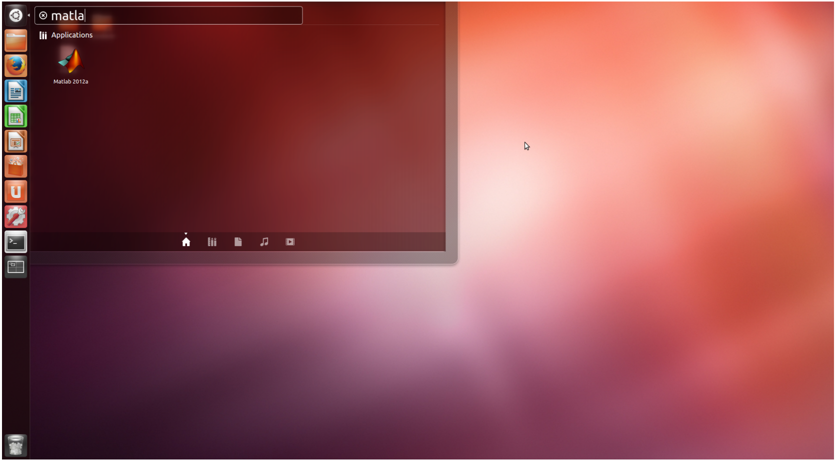
Introduction to Matlab

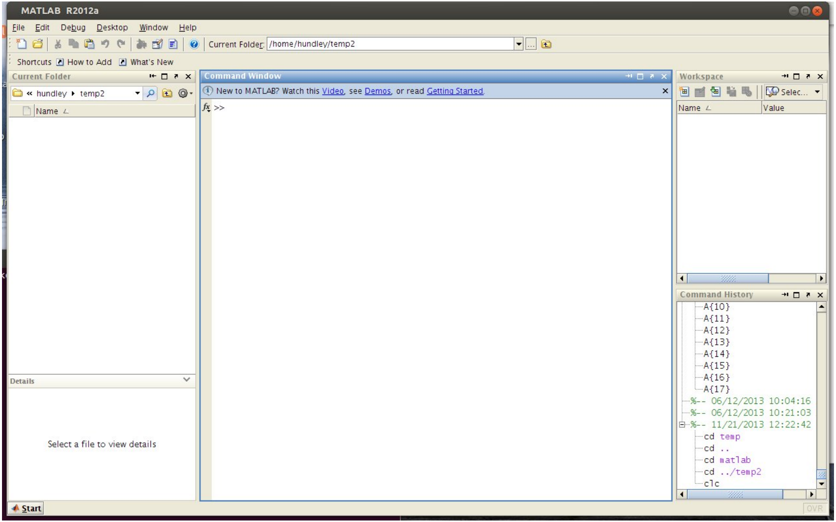
Math 339

Fall 2013

First, put the icon in the launcher: Drag and drop



Now, open Matlab:



* Current Folder * Command Window * Workspace * Command History

Operations in Matlab

Description:	In Matlab:	Try typing:
Assignment is $=$	x=3	x=3 versus 3=x
The constant π	pi	$a = \cos(\pi/3)$
The exponential e^x	exp(x)	exp(a)
Complex numbers	i or j	(1-3*i)*(5-2*i)
Go to previous line	Up arrow key	Change x=3 to x=5;
Suppress output	;	
Clear memory	clear	
Clear the screen	clc	

(You don't need the * for complex numbers, but it's good practice)

Entering Arrays:

- A row vector stored in variable xr:

  `xr=[1,2,3,4,5]`

Entering Arrays:

- A row vector stored in variable xr:

```
xr=[1,2,3,4,5]
```

- Inside an array, semi-colon ends a row. To enter vector xc:

```
xc=[1;2;3;4;5];
xc1=xr';    %Transpose is the apostrophe
```

Entering Arrays:

- A row vector stored in variable xr:

```
xr=[1,2,3,4,5]
```

- Inside an array, semi-colon ends a row. To enter vector xc:

```
xc=[1;2;3;4;5];
xc1=xr';    %Transpose is the apostrophe
```

- An array can be entered row-wise with semicolons ending each row:

```
A=[1 2 3;4 5 6];
```

Entering Arrays:

- A row vector stored in variable xr:

```
xr=[1,2,3,4,5]
```

- Inside an array, semi-colon ends a row. To enter vector xc:

```
xc=[1;2;3;4;5];
xc1=xr';    %Transpose is the apostrophe
```

