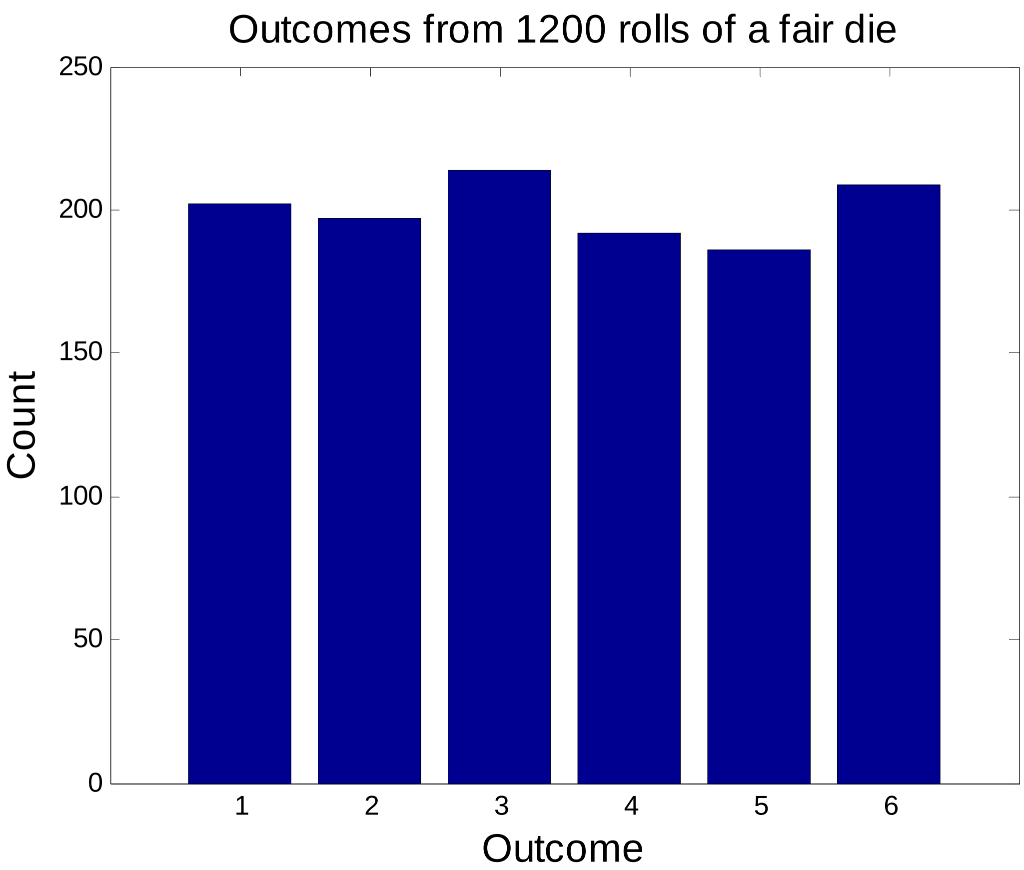
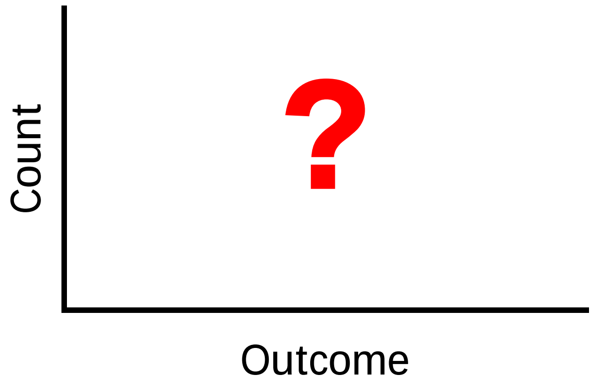
- Previous Lecture:
  - Executing a user-defined function
  - Function scope
  - Subfunction

- Today's Lecture:
  - 1-d array—vector
  - Probability and random numbers
  - Simulation using random numbers, vectors

- Announcement:
  - Project 3 due Monday10/3 at 11pm



Roll 2 fair dice



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

Execute the statement

```
y= foo(x);
```

```
function z = foo(a)
z= a + rand;
```

- Matlab looks for a function called foo (m-file called foo.m)
- Argument (value of x) is copied into function foo's local parameter
  - called "pass-by-value," one of several argument passing schemes used by programming languages
- Function code executes within its own workspace
- At the end, the function's output argument (value) is sent from the function to the place that calls the function. E.g., the value is assigned to y.
- Function's workspace is deleted
  - If foo is called again, it starts with a new, empty workspace

