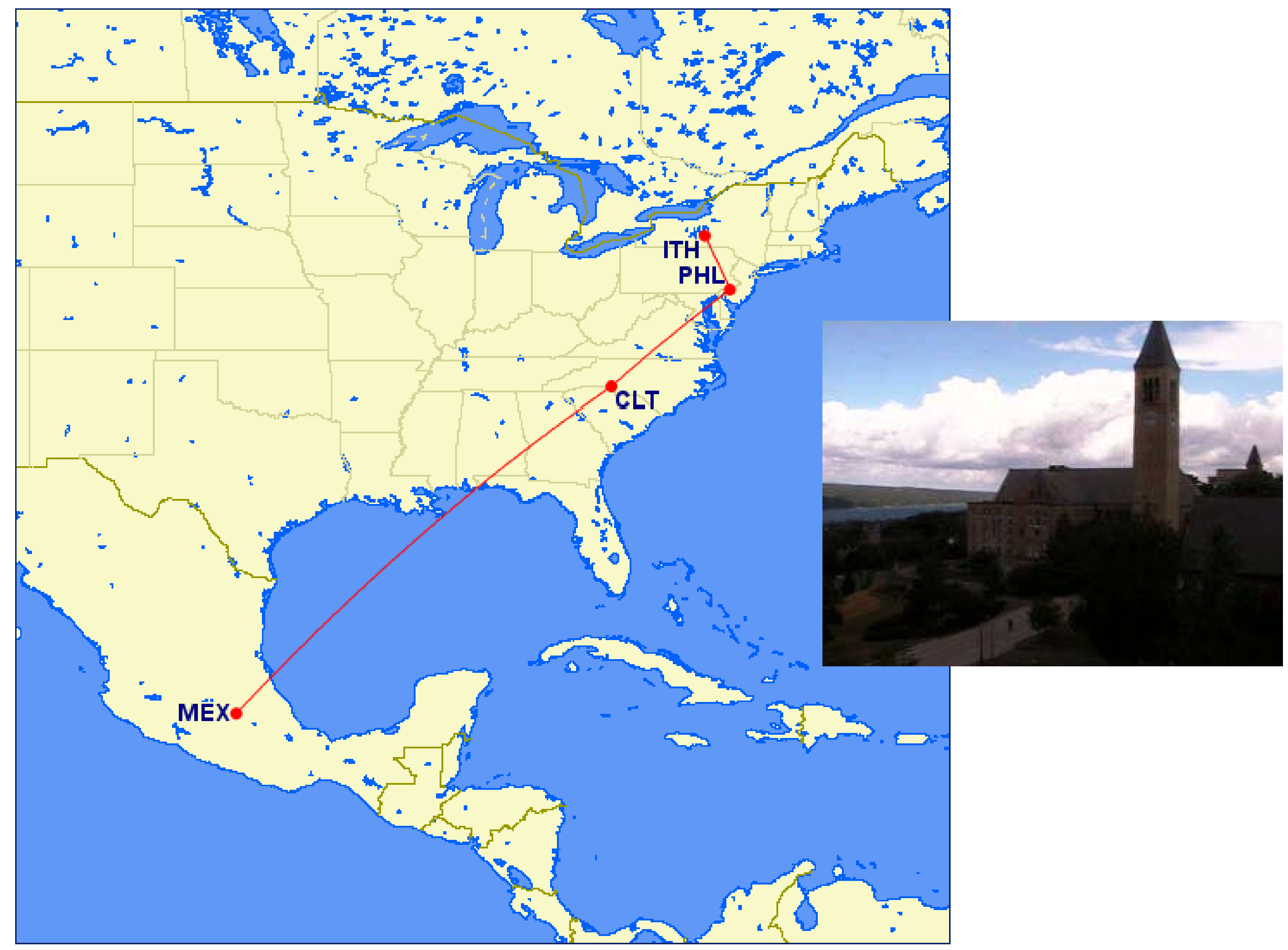
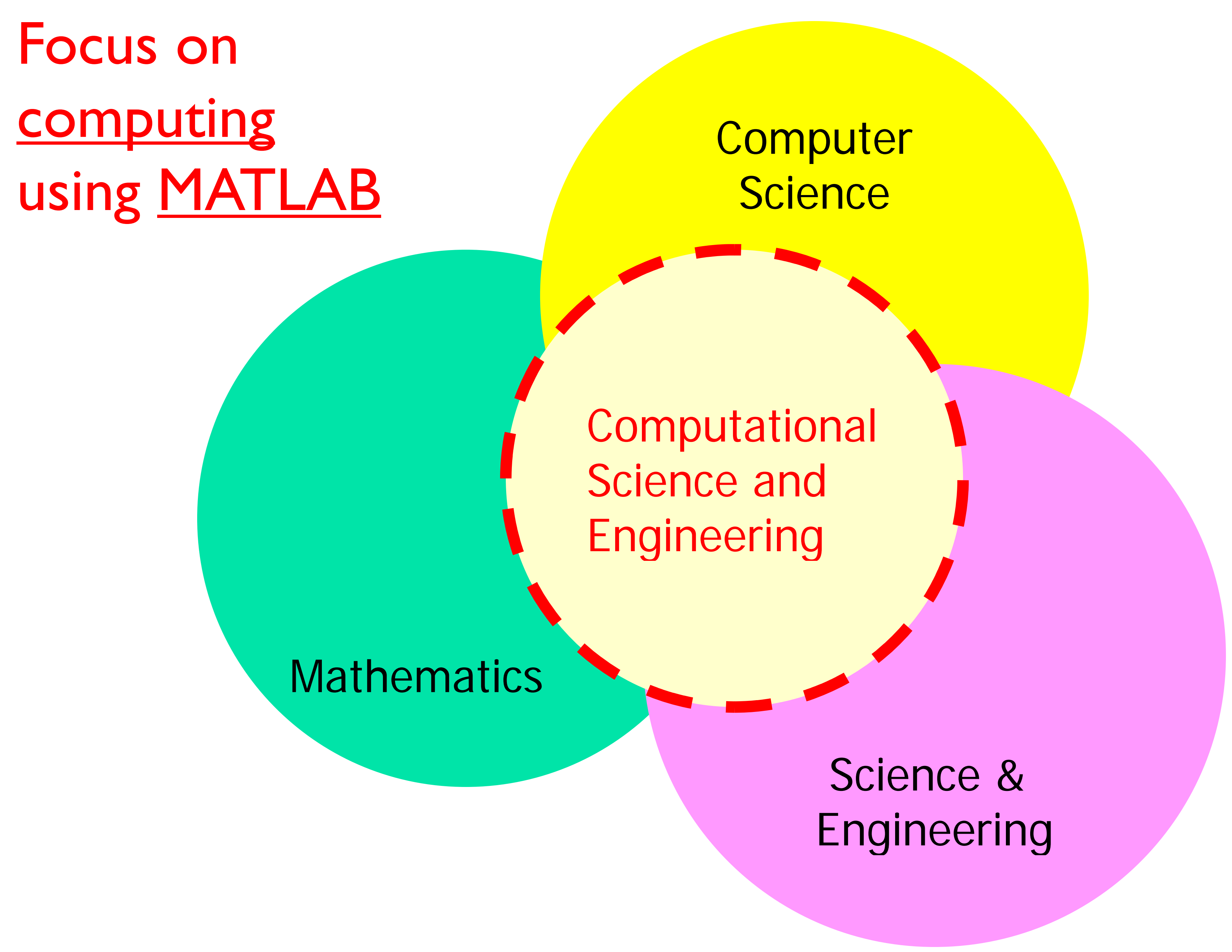
Scientific Computing with MATLAB

Dra. K.-Y. Daisy Fan
Department of Computer Science
Cornell University
Ithaca, NY, USA

UNAM IIM 2012


ITH
PHL
CLT
MEX

Focus on computing using MATLAB

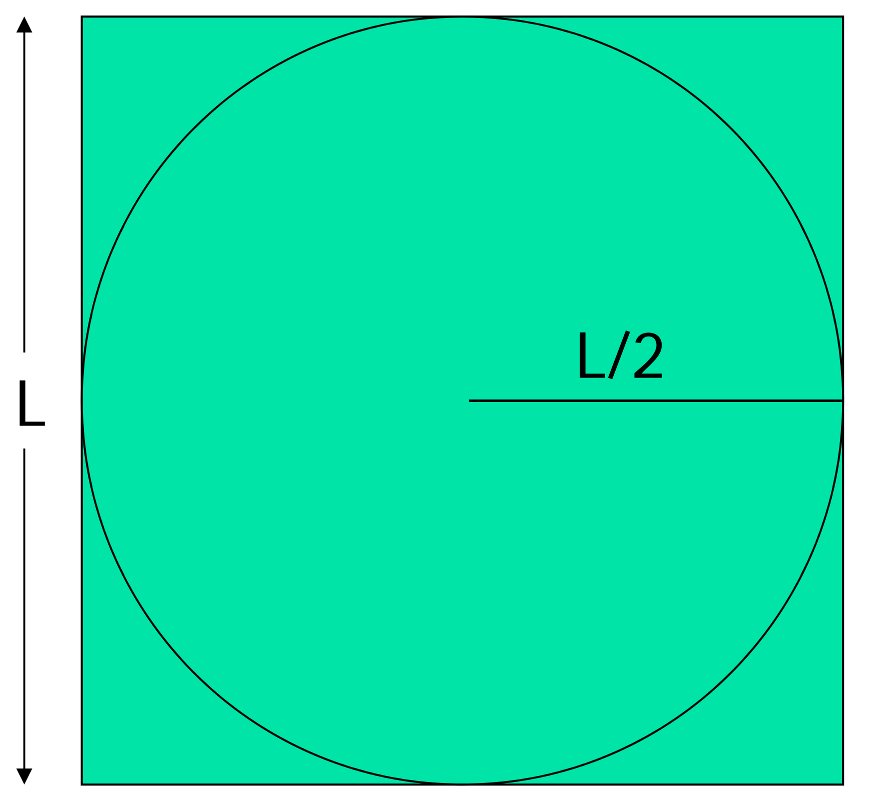


Scientific Computing with MATLAB

1. Programming Fundamentals
   - Functions, control flow, arrays and vectorized computation
2. Graphics for Research and Publication
3. Manipulating Data Sets
   - Working with external data files, including numeric, image, and text
4. Numerical Simulation
   - Fundamental ideas, probability, and numerical methods
5. Specialized Topics
   - Advanced data objects in MATLAB, …

