- Previous Lecture:
  - User-defined functions
    - Examples with varying numbers of input and output parameters
    - Local memory space
- Today's Lecture:
  - Subfunctions
  - 1-d array—vector
  - More MATLAB graphics
  - Probability and random numbers
- Announcements:
  - Discussion section this week in the lab, UP B7
  - Please register your clicker online even if you've registered in class

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

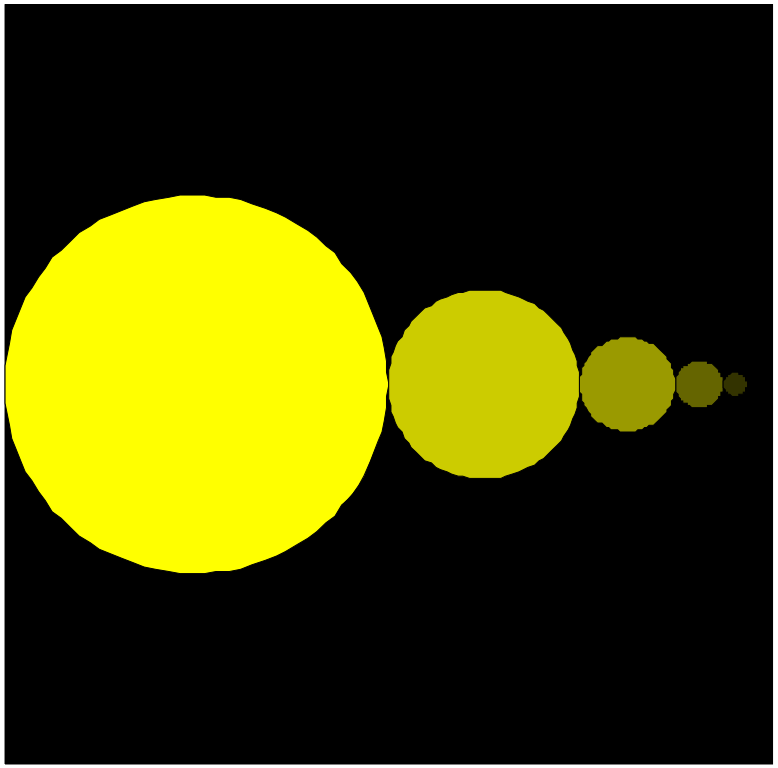
Execute the statement `y= foo(x)`

- 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

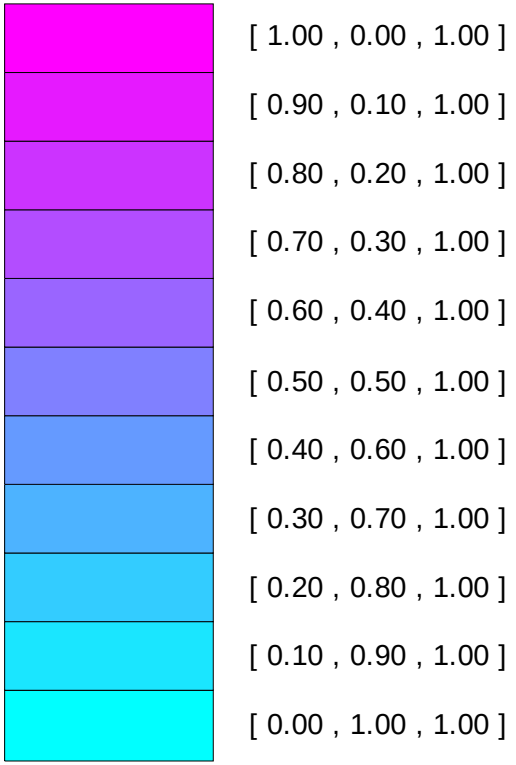
Subfunction

- 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

Graphics and color interpolation



Used given **`DrawDisk`**

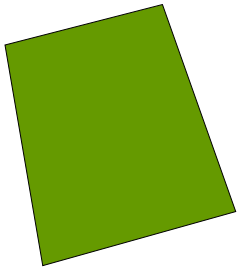


Learn about **`fill`**, **`text`** and practice working with *vectors*

Color computation

- Color is a 3-vector, sometimes called the RGB values
- Any color is a mix of red, green, and blue
- Example:

