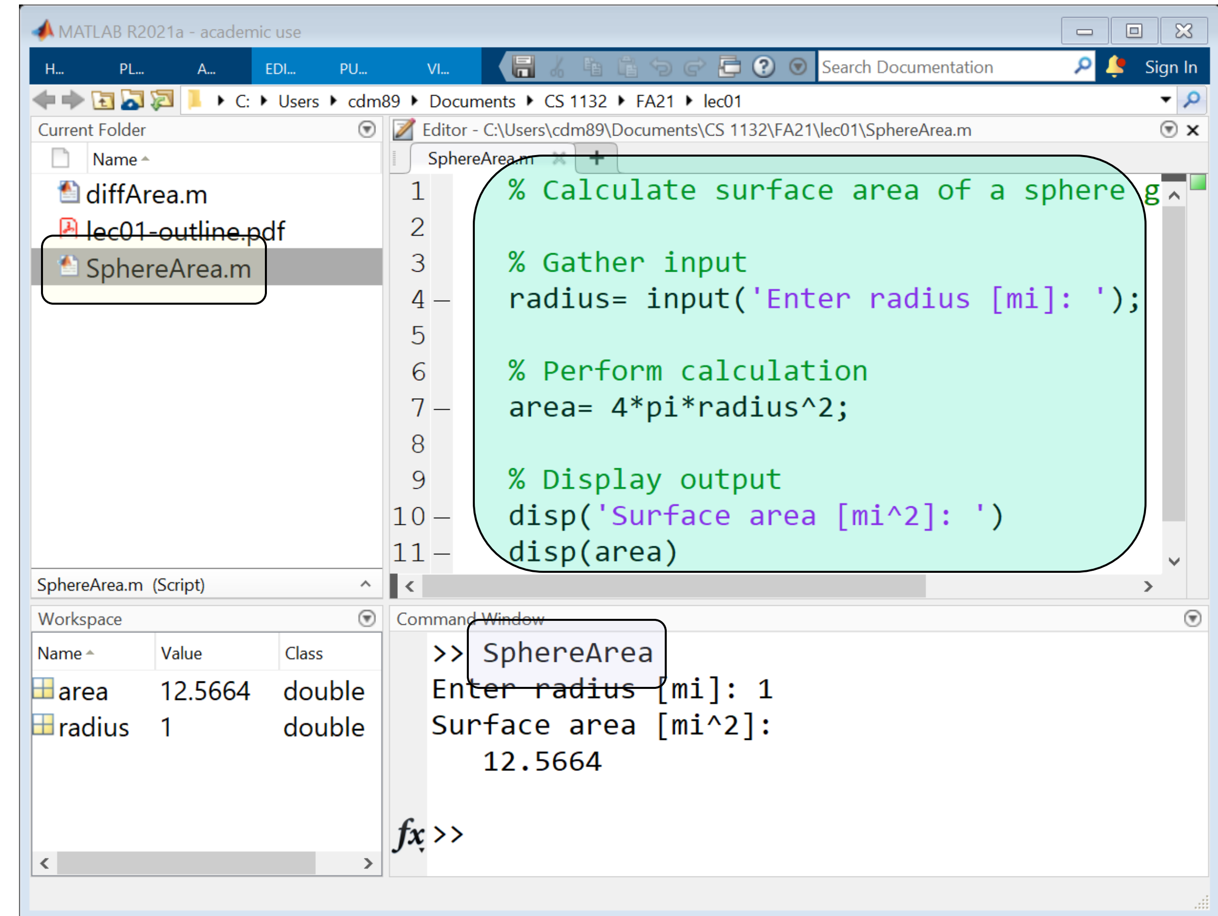
- Today's topics
  - User-defined function
  - `for`-loop
  - Conditionals

- Announcement/Reminder:
  - Be sure to read from textbook and do the exercises (or use another reference book of your choice)
  - Assignment 1 will be released before next class
  - Lab Wednesday (Upson 225)
  - Ed Discussion forum

Last time: scripts

- Execute multiple commands as a batch
- Name of script is name of file (w/o ".m" extension)
- File must be in current folder so MATLAB can find code when it sees its name
- To run, type its name as a command
  - Scripts can run other scripts
  - Alternatives: "Run" button, F5 key



Key features of scripts

- Bundle complicated logic conveniently under one name
  - Modularity
- Inputs and outputs interact with *humans*
  - `input()`, `disp()`, `fprintf()`
- Variables live in common workspace
  - Danger!

