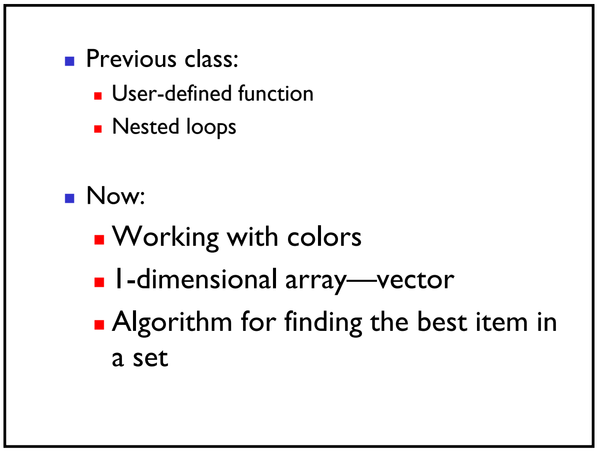
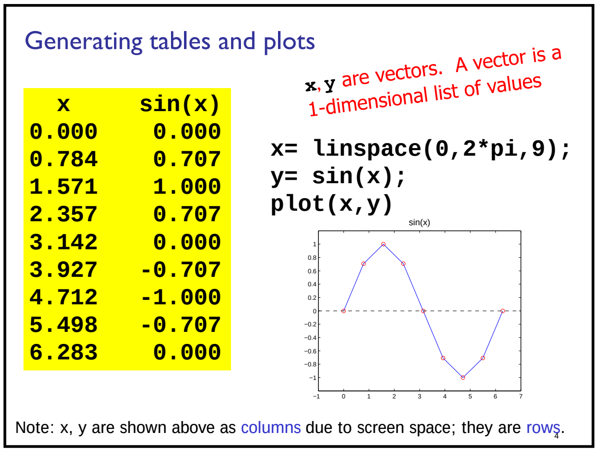


- Previous class:
  - User-defined function
  - Nested loops
- Now:
  - Working with colors
  - 1-dimensional array—vector
  - Algorithm for finding the best item in a set



Generating tables and plots

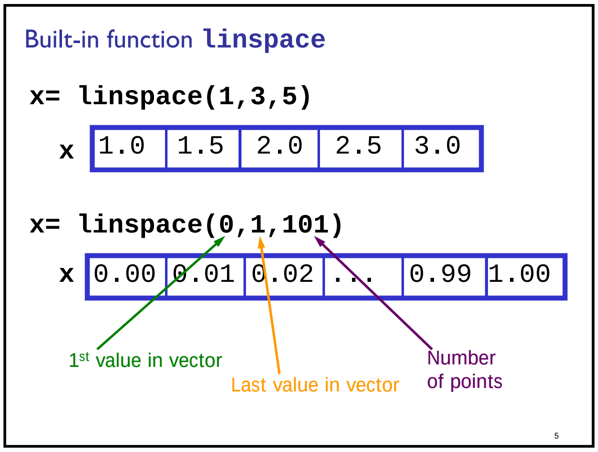
x	sin(x)
0.000	0.000
0.784	0.707
1.571	1.000
2.357	0.707
3.142	0.000
3.927	-0.707
4.712	-1.000
5.498	-0.707
6.283	0.000

`x`, `y` are vectors. A vector is a 1-dimensional list of values

```
x= linspace(0,2*pi,9);
y= sin(x);
plot(x,y)
```

sin(x)
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
−1
0
1
2
3
4
5
6
7

Note: x, y are shown above as columns due to screen space; they are rows.



Built-in function `linspace`

```
x= linspace(1,3,5)
```

x	1.0	1.5	2.0	2.5	3.0

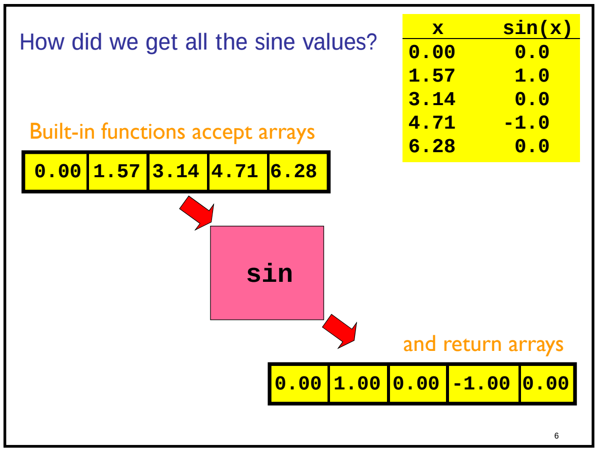
```
x= linspace(0,1,101)
```

x	0.00	0.01	0.02	. . .	0.99	1.00

1st value in vector

Last value in vector

Number of points



How did we get all the sine values?