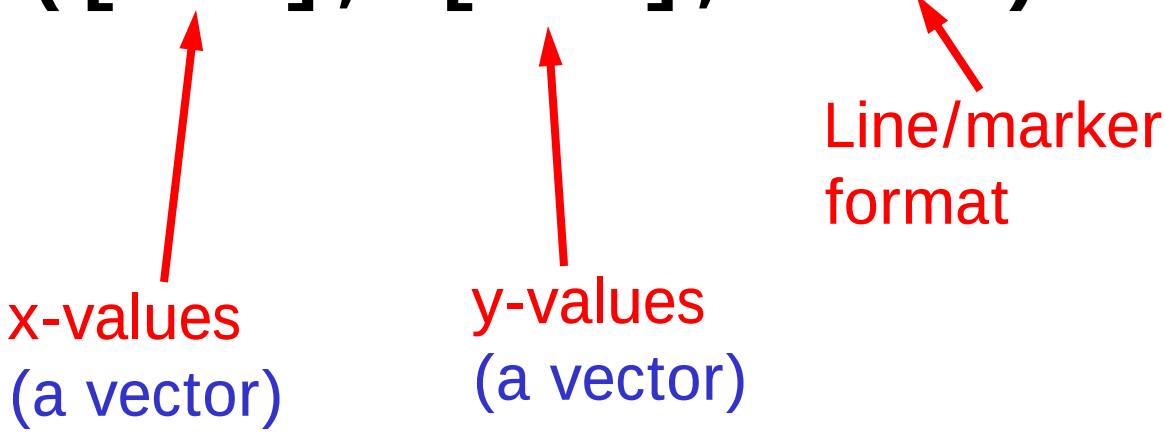
```
c= [0.4  0.6  0]
```



- Each component is a real value in [0,1]
- [0 0 0] is black
- [1 1 1] is white

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], '-*')
```



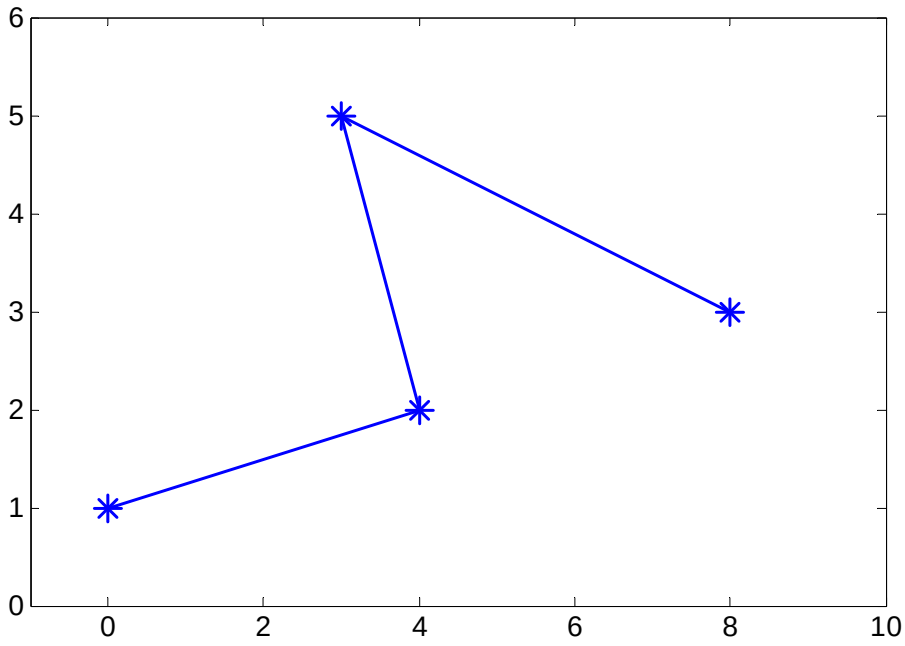
Making an x-y plot

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

x-values
(a vector)

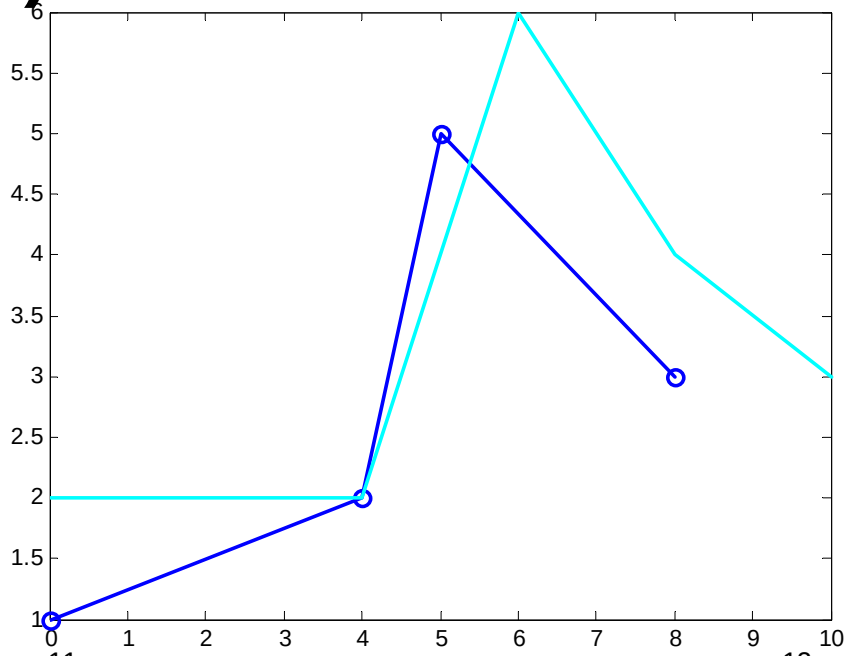
y-values
(a vector)

Line/marker
format



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

```
a= [0 4 3 8];
b= [1 2 5 3];
f= [0 4 6 8 10];
g= [2 2 6 4  3];
plot(a,b,'-*',f,g,'c')
```



Drawing a polygon (multiple line segments)

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

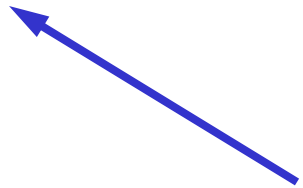
Fill in the missing vector values!

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)
```



A built-in function

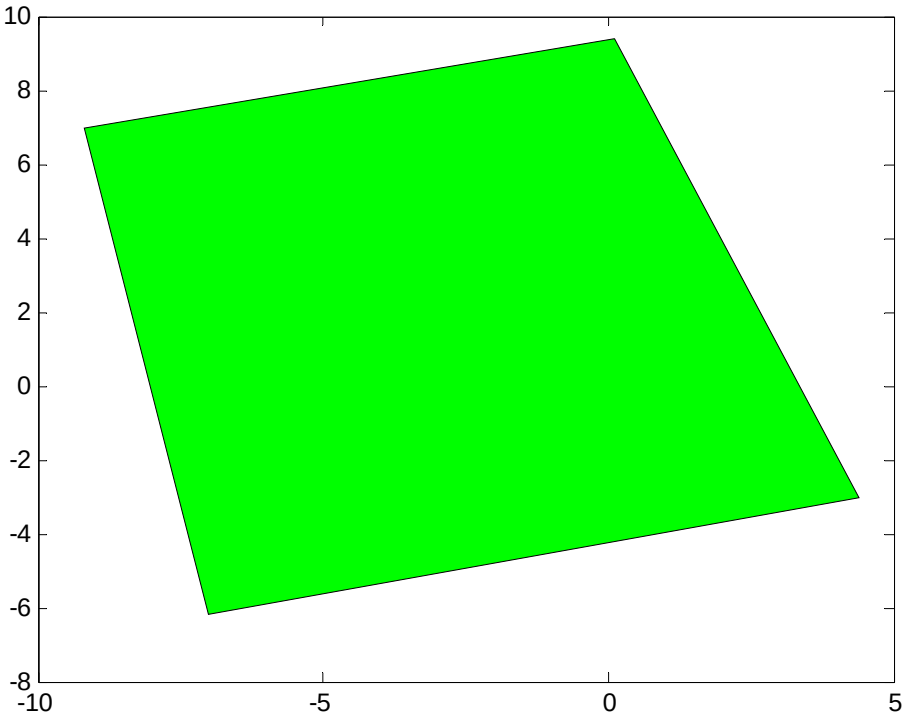
Coloring a polygon (fill)

```
% 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)
```

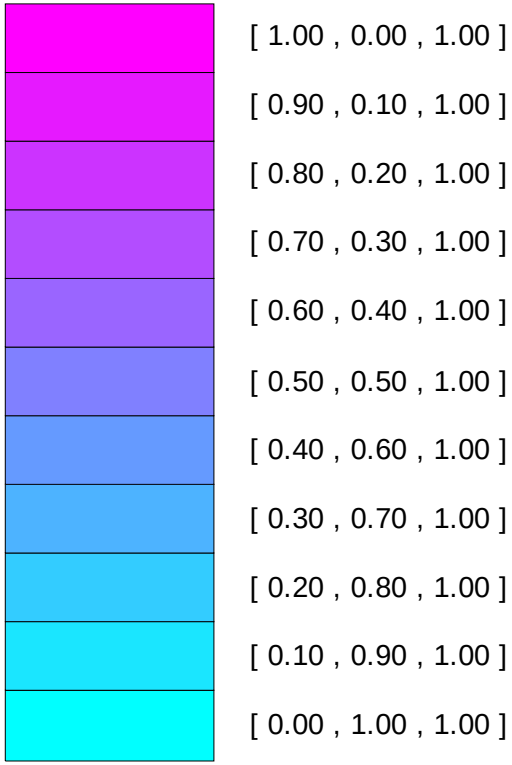
Built-in function **fill** actually does the "wrap-around" automatically.