Alternative:

- Inputs and outputs interact with other code
  - Interaction with humans considered a “side effect”
- Behavior not affected by other computations/commands

<u>Functions</u>

General form of a user-defined function

```
function [out1, out2, …]= functionName (in1, in2, …)
% 1-line comment to describe the function
% Additional description of function

    Executable code that at some point assigns
    values to output parameters out1, out2, …
```

- *in1*, *in2*, … are defined when the function begins execution. Variables *in1*, *in2*, … are called function *parameters* and they hold the function *arguments* used when the function is invoked (called).
- *out1*, *out2*, … are not defined until the executable code in the function assigns values to them.

```
function [x, y] = polar2xy(r,theta)
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

rads= theta*pi/180;  % radian
x= r*cos(rads);
y= r*sin(rads);
```

A function file
`polar2xy.m`

Function header is the "contract" for how the function will be used (called)

You have this function:

```
function  [x, y] = polar2xy(r, theta)
% Convert polar coordinates (r, theta) to
% Cartesian coordinates (x,y).  Theta in degrees.
…
```

Code to call the above function:

```
% Convert polar (r1,t1) to Cartesian (x1,y1)
r1= 1;  t1= 30;
[x1, y1]= polar2xy(r1, t1);
plot(x1, y1,'b*')
…
```

Convert SphereArea from script to function

DEMO

Variable scope

- *Scripts* place variables on the workspace, but variables in *functions* have "local scope"
  - A new, private workspace every time function is called
- Analogy: stack of scratch paper
  - Evaluate arguments, copy to parameters on next page
  - Do all work on new page (can't flip back)
  - Copy values of output variables to previous page

Printing a value is not the same as returning a value

Given this function:

```
function m = convertLength(ft,in)
% Convert length from feet (ft) and inches (in)
% to meters (m).

   . . .
```

How many proper calls to convertLength are shown below?

```
% Given f and n
d= convertLength(f,n);
d= convertLength(f*12+n);
d= convertLength(f+n/12);
x= min(convertLength(f,n), 1);
y= convertLength(pi*(f+n/12)^2);
```

A: 1

B: 2

C: 3

D: 4

E: 5 or 0

Comments in functions

- Block of comments after the function header is printed whenever a user types

  help <*functionName*>

  at the Command Window
- 1st line of this comment block is searched whenever a user types