MATLAB desktop environment

Monte Carlo Approximation of π



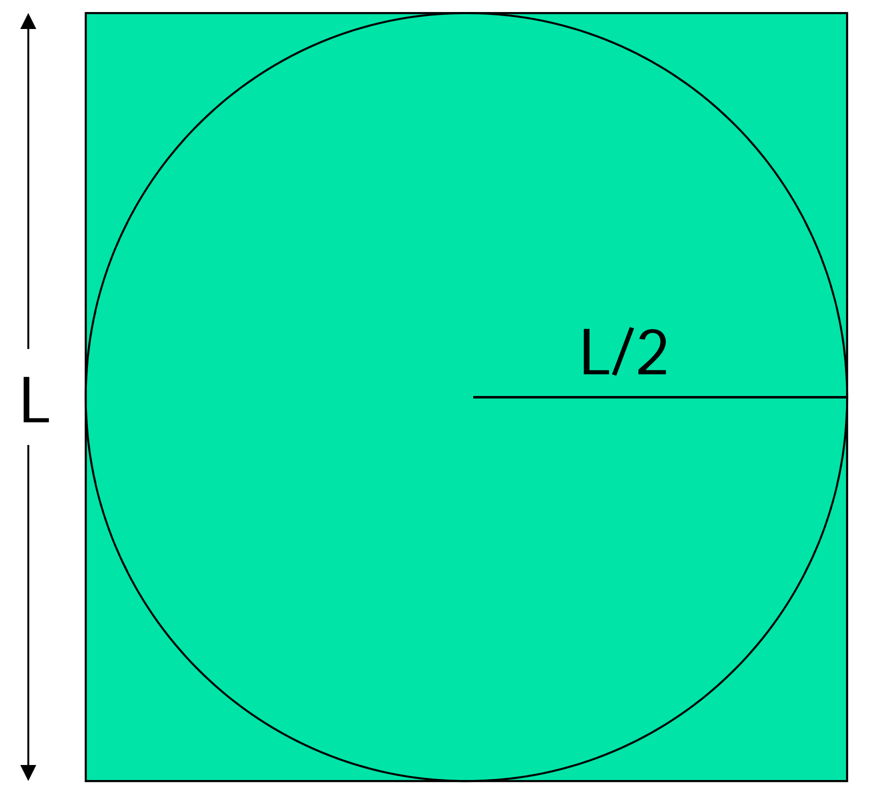
Throw N darts

Sq. area $= \boldsymbol{N} = \boldsymbol{L} \times \boldsymbol{L}$

Circle area $= \boldsymbol{N_{in}}$

$= \pi \boldsymbol{L^2/4}$

Monte Carlo Approximation of π



Throw N darts

Sq. area = $\boldsymbol{N} = \boldsymbol{L} \times \boldsymbol{L}$

Circle area = $\boldsymbol{N_{in}}$

$= \pi \boldsymbol{L^2/4}$

$\pi = \mathbf{4}\ \boldsymbol{N_{in} / N}$

Monte Carlo Approximation of π

For each of N trials

Throw a dart

If it lands in circle

add 1 to total # of hits

Pi is 4*hits/N

Monte Carlo π with N darts on L-by-L board

```
N= ____;
for k = 1:N




end
myPi = 4*hits/N;
```

Monte Carlo π with N darts on L-by-L board

```
N= ____;
for k = 1:N
    % Throw kth dart



    % Count it if it is in the circle




end
myPi = 4*hits/N;
```

Monte Carlo π with N darts on L-by-L board

```
N= ____;  L= ____;
for k = 1:N
    % Throw kth dart
    x = rand(1)*L - L/2;
    y = rand(1)*L - L/2;
    % Count it if it is in the circle



end
myPi = 4*hits/N;
```

Monte Carlo π with N darts on L-by-L board

```
N= ___;  L= ___;  hits= 0;
for k = 1:N
   % Throw kth dart
   x = rand(1)*L - L/2;
   y = rand(1)*L - L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
end
myPi = 4*hits/N;
```

Input & output

- *variable* = **input** ('*prompt* ')

```
N= input('How many darts? ')
```

- **fprintf** ('*message to print* ')

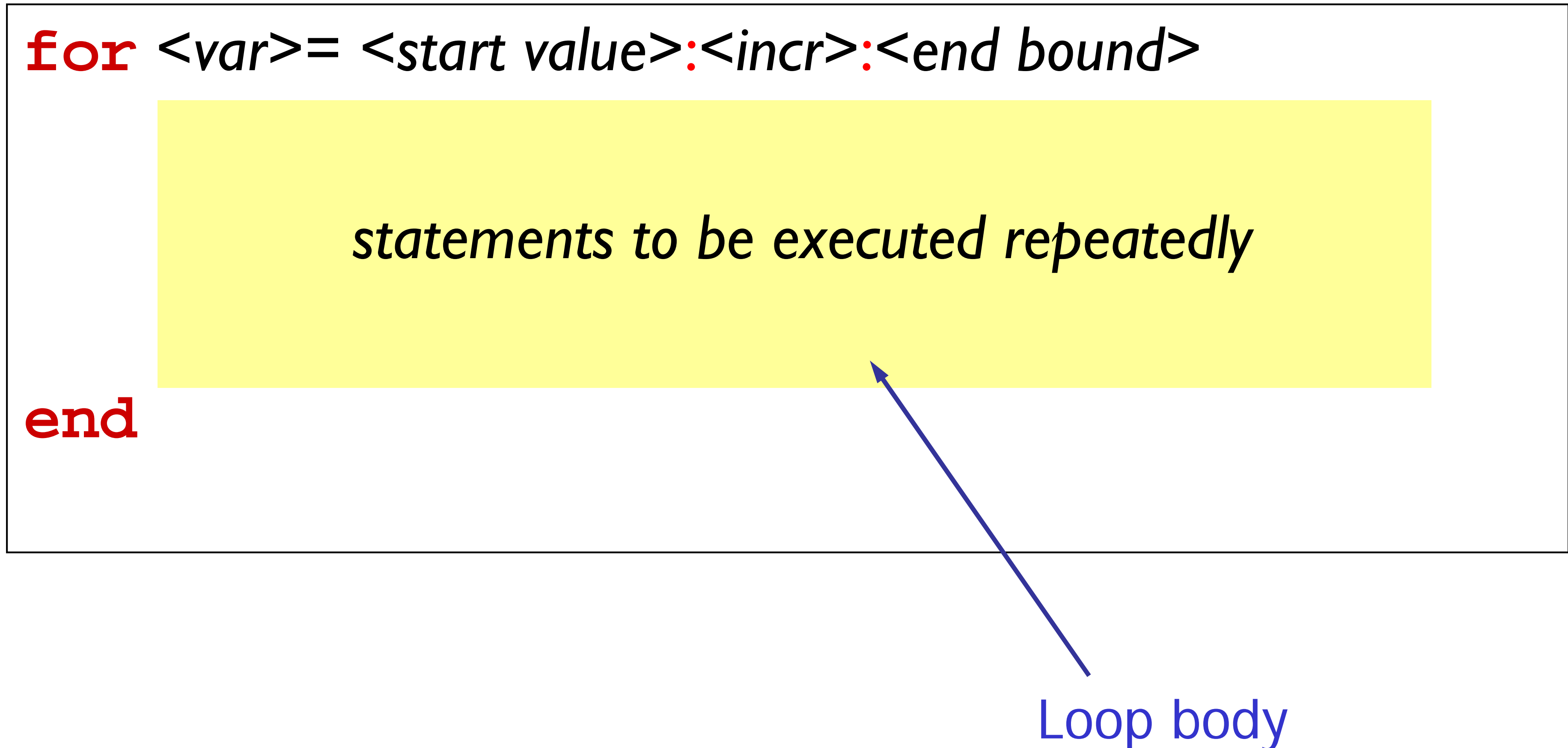
```
fprintf('My pi ')
fprintf('is %f\n', p)
fprintf('Error is %f, %d darts\n', e,N)
```

Substitution sequences (conversion specifications)

`%f`	fixed point (or floating point)
`%d`	decimal—whole number
`%e`	exponential
`%g`	general—Matlab chooses a format
`%c`	character
`%s`	string

Examples: `%f` `%15.2f`

Syntax of the `for` loop



Syntax of the `for` loop

```
for <var>= <start value>:<incr>:<end bound>

        statements to be executed repeatedly

end
```

Loop header specifies all the values that the index variable will take on, one for each pass of the loop.

E.g, `k= 3:1:7` means `k` will take on the values 3, 4, 5, 6, 7, one at a time.

`for` loop examples

```
for k= 2:0.5:3
     disp(k)
end
```

`k` takes on the values ____________

Non-integer increment is OK

```
for k= 1:4
     disp(k)
end
```

`k` takes on the values ____________

Default increment is 1

```
for k= 0:-2:-6
     disp(k)
end
```

`k` takes on the values ____________

"Increment" may be negative

```
for k= 0:-2:-7
     disp(k)
end
```

`k` takes on the values ____________

Colon expression specifies a *bound*

```
for k= 5:2:1
     disp(k)
end
```

`for` loop examples

```
for k= 2:0.5:3
     disp(k)
end
```

`k` takes on the values 2,2.5,3

Non-integer increment is OK

```
for k= 1:4
     disp(k)
end
```

`k` takes on the values 1,2,3,4

Default increment is 1

```
for k= 0:-2:-6
     disp(k)
end
```

`k` takes on the values 0,-2,-4,-6

"Increment" may be negative

```
for k= 0:-2:-7
     disp(k)
end
```

`k` takes on the values 0,-2,-4,-6

Colon expression specifies a *bound*

```
for k= 5:2:1
     disp(k)
end
```

The set of values for k is the empty set: the loop body won't execute

The **if** construct

if *boolean expression1*

statements to execute if *expression1* *is true*

elseif *boolean expression2*

statements to execute if *expression1* *is false*

but *expression2* *is true*

:

else

statements to execute if all previous conditions

are false

end

Can have any number of elseif branches but at most one else branch

Relational operators

Operator	Meaning
`<`	Less than
`>`	Greater than
`<=`	Less than or equal to
`>=`	Greater than or equal to
`==`	Equal to
`~=`	Not equal to

“Truth table”

X, Y represent boolean expressions.
E.g., d>3.14

X	Y	X && Y “and”	X || Y “or”	~y “not”
F	F	F	F	T
F	T	F	T	F
T	F	F	T	T
T	T	T	T	F

"Truth table"

Matlab uses **0** to represent false,
1 to represent true

X	Y	X && Y "and"	X || Y "or"	~y "not"
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	1	0

Logical operators “short-circuit”

a > b && c > d

true

Go on

a > b && c > d

false

Stop

Entire expression is false since the first part is false

A **&&** condition short-circuits to false if the left operand evaluates to *false*.

A **||** condition short-circuits to true if the left operand evaluates to *true*.

