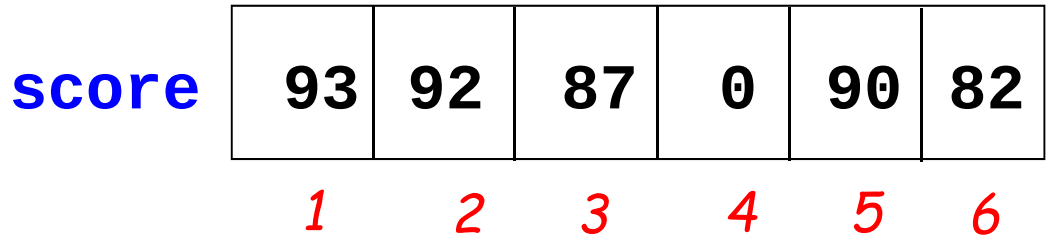
- Previous Lecture:
  - Executing a user-defined function
  - Function scope
- Today's Lecture:
  - Subfunction
  - 1-d array—vector
  - Probability and random numbers
  - Simulation using random numbers, vectors
- Announcement:
  - Project 3 due Thursday, March 5 at 11pm
  - Check Staff page for updates to office hours

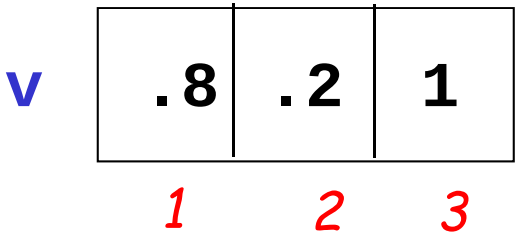
Subfunction

- 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

1-d array: **vector**

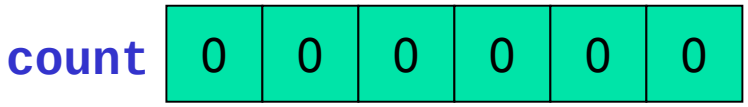
- An array is a named collection of like data organized into rows or columns
- A 1-d array is a row or a column, called a *vector*
- An *index* identifies the position of a value in a vector





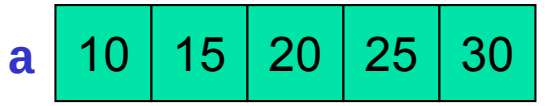
Here are a few different ways to create a vector