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

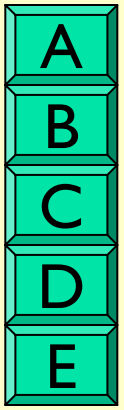
Can be a vector
(RGB values)



```
function paintChips(c1,c2,n)
% n tiles from color c1 to c2



for k= 0:n-1
    % Compute color of kth tile
    f= ???
    v= (1-f)*c1 + f*c2;
    % Draw kth tile



end
```

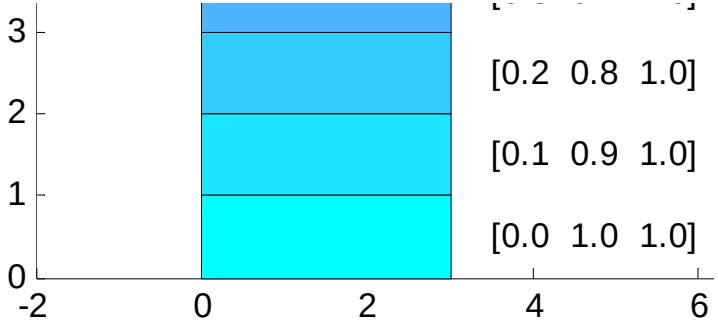


```
function paintChips(c1,c2,n)
% n tiles from color c1 to c2



for k= 0:n-1
    % Compute color of kth tile
    f= ???
    v= (1-f)*c1 + f*c2;
    % Draw kth tile




end
```

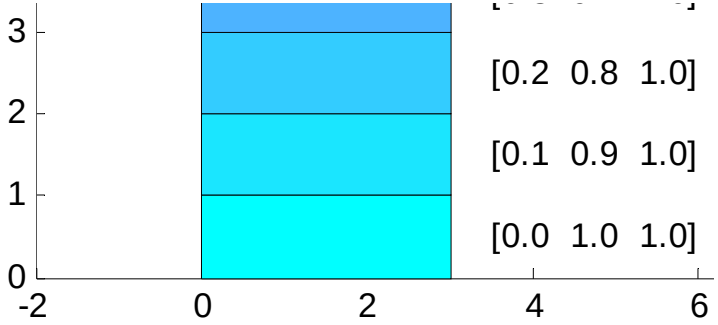


k/n
k/(n-1)
(k-1)/n
(k-1)/(n-1)
(k-1)/(n+1)

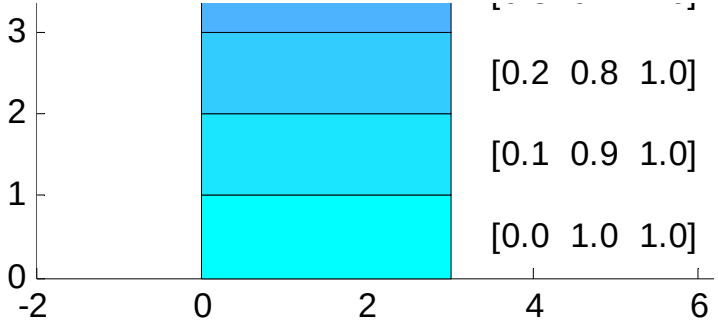
```
function paintChips(c1,c2,n)
% n tiles from color c1 to c2




for k= 0:n-1
    % Compute color of kth tile
    f= ???
    v= (1-f)*c1 + f*c2;
    % Draw kth tile




end
```



```
function paintChips(c1,c2,n)
% n tiles from color c1 to c2

x= [0 3 3 0];
y= [0 0 1 1];
for k= 0:n-1
    % Compute color of kth tile
    f= k/(n-1);
    v= (1-f)*c1 + f*c2;
    % Draw kth tile
    fill(______, _______, v)



end
```

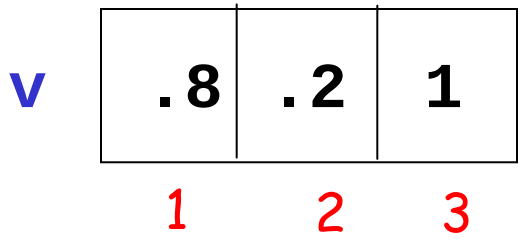


```
function paintChips(c1,c2,n)
% n tiles from color c1 to c2

x= [0 3 3 0];
y= [0 0 1 1];
for k= 0:n-1
    % Compute color of kth tile
    f= k/(n-1);
    v= (1-f)*c1 + f*c2;
    % Draw kth tile
    fill(x, y+k, v)
    text(3.5, k+.5, ...
         sprintf('[%.1f %.1f %.1f]',...
         v(1),v(2),v(3) )
end
```

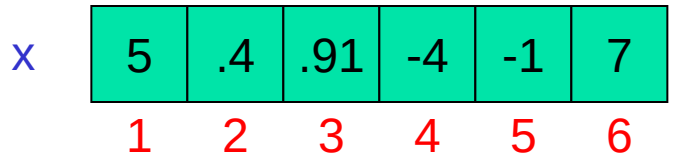


1-d array: **vector**

- 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



Array index starts at 1

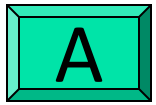


Let k be the index of vector x, then

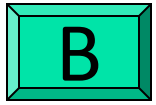
- k must be a positive integer
- 1<= k <= length(x)
- To access the k^{th} element: x(k)

Simulate a fair 6-sided die

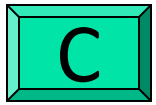
Which expression(s) below will give a random *integer* in [1..6] with equal likelihood?



`round(rand(1)*6)`

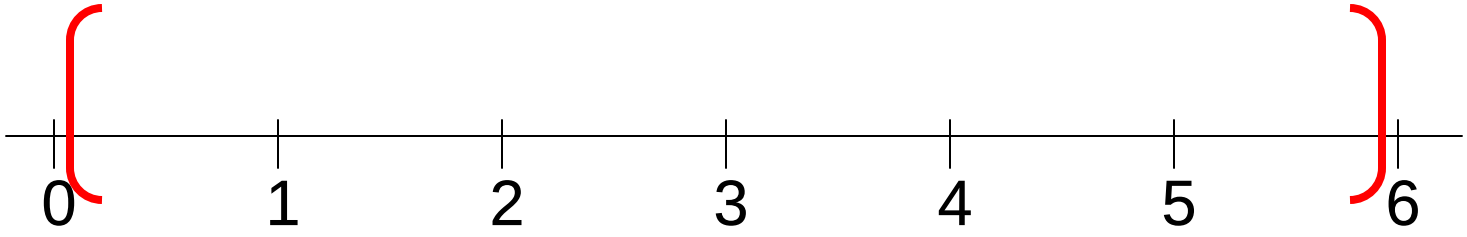


`ceil(rand(1)*6)`

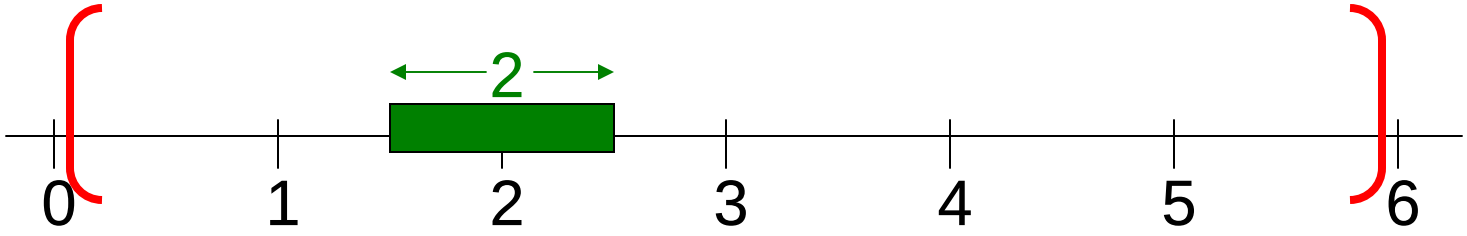


Both expressions above

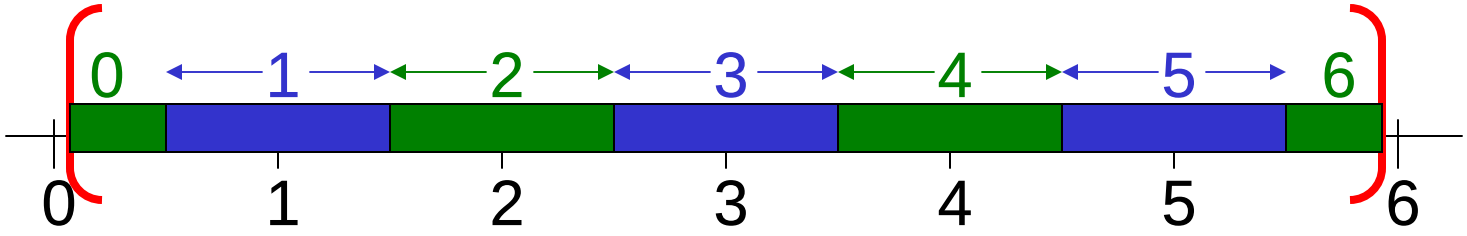
(rand(1)*6)