- An array can be entered row-wise with semicolons ending each row:

```
A=[1 2 3;4 5 6];
```

- You can find the length of a vector or the size of a matrix:

```
n1=length(xc)
[numrows,numcols]=size(A)
```

More on Arrays:

- Arrays can be accessed (and changed) element-wise.
  For example, change the $(1, 2)$ entry in matrix A to -3:

  ```
  A(1,2)=-3;
  ```

More on Arrays:

- Arrays can be accessed (and changed) element-wise.
  For example, change the $(1,2)$ entry in matrix A to -3:

  ```
  A(1,2)=-3;
  ```

- What does the following command do?

  ```
  B=A([1,1,2],[2,1,3])
  ```

More on Arrays:

- Arrays can be accessed (and changed) element-wise.
  For example, change the $(1,2)$ entry in matrix A to -3:

  `A(1,2)=-3;`

- What does the following command do?

  `B=A([1,1,2],[2,1,3])`

$$B = \begin{bmatrix} -3 & 1 & 3 \\ -3 & 1 & 3 \\ 5 & 4 & 6 \end{bmatrix} = \begin{bmatrix} A(1,2) & A(1,1) & A(1,3) \\ A(1,2) & A(1,1) & A(1,3) \\ A(2,2) & A(2,1) & A(2,3) \end{bmatrix}$$

The colon operator: A quick way to make a vector

Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

The colon operator: A quick way to make a vector

Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

  2 3 4 5 6 7 8 9

The colon operator: A quick way to make a vector
Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

2	3	4	5	6	7	8	9

- `x=8:-2:1` Start at 8, decrease by 2 until 1 (or just before).

The colon operator: A quick way to make a vector

Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

  `2 3 4 5 6 7 8 9`

- `x=8:-2:1` Start at 8, decrease by 2 until 1 (or just before).

  `8 6 4 2`

- `x=2:3:10` Start at 2, increase by 3 until 10 (or just before)

The colon operator: A quick way to make a vector

Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

  ```
  2    3    4    5    6    7    8    9
  ```

- `x=8:-2:1` Start at 8, decrease by 2 until 1 (or just before).

  ```
   8    6    4    2
  ```

- `x=2:3:10` Start at 2, increase by 3 until 10 (or just before)

  ```
  2 5 8
  ```

The colon operator: A quick way to make a vector
Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

  ```
  2    3    4    5    6    7    8    9
  ```

- `x=8:-2:1` Start at 8, decrease by 2 until 1 (or just before).

  ```
   8    6    4    2
  ```

- `x=2:3:10` Start at 2, increase by 3 until 10 (or just before)

  ```
  2  5  8
  ```

- What is the Matlab command to produce the odd numbers between 3 and 11?

The colon operator: A quick way to make a vector
Examples:

- `x=2:9` The vector x is the set of integers from 2 to 9.

  `2 3 4 5 6 7 8 9`

- `x=8:-2:1` Start at 8, decrease by 2 until 1 (or just before).

  `8 6 4 2`

- `x=2:3:10` Start at 2, increase by 3 until 10 (or just before)

  `2 5 8`

- What is the Matlab command to produce the odd numbers between 3 and 11?

  `3:2:11`

How would I get 5 evenly spaced points between (and including) 1.3 and 4.6?

How would I get 5 evenly spaced points between (and including) 1.3 and 4.6?

```
\linspace(1.3,4.6,5);
```

How would I get 5 evenly spaced points between (and including) 1.3 and 4.6?

```
\linspace(1.3,4.6,5);
```

General command:

```
linspace(a,b)    (Default is 100 points)
linspace(a,b,c)  (c points evenly spaced between a and b)
```

How would I get 5 evenly spaced points between (and including) 1.3 and 4.6?

```
\linspace(1.3,4.6,5);
```

General command:

```
linspace(a,b)     (Default is 100 points)
linspace(a,b,c)   (c points evenly spaced between a and b)
```

Get 100 points between -1 and 10:

How would I get 5 evenly spaced points between (and including) 1.3 and 4.6?

```
\linspace(1.3,4.6,5);
```

General command:

```
linspace(a,b)     (Default is 100 points)
linspace(a,b,c)   (c points evenly spaced between a and b)
```

Get 100 points between -1 and 10: `linspace(-1,10)`

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)`

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.
- `A=randn(4,5)`

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.
- `A=randn(4,5)` Matrix A is filled with random[2] numbers.