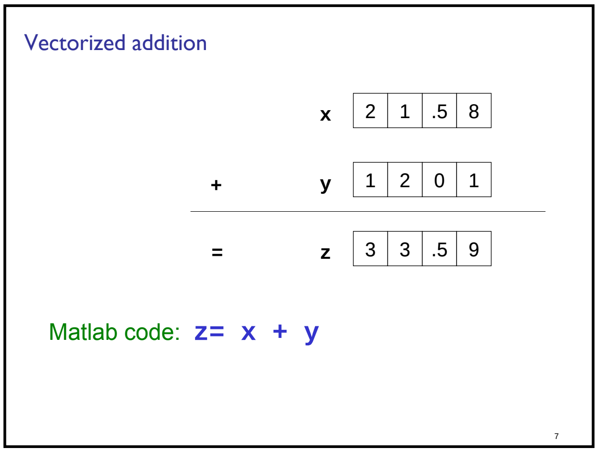
x	sin(x)
0.00	0.0
1.57	1.0
3.14	0.0
4.71	-1.0
6.28	0.0

Built-in functions accept arrays

0.00	1.57	3.14	4.71	6.28

and return arrays

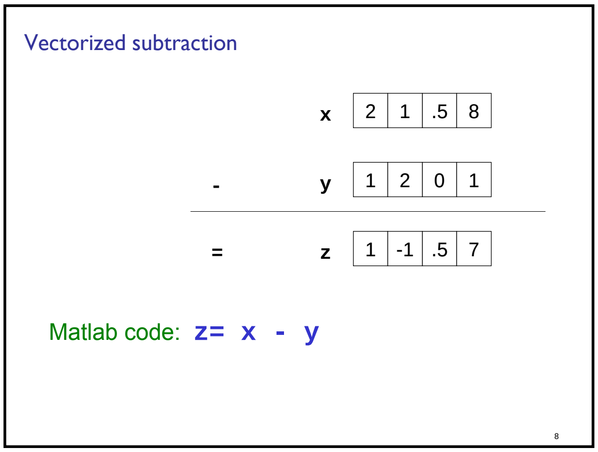
0.00	1.00	0.00	-1.00	0.00



Vectorized addition

	x	2	1	.5	8
+	y	1	2	0	1
=	z	3	3	.5	9

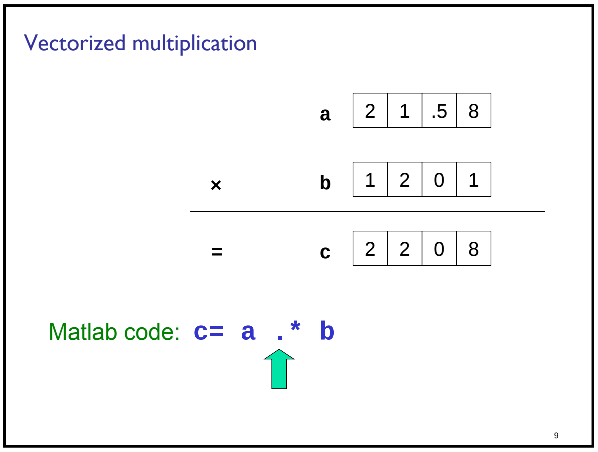
Matlab code: `z= x + y`



Vectorized subtraction

	x	2	1	.5	8
-	y	1	2	0	1
=	z	1	-1	.5	7

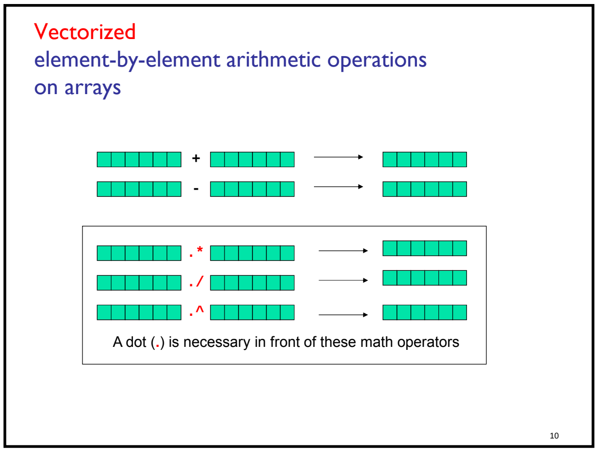
Matlab code: `z= x - y`



Vectorized multiplication

	a	2	1	.5	8
×	b	1	2	0	1
=	c	2	2	0	8

Matlab code: `c= a .* b`

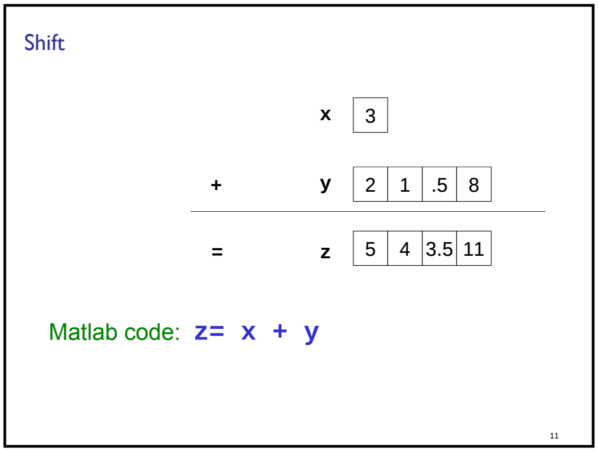


Vectorized
element-by-element arithmetic operations