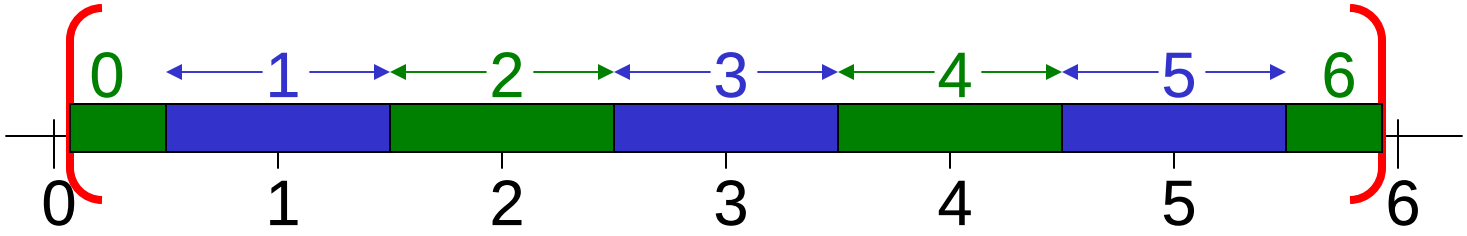
`round(rand(1)*6)`



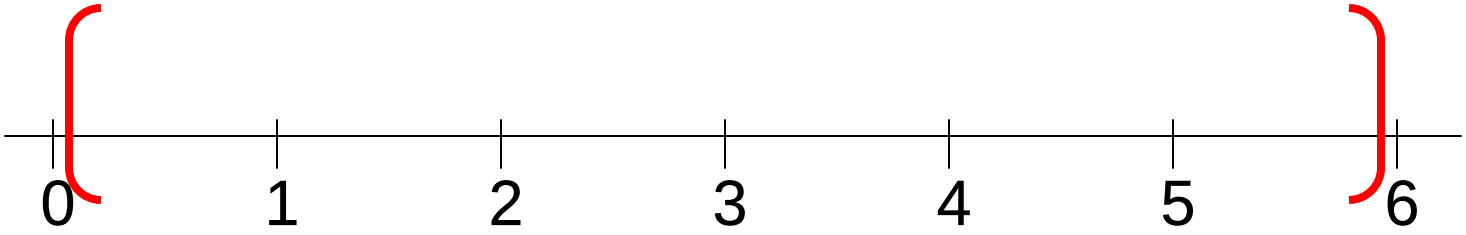
`round(rand(1)*6)`



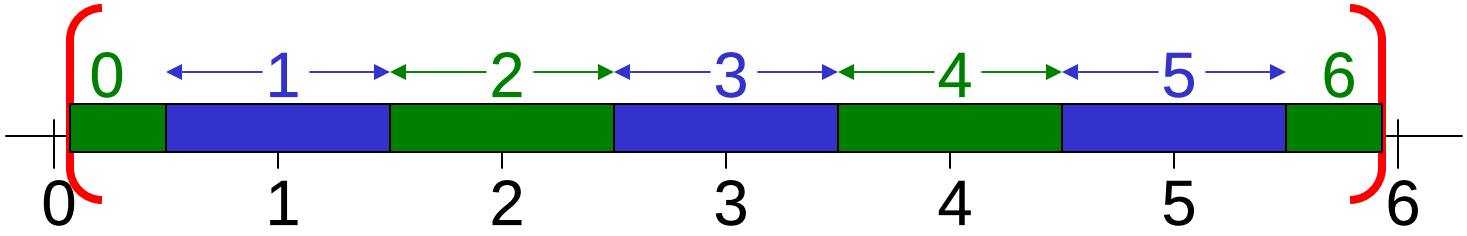
`round(rand(1)*6)`



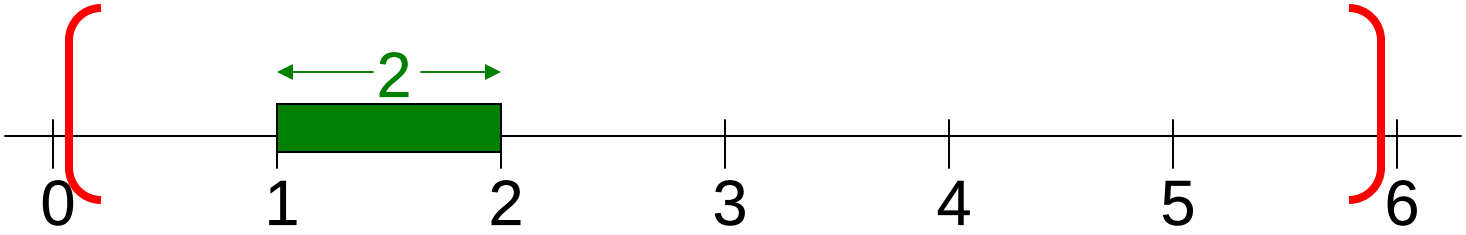
`(rand(1)*6)`



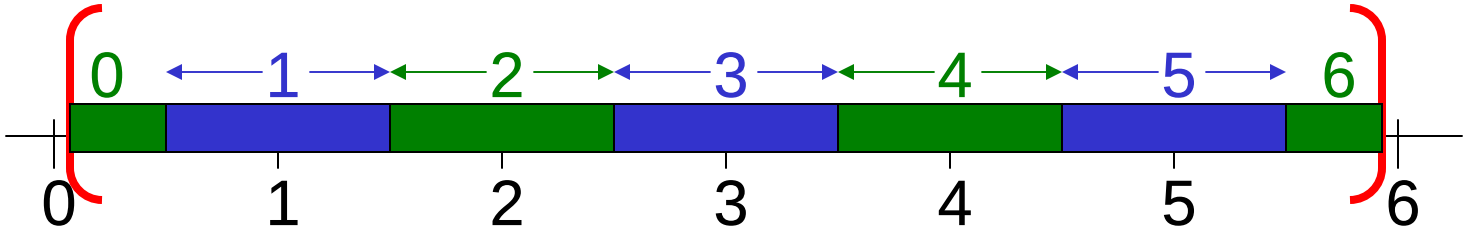