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

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)

b= 7:-2:0

c= [3 7 2 1]

d= [3; 7; 2]

e= d'
```

a	10	15	20	25	30

b	7	5	3	1

c	3	7	2	1

d	3
	7
	2

e	3	7	2

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], ‘-*’)
```

Line/marker
format

x-values
(a vector)

y-values
(a vector)

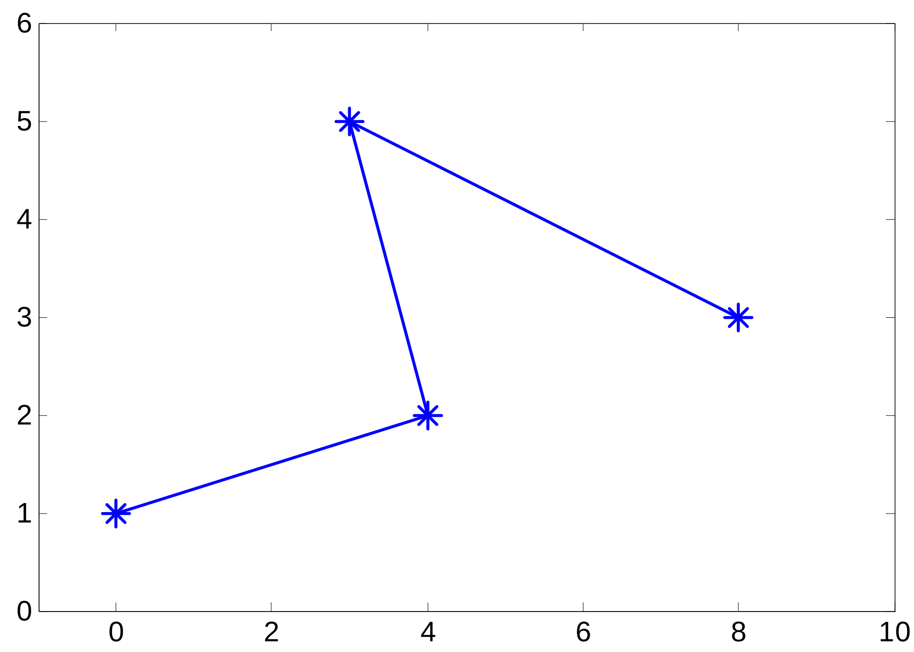
Making an x-y plot

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

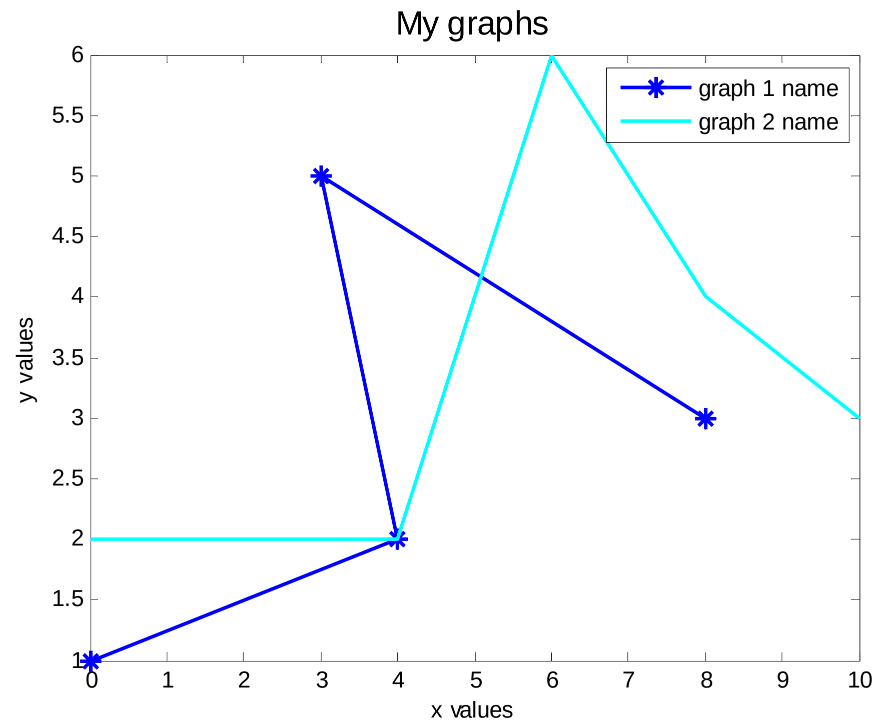
x-values
(a vector)