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.
- `A=randn(4,5)` Matrix A is filled with random[2] numbers.
- `A=eye(4)`

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.
- `A=randn(4,5)` Matrix A is filled with random[2] numbers.
- `A=eye(4)` The 4×4 identity matrix.

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.
- `A=randn(4,5)` Matrix A is filled with random[2] numbers.
- `A=eye(4)` The 4×4 identity matrix.
- Let `B=[1 2;3 4]`. What does `A=repmat(B,2,3)` do?

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Special Arrays: Try typing these in- What does it mean?

- `A=rand(3,2)` Matrix A is filled with random[1] numbers.
- `A=randn(4,5)` Matrix A is filled with random[2] numbers.
- `A=eye(4)` The 4×4 identity matrix.
- Let `B=[1 2;3 4]`. What does `A=repmat(B,2,3)` do?

$$A = \begin{bmatrix} B & B & B \\ B & B & B \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \end{bmatrix}$$

[1] Random here means uniformly distributed between 0 and 1.

[2] Random here means a normal distribution with zero mean and unit std.

Linear Algebra works in a natural way.
Define x as a random 3×1 vector, A as a random 3×2 matrix, B as a random 3×3 matrix, and C as 2×3 random matrix. (Use either kind of random number)

Linear Algebra works in a natural way.
Define x as a random 3×1 vector, A as a random 3×2 matrix, B as a random 3×3 matrix, and C as 2×3 random matrix. (Use either kind of random number)

```
x=rand(3,1);
A=randn(3,2);
B=rand(3,3);
C=randn(2,3);
```

Linear Algebra works in a natural way.
Define x as a random 3×1 vector, A as a random 3×2 matrix, B as a random 3×3 matrix, and C as 2×3 random matrix. (Use either kind of random number)

```
x=rand(3,1);
A=randn(3,2);
B=rand(3,3);
C=randn(2,3);
```

Are the following defined?

```
A*x     C*x     A*C     C*B    x'*A
```

Linear Algebra works in a natural way.
Define x as a random 3×1 vector, A as a random 3×2 matrix, B as a random 3×3 matrix, and C as 2×3 random matrix. (Use either kind of random number)

```
x=rand(3,1);
A=randn(3,2);
B=rand(3,3);
C=randn(2,3);
```

Are the following defined?

```
A*x      C*x      A*C     C*B    x'*A
```

(The only expression not defined is $A\mathbf{x}$)

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`
Other examples:

- Raise all the entries in the vector x to the third power:

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`
Other examples:

- Raise all the entries in the vector x to the third power: `y=x.^3`

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`
Other examples:

- Raise all the entries in the vector x to the third power: `y=x.^3`
- Add 2 to every element in the matrix C: `C+2` (No dot needed)

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`
Other examples:

- Raise all the entries in the vector x to the third power: `y=x.^3`
- Add 2 to every element in the matrix C: `C+2` (No dot needed)
- Is there a difference between B^2 and `B.^2`?

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`
Other examples:

- Raise all the entries in the vector x to the third power: `y=x.^3`
- Add 2 to every element in the matrix C: `C+2` (No dot needed)
- Is there a difference between B^2 and `B.^2`? (Yes)

The Dot Operator

The dot operator tells Matlab to perform the operation following it, element-by-element.
For example: `A.*C'`
Other examples:

- Raise all the entries in the vector x to the third power: `y=x.^3`
- Add 2 to every element in the matrix C: `C+2` (No dot needed)
- Is there a difference between B^2 and `B.^2`? (Yes)
- What happens: `sin(A)` and `exp(-B)`

Other linear algebra operations:

- `det(A)` is the determinant of A
- `[V,D]=eig(A);` Matrix V holds the eigenvectors, D the eigenvalues of A.
- `X=linsolve(A,B)` Solve the system $AX = B$ for X.

More with Arrays: (For demonstrations, let A be a random 6×6 matrix).

