Lecture 14
MATLAB I: Welcome to Matlab!
(Programs and Functions)

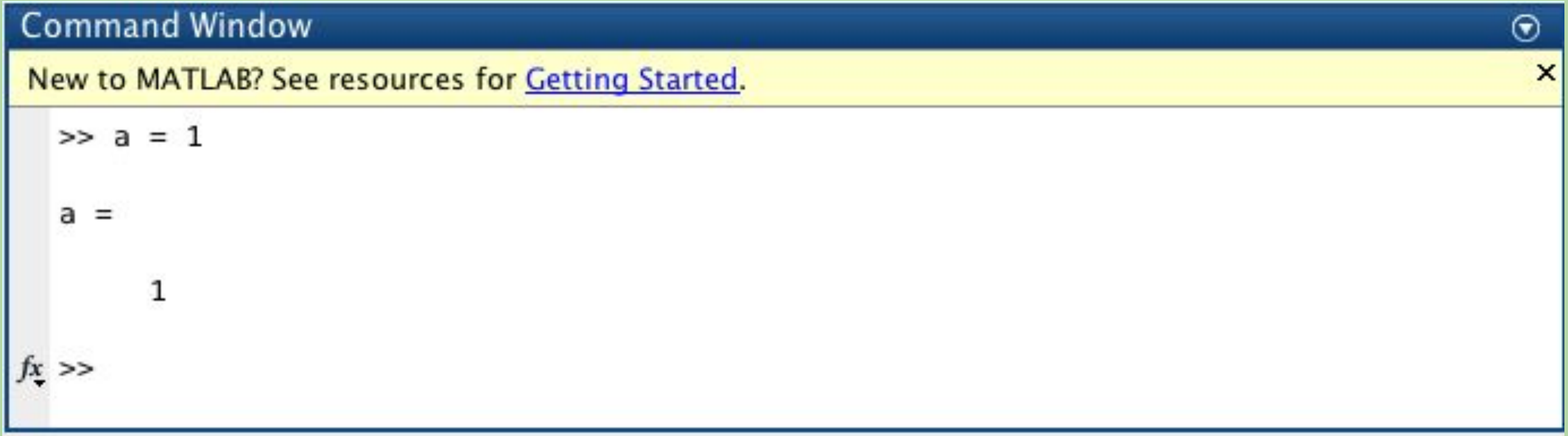
MATLAB Resources

- https://www.mathworks.com
  - Excellent Documentation
  - Intro Videos
- We will be using version MATLAB R2018A
  - Free academic license

  https://www.brown.edu/information-technology/software/catalog/matlab
- Matlab primer located on course website
- Can download on department machine using command `cs4_matlab`

Command Window

- The command window at the bottom of the interface allows you to interact with MATLAB: you can define variables, call functions, and so much more!
- Similar to using the Python shell



New to MATLAB? See resources for Getting Started.

```
>> a = 1

a =

     1

>>
```

Workspaces

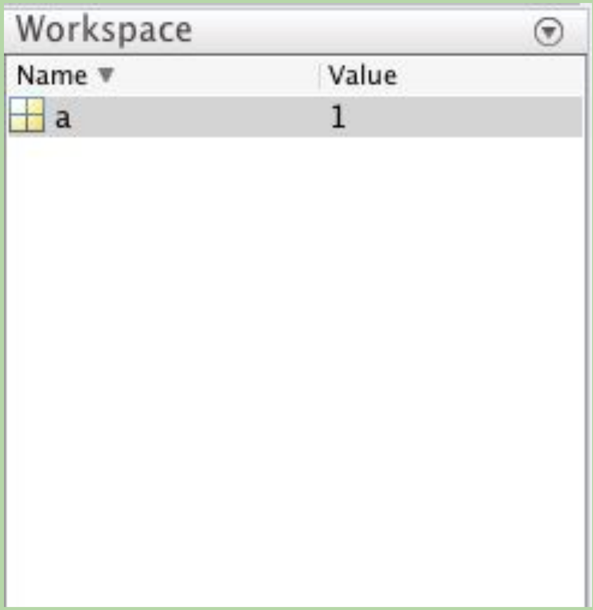
- Variables defined in the command window are said to be stored in the ‘Global Workspace’.
- **whos** displays dimensions, amount of storage and class of variables in workspace, e.g.

```
>> whos
  Name      Size            Bytes  Class
  myPi      1x1                 8  double
  name      1x11               22  char
  b         1x1                 1  logical
```