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)
```

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**	**92**	**87**	**0**	**90**	**82**
	1	*2*	*3*	*4*	*5*	*6*

Given the vector **`score`** …

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;
```

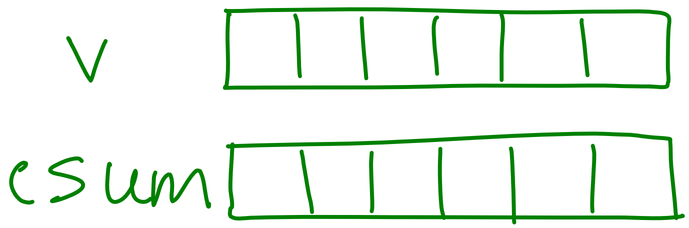
See **`plotComparison2.m`**

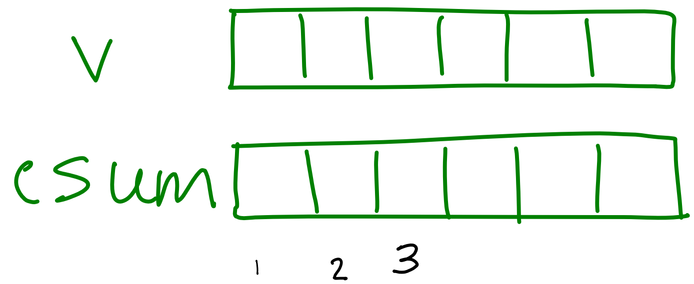
Example

- 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



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

