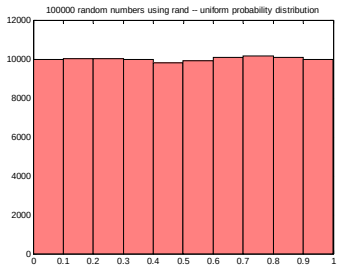
- Previous Lecture:
  - User-defined functions
    - Examples with varying numbers of input and output parameters
    - Local memory space
- Today's Lecture:
  - Probability and random numbers
  - 1-d array—vector
  - More MATLAB graphics
- Announcement:
  - Section this week in the computer labs
  - P3 posted, due 3/6 at 6pm

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

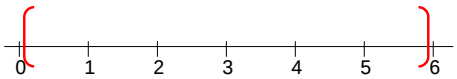


Simulate a fair 6-sided die

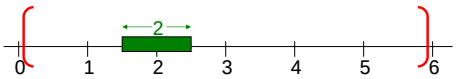
Which expression(s) below will give a random *integer* in [1..6] with equal likelihood?

A **`round(rand(1)*6)`**
B **`ceil(rand(1)*6)`**
C *Both expressions above*

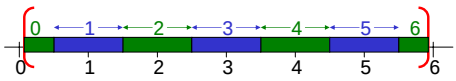
`(rand(1)*6)`



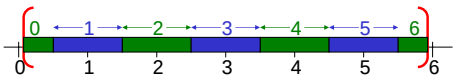
`round(rand(1)*6)`



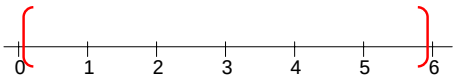
round(rand(1)*6)



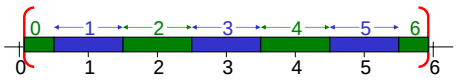
round(rand(1)*6)



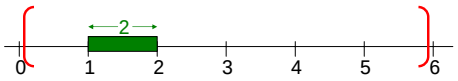
(rand(1)*6)



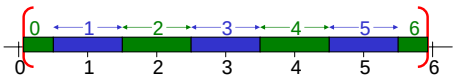
round(rand(1)*6)



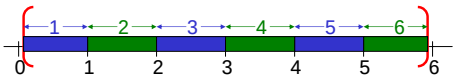
ceil(rand(1)*6)



round(rand(1)*6)

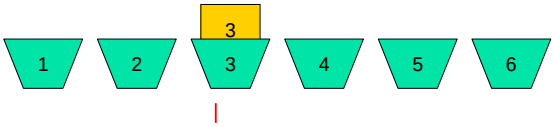


ceil(rand(1)*6)

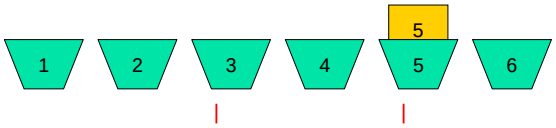


Possible outcomes from rolling a fair 6-sided die

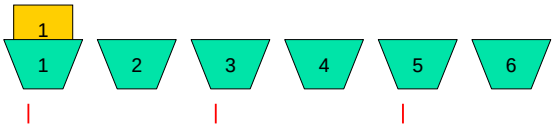
Simulation



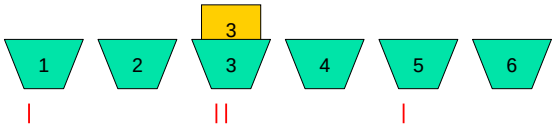
Simulation



Simulation



Simulation



Algorithm

```
% roll a die

% increment correct "bin"
```


rollDieV1.m

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

A **ceil(rand(1)*12)**
B **ceil(rand(1)*11)+1**
C **floor(rand(1)*11)+2**
D *2 of the above*
E *None of the above*

Discover the answer in section this week!

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

score	93	92	87	0	90	82
	1	2	3	4	5	6


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

Improve the implementation in section this week!

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

count | 0 | 0 | 0 | 0 | 0 | 0 |

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)


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

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

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;
```


A few different ways to create a vector
(See Lab 6 for more!)

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

count: 0 0 0 0 0 0

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

x: 10 15 20 25 30

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

y: 3 7 2 1

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!

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)
```

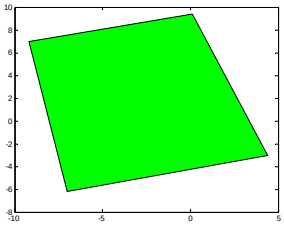
A built-in function

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')
```



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 white
- [1 1 1] is black

Let's *compute* colors!

Show "all combinations" of red and blue

- Assume some kind of granularity—discretize the color value range for red and blue
- Assume no contribution from green (set to 0)

Program development:

- Compute the color first; worry about drawing later
- Decide on granularity, say, Δ=.25


```
% All combinations of R and B

gran= 0.25;  %granularity
```



```
% All combinations of R and B

gran= 0.25;  %granularity
For all red values

    Set color vector
```



```
% All combinations of R and B

gran= 0.25;  %granularity
for r= 0:gran:1

        c= [? 0 ?];


end
```

```
% All combinations of R and B

gran= 0.25;  %granularity
for r= 0:gran:1

        c= [r 0 ?];


end
```



```
% All combinations of R and B

gran= 0.25;  %granularity
for r= 0:gran:1
    For all blue values
        c= [r 0 ?];


end
```



```
% All combinations of R and B

gran= 0.25;  %granularity
for r= 0:gran:1
    for b= 0:gran:1
        c= [r 0 b];

    end
end
```



```
% All combinations of R and B

gran= 0.25;  %granularity
for r= 0:gran:1
    for b= 0:gran:1
        c= [r 0 b];
        % Add code to display color
    end
end
```


RBCombinations.m

Things to consider/try on the color computation problem

- The granularity was the programmer's choice
- Choosing how to display the colors was a design problem!
- What if you want compute "all combinations" of the R, G, and B values? How would the program change?
- Another design problem: how to show all color combinations of the ***3***-vector on a ***2***-dimensional plot?

Prelim 1

- Q1: program trace (assignment, conditional, for-loop) ☺
- Q2: rand and representation ☺
  if, elseif, nesting (tough question!!) 😐
- Q3: simulation, while-loop 😐☹
- Q4: for-loop ☺☺☺☺
- Q5: nested for-loops 😐☹

  - Median 86
  - Mean 82.2; Standard Deviation 14.0

February 26, 2008 Lecture 11 55