  lookfor <*someWord*>

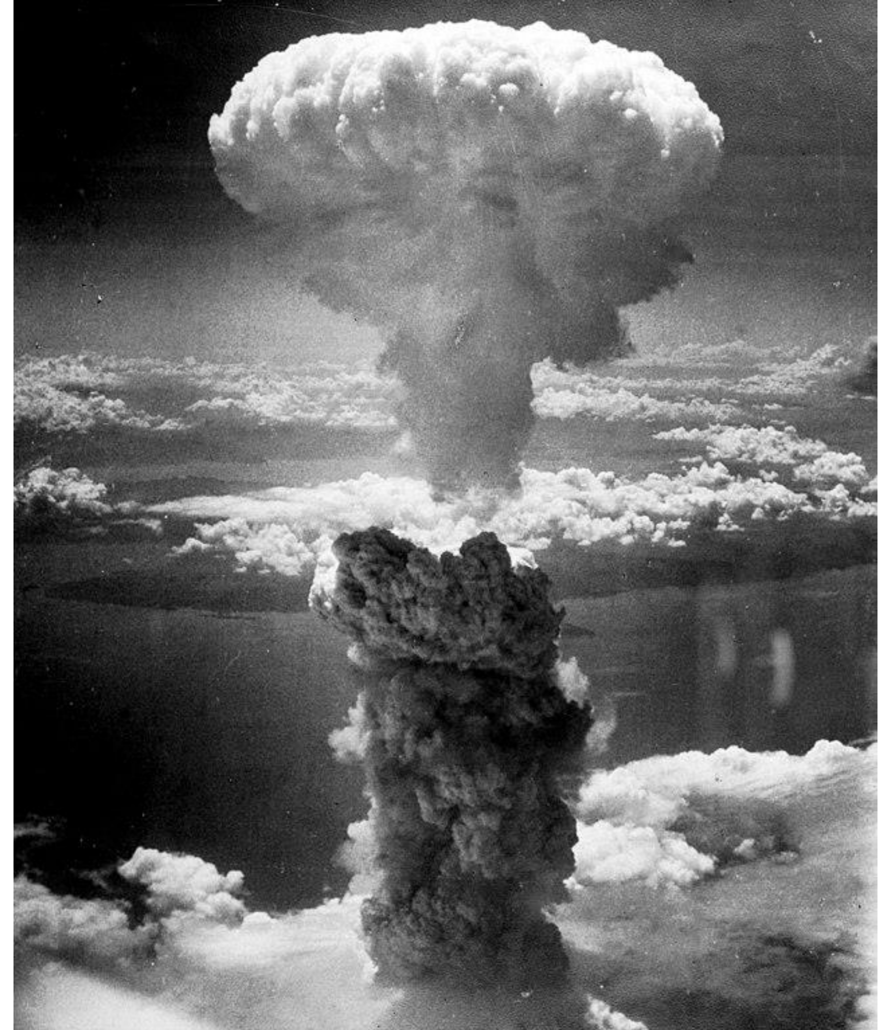
  at the Command Window
- Every function should have a comment block after the function header that says what the function does concisely

Accessing a function

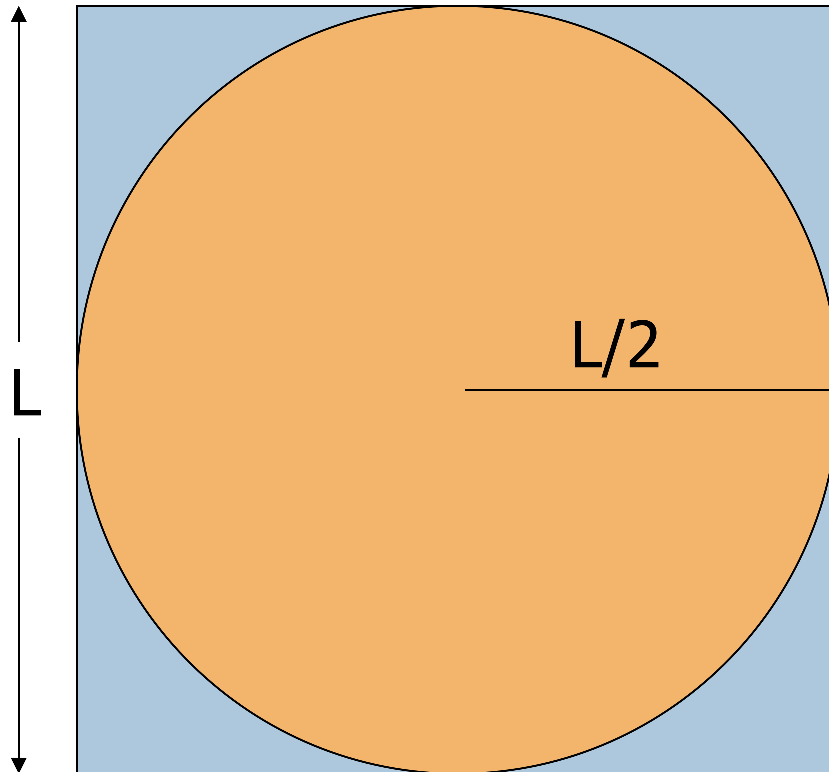
- A function is accessible if it is in the current directory or if it is on the search path
- Easy: put all related m-files in the same directory
- Better: the `path` function gives greater flexibility
- Precedence order:
  - Variables in workspace (if script)
  - Functions in current directory
  - Search path, left-to-right

Monte Carlo methods

1. Derive a relationship between some *desired quantity* and a *probability*
2. Use simulation to estimate the probability
   - Computer-generated random numbers
3. Approximate desired quantity based on prob. estimate