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

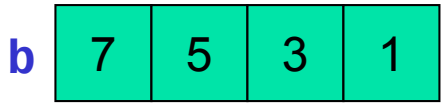


Similar functions: **ones**, **rand**

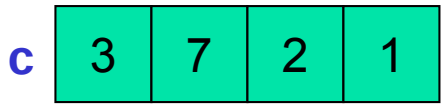
```
a= linspace(10,30,5)
```



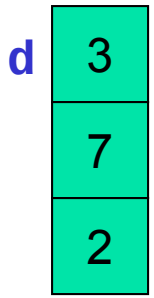

```
b= 7:-2:0
```



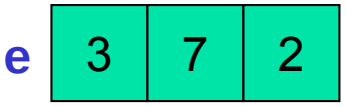

```
c= [3 7 2 1]
```



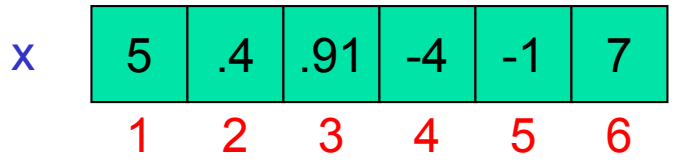

```
d= [3; 7; 2]
```




```
e= d'
```



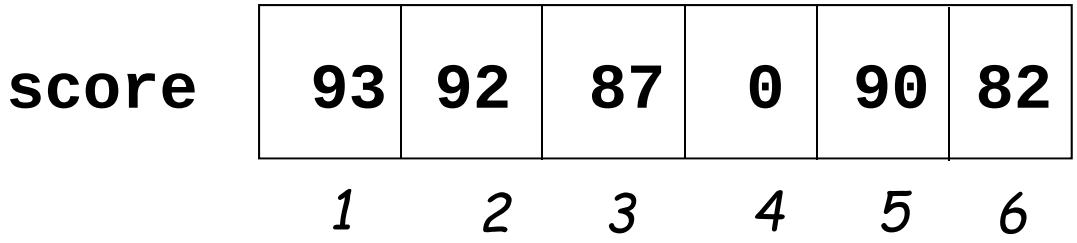
Array index starts at 1



Let k be the index of vector x, then

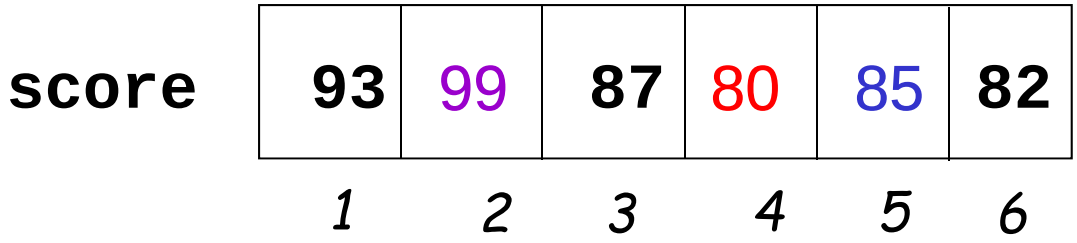
- k must be a positive integer
- 1<= k <= length(x)
- To access the k^{th} element: x(k)

Accessing values in a vector



Given the vector **`score`** …

Accessing values in a vector



Given the vector **score** …

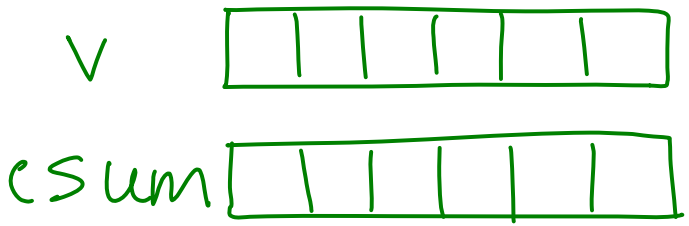
```
score(4)= 80;
score(5)= (score(4)+score(5))/2;
k= 1;
score(k+1)= 99;
```