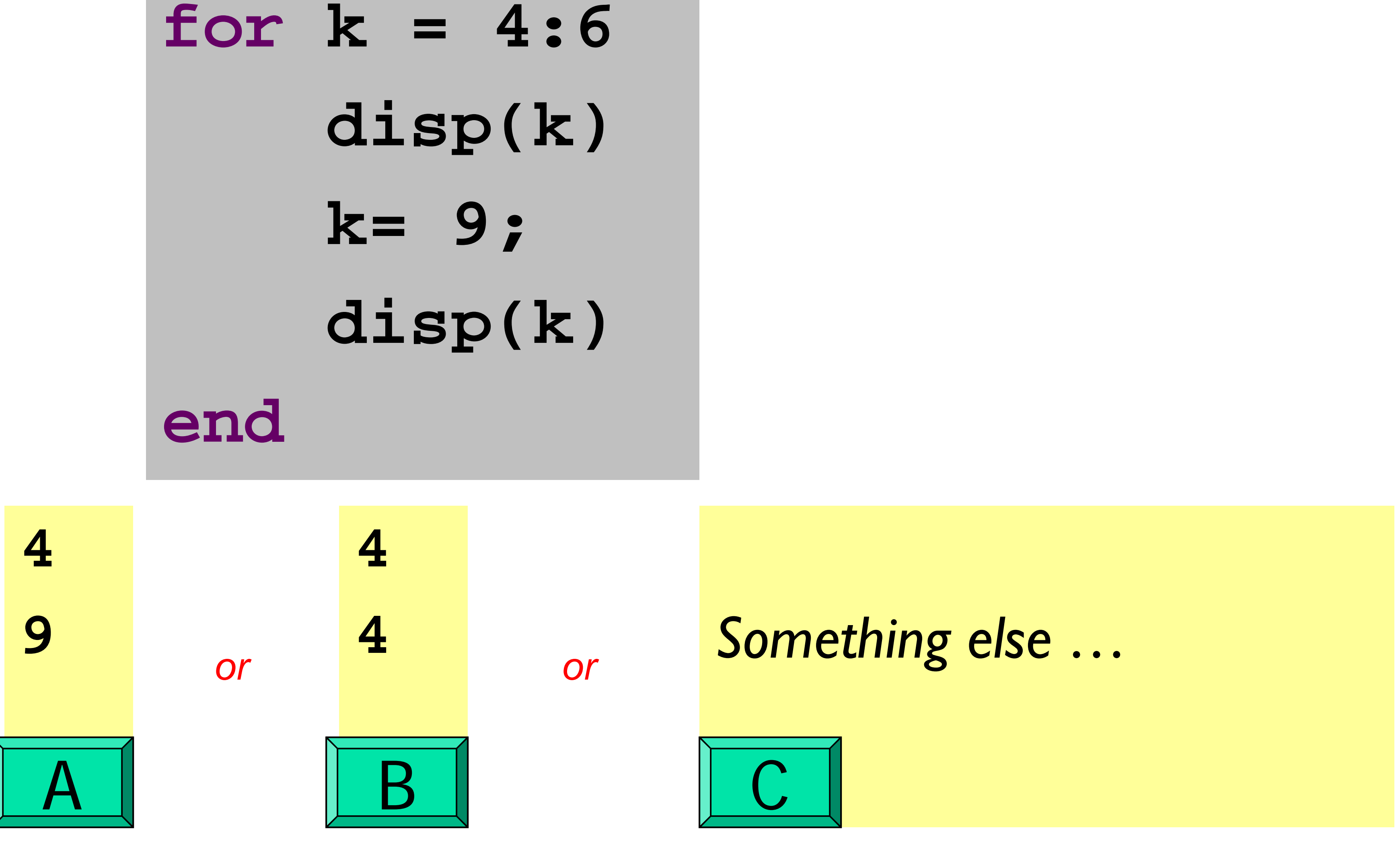
Exercise: Print a table of function values

$$f(x) = \begin{cases} \sin(x) & x < 0 \\ x^2 - 2x & 0 \leq x < 2 \\ x^3/4 - x & x \geq 2 \end{cases}$$

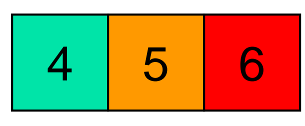
Solicit a range of x values and the step size from the user and print the values of $f(x)$. Example run:

```
Enter smallest value of x: -1
Enter largest value of x: 4
Enter the step size:
```

What will be displayed when you run the following script?

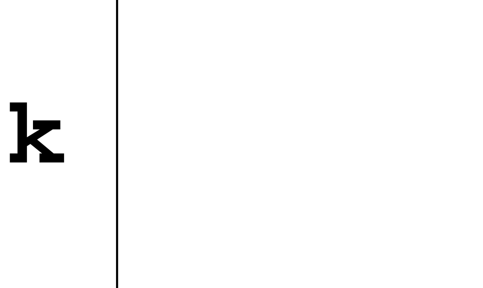


```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

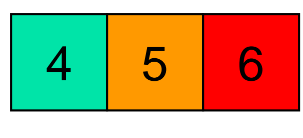


With this loop header, **k** "promises" to be these values, one at a time

Output in Command Window



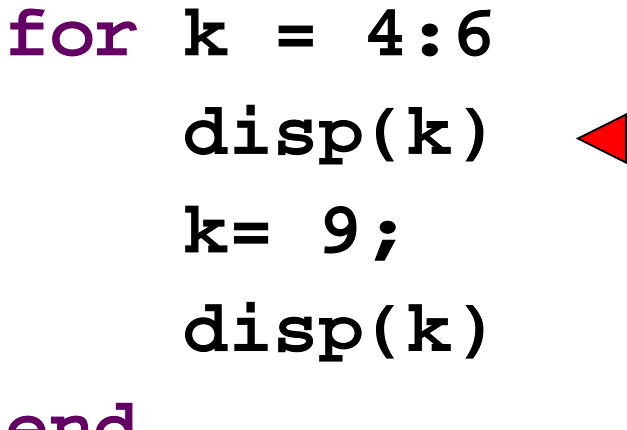
```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

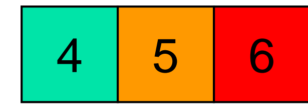


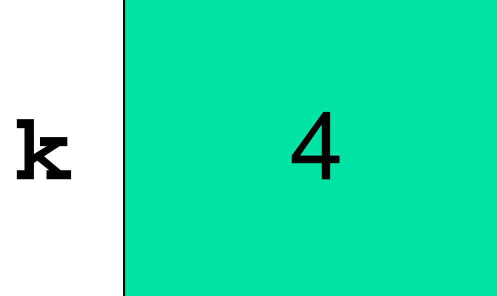
With this loop header, **k** "promises" to be these values, one at a time