on arrays

🟩🟩🟩🟩🟩🟩 + 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 - 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

🟩🟩🟩🟩🟩🟩 .* 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 ./ 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 .^ 🟩🟩🟩🟩🟩🟩 → 🟩🟩🟩🟩🟩🟩

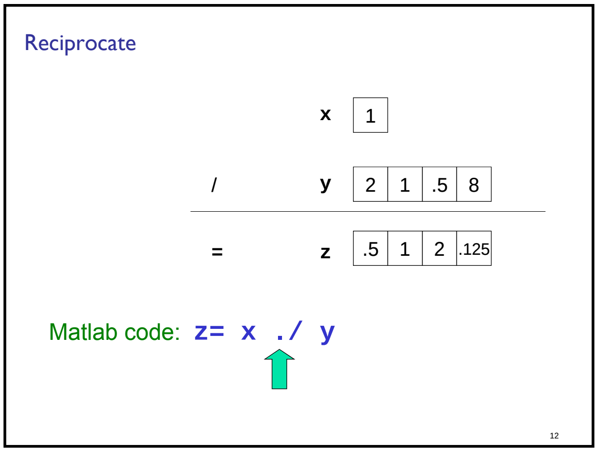
A dot (.) is necessary in front of these math operators



Shift

	x	3			
+	y	2	1	.5	8
=	z	5	4	3.5	11

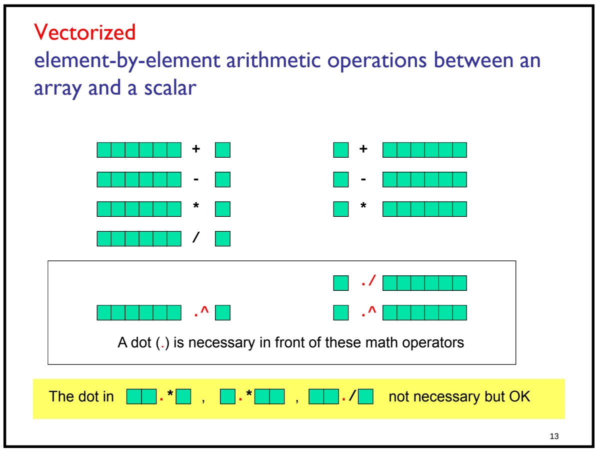
Matlab code: `z= x + y`



Reciprocate

	x	1			
/	y	2	1	.5	8
=	z	.5	1	2	.125

Matlab code: `z= x ./ y`



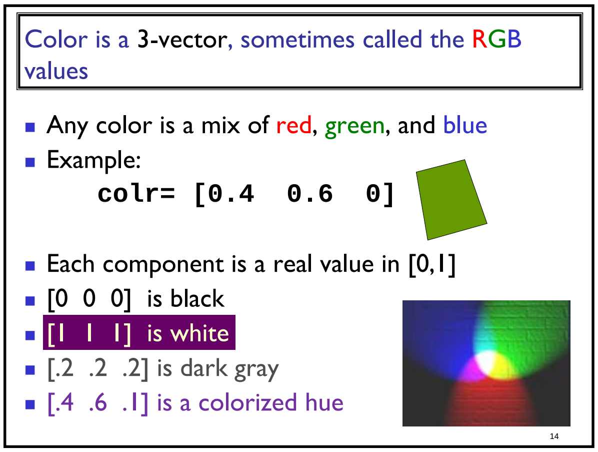
Vectorized
element-by-element arithmetic operations between an array and a scalar