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

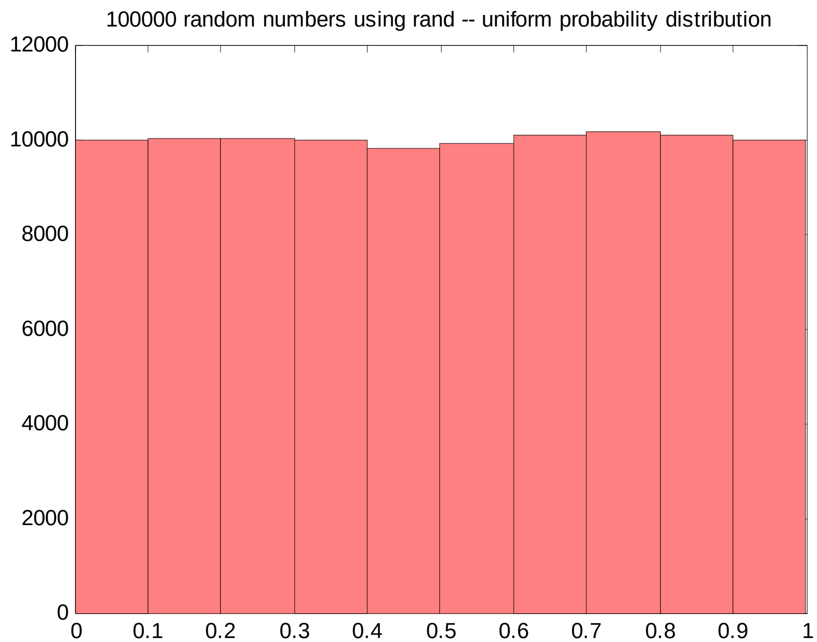
$$csum(4) = \underbrace{v(1) + v(2) + v(3)}_{csum(3)} + v(4)$$

```
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`	one random # in (0,1)
`6*rand`	one random # in (0,6)
`6*rand+1`	one random # in (1,7)

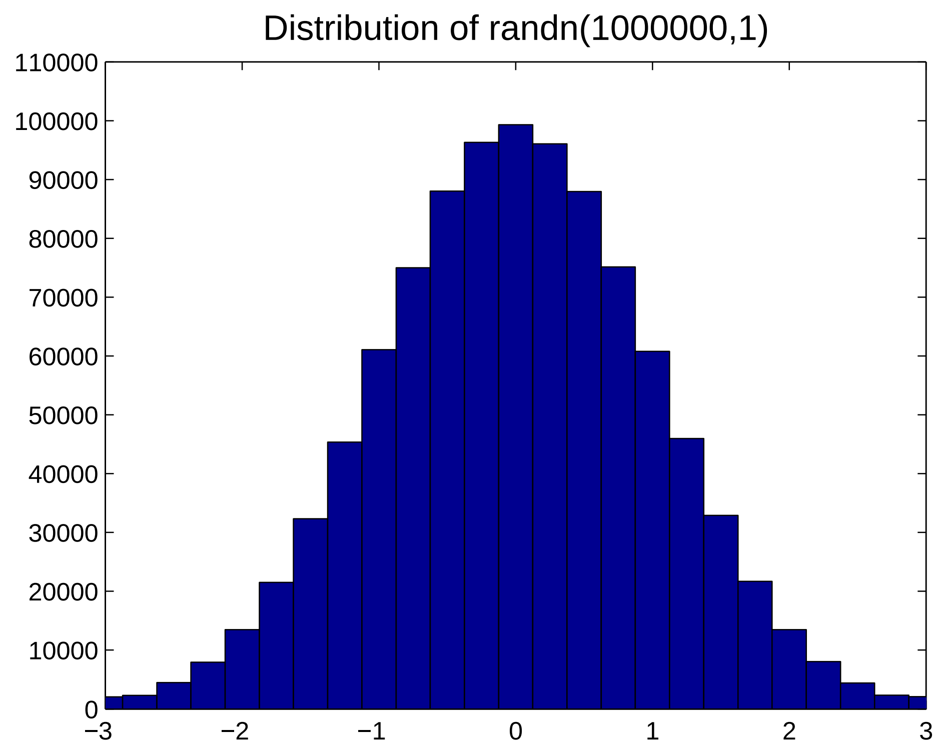


Uniform probability distribution in (0,1)

`rand`

Normal distribution with zero mean and unit standard deviation

`randn`



Simulate a fair 6-sided die