Output in Command Window

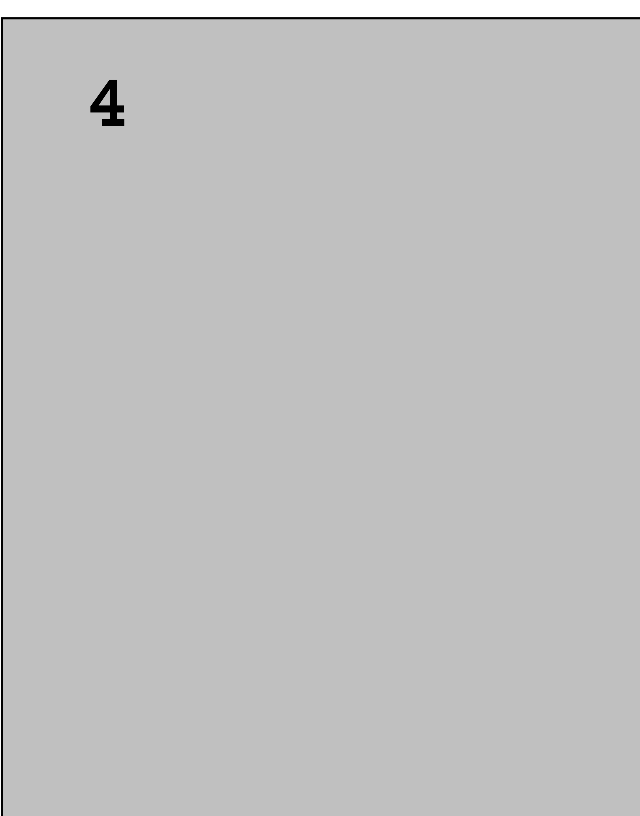
k 4

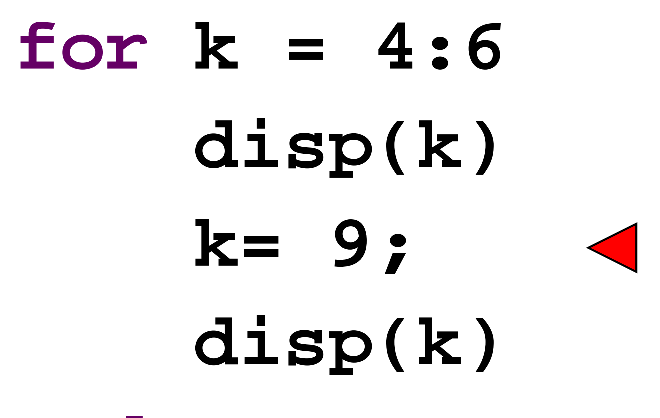


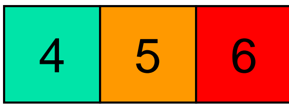


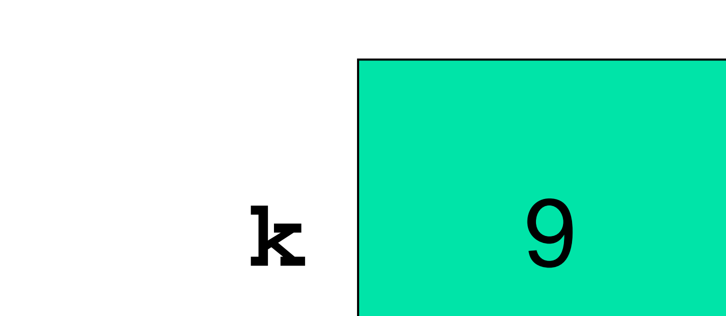


Output in Command Window

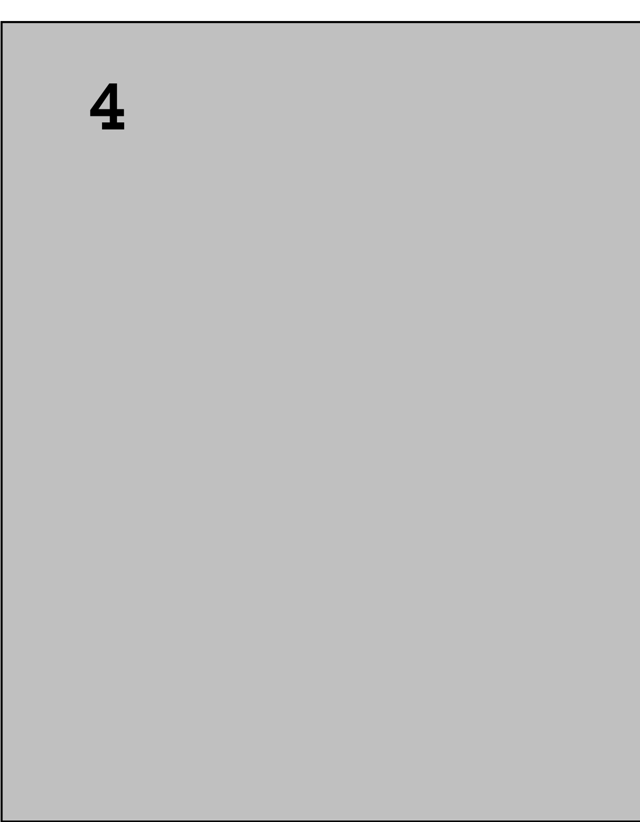


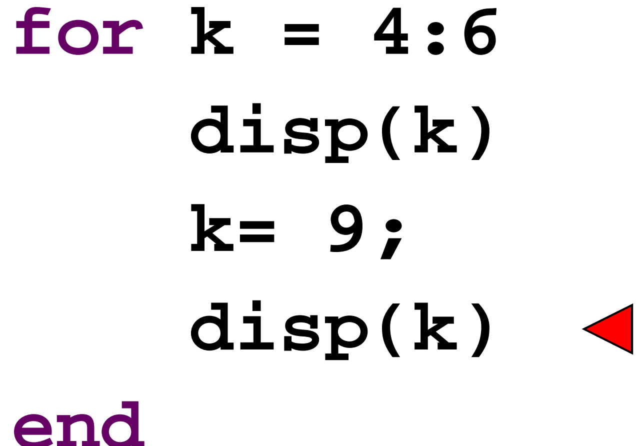


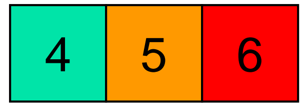




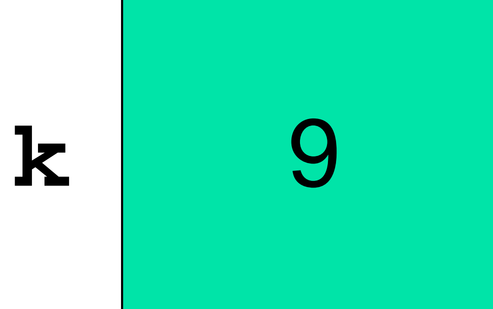
Output in Command Window

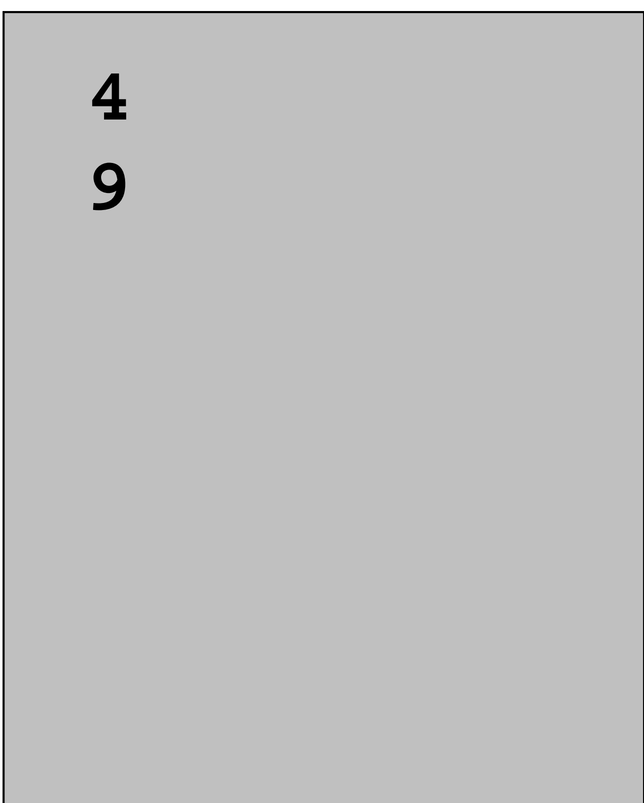


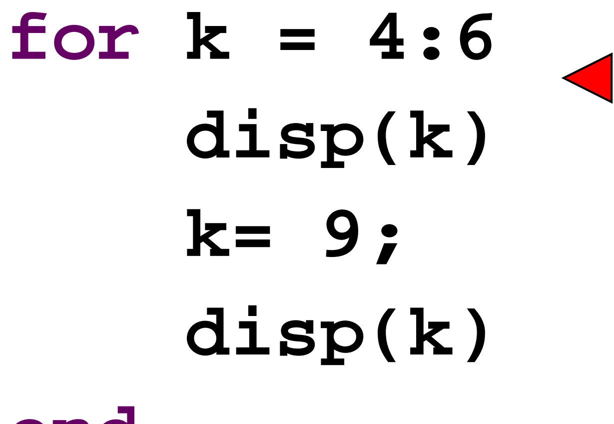


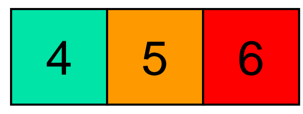


Output in Command Window

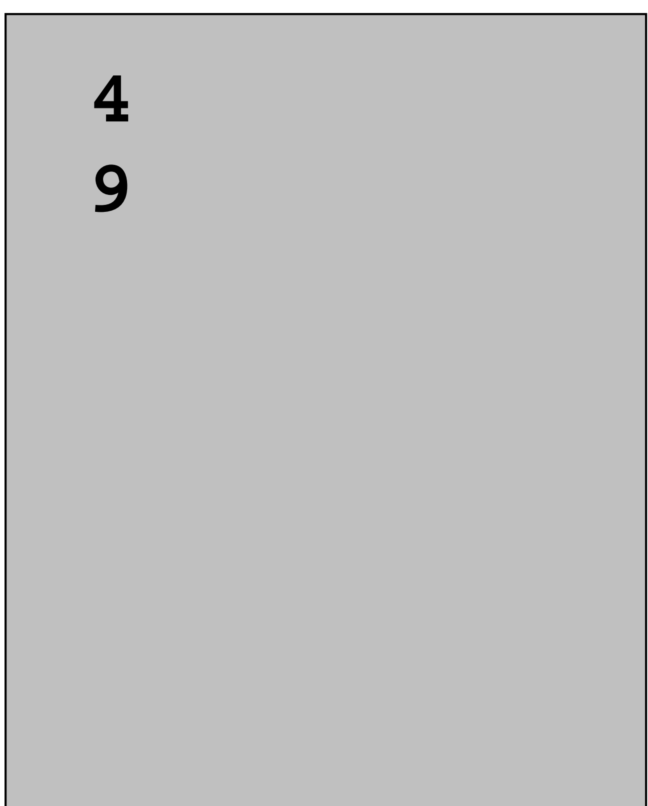


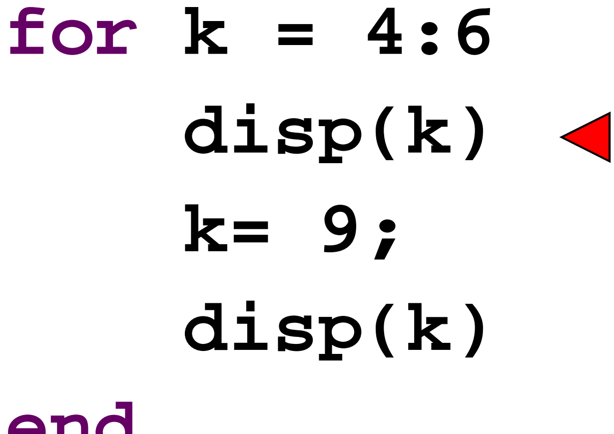


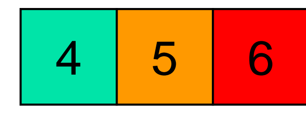




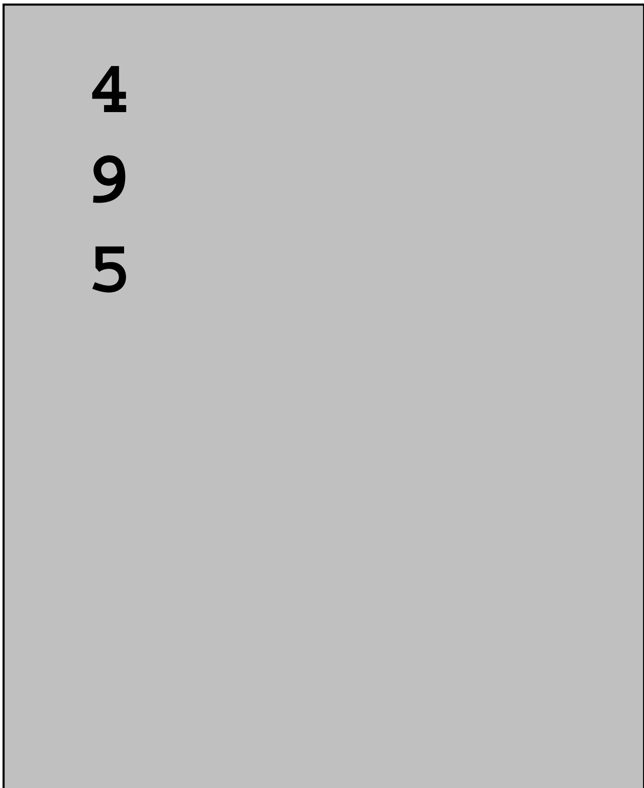
Output in Command Window

