- Previous Lecture:
  - Developing algorithms
  - Finite/inexact arithmetic
  - Discrete vs. continuous
- Today's Lecture:
  - User-defined functions
- Announcements:
  - Section this week in regular classrooms
  - Prelim 1 on 2/21 (Thurs) 7:30pm

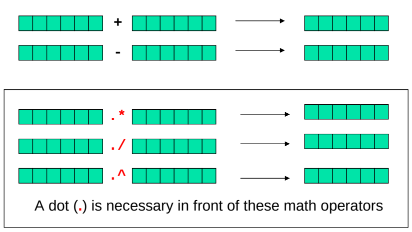
## Vectorized addition

| | | | | | |
|---|---|---|---|---|---|
| | x | 2 | 1 | .5 | 8 |
| + | y | 1 | 2 | 0 | 1 |
| = | z | 3 | 3 | .5 | 9 |

Matlab code: `z= x + y`



## Vectorized element-by-element arithmetic operations on arrays





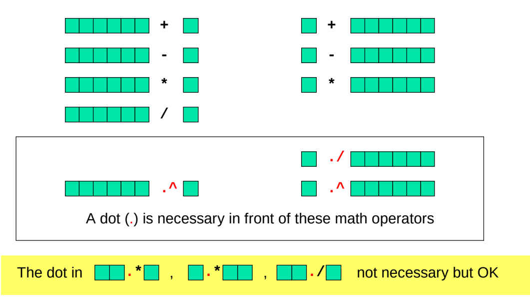

## Shift

| | | | | | |
|---|---|---|---|---|---|
| | x | 3 | | | |
| + | y | 2 | 1 | .5 | 8 |
| = | z | 5 | 4 | 3.5 | 11 |

Matlab code: `z= x + y`



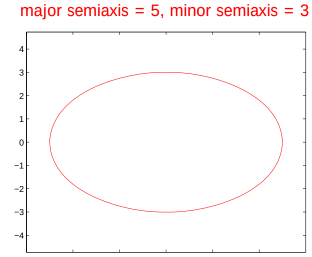
## Vectorized element-by-element arithmetic operations between an array and a scalar






## Estimate the perimeter of an ellipse

major semiaxis = 5, minor semiaxis = 3




Different methods based on different ways to "average" the major and minor axes

How many errors in the following statement given that `x = linspace(0,1,100)` ?

```
Y = (3*x .+ 1)/(1 + x^2)
```

A: 1  B: 2  C: 3  D: 4



## Plotting an Ellipse

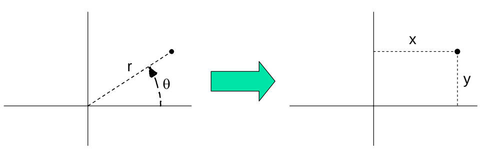
$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$

Easier representation for plotting:

$$\big(a\cos(t), b\sin(t)\big) \qquad 0 <= t <= 2\pi$$



## Convert from polar to Cartesian coordinates




Polar coordinates    Cartesian coordinates



```
% Convert polar coordinates (r,theta) to
% Cartesian coordinates (x,y).
% theta is in degrees.

r= input('Enter radius: ');
theta= input('Enter angle in degrees: ');

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



```
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);
```

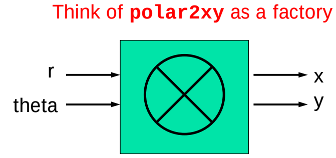
A function file `polar2xy.m`

```
r= input('Enter radius: ');
theta= input('Enter angle in degrees: ');

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

(Part of) a script file

```
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);
```

Think of **`polar2xy`** as a factory



## Why write user-defined functions?

1. Elevate reasoning by hiding details
2. Facilitate top-down design
3. Software management



## Elevates reasoning

Nice to have **sqrt** function when designing a quadratic equation solver.

You get to think at the level of

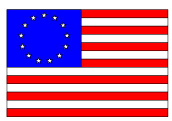
$$ax^2 + bx + c = 0$$

Easier to understand the finished quadratic equation solving code:

```
    :
r1 = (-b+sqrt(b^2-4*a*c))/(2*a);
r2 = (-b-sqrt(b^2-4*a*c))/(2*a);
    :
```



## Facilitates top-down design



1. Focus on how to draw the flag given just a specification of what the functions drawRect and drawStar do.

2. Figure out how to implement drawRect and drawStar.



## To specify a function...

... you describe how to use it, e.g.,

```
function DrawRect(a,b,L,W,c)
% Adds rectangle to current window.
% Assumes hold is on. Vertices are
% (a,b),(a+L,b),(a+L,b+W), & (a,b+W).
% The color c is one of 'r','g',
%'y','b','w','k','c',or 'm'.
```

*Given the specification, the user of the function doesn't need to know the detail of the function—they can just use it!*

## To implement a function...

... you write the code so that the function "lives up to" the specification. E.g.,

```
x = [a a+L a+L a a];
y = [b b b+W b+W b];
fill(x,y,c);
```

Don't worry—you'll learn these graphics functions soon.



## Software Management

Today:

I write a function

**EPerimeter(a,b)**

that computes the perimeter of the ellipse

$$\left(\frac{x}{a}\right)^2+\left(\frac{y}{b}\right)^2=1$$

## Software Management

During this year :

You write software that makes extensive use of

`EPerimeter(a,b)`

Imagine 100s of programs each with several lines that reference `EPerimeter`



## Software Management

Next year:

I discover a more efficient way to approximate ellipse perimeters. I change the implementation of

`EPerimeter(a,b)`

You do not have to change your software at all.



```
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`

```
r= input('Enter radius: ');
theta= input('Enter angle in degrees: ');

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

(Part of) a script file



```
function [x, y] = polar2xy(r,theta)
```

Output parameter list

Function name (This file's name is `polar2xy.m`)

Input parameter list



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, '*')
...
```



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, '*')
...
```

## 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.



## Comments in functions

- Block of comments after the function header is printed whenever a user types **`help functionName`** at the Command Window
- The 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:
  - 1st line succinctly describes what the function does
  - Additional lines for more detail, if necessary



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?

```
f= …;  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

dotsInCircles.m

(functions with multiple input parameters)
(functions with a single output parameter)
(functions with multiple output parameters)
(functions with no output parameter)



## Accessing your functions

For now*, put your related functions and scripts in the same directory.

MyDirectory

- `dotsInCircles.m`
- `polar2xy.m`
- `randDouble.m`
- `drawColorDot.m`
- *Any script/function that calls* `polar2xy.m`

*The **`path`** function gives greater flexibility. Not required in CS100M.

## Why write user-defined function?

1. Elevate reasoning by hiding details
2. Facilitate top-down design
3. Software management
4. A function can be independently tested easily
5. Keep a driver program clean by keeping detail code in functions—separate, non-interacting files