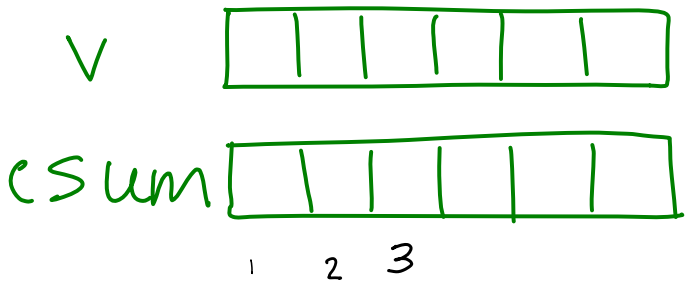
Example

- 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**


v
csum



$csum(k) = csum(k-1) + v(k)$

$csum(3) = v(1) + v(2) + v(3)$

$csum(4) = v(1) + v(2) + v(3) + v(4)$

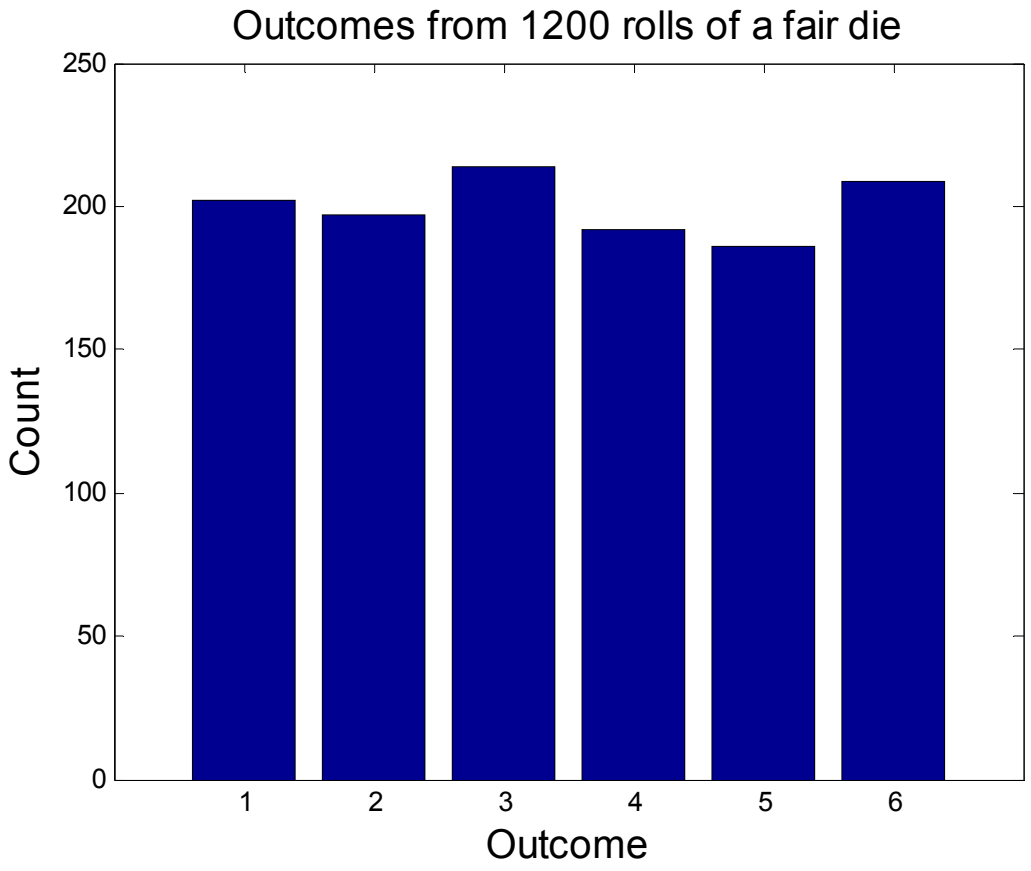
$csum(3)$

```
csum(1) = v(1);
for k = 2 : length(v)
    csum(k) = csum(k-1) + v(k);
end
```

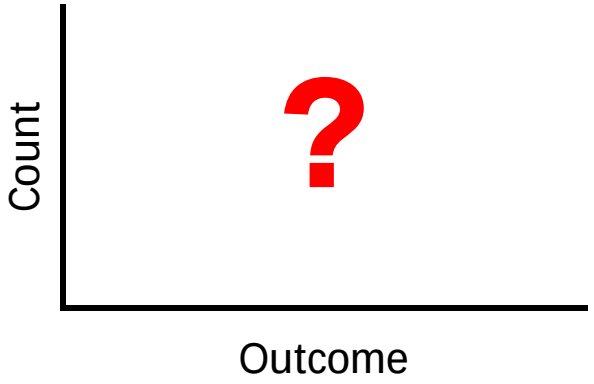
Random numbers

- *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)