🟩🟩🟩🟩🟩🟩 + 🟩 🟩 + 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 - 🟩 🟩 - 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 * 🟩 🟩 * 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 / 🟩

🟩 ./ 🟩🟩🟩🟩🟩🟩
🟩🟩🟩🟩🟩🟩 .^ 🟩 🟩 .^ 🟩🟩🟩🟩🟩🟩

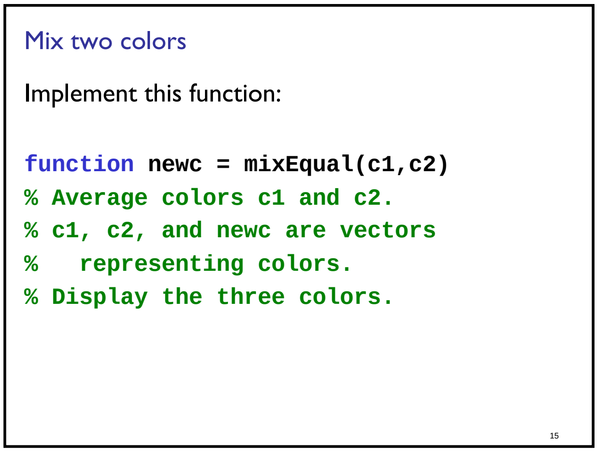
A dot (.) is necessary in front of these math operators

The dot in 🟩🟩.*🟩 , 🟩.*🟩🟩 , 🟩🟩./🟩 not necessary but OK



Color is a 3-vector, sometimes called the RGB values

- Any color is a mix of red, green, and blue
- Example:
  `colr= [0.4 0.6 0]`
- Each component is a real value in [0,1]
- [0 0 0] is black
- [1 1 1] is white
- [.2 .2 .2] is dark gray
- [.4 .6 .1] is a colorized hue

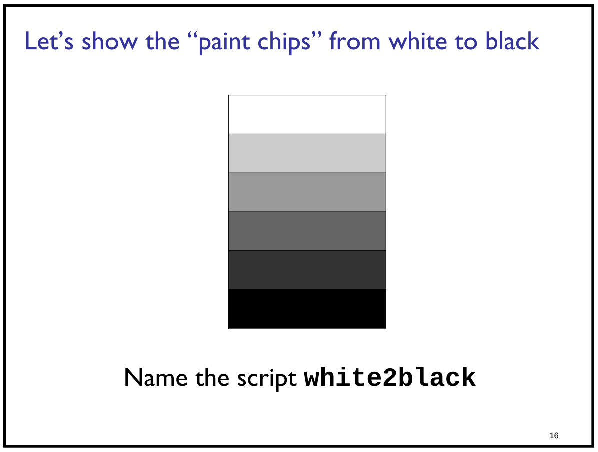


Mix two colors

Implement this function:

```
function newc = mixEqual(c1,c2)
% Average colors c1 and c2.
% c1, c2, and newc are vectors
%   representing colors.
% Display the three colors.
```