Monte Carlo Approximation of π



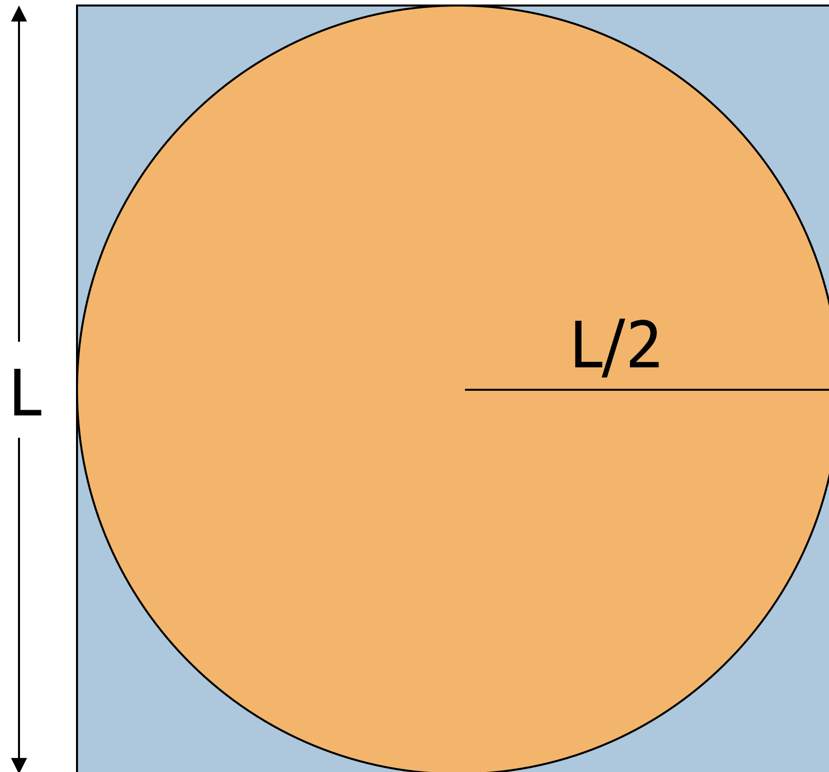
Throw N darts

Sq. area = $\boldsymbol{L \times L}$

Circle area = $\boldsymbol{\pi L^2/4}$

Prob. landing in circle
= (circle area)/(sq. area)
= $\boldsymbol{\pi/4}$
$\cong \boldsymbol{N_{in}/N}$

Monte Carlo Approximation of π



Throw N darts

$\pi \cong \mathbf{4}\ \boldsymbol{N_{in}}\ /\ \boldsymbol{N}$

Monte Carlo Approximation of π

For each of N trials

Throw a dart

If it lands in circle

add 1 to total # of hits

Pi is 4*hits/N

Repetition

Syntax of the for loop

```
for <var>= <start value>:<incr>:<end bound>

        statements to be executed repeatedly

end
```

Loop header specifies all the values that the index variable will take on, one for each pass of the loop.

E.g, `k= 3:1:7` means k will take on the values 3, 4, 5, 6, 7, one at a time.