Roll 2 fair dice



Simulate a fair 6-sided die

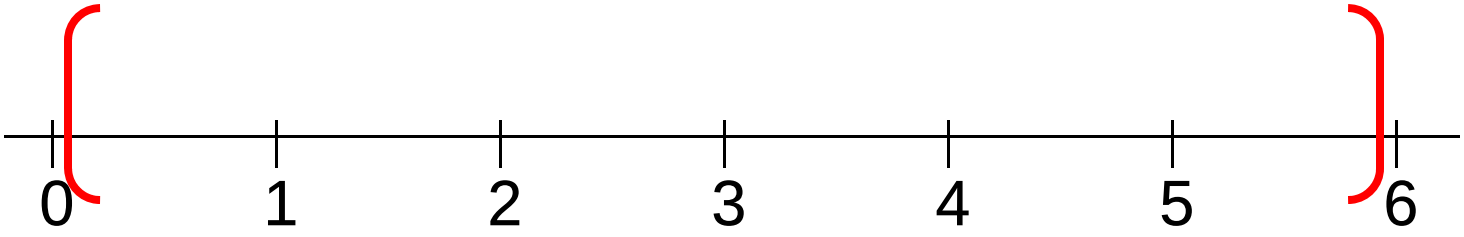
Which expression(s) below will give a random *integer* in [1..6] with equal likelihood?