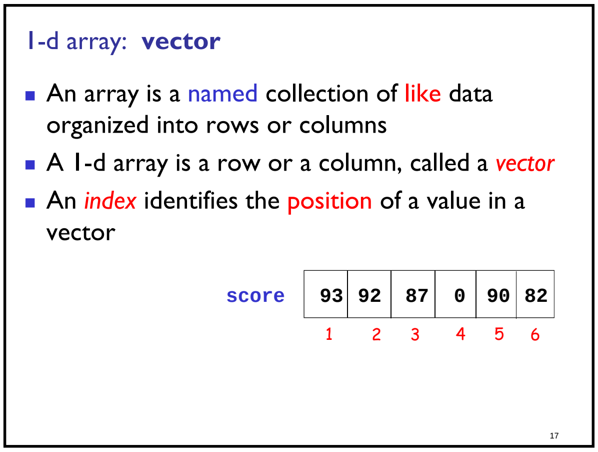
15



Let’s show the “paint chips” from white to black

Name the script **white2black**

16

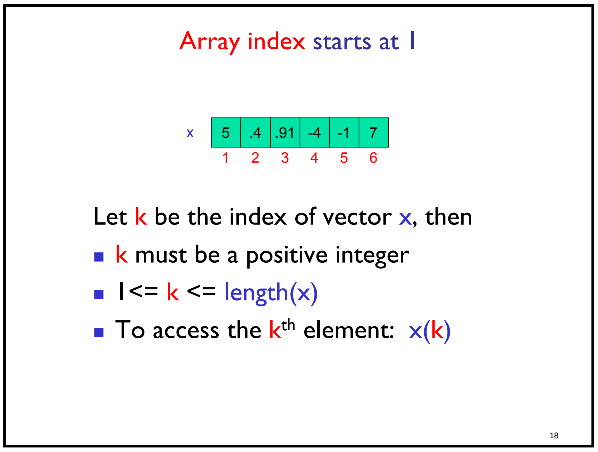


1-d array: **vector**

- An array is a named collection of like data organized into rows or columns
- A 1-d array is a row or a column, called a *vector*
- An *index* identifies the position of a value in a vector

score	93	92	87	0	90	82
	1	2	3	4	5	6

17



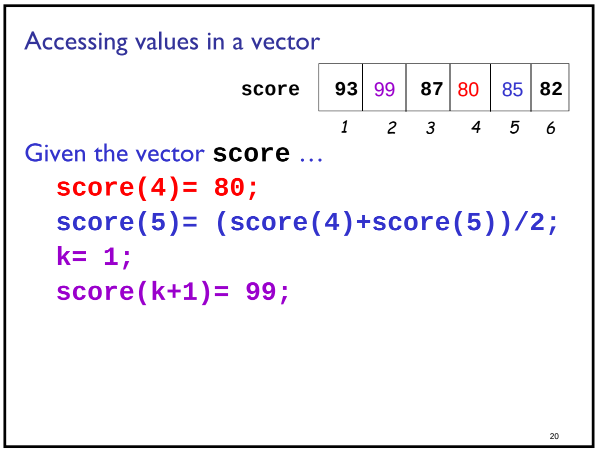
Array index starts at 1

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)

18



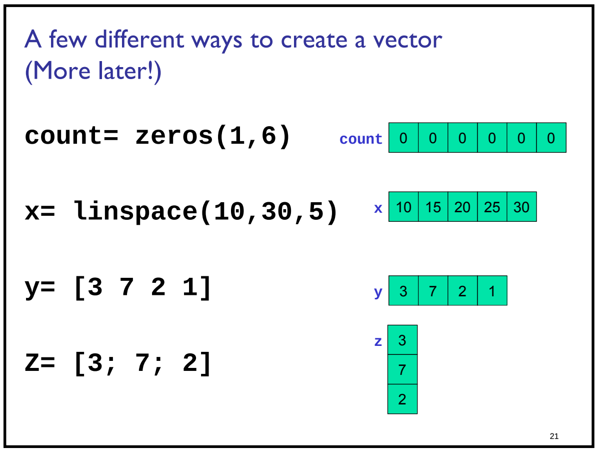
Accessing values in a vector

score	93	99	87	80	85	82
	1	*2*	*3*	*4*	*5*	*6*

Given the vector **score** …

```
score(4)= 80;
score(5)= (score(4)+score(5))/2;
k= 1;
score(k+1)= 99;
```

20



A few different ways to create a vector (More later!)

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

count	0	0	0	0	0	0

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

x	10	15	20	25	30

```
y= [3 7 2 1]
```

y	3	7	2	1

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

z	3
	7
	2

21

Drawing a single line segment

```
a= 0;  % x-coord of pt 1
b= 1;  % y-coord of pt 1
c= 5;  % x-coord of pt 2
d= 3;  % y-coord of pt 2
plot([a c], [b d], '-*')
```

x-values (a vector)

y-values (a vector)

Line/marker format

Drawing a polygon (multiple line segments)

