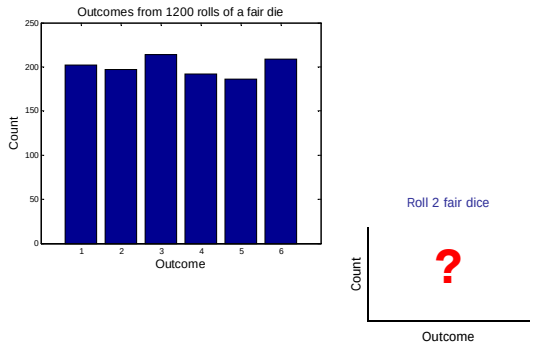
- Previous Lecture:
  - Executing a user-defined function
  - Function scope
  - Subfunction
- Today's Lecture:
  - 1-d array—vector
  - Probability and random numbers
  - Simulation using random numbers, vectors
- Announcement:
  - Project 3 due Monday10/3 at 11pm



Lecture 10 3

What is the output?

```
x = 1;
x = f(x+1);
y = x+1;
disp(y)
```

```
function y = f(x)
x = x+1;
y = x+1;
```

A: 1 B: 2 C: 3 D: 4 E: 5

Lecture 9 4

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

Lecture 10 9

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 |

10

Start with 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

Lecture 10 11

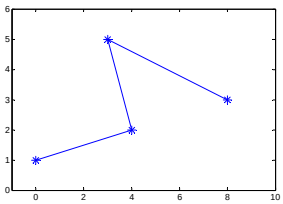
Making an x-y plot

```
a= [0 4 3 8];  % x-coords
b= [1 2 5 3];  % y-coords
plot(a, b, '-*')
```

x-values (a vector)

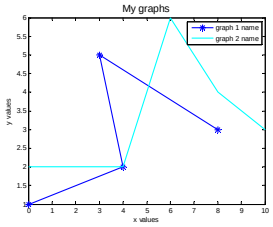
y-values (a vector)

Line/marker format



Making an x-y plot with multiple graphs (lines)

```
a= [0 4 5 8];
b= [1 2 5 3];
f= [0 4 6 8 10];
g= [2 2 6 4  3];
plot(a,b,'-*',f,g,'c')
legend('graph 1 name', 'graph 2 name')
xlabel('x values')
ylabel('y values')
title('My graphs', 'Fontsize',14)
```



13

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)

Lecture 10 16

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;
```

See **plotComparison2.m**

Lecture 10 18

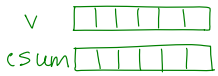
Example

- 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**

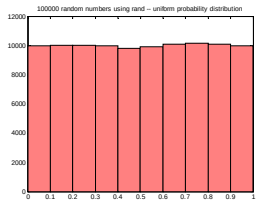
Lecture 10 19



Random numbers

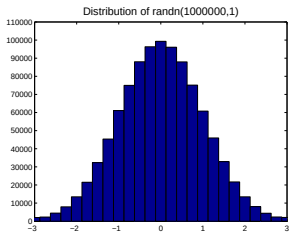
- *Pseudorandom* numbers in programming
- Function **`rand(…)`** generates random real numbers in the interval (0,1). All numbers in the interval (0,1) are equally likely to occur—uniform probability distribution.
- Examples:

  `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`

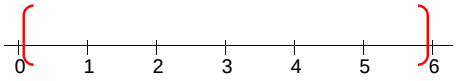


Simulate a fair 6-sided die

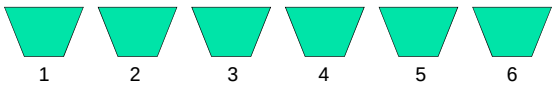
Which expression(s) below will give a random *integer* in [1..6] with equal likelihood?

A **`round(rand*6)`**
B **`ceil(rand*6)`**
C *Both expressions above*

`(rand*6)`



Possible outcomes from rolling a fair 6-sided die

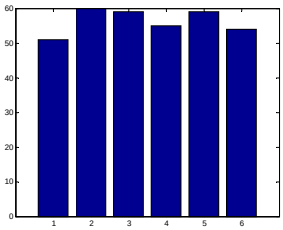


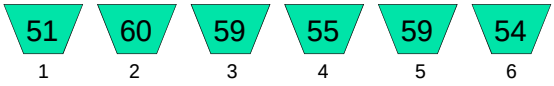
Simulation result

Data in bins

`bar(1:6, count)`

Bin numbers





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);
```

	1	2	3	4	5	6
count	0	0	0	0	0	0

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

    % Increment the appropriate bin

end

% Show histogram of outcome
```


rollDieV1.m


```
% 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
```

	1	2	3	4	5	6
count	0	0	0	0	0	0

Looks redundant … is it? Is there a more concise way?

```
function count = rollDie(rolls)

FACES= 6;                   % #faces on die
count= zeros(1,FACES);
```

	1	2	3	4	5	6
count	0	0	0	0	0	0

```
% 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
```