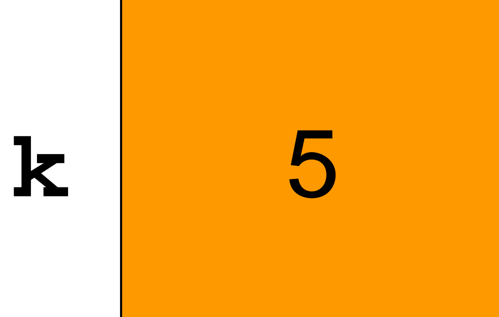


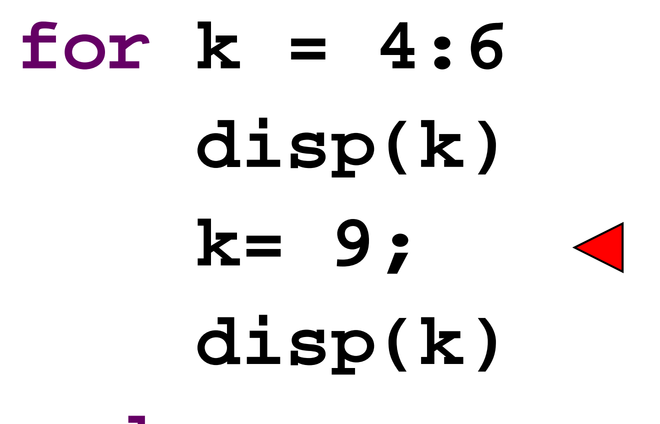


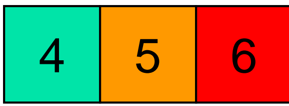


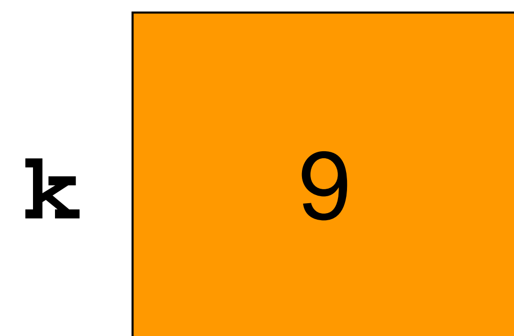
Output in Command Window



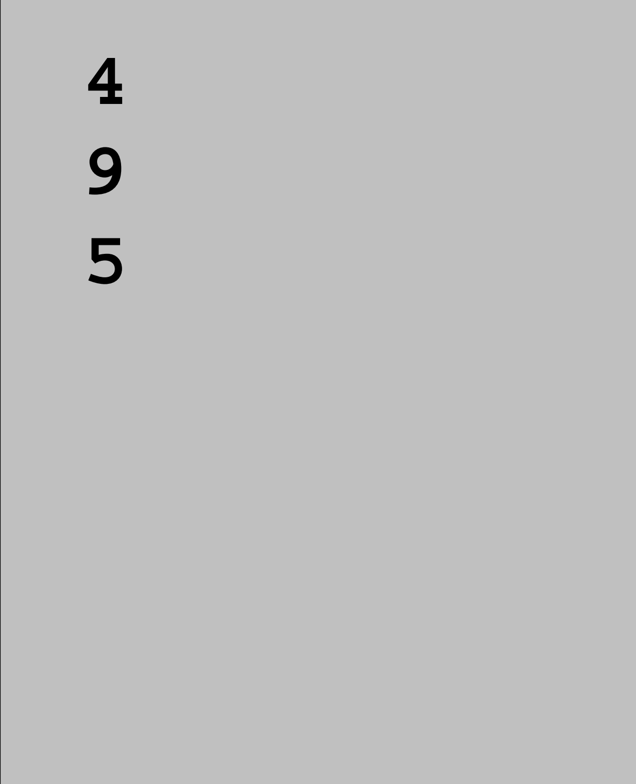


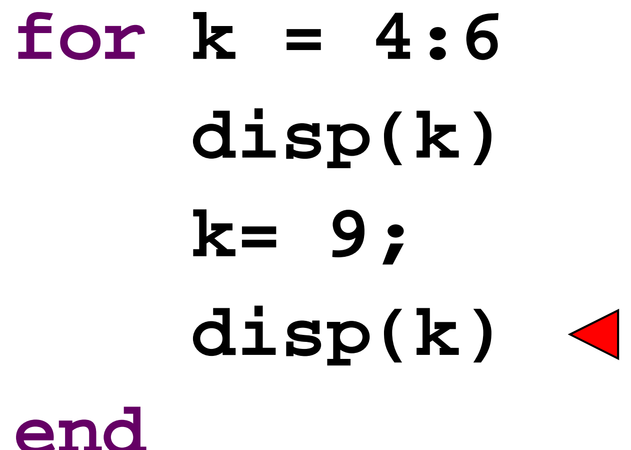


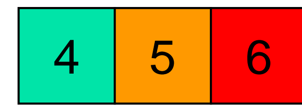




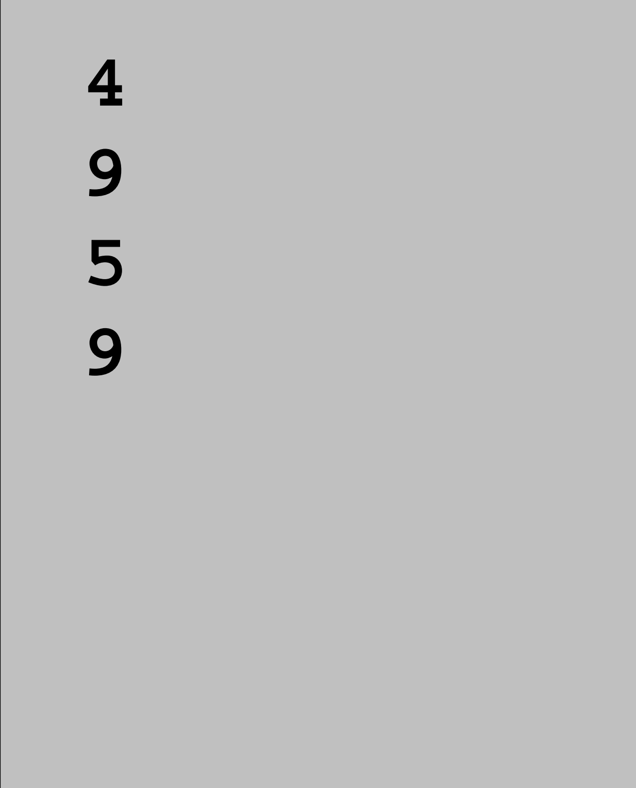
Output in Command Window

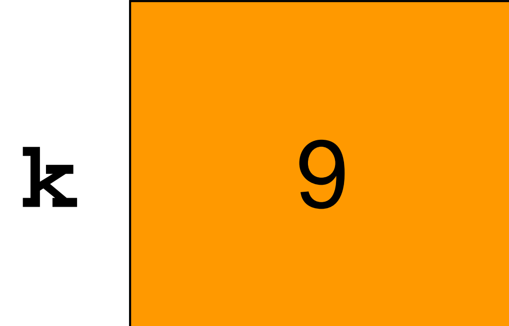


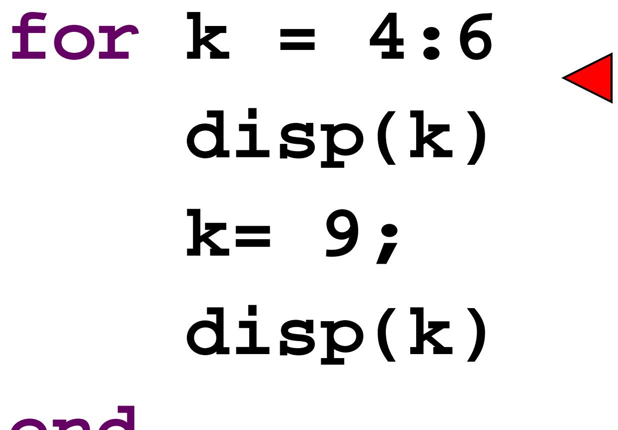


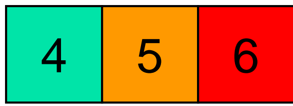


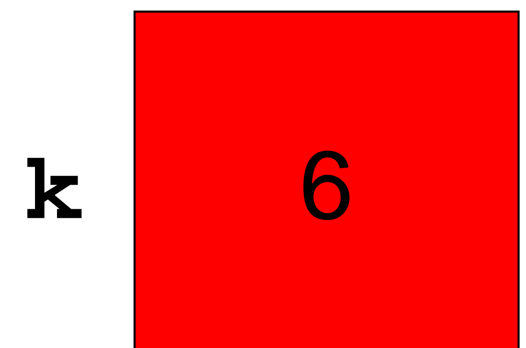
Output in Command Window



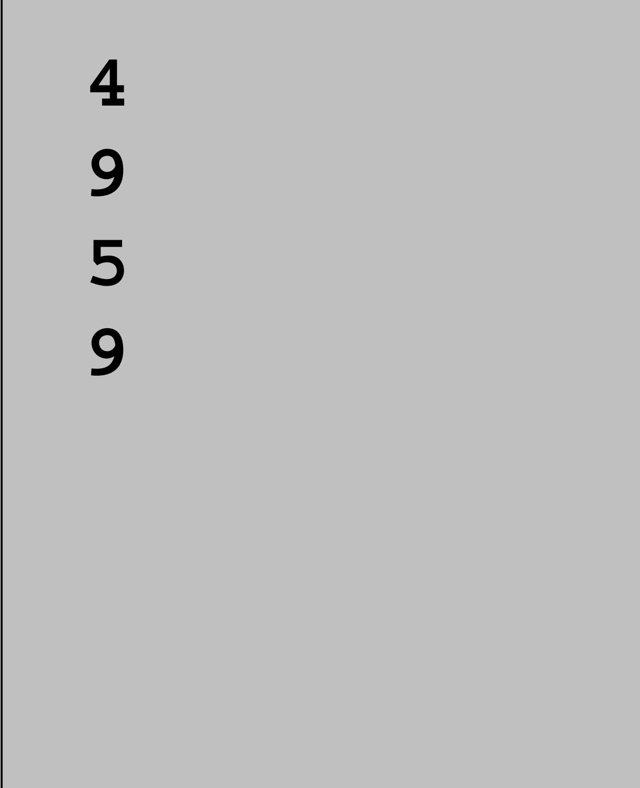


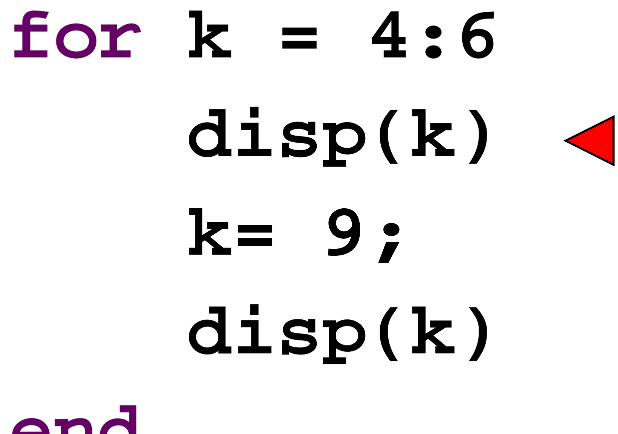


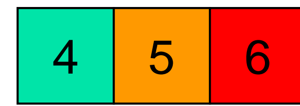


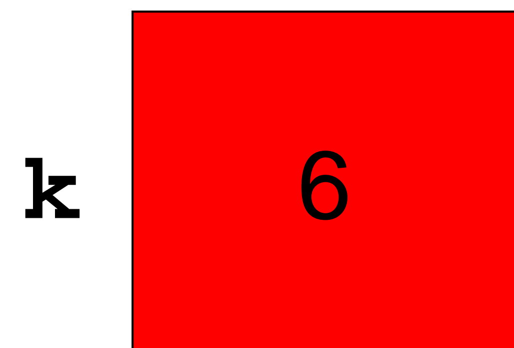


Output in Command Window

