- Previous Lecture:
  - User-defined functions
    - Local memory space
    - Subfunctions
- Today's Lecture:
  - 1-d array—vector
  - Probability and random numbers
  - Simulation using random numbers, vectors
- Announcements:
  - Project 3 posted; due Monday 10/1
  - Discussion this week in classrooms as listed on the roster
  - Prelim 1 next Thursday, 10/4, at 7:30pm

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

v	.8	.2	1
	1	2	3

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 k^{th} element: x(k)

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;
```


Here are a few different ways to create a vector

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

count	0	0	0	0	0	0

Similar functions: **ones**, **rand**

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

a	10	15	20	25	30

```
b= 7:-2:0
```

b	7	5	3	1

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

c	3	7	2	1

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

d
3
7
2

```
e= d'
```

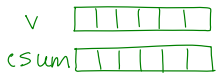
e	3	7	2

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**

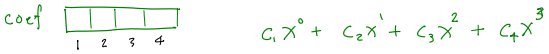


Example

- Write a function **`evalPoly`** to evaluate an n^{th} order polynomial of x :

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- Input parameter **`coef`** has length n+1, contains the coefficients of the polynomial
- **`coef(1)`** is the coefficient for the term x^0
- Input parameter **`x`**
- Return the value of the polynomial evaluated at x
- No Matlab predefined function other than **`length`**



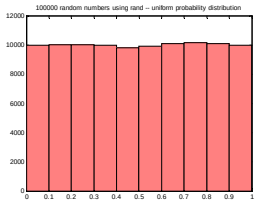
Random numbers

- *Pseudorandom* numbers in programming
- Function **`rand(…)`** generates random real numbers in the interval (0,1). All numbers in the interval (0,1) are equally likely to occur—uniform probability distribution.
- Examples:

  `rand(1)` one random # in (0,1)

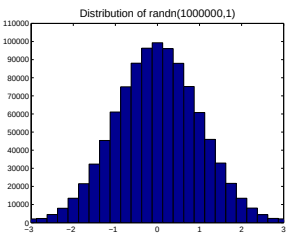
  `6*rand(1)` one random # in (0,6)

  `6*rand(1)+1` one random # in (1,7)



Uniform probability distribution in (0,1)
`rand`

Normal distribution with zero mean and unit standard deviation
`randn`



Simulate a fair 6-sided die

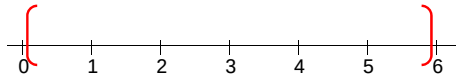
Which expression(s) below will give a random *integer* in [1..6] with equal likelihood?

A **`round(rand*6)`**

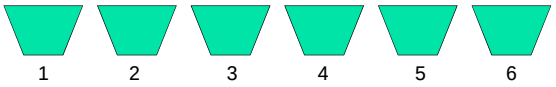
B **`ceil(rand*6)`**

C *Both expressions above*

(rand(1)*6)



Possible outcomes from rolling a fair 6-sided die



Keep tally on repeated rolls of a fair die

Repeat the following:

```
% roll the die

% increment correct "bin"
```



```
function count = rollDie(rolls)

FACES= 6;               % #faces on die
count= zeros(1,FACES);  % bins to store counts

% Count outcomes of rolling a FAIR die
for k= 1:rolls
    % Roll the die

    % Increment the appropriate bin

end

% Show histogram of outcome
```



```
% Count outcomes of rolling a FAIR die
count= zeros(1,6);
for k= 1:100
    face= ceil(rand*6);
    if face==1
        count(1)= count(1) + 1;
    elseif face==2
        count(2)= count(2) + 1;
    ⋮
    elseif face==5
        count(5)= count(5) + 1;
    else
        count(6)= count(6) + 1;
    end
end
```

```
function count = rollDie(rolls)

FACES= 6;               % #faces on die
count= zeros(1,FACES);  % bins to store counts

% Count outcomes of rolling a FAIR die
for k= 1:rolls
    % Roll the die
    face= ceil(rand*FACES);
    % Increment the appropriate bin
    count(face)= count(face) + 1;
end

% Show histogram of outcome
```

Simulate twinkling stars

- Get 10 user mouse clicks as locations of 10 stars—our constellation
- Simulate twinkling
  - Loop through all the stars; each has equal likelihood of being bright or dark
  - Repeat many times
- Can use DrawStar, DrawRect

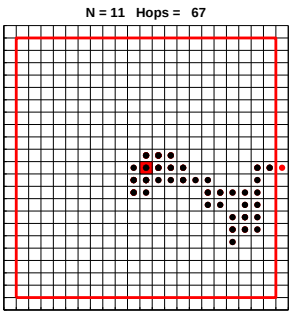

```
% No. of stars and star radius
  N=10;  r=.5;
% Get mouse clicks, store coords in vectors x,y
  [x,y] = ginput(N);
% Twinkle!
  for k= 1:20  % 20 rounds of twinkling









  end
```

2-dimensional random walk

Start in the middle tile, (0,0).

For each step, randomly choose between N,E,S,W and then walk one tile. Each tile is 1×1.

Walk until you reach the boundary.




```
function [x, y] = RandomWalk2D(N)
% 2D random walk in 2N-1 by 2N-1 grid.
% Walk randomly from (0,0) to an edge.
% Vectors x,y represent the path.
```



```
function [x, y] = RandomWalk2D(N)

k=0;  xc=0;  yc=0;

while   not at an edge
   % Choose random dir, update xc,yc




   % Record new location in x, y

end
```

```
% Standing at (xc,yc)
% Randomly select a step
    r= rand(1);
    if r < .25
        yc= yc + 1;  % north
    elseif r < .5
        xc= xc + 1;  % east
    elseif r < .75
        yc= yc -1;   % south
    else
        xc= xc -1;   % west
    end
```