`round(rand(1)*6)`



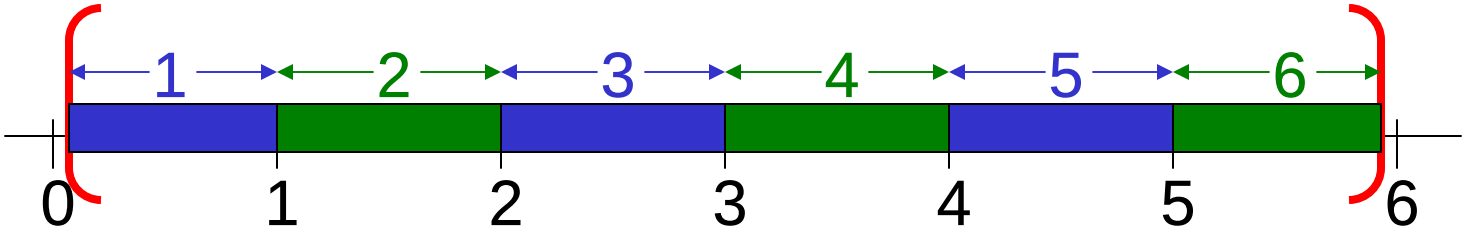
`ceil(rand(1)*6)`



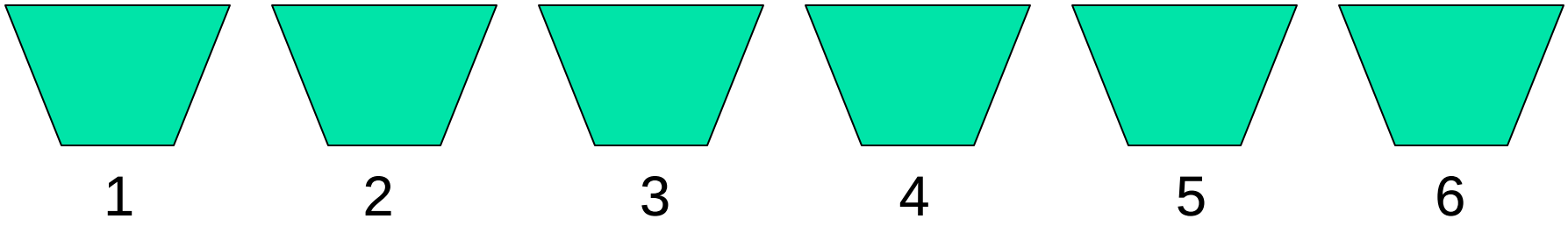
`round(rand(1)*6)`



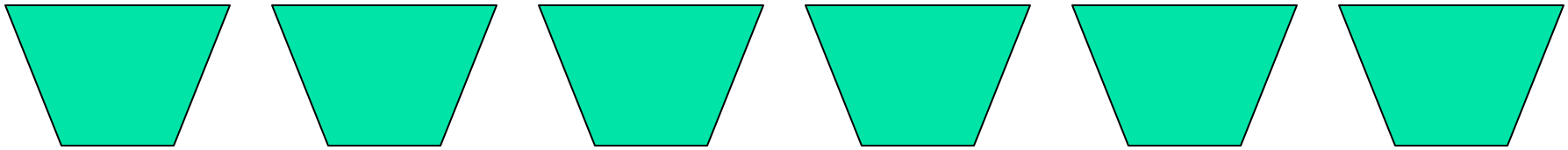
`ceil(rand(1)*6)`



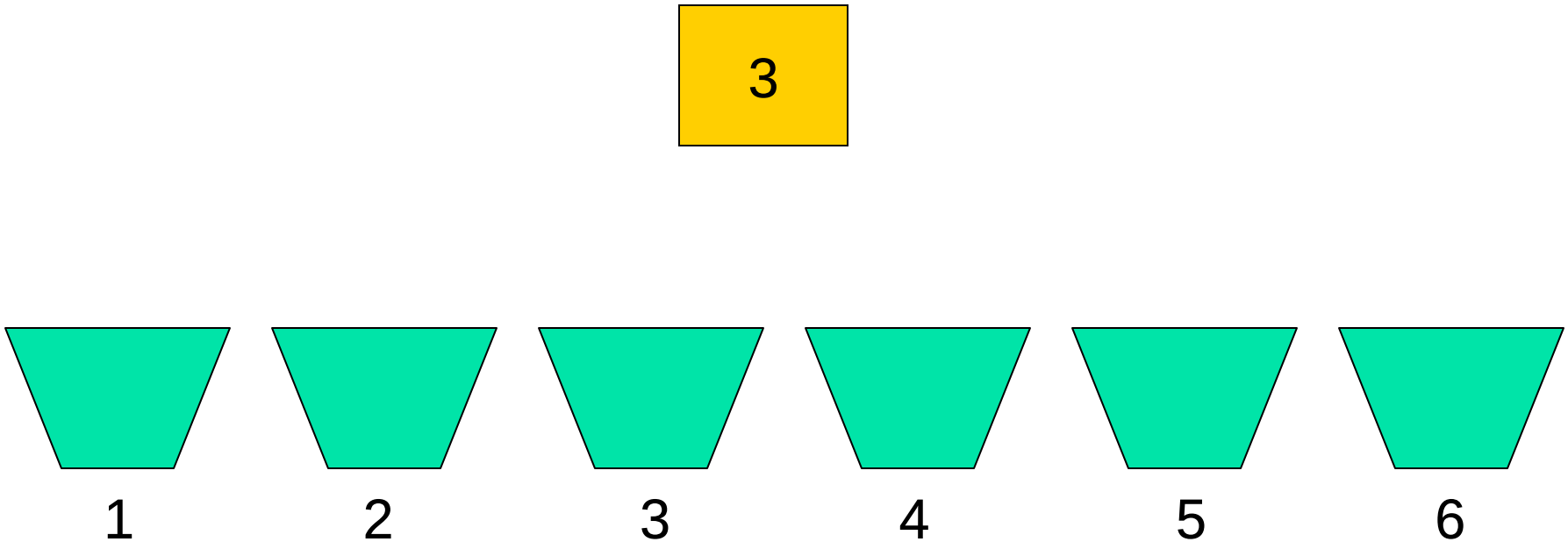
Possible outcomes from rolling a fair 6-sided die