for loop examples

```
for k = 2:0.5:3
    disp(k)
end
```

k takes on the values 2, 2.5, 3

Non-integer increment is OK

```
for k = 1:4
    disp(k)
end
```

k takes on the values 1, 2, 3, 4

Default increment is 1

```
for k = 0:-2:-6
    disp(k)
end
```

k takes on the values 0, -2, -4, -6

"Increment" may be negative

```
for k = 0:-2:-7
    disp(k)
end
```

k takes on the values 0, -2, -4, -6

Colon expression specifies *bounds*

```
for k = 5:2:1
    disp(k)
end
```

The set of values for k is the empty set: the loop body won't execute

Monte Carlo Approximation of π

For each of N trials

Throw a dart

If it lands in circle

add 1 to total # of hits

Pi is 4*hits/N

Monte Carlo Approximation of π with N darts on L-by-L board

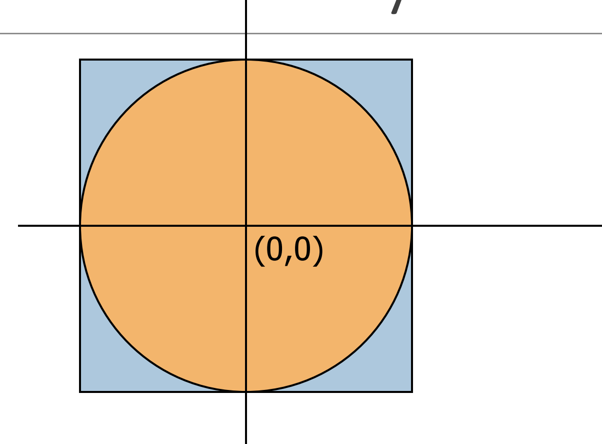
```
N=__;
for k = 1:N





end
myPi= 4*hits/N;
```

Monte Carlo Approximation of π with N darts on L-by-L board




```
N=__;
for k = 1:N
   % Throw kth dart



   % Count it if it is in the circle




end
myPi= 4*hits/N;
```

Generating random dart coordinates

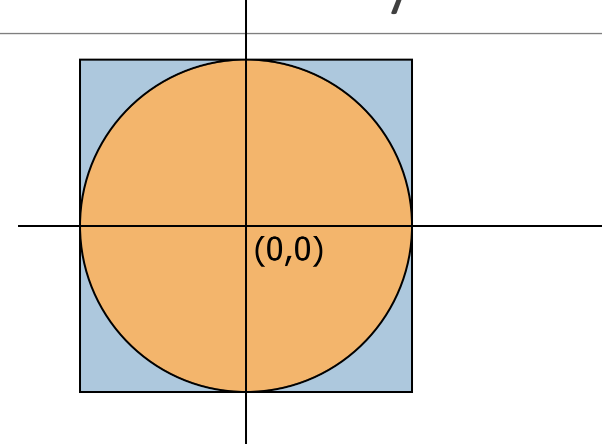
- Want: Sample from uniform distribution between –L/2 and L/2
- Have: Sample from uniform distribution between 0 and 1
- Transform “have” to “want” by scaling and shifting

Monte Carlo Approximation of π with N darts on L-by-L board

```
N=__; L=__;
for k = 1:N
   % Throw kth dart
   x= rand()*L - L/2;
   y= rand()*L - L/2;
   % Count it if it is in the circle



end
myPi= 4*hits/N;
```



Decision-making

The `if` construct

```
if boolean expression1
  statements to execute if expression1 is true

elseif boolean expression2
  statements to execute if expression1 is false
  but expression2 is true
:
else
  statements to execute if all previous conditions
  are false
end
```

Can have any number of elseif branches but at most one else branch

Boolean expressions

RELATIONAL OPERATORS

- <   Less than
- >   Greater than
- <=   Less than or equal to
- >=   Greater than or equal to
- ==   Equal to
- ~=   Not equal to

LOGICAL OPERATORS

- &&   Logical AND (are both true?)
- || Logical OR (is at least one true?)
- ~   Logical NOT (negation)

Logical operator examples

`&&` logical and: Are both conditions true?

E.g., we ask "is $L \leq x_c$ and $x_c \leq R$?"

In our code: `L<=xc && xc<=R`

`||` logical or: Is at least one condition true?

E.g., we can ask if x_c is outside of $[L,R]$,

i.e., "is $x_c < L$ or $R < x_c$?"

In code: `xc<L || R<xc`

`~` logical not: Negation

E.g., we can ask if x_c is not outside $[L,R]$.

In code: `~(xc<L || R<xc)`

Monte Carlo Approximation of π

For each of N trials

Throw a dart

If it lands in circle

add 1 to total # of hits

Pi is 4*hits/N

Monte Carlo Approximation of π with N darts on L-by-L board

```
N=__; L=__;
for k = 1:N
   % Throw kth dart
   x= rand()*L - L/2;
   y= rand()*L - L/2;
   % Count it if it is in the circle
   if sqrt(x^2 + y^2) <= L/2
      hits= hits + 1;
   end
end
myPi= 4*hits/N;
```