The notation:	Yields:
`A(i,j)`	The (i,j)th element
`A(i,:)`	The entire ith row
`A(:,j)`	The entire jth column
`A(:,2:5)`	The 2d to fifth columns, all rows
`A(1:4,2:3)`	A 4×2 submatrix

Examples:

1. Assign vector x to the 3rd column of A:

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A:

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A: `y=A(4,:);`

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A: `y=A(4,:);`
3. Append the vector x to the last column of A:

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A: `y=A(4,:);`
3. Append the vector x to the last column of A: `A=[A, x];`

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A: `y=A(4,:);`
3. Append the vector x to the last column of A: `A=[A, x];`
4. Solve $A\mathbf{c} = \mathbf{x}$ for $\mathbf{c}$:

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A: `y=A(4,:);`
3. Append the vector x to the last column of A: `A=[A, x];`
4. Solve $A\mathbf{c} = \mathbf{x}$ for $\mathbf{c}$: `c=linsolve(A,x)`

Examples:

1. Assign vector x to the 3rd column of A: `x=A(:,3);`
2. Assign vector y to the 4th row of A: `y=A(4,:);`
3. Append the vector x to the last column of A: `A=[A, x];`
4. Solve $A\mathbf{c} = \mathbf{x}$ for $\mathbf{c}$: `c=linsolve(A,x)`

To delete rows/columns, assign the row/column to the "empty array": `[]`. For example, delete row 3 from the matrix A:

```
size(A)
A(3,:)=[];
size(A)
```

To delete rows/columns, assign the row/column to the "empty array": []. For example, delete row 3 from the matrix A:

```
size(A)
A(3,:)=[];
size(A)
```

For a new array, let's load an image. A picture of a clown is built-in to Matlab for demonstrations:

```
clear
clc
load clown
whos
image(X);
colormap(map);
```

Delete all of the odd rows and even columns out of the image, and show the result (we'll save the original image in X and put the modified matrix in Y):

```
Y=X;
Y(1:3:end,:)=[];
Y(:,2:2:end)=[];
image(Y);
```

Plotting functions: You need both a domain and a range.

- Example: Plot $y = \sin(x)$ for $-\pi \leq x \leq 3\pi$.

```
x=linspace(-pi,3*pi,200);
y=sin(x);
plot(x,y);
```

Plotting functions: You need both a domain and a range.

- Example: Plot $y = \sin(x)$ for $-\pi \leq x \leq 3\pi$.

```
x=linspace(-pi,3*pi,200);
y=sin(x);
plot(x,y);
```

- Multiple plots on one graph: Plot the sine using green solid line, the parabola using black dash-dotted line, and the exponential using magenta dotted line:

```
x1=linspace(-2,2);
y1=sin(x1);
y2=x1.^2;
x2=linspace(-2,1);
y3=exp(x2);
plot(x1,y1,'g-',x1,y2,'k-.',x2,y3,'m:');
```

To see the plotting options, type `help plot`

Code	Color	Symbol	
y	yellow	.	point
m	magenta	o	circle
c	cyan	x	x-mark
r	red	+	plus
g	green	-	solid
b	blue	*	star
w	white	:	dotted
k	black	-.	dashdot
		--	dashed

For more, type `doc plot`

Files called “scripts” are text files with Matlab commands that are executed when they are called in the command window. These take the place of the Maple worksheet.

EXAMPLE: Write a script function that will perform Newton’s Method on the function $x - e^{-x}$ starting at $x = -1$ until the solution is gives f to within 10^{-6}.
SOLUTION:

- Open the editor from the command window: `edit`
- Type the following:

```
% Script file that performs Newton's Method

f=inline('x-exp(-x)');  df=inline('1+exp(-x)');

x(1)=-1;

for j=1:100
  y(j)=f(x(j));
  dy(j)=df(x(j));
  x(j+1)=x(j)-y(j)/dy(j);

  if abs(y(j))<10^(-6)
     break;
  end

end
```

Save the result as “Script01.m”