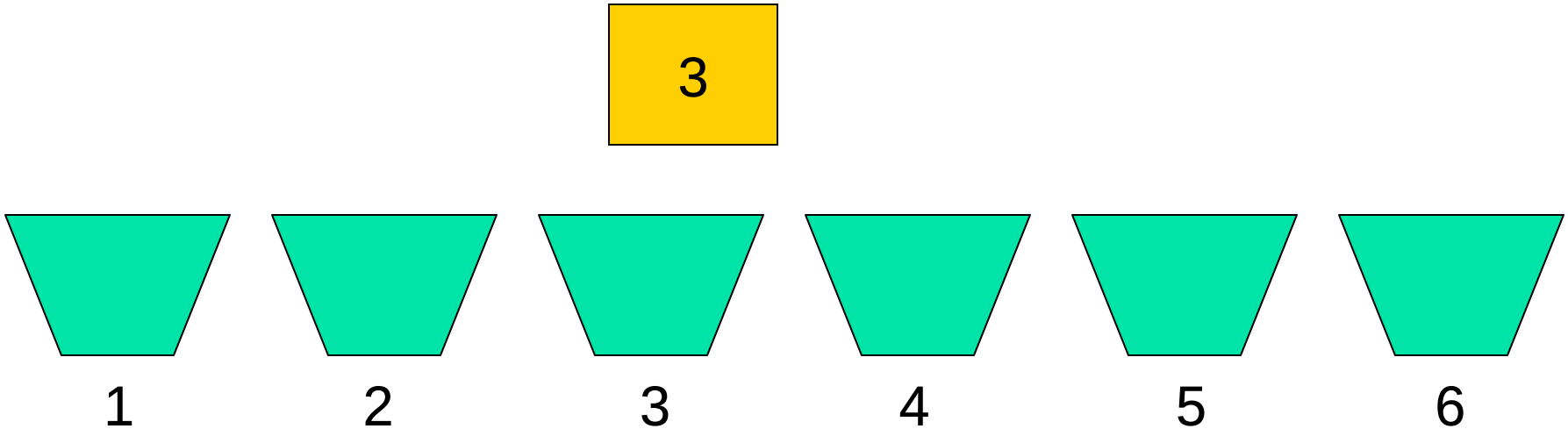
Simulation



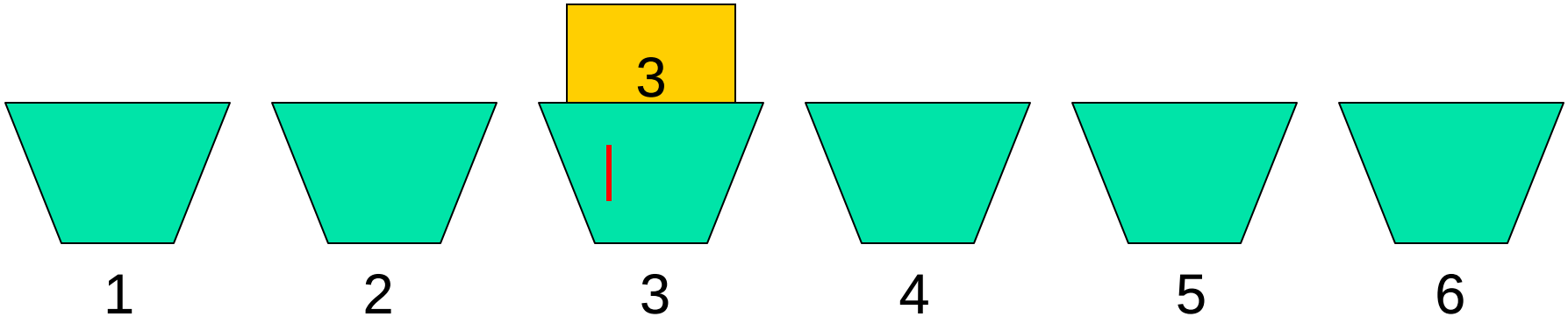
Simulation



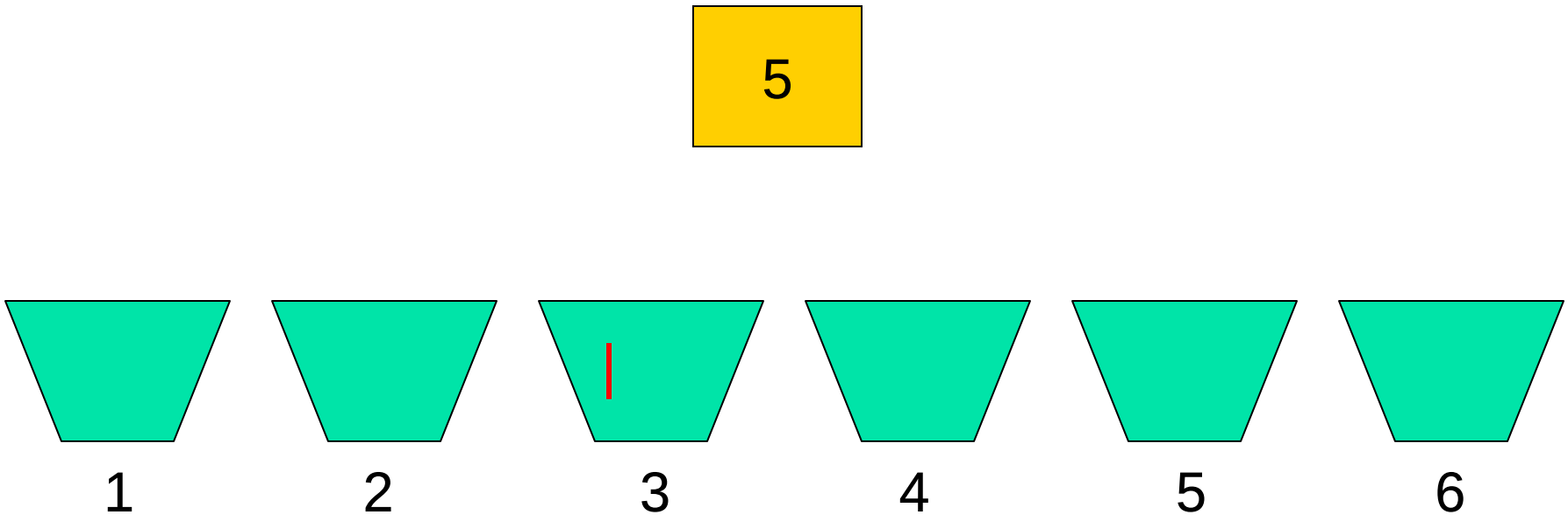
Simulation



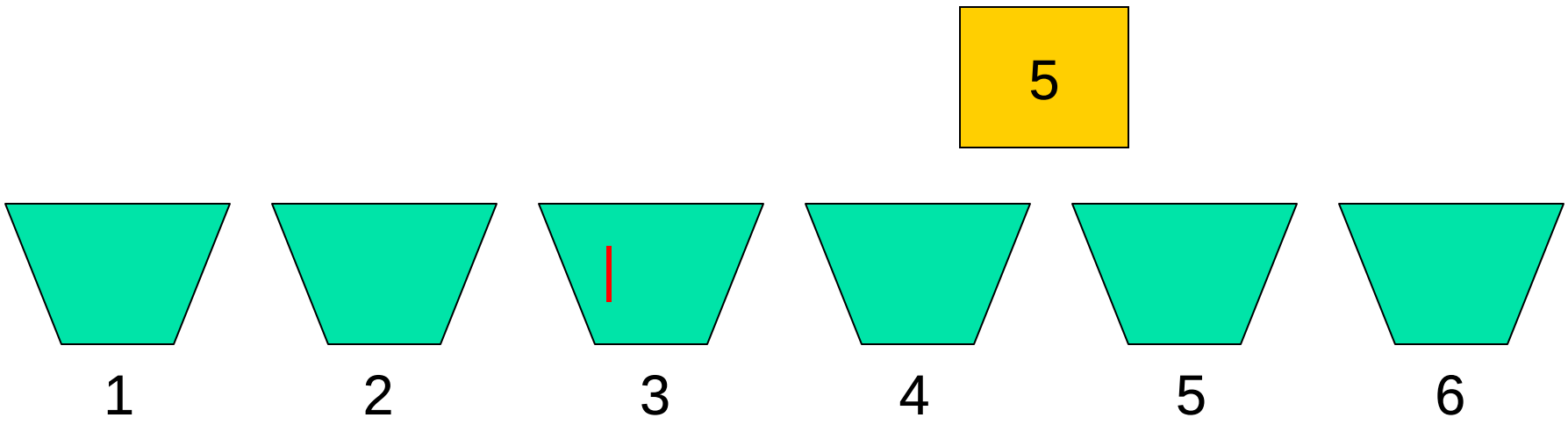
Simulation



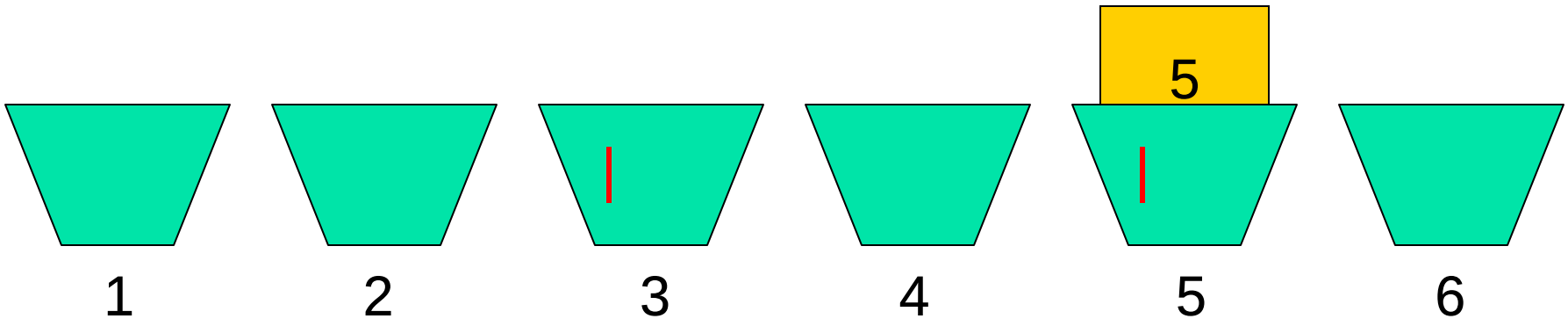
Simulation



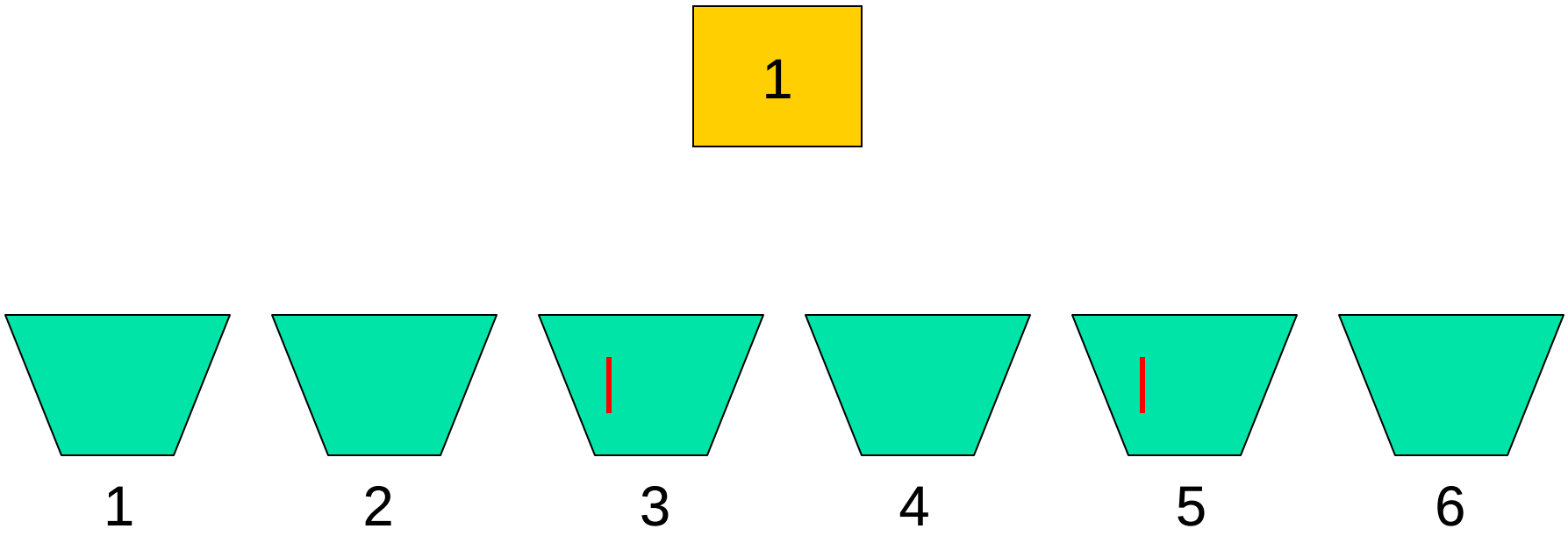
Simulation