A `round(rand*6)`

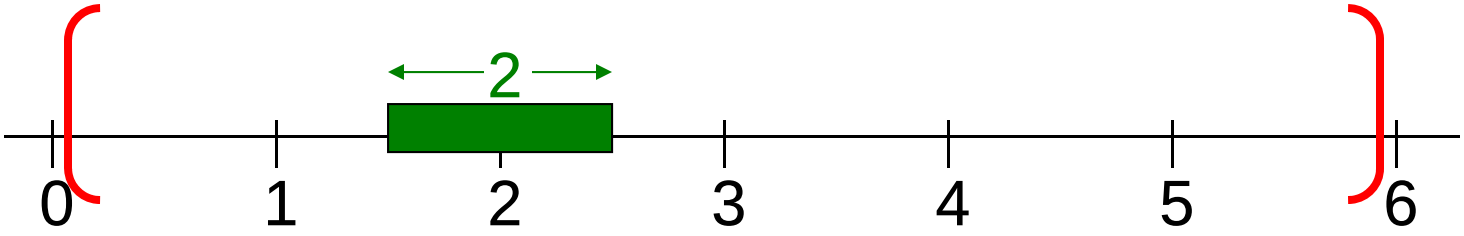
B `ceil(rand*6)`

C *Both expressions above*

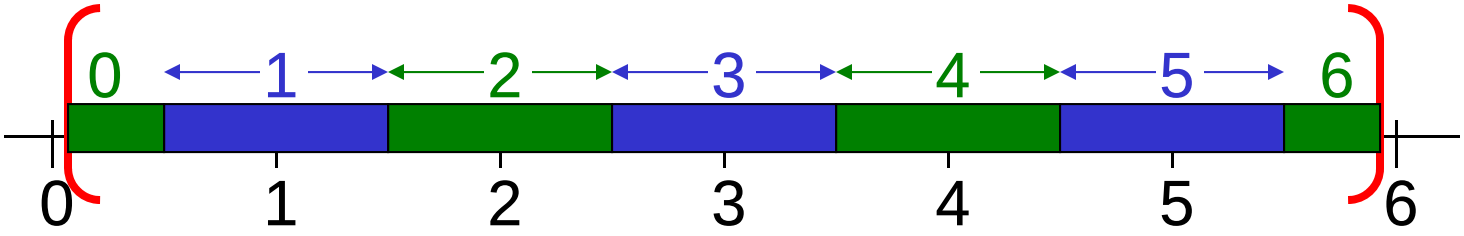
`(rand*6)`



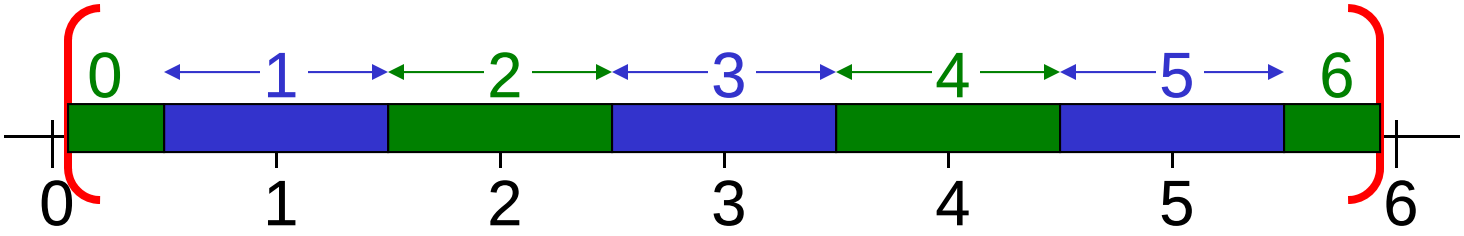
`round(rand*6)`



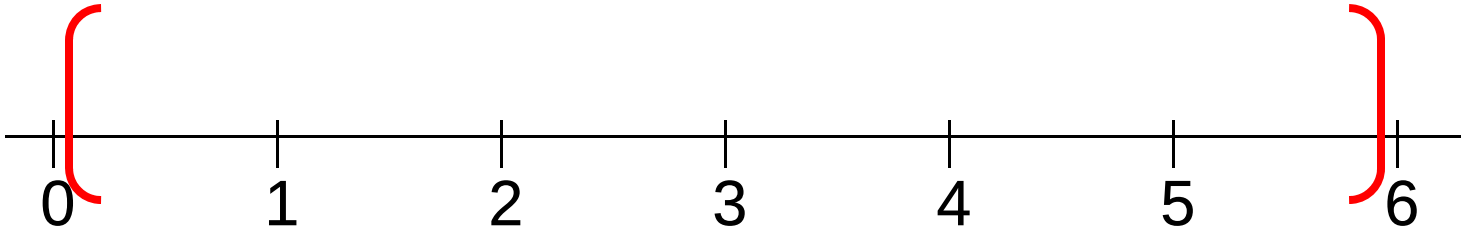
`round(rand*6)`



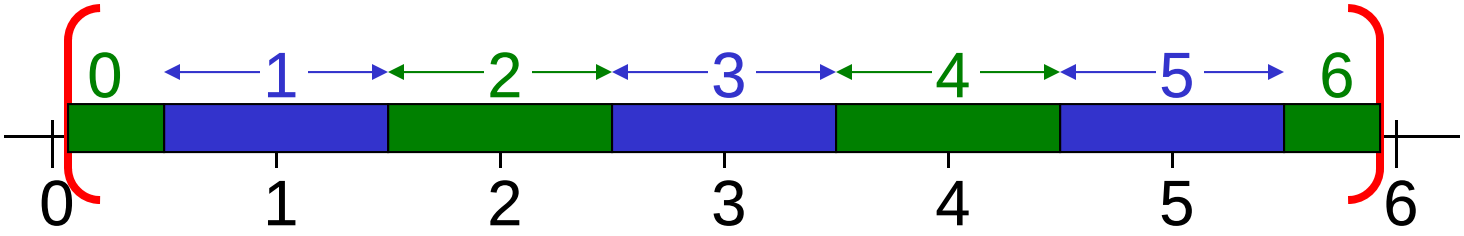
`round(rand*6)`



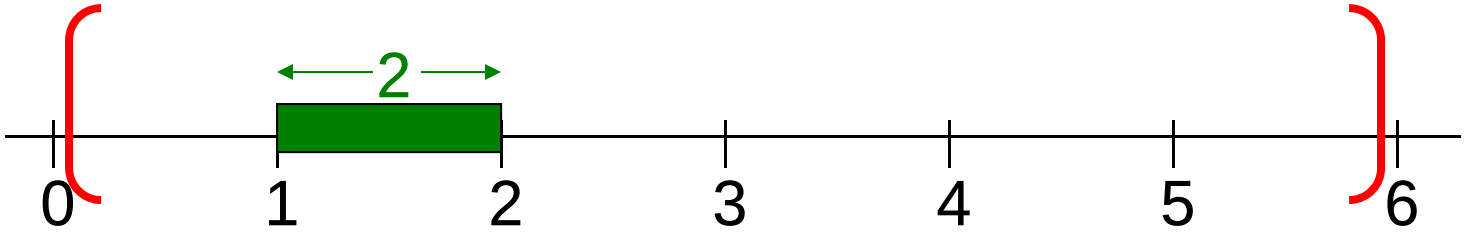
`(rand*6)`



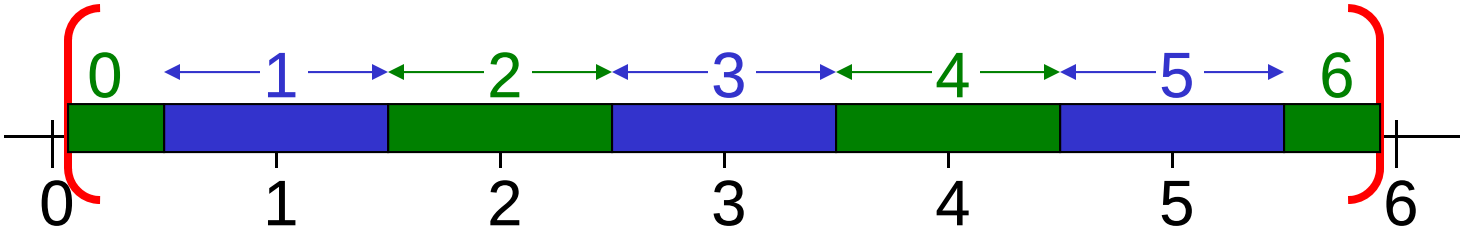