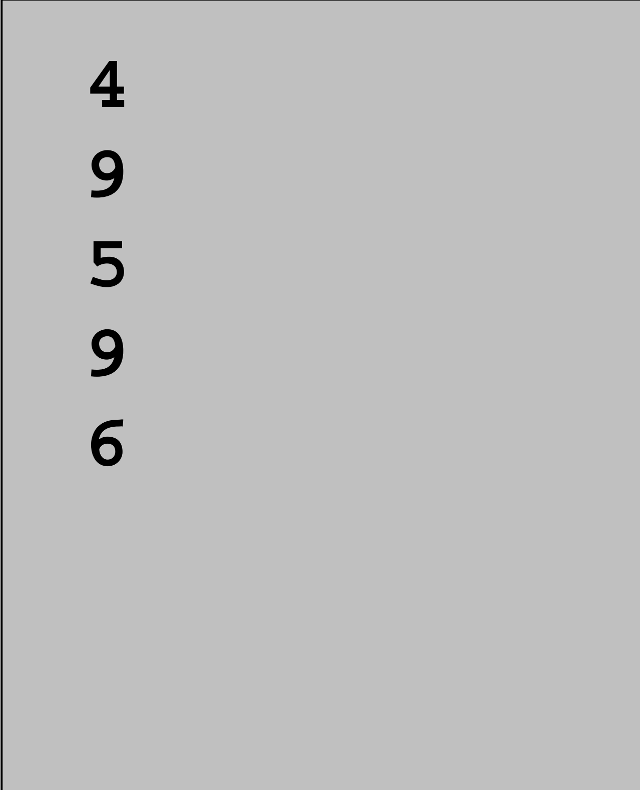


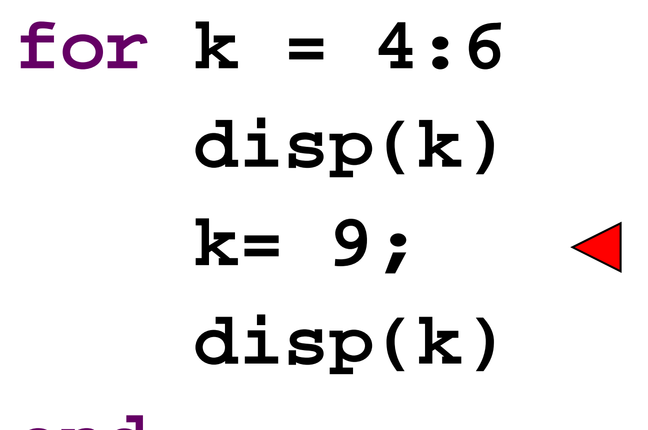


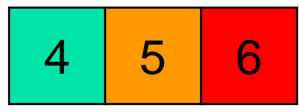


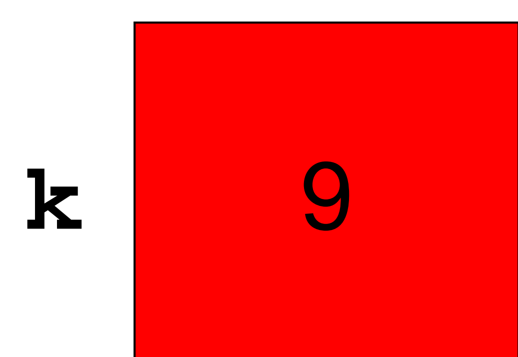


Output in Command Window

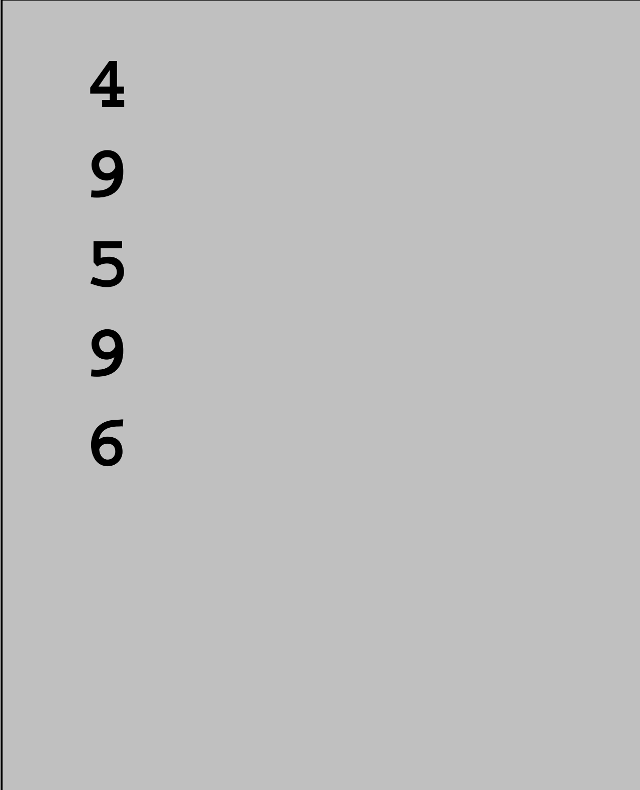


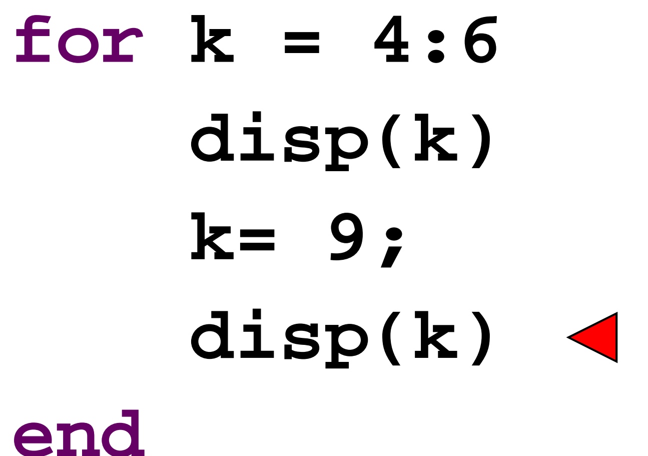


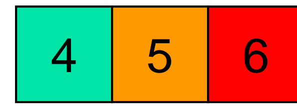


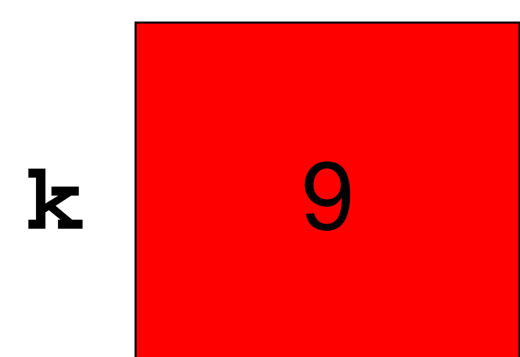


Output in Command Window

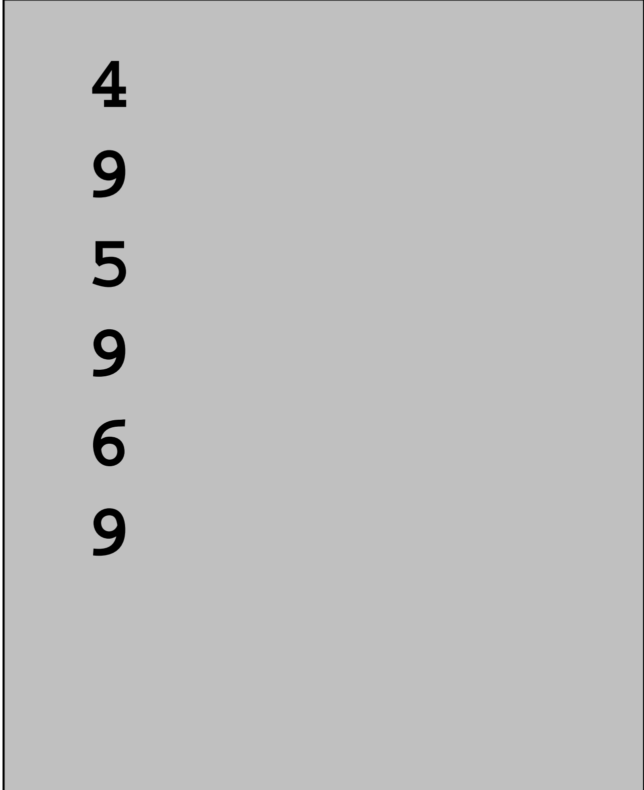


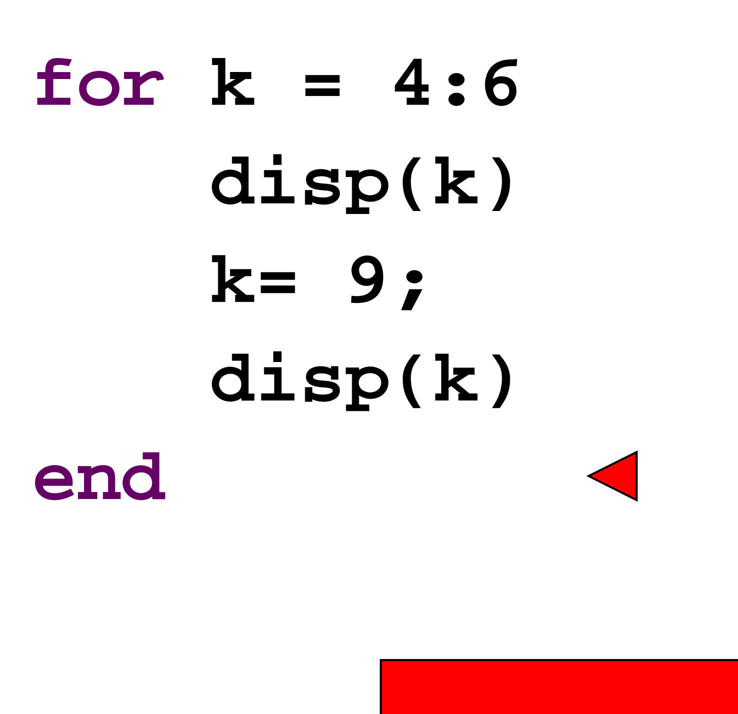


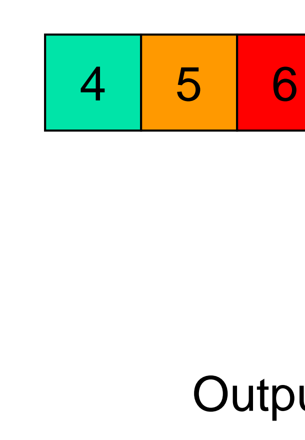




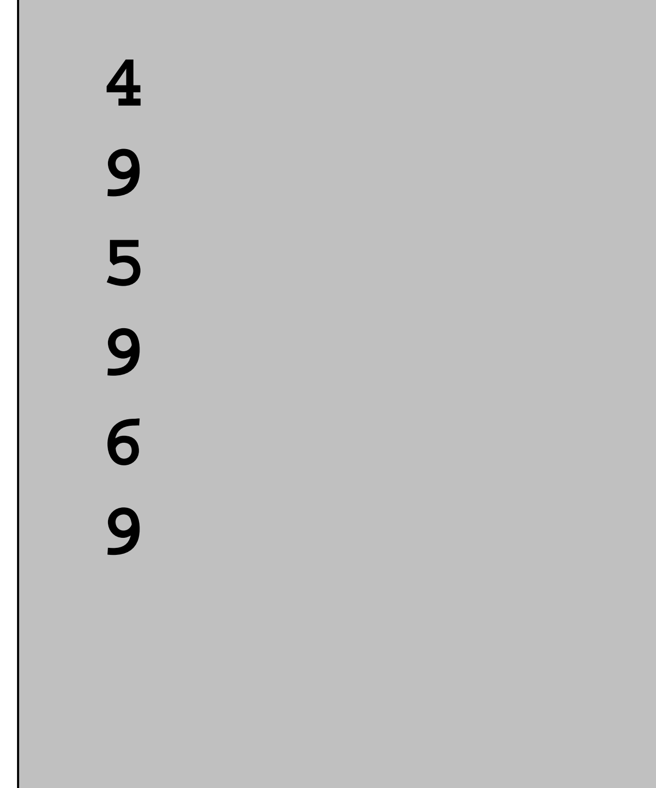
Output in Command Window




for k = 4:6
disp(k)
k= 9;
disp(k)
end
k
9


4
5
6

Output in Command Window


4
9
5
9
6
9

```
for k = 4:6
    disp(k)
    k= 9;
    disp(k)
end
```

Not a condition (boolean expression) that checks whether k<=6.