Can also use **Workspace Window**

Workspaces

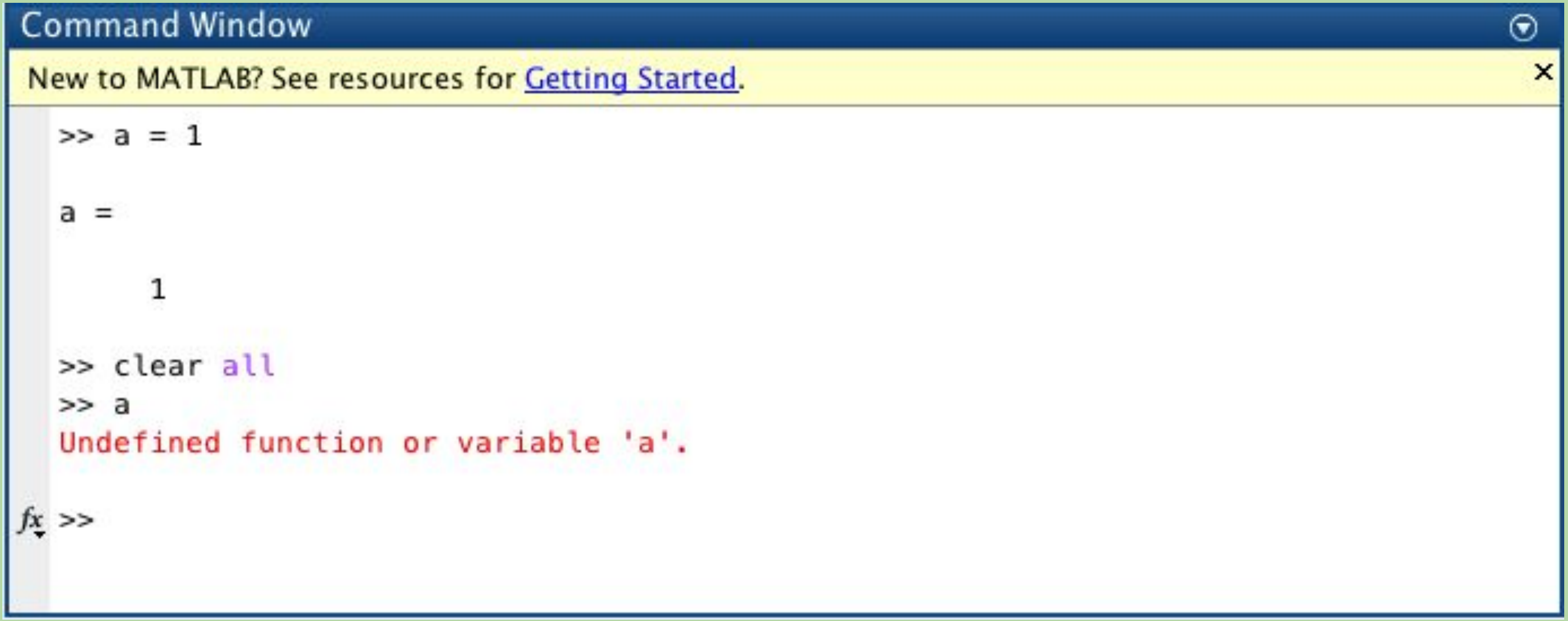
- If you’ve forgotten a variable name, you can use whos or the **Workspace Window** (located all the way to the right) to find it
- The Workspace Window provides the ability to interactively examine and change variable values
- When a function in MATLAB executes, it gets its own Workspace. This is done to avoid clashes with variables that you have already defined.