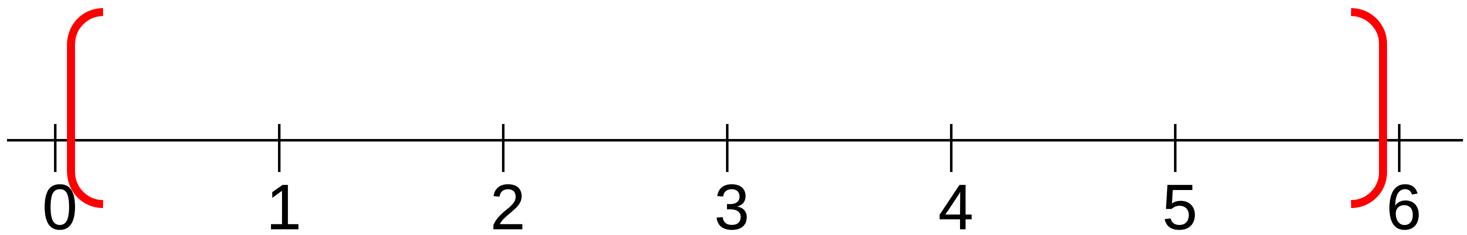
Which expression(s) below will give a random *integer* in [1..6] with equal likelihood?

A `round(rand*6)`

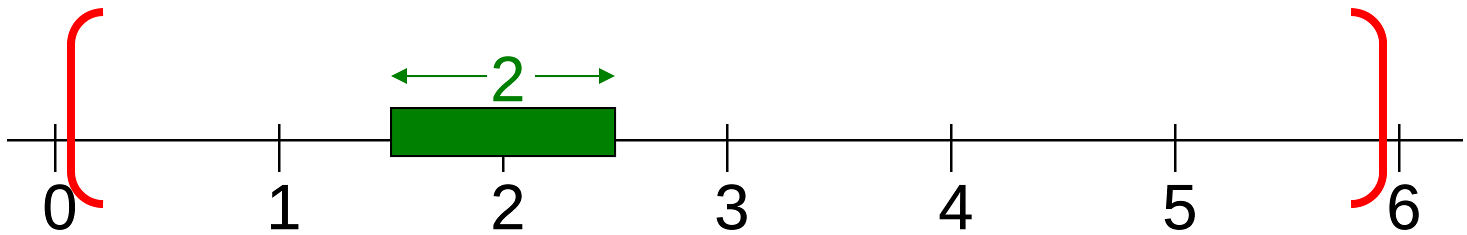
B `ceil(rand*6)`

C *Both expressions above*

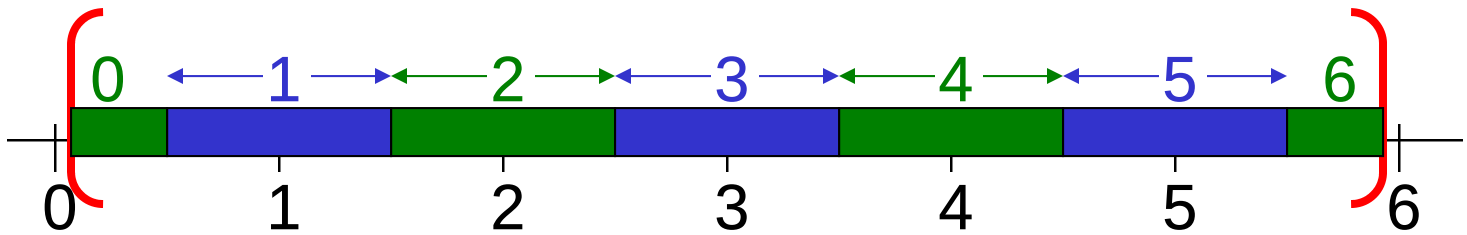
`(rand*6)`



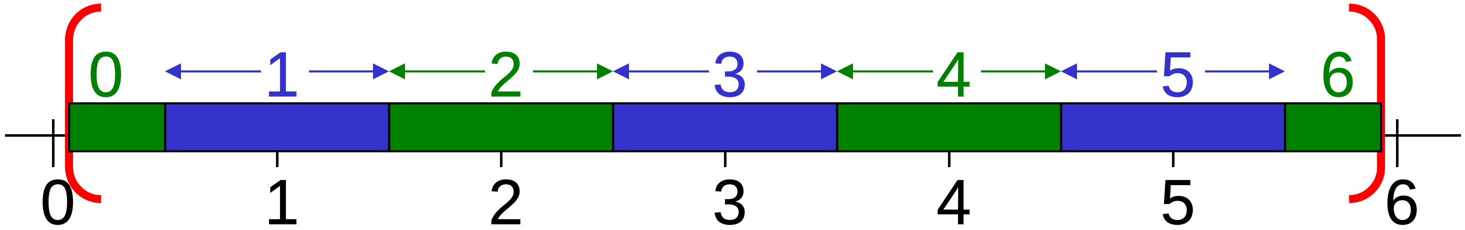
```
round(rand*6)
```



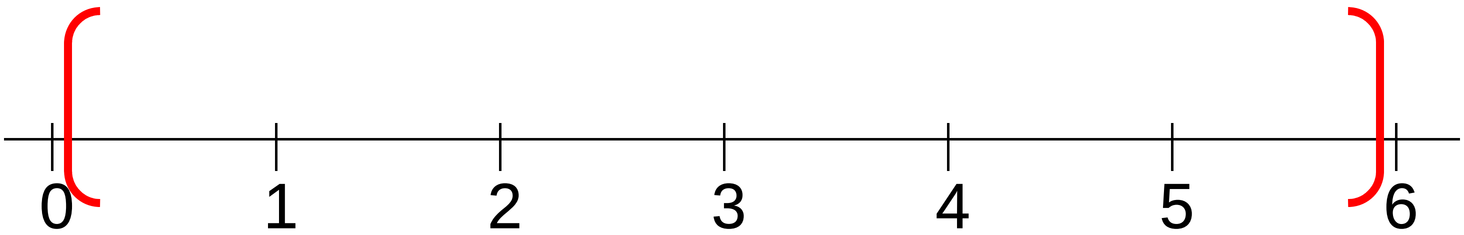
`round(rand*6)`



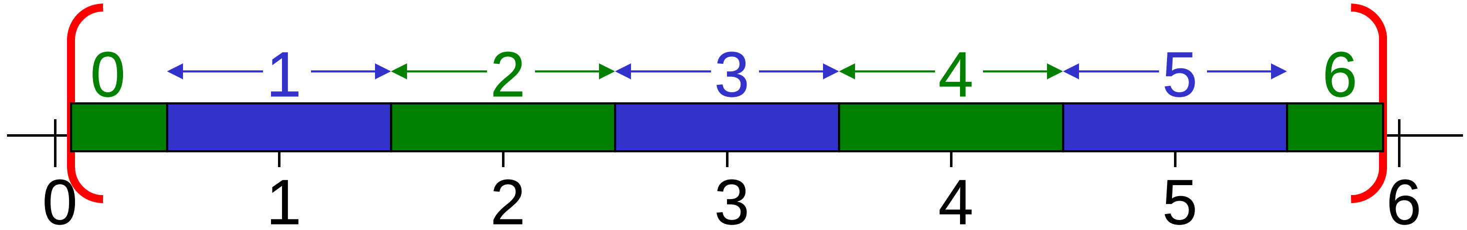
```
round(rand*6)
```



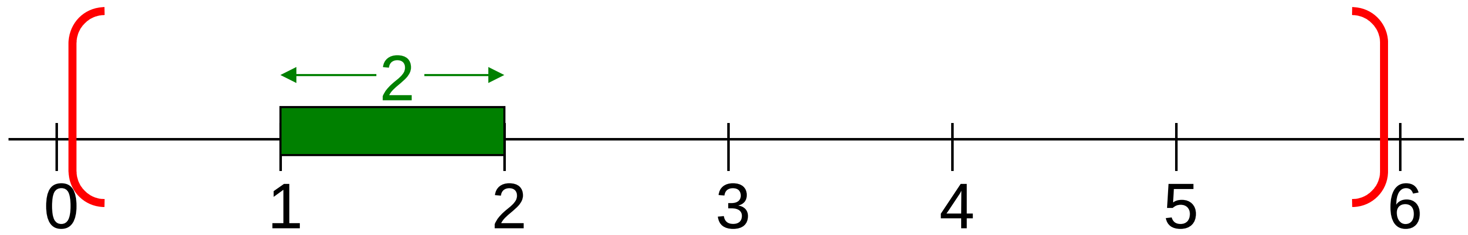