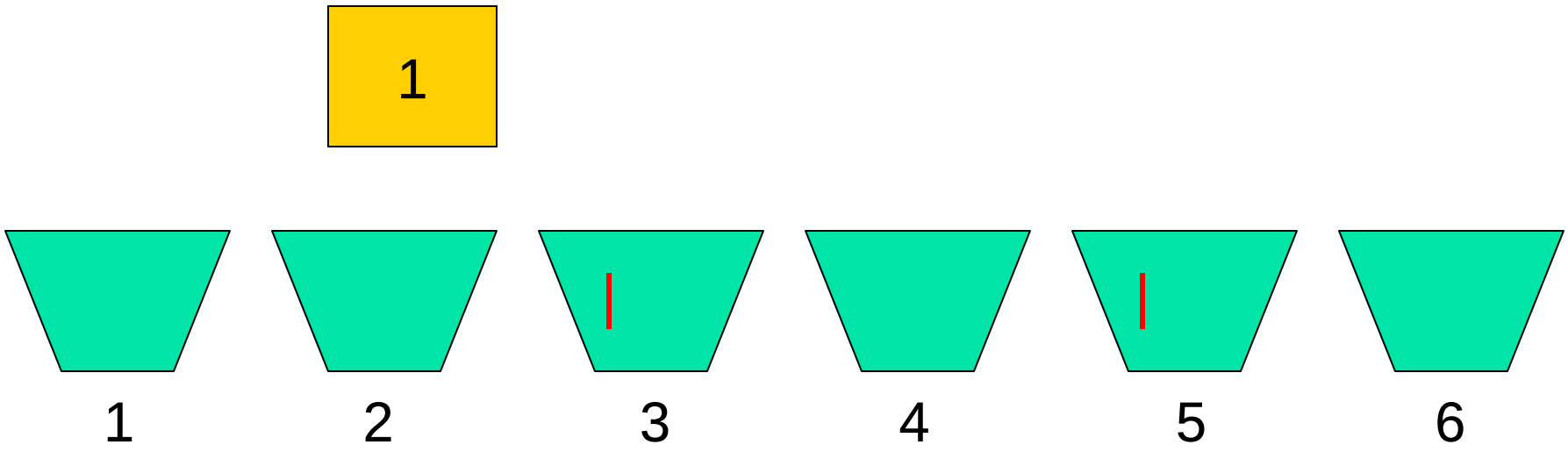
Simulation



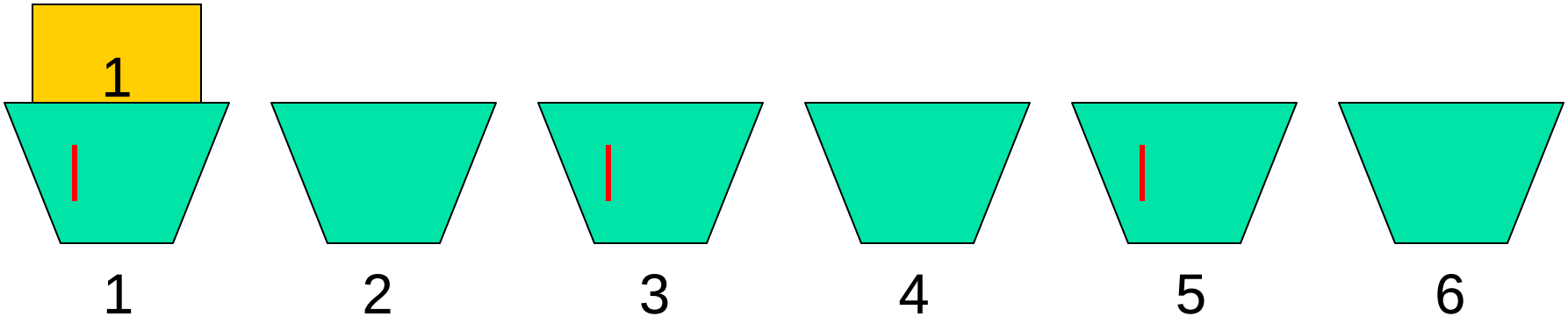
Simulation



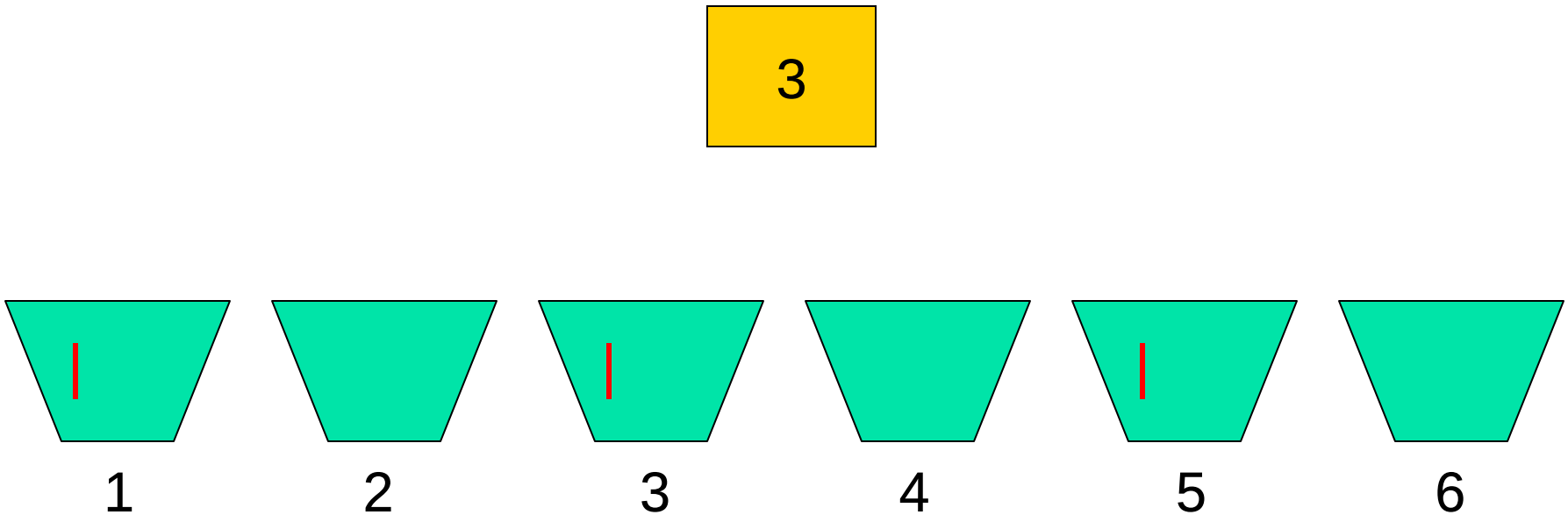
Simulation



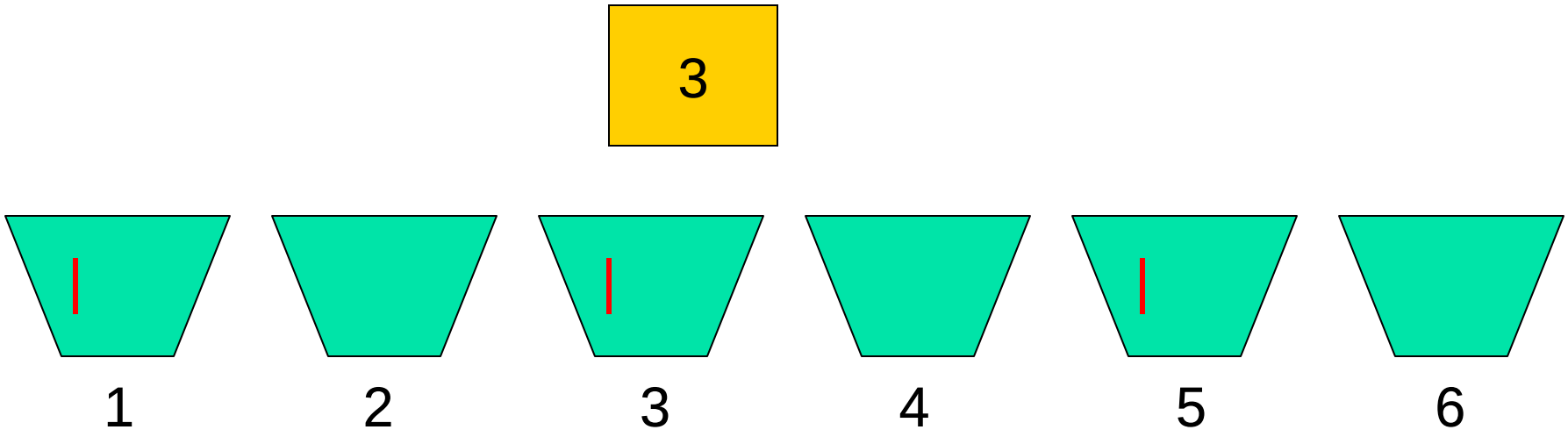
Simulation



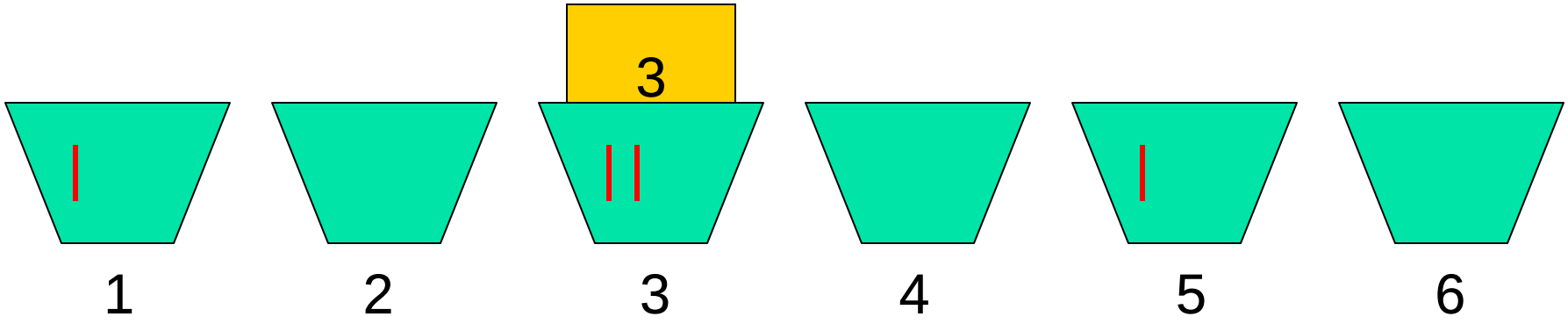
Simulation



Simulation



Simulation



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
```

```
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(1)*FACES);
    % Increment the appropriate bin

end

% Show histogram of outcome
```

```
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(1)*FACES);
    % Increment the appropriate bin
    count(face)= count(face) + 1;
end

% Show histogram of outcome
```