`round(rand*6)`



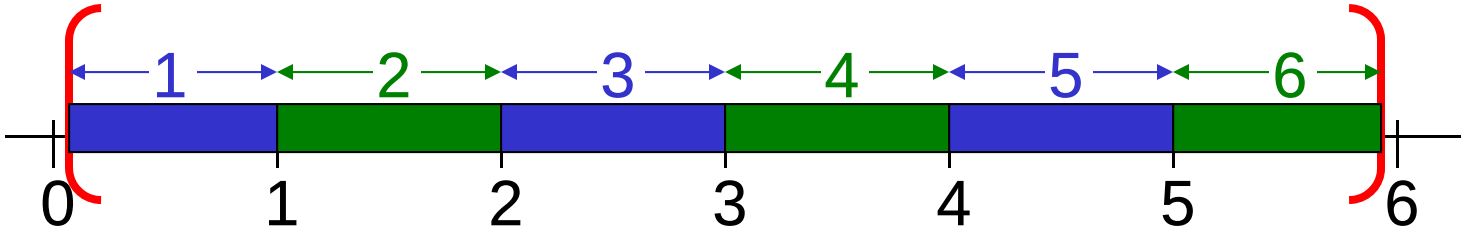
`ceil(rand*6)`



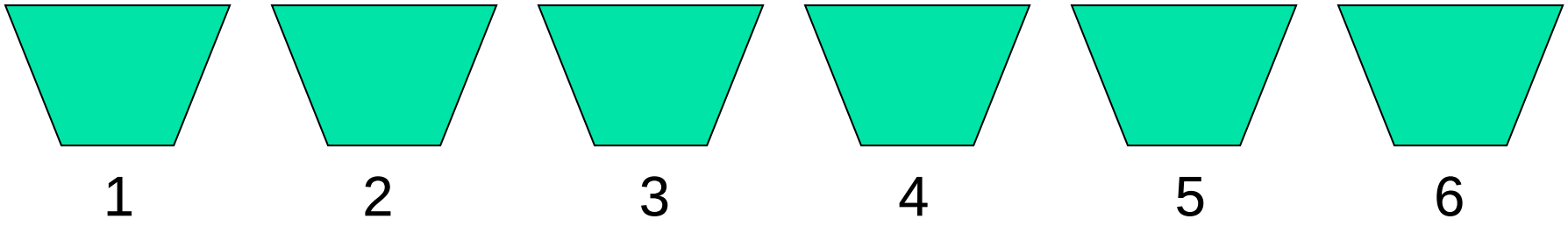
`round(rand*6)`



`ceil(rand*6)`



Possible outcomes from rolling a fair 6-sided die

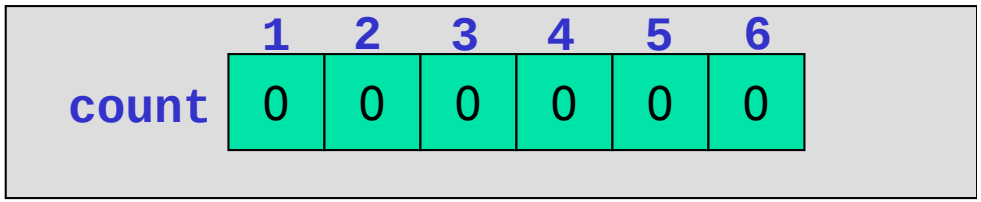


Keep tally on repeated rolls of a fair die

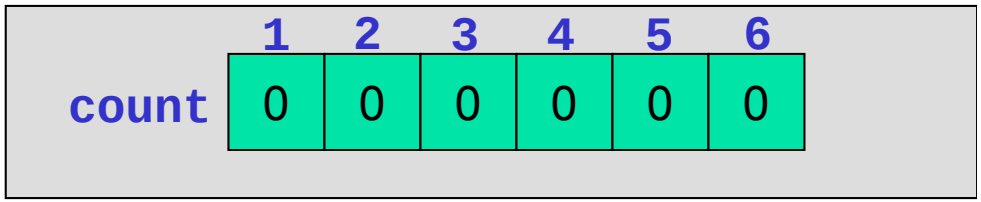
Repeat the following:

```
% roll the die

% increment correct "bin"
```

```
function count = rollDie(rolls)

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

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

    % Increment the appropriate bin

end

% Show histogram of outcome
```

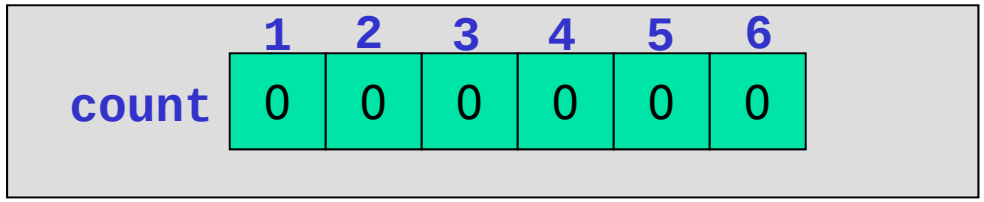


```
function count = rollDie(rolls)

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

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

end

% Show histogram of outcome
```