```
(rand*6)
```



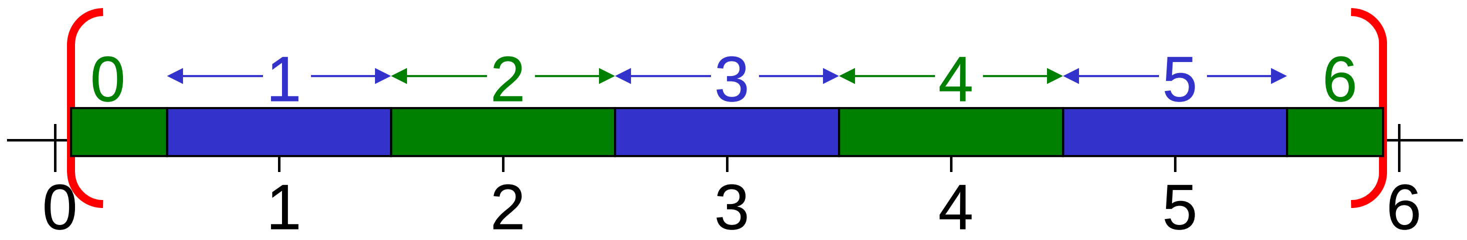
`round(rand*6)`



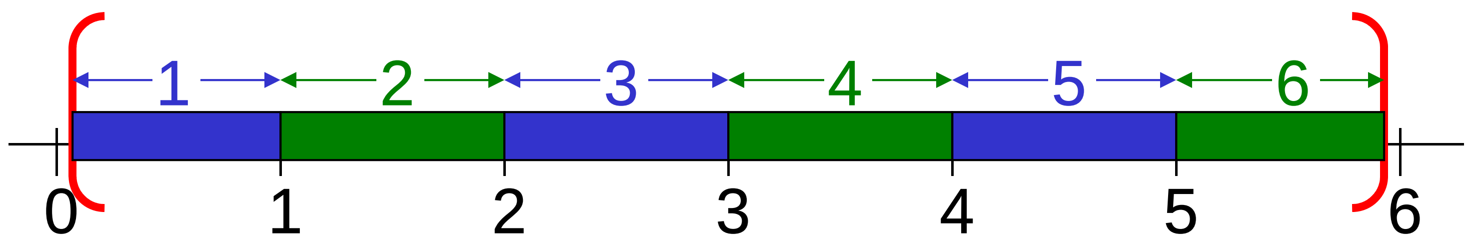
`ceil(rand*6)`



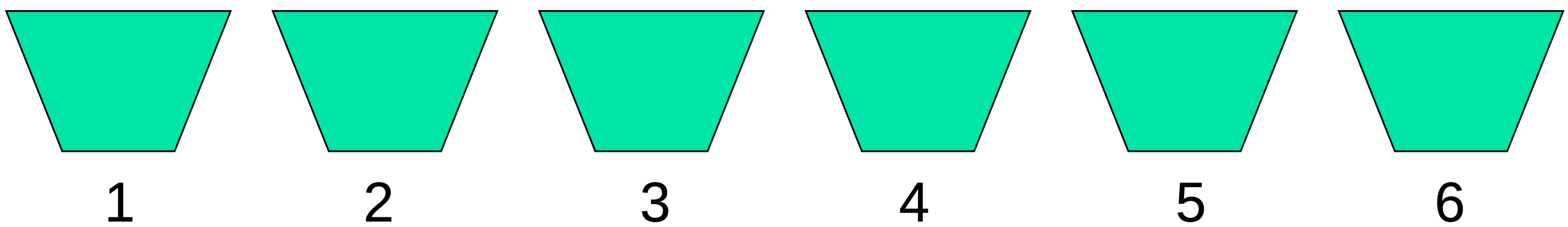
`round(rand*6)`



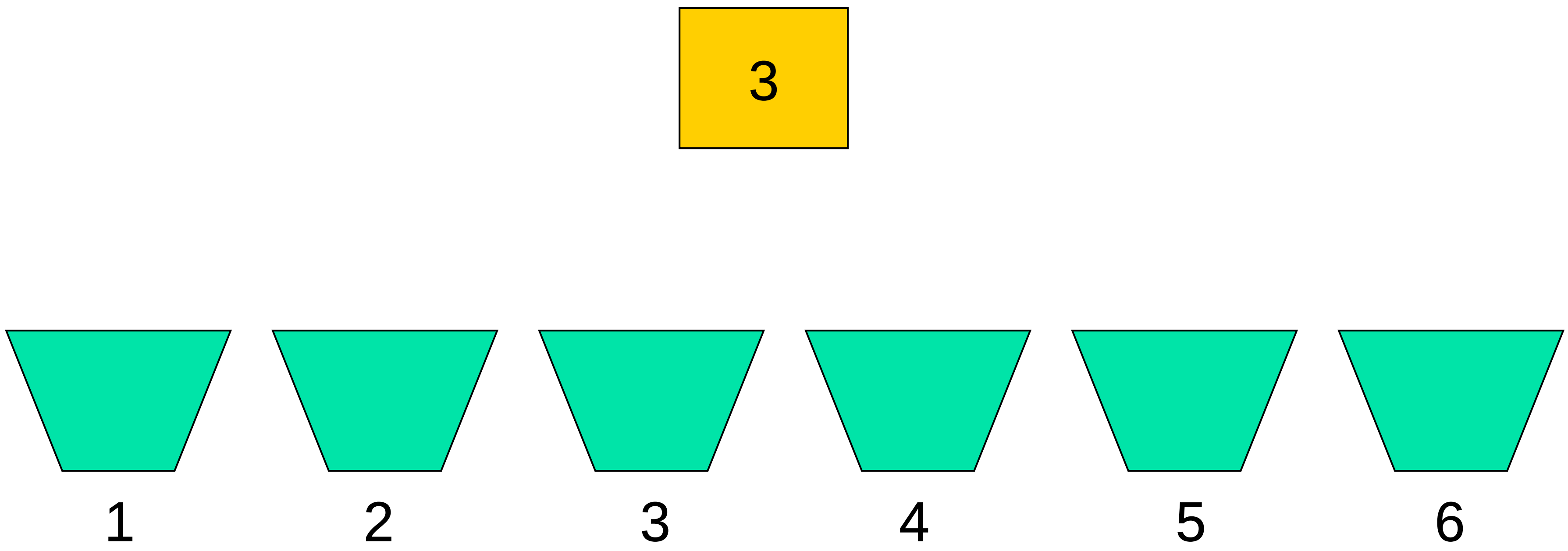