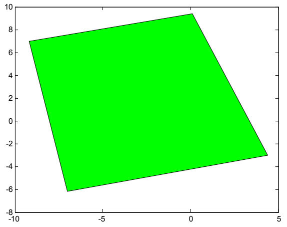
```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h.
x= [a  a+w  a+w  a    a ];  % x data
y= [b  b    b+h  b+h  b ];  % y data
plot(x, y)
```

Coloring a polygon (fill)

```
% Draw a rectangle with the lower-left
% corner at (a,b), width w, height h,
% and fill it with a color named by c.
x= [a  a+w  a+w  a   ~~a~~];  % x data
y= [b  b    b+h  b+h ~~b~~];  % y data
fill(x, y, c)
```

Built-in function **fill** actually does the "wrap-around" automatically.

```
x= [0.1 -9.2  -7  4.4];
y= [9.4  7   -6.2  -3];
fill(x,y,'g')
```

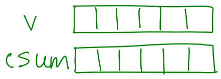


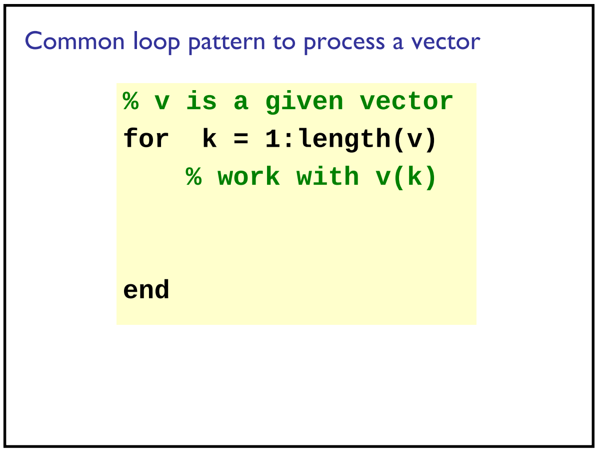
Example

- Write a program fragment that calculates the cumulative sums of a given vector **v**.
- The cumulative sums should be stored in a vector of the same length as **v**.

1, 3, 5, 0   **v**

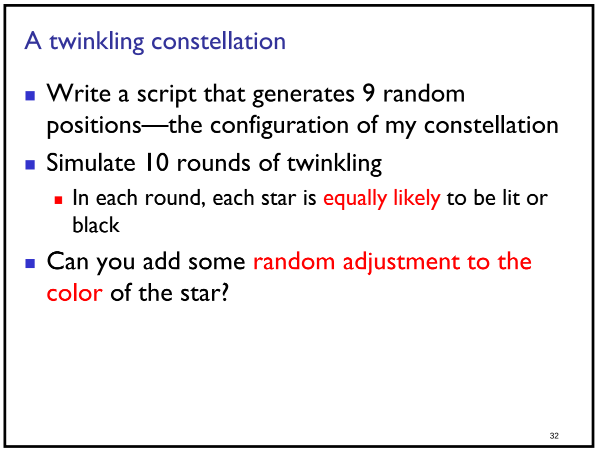
1, 4, 9, 9   cumulative sums of **v**





Common loop pattern to process a vector

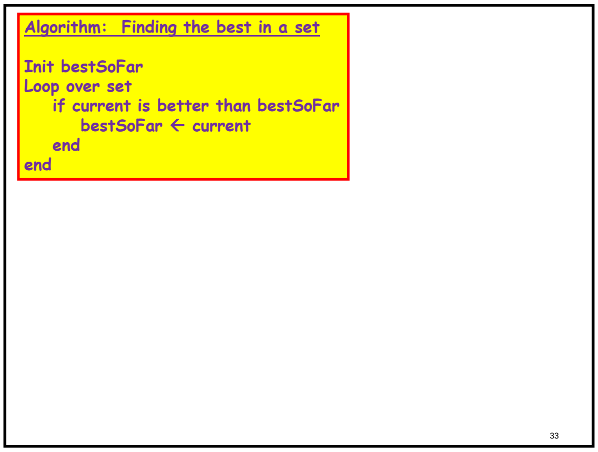
```
% v is a given vector
for  k = 1:length(v)
    % work with v(k)


end
```



A twinkling constellation

- Write a script that generates 9 random positions—the configuration of my constellation
- Simulate 10 rounds of twinkling
  - In each round, each star is equally likely to be lit or black
- Can you add some random adjustment to the color of the star?

32



```
Algorithm:  Finding the best in a set

Init bestSoFar
Loop over set
    if current is better than bestSoFar
        bestSoFar ← current
    end
end
```

33