It is an expression that specifies values:

Monte Carlo π with N darts on L-by-L board

```
N= ___;  L= ___;  hits= 0;
for k = 1:N
   % Throw kth dart
   x = rand(1)*L - L/2;
   y = rand(1)*L - L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
end
myPi = 4*hits/N;
```

Using a while-loop

```
N= ___;  L= ___;  hits= 0;  k= 1;
while k <= N
   % Throw kth dart
   x = rand(1)*L - L/2;
   y = rand(1)*L - L/2;
   % Count it if it is in the circle
   if sqrt(x^2+y^2) <= L/2
      hits = hits + 1;
   end
   k = k+1;
end
myPi = 4*hits/N;
```

Common loop patterns

Do something *n* times

```
for k= 1:1:n
    % Do something

end
```

Do something an indefinite number of times

```
%Initialize loop variables

while ( not stopping signal )
    % Do something

    % Update loop variables

end
```

Patterns to do something n times

```
for k= 1:1:n
    % Do something

end
```

```
%Initialize loop variables
k= 1;
while ( k <= n )
    % Do something


    % Update loop variables
    k= k+1;
end
```

General form of a user-defined function

```
function [out1, out2, …]= functionName (in1, in2, …)
% 1-line comment to describe the function
% Additional description of function

    Executable code that at some point assigns
    values to output parameters out1, out2, …
```

- *in1*, *in2*, … are defined when the function begins execution. Variables *in1*, *in2*, … are called function *parameters* and they hold the function *arguments* used when the function is invoked (called).
- *out1*, *out2*, … are not defined until the executable code in the function assigns values to them.

```
function myPi = mcPiFun(N)
% myPi is Monte Carlo estimate of pi by
% throwing N darts

N= ___;  L= ___;  hits= 0;
for k = 1:N
    % Throw kth dart
    x = rand(1)*L - L/2;
    y = rand(1)*L - L/2;
    % Count it if it is in the circle
    if sqrt(x^2+y^2) <= L/2
        hits = hits + 1;
    end
end
myPi = 4*hits/N;
```

(The "N= ___;" assignment is struck through on the slide.)

```
function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

rads= theta*pi/180;  % radian
x= r*cos(rads);
y= r*sin(rads);
```

A function file
`polar2xy.m`

Function header is the “contract” for how the function will be used (called)

You have this function:

```
function  [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r, theta) to
% Cartesian coordinates (x,y).  Theta in degrees.
…
```

Code to call the above function:

```
% Convert polar (r1,t1) to Cartesian (x1,y1)
r1= 1;  t1= 30;
[x1, y1]= polar2xy(r1, t1);
plot(x1, y1, 'b*')
…
```

Given this function:

```
function m = convertLength(ft,in)
% Convert length from feet (ft) and inches (in)
% to meters (m).

   . . .
```

How many proper calls to convertLength are shown below?

```
% Given f and n
d= convertLength(f,n);
d= convertLength(f*12+n);
d= convertLength(f+n/12);
x= min(convertLength(f,n), 1);
y= convertLength(pi*(f+n/12)^2);
```

A: 1

B: 2

C: 3

D: 4

E: 5 or 0

Comments in functions

- Block of comments after the function header is printed whenever a user types

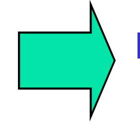
  help <*functionName*>

  at the Command Window

- 1st line of this comment block is searched whenever a user types

  lookfor <*someWord*>

  at the Command Window



- Every function should have a comment block after the function header that says what the function does concisely

Subfunction

- There can be more than one function in an M-file
- top function is the main function and has the name of the file
- remaining functions are subfunctions, accessible only by the functions in the same m-file
- Each (sub)function in the file begins with a function header
- Keyword `end` is not necessary at the end of a (sub)function

Arrays

The basic variable in Matlab is a matrix:

- Scalar → 1 x 1 matrix

- 1-d array of length 4 →
  1 x 4 matrix   or   4 x 1 matrix

- 2-d array → a matrix, naturally

Array index starts at 1

x	5	.4	.91	-4	-1	7
	1	2	3	4	5	6

Let k be the index of vector x, then

- k must be a positive integer
- 1<= k <= length(x)
- To access the kth element: x(k)

Here are a few different ways to create a vector

```
count= zeros(1,6)
```

Similar functions: `ones`, `rand`

```
a= linspace(10,30,5)

b= 7:-2:0

c= [3 7 2 1]

d= [3; 7; 2]
```

count

0	0	0	0	0	0

a

10	15	20	25	30

b

7	5	3	1

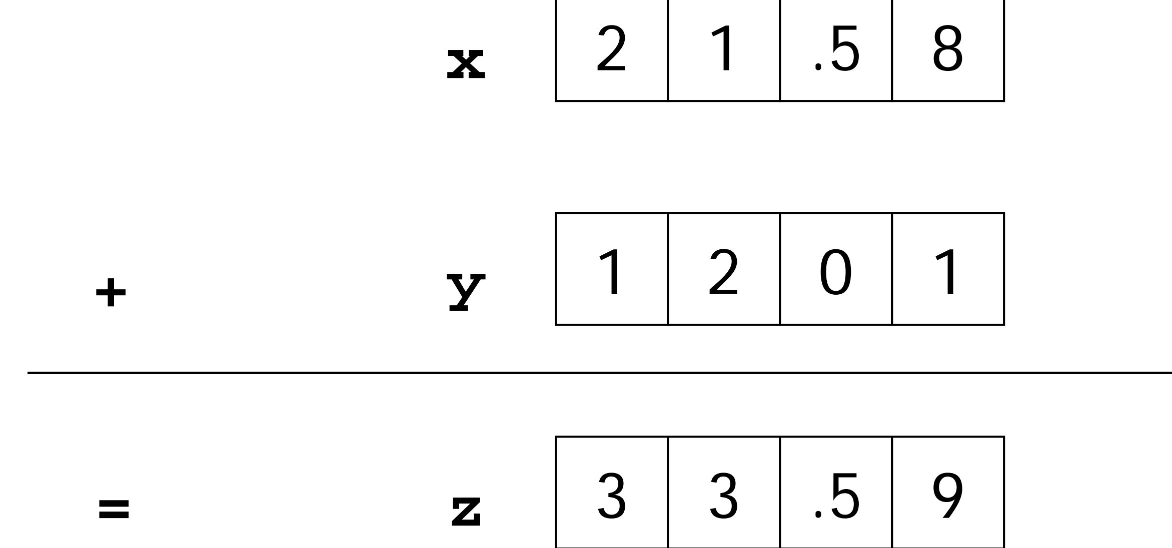
c

3	7	2	1

d

3
7
2

Vectorized addition



Matlab code: `z= x + y`

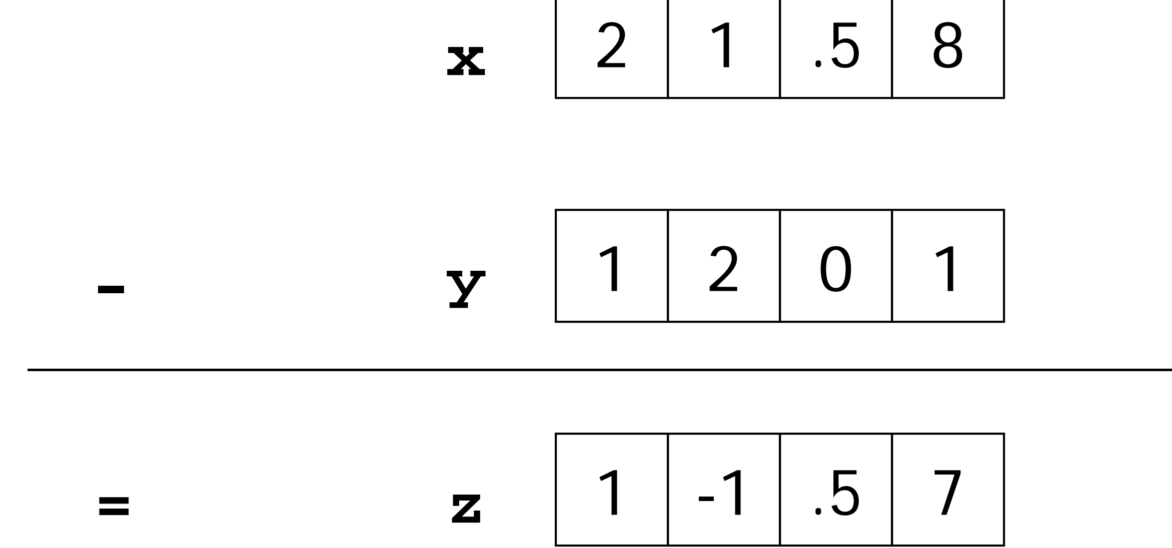
Vectorized code
—a Matlab-specific feature

- Code that performs element-by-element arithmetic/relational/logical operations on array operands in one step

- Scalar operation: x + y
  where x, y are scalar variables

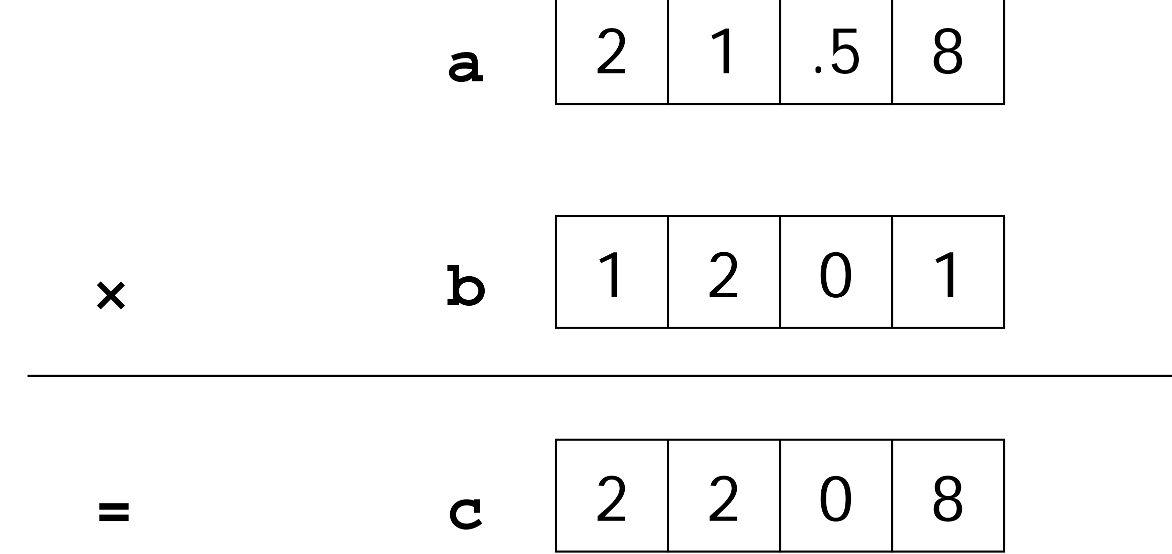
- Vectorized code: x + y
  where x and/or y are vectors. If x and y are both vectors, they must be of the same shape and length