`ceil(rand*6)`



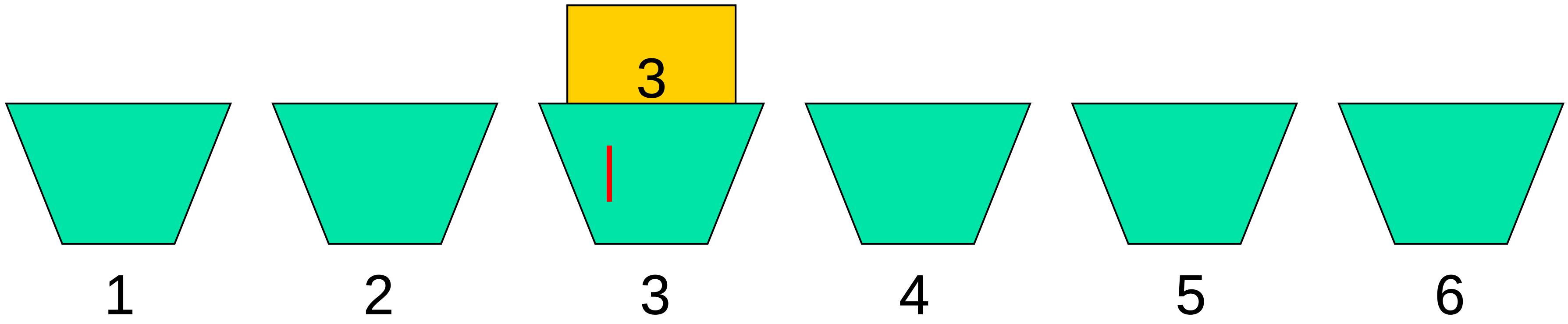
Possible outcomes from rolling a fair 6-sided die



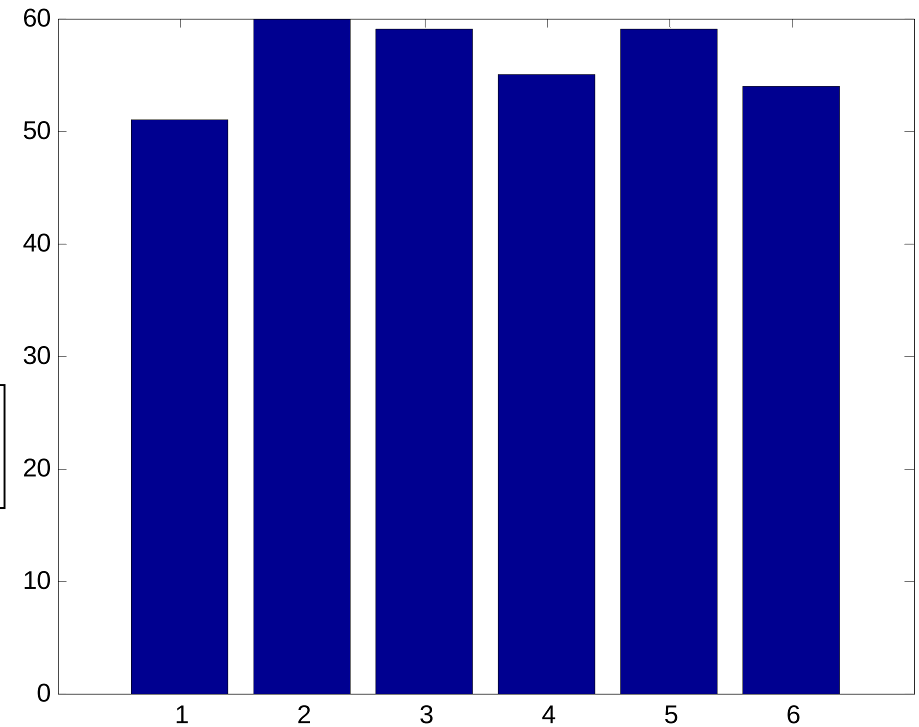
Simulation



Simulation



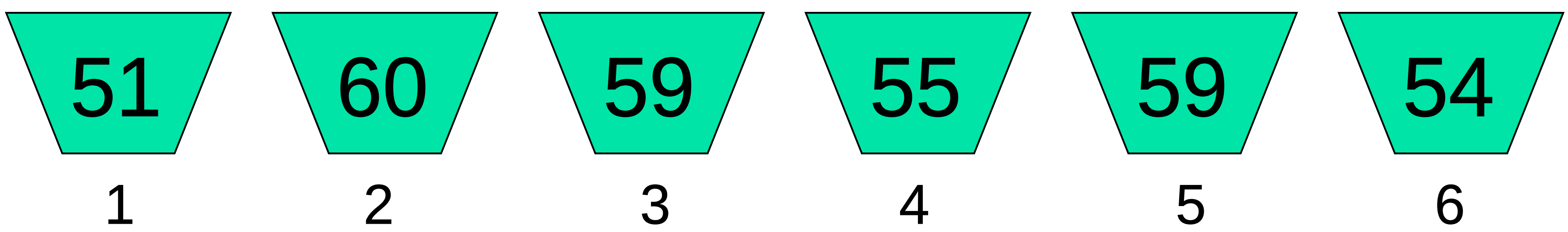
Simulation result



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);

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

    % Increment the appropriate bin

end

% Show histogram of outcome
```

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