To run the script, in the command window, type

```
Script01
```

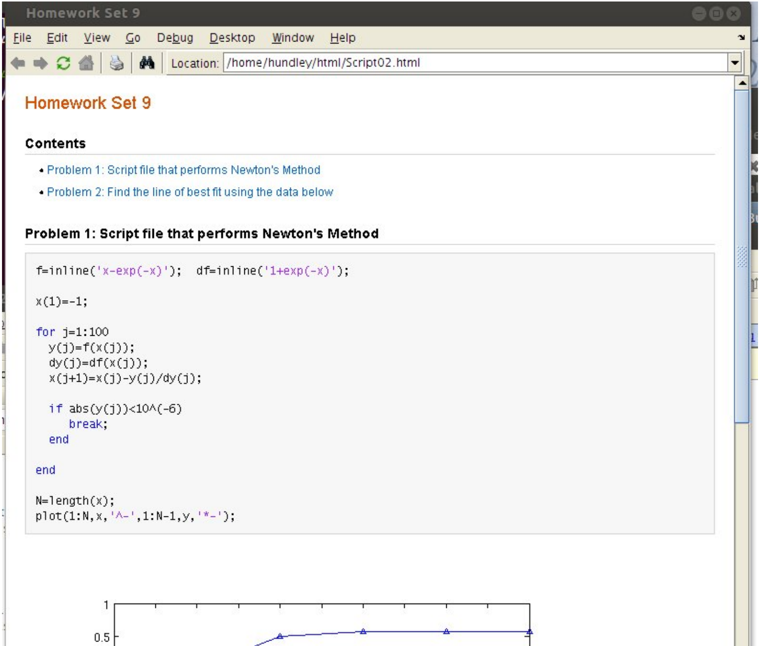
(Do not type the file suffix (.m)).
To see the variables, type x and y:

```
x
y
```

We can't see many of the digits! To see more, type

```
format long
y
format short
y
```

To publish: Example is Script02.m
(Open editor, then `File`, then `Publish Script02`)



Homework Set 9

File Edit View Go Debug Desktop Window Help

Location: /home/hundley/html/Script02.html

Homework Set 9

Contents

- Problem 1: Script file that performs Newton's Method
- Problem 2: Find the line of best fit using the data below

Problem 1: Script file that performs Newton's Method

```
f=inline('x-exp(-x)');  df=inline('1+exp(-x)');

x(1)=-1;

for j=1:100
  y(j)=f(x(j));
  dy(j)=df(x(j));
  x(j+1)=x(j)-y(j)/dy(j);

  if abs(y(j))<10^(-6)
     break;
  end

end

N=length(x);
plot(1:N,x,'^-',1:N-1,y,'*-');
```

Functions:

Functions:

- Are text files with .m suffix (just like a script)

Functions:

- Are text files with .m suffix (just like a script)
- Have inputs and produce outputs (not like a script)

Functions:

- Are text files with .m suffix (just like a script)
- Have inputs and produce outputs (not like a script)
- Use local variables (not like a script)

Functions:

- Are text files with .m suffix (just like a script)
- Have inputs and produce outputs (not like a script)
- Use local variables (not like a script)
- The first line of the .m file is the key

Example: Compute cost to produce a cylindrical can.
What should be the inputs and outputs?

Example: Compute cost to produce a cylindrical can.
What should be the inputs and outputs?

- Input: Radius r, Height h.
- Also:

Example: Compute cost to produce a cylindrical can.
What should be the inputs and outputs?

- Input: Radius r, Height h.
- Also: Cost for top/bottom, Ct, Cost for sides: Cs
- Output:

Example: Compute cost to produce a cylindrical can.
What should be the inputs and outputs?

- Input: Radius r, Height h.
- Also: Cost for top/bottom, Ct, Cost for sides: Cs
- Output: Cost C and perhaps Surface Area as well.

Cost:

Example: Compute cost to produce a cylindrical can.
What should be the inputs and outputs?

- Input: Radius r, Height h.
- Also: Cost for top/bottom, Ct, Cost for sides: Cs
- Output: Cost C and perhaps Surface Area as well.

Cost:

$$C = C_t(2\pi r^2) + C_s(2\pi rh)$$

Surface Area:

$$A = 2\pi r^2 + 2\pi rh$$

```
function [C,A]=canFunction(r,h,Ct,Cs)
% function [C,A]=canFunction(r,h,Ct,Cs)
%  Computes the cost C and surface area A of a can.
%  Input:  radius r, height h, Ct, Cs are costs of
%       top/bottom and sides.
%  Output:  Cost and Surface Area (in that order)

TopBottom=2*pi*r^2;
Sides=2*pi*r*h;

C=Ct*TopBottom+Cs*Sides;
A=TopBottom+Sides;
```

Save this file as the function name with a `.m.` suffix, or, `canFunction.m`.

Some things to notice about a function:

- The first line should always begin with the word "function". This is how Matlab distinguishes between a script and a function.
- You should always include remarks that tell you how to use the function.

Now in the command window, we can type things like:

```
help canFunction
[C,A]=canFunction(3,6,10,15);
```

You should notice that when the function is called, only the output variable names are present- that is, the variables `TopBottom` and `Sides` that the function uses are only present for the function itself (these are called "local variables" in computer programming).