Vectorized subtraction



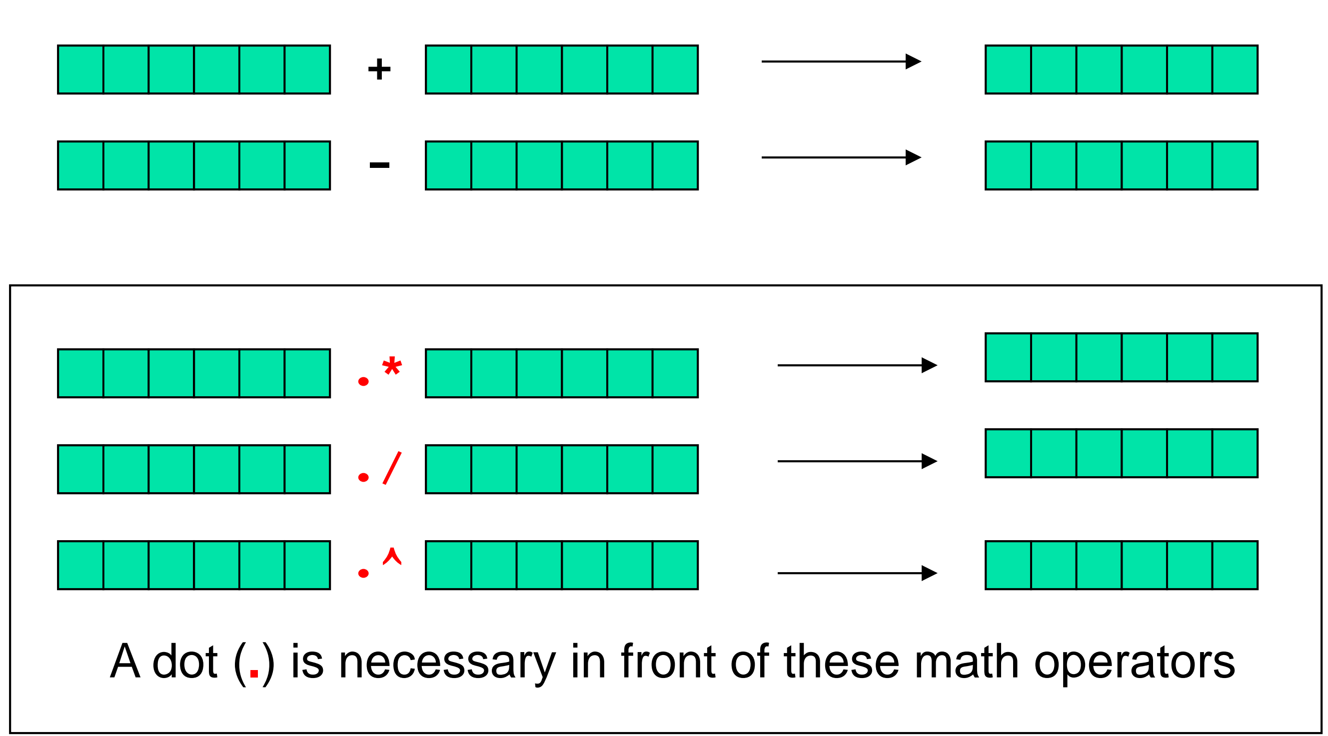
Matlab code: `z= x - y`

Vectorized multiplication

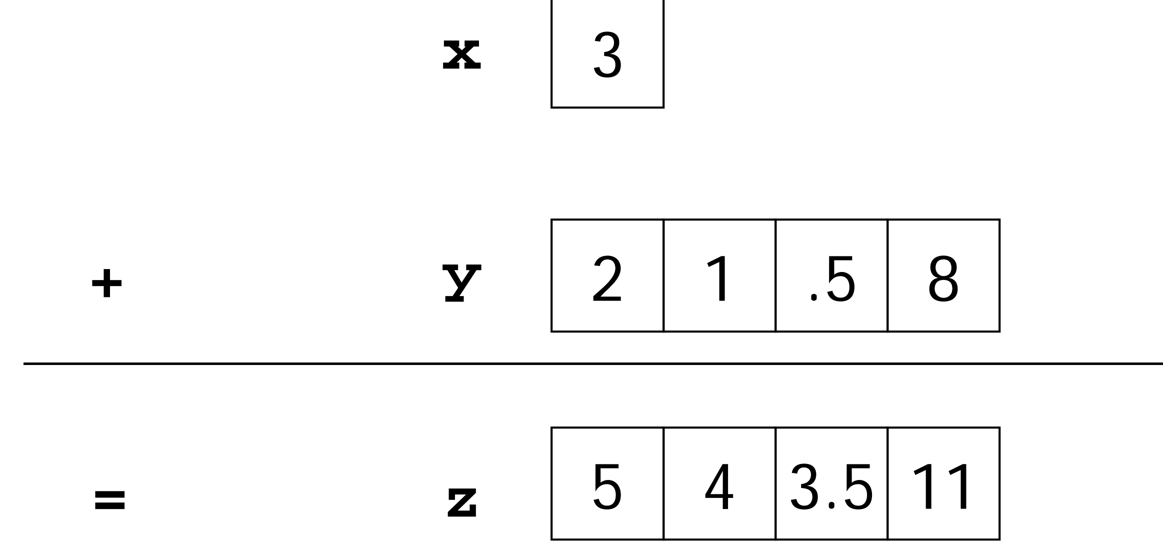


Matlab code: `c= a .* b`

Vectorized
element-by-element arithmetic operations on arrays

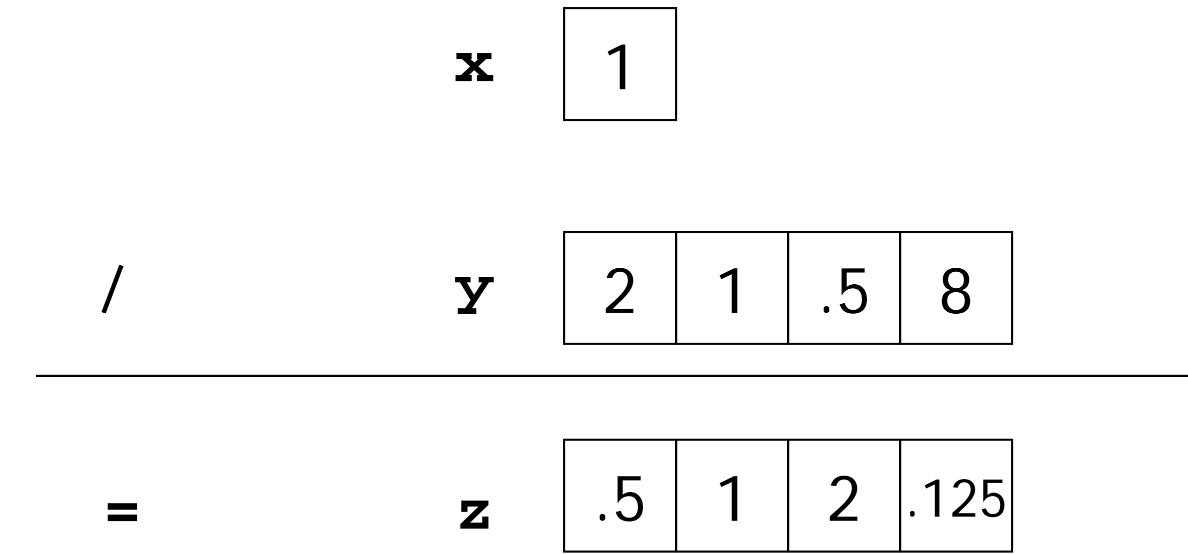


Shift



Matlab code: `z= x + y`

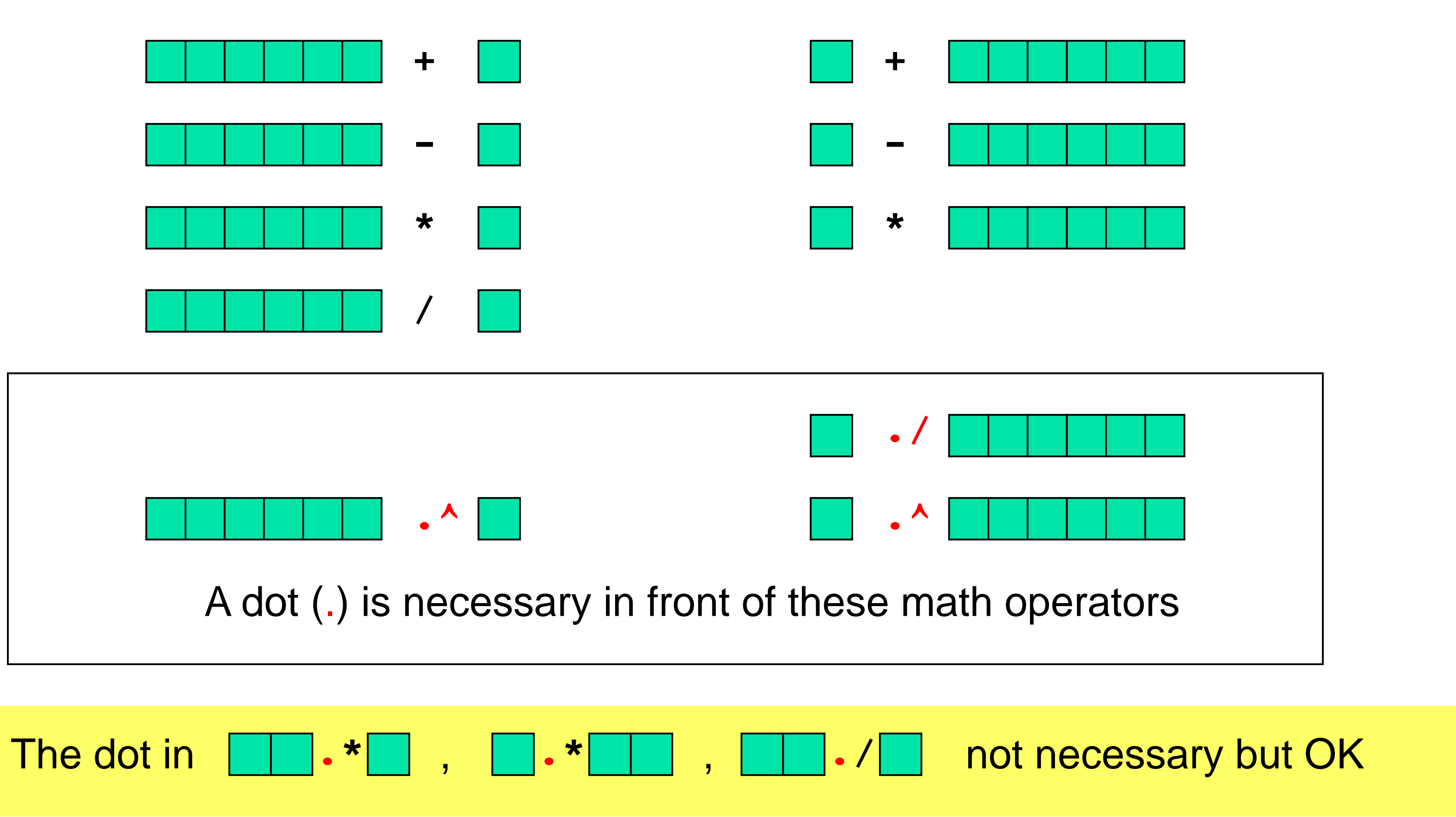
Reciprocate



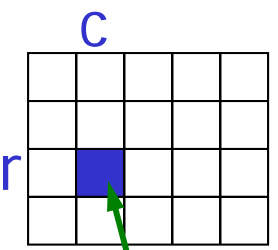
Matlab code:

```
z= x ./ y
```

Vectorized
element-by-element arithmetic operations between an array and a scalar



2-d array: **matrix**



- An array is a named collection of like data organized into rows and columns
- A 2-d array is a table, called a *matrix*
- Two *indices* identify the position of a value in a matrix, e.g.,

  `mat(r,c)`

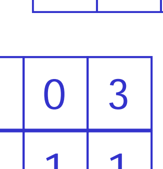
  refers to component in row r, column c of matrix mat
- Array index starts at 1
- Rectangular: all rows have the same #of columns

Creating a matrix

- Built-in functions: ones, zeros, rand
  - E.g., zeros(2,3) gives a 2-by-3 matrix of 0s
- "Build" a matrix using square brackets, [ ], but the dimension must match up:
  - [x y] puts y to the right of x
  - [x; y] puts y below x
  - [4 0 3; 5 1 9] creates the matrix

4	0	3
5	1	9

  - [4 0 3; ones(1,3)] gives
  - [4 0 3; ones(3,1)] doesn't work



What will **A** be?

```
A= [0  0]
A= [A'  ones(2,1)]
A= [0  0  0  0;  A  A]
```

```
% Given an nr-by-nc matrix M.
% What is A?
for  r= 1: nr
    for c= 1: nc
        A(c,r)= M(r,c);
    end
end
```