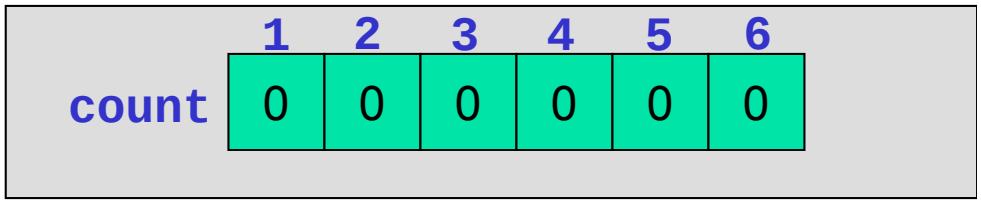


rollDieV1.m

```
% 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
```



```
function count = rollDie(rolls)

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

% 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 the rolling of 2 fair dice
totalOutcome= ???

A `ceil(rand*12)`

B `ceil(rand*11)+1`

C `floor(rand*11)+2`

D *2 of the above*

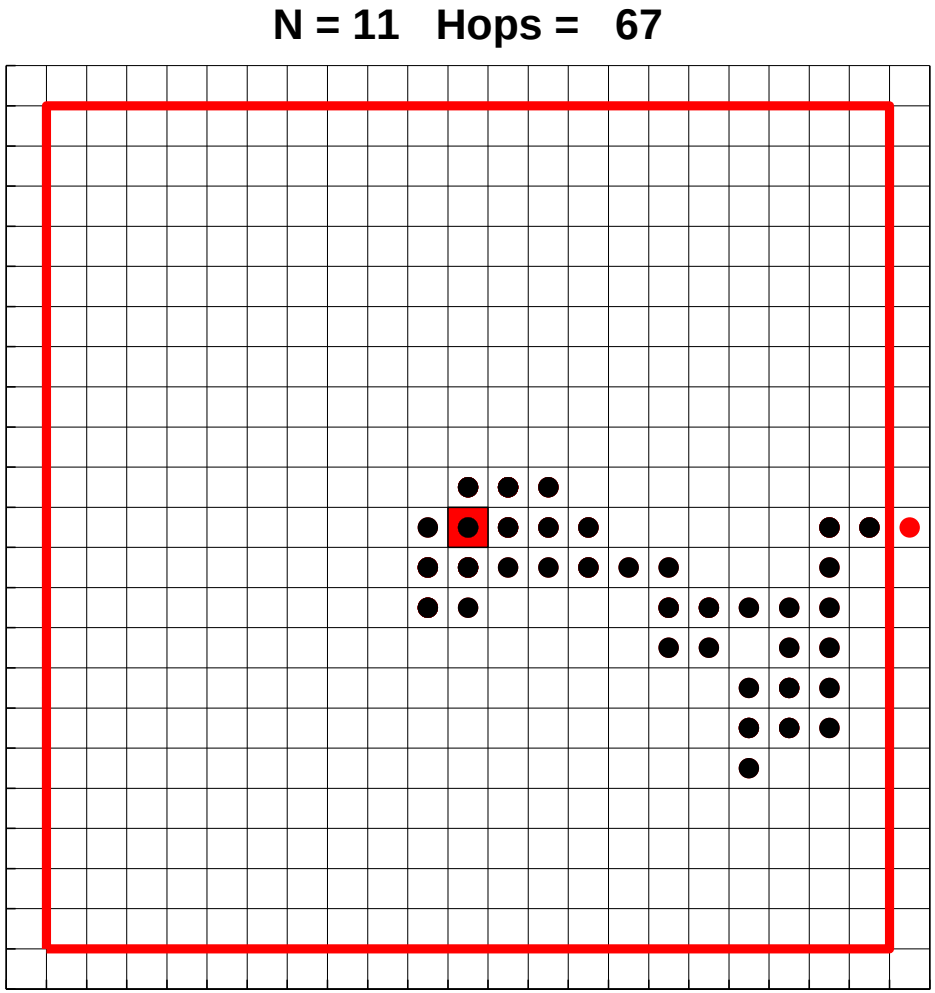
E *None of the above*

2-dimensional random walk

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

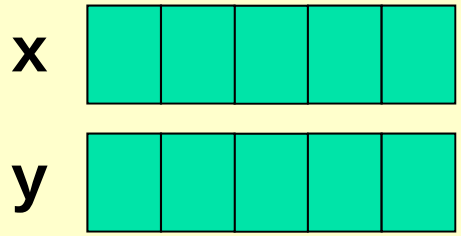
For each step, randomly choose between N,E,S,W and then walk one tile. Each tile is 1×1.

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.
```

By the end of the function …



```
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
```

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

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

while  abs(xc)<N && abs(yc)<N
    % Choose random dir, update xc,yc



    % Record new location in x, y

end
```

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

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

while  abs(xc)<N && abs(yc)<N
   % Choose random dir, update xc,yc



   % Record new location in x, y
   k=k+1;  x(k)=xc;  y(k)=yc;
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
```

See **RandomWalk2D.m**