Name ▼	Value
a	1

Housekeeping

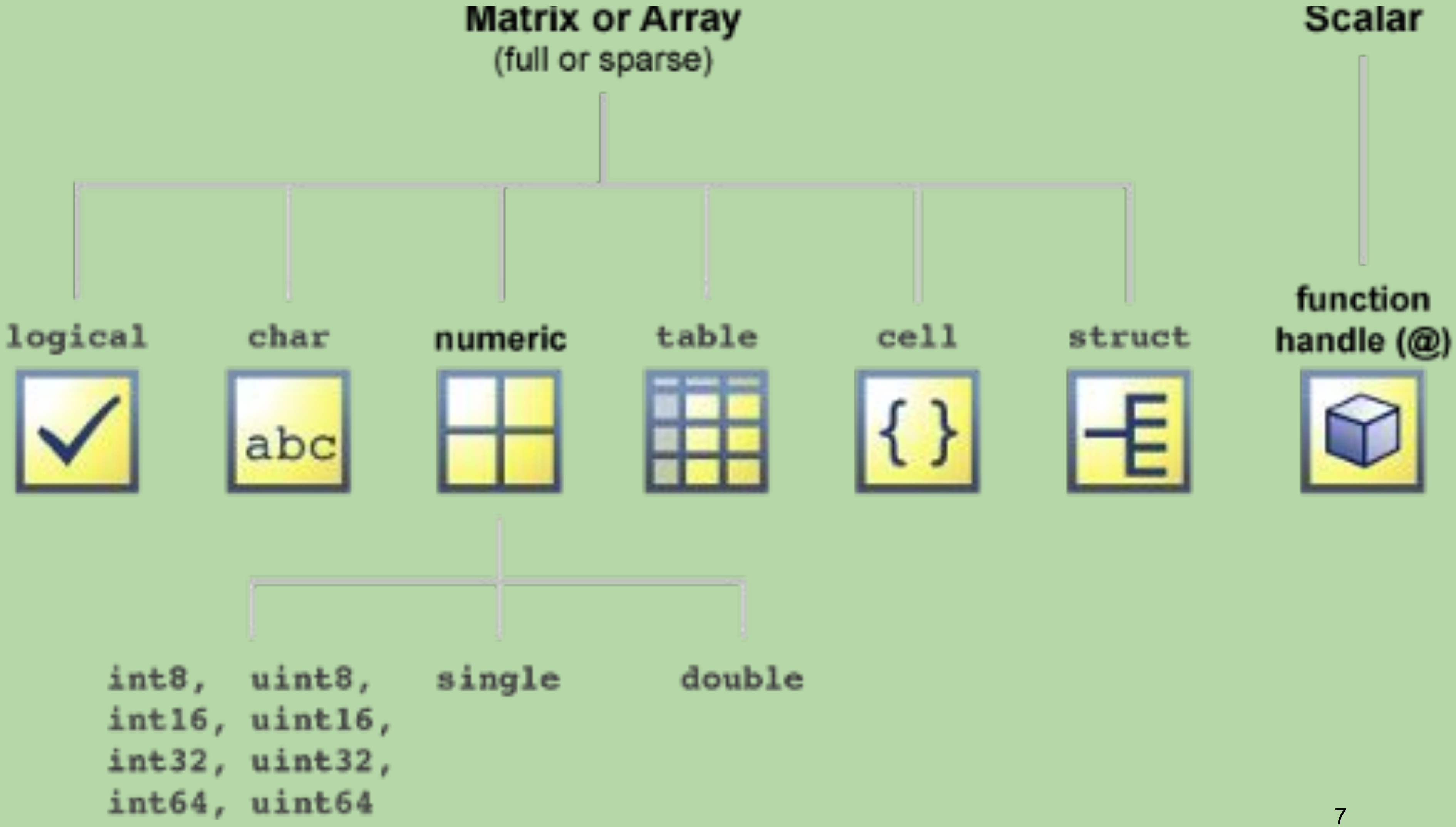
□ **`clear x`** deletes variable x (and frees up storage)
□ **`clear all`** deletes all variable in workspace



New to MATLAB? See resources for Getting Started.

```
>> a = 1

a =

     1

>> clear all
>> a
Undefined function or variable 'a'.

>>
```

Fundamental MATLAB Classes



Working with Classes

class(a) returns the class name of variable a

```
>> class(a)
ans = double
```

classname*(value)** returns a value of class ***classname

```
>> class(logical(0))
ans = logical
```

Mixing Types

- What happens when we mix variable types in an arithmetic expression?

```
>> class(myPi + b) % double + logical
ans = double
>> class(‘Walt Disney’ + 1) % char + double
ans = double
>> class(‘Walt Disney’ + true) % char + logical
ans = double
```

- Three most common classes promote to double

Additional Numeric Classes

- Integers
  - **int8**, **int16**, **int32**, **int64 (signed)**
  - **uint8, uint16, uint32, uint64 (unsigned)**
    - Unsigned means that the integer will only be positive
  - Number in name represents number of bits required for storage
  - Values in {0,1,…, 2^N} or {-2^(N-1), …, 2^(N-1)-1}
  - Use when more compact or accurate than double

Additional Numeric Classes

- Real numbers: **single**
  - More compact (4 bytes), less accurate than double
  - Follows IEEE 754 standard for single precision floating point numbers (1 bit for sign, 23 for fraction, 8 bits for exponent)

Additional Numeric Classes

- Complex Numbers: **complex**
  - stores real and imaginary part as double

```
>> z=2+3*sqrt(-1)
z = 2.0000 + 3.0000i
>> z*conj(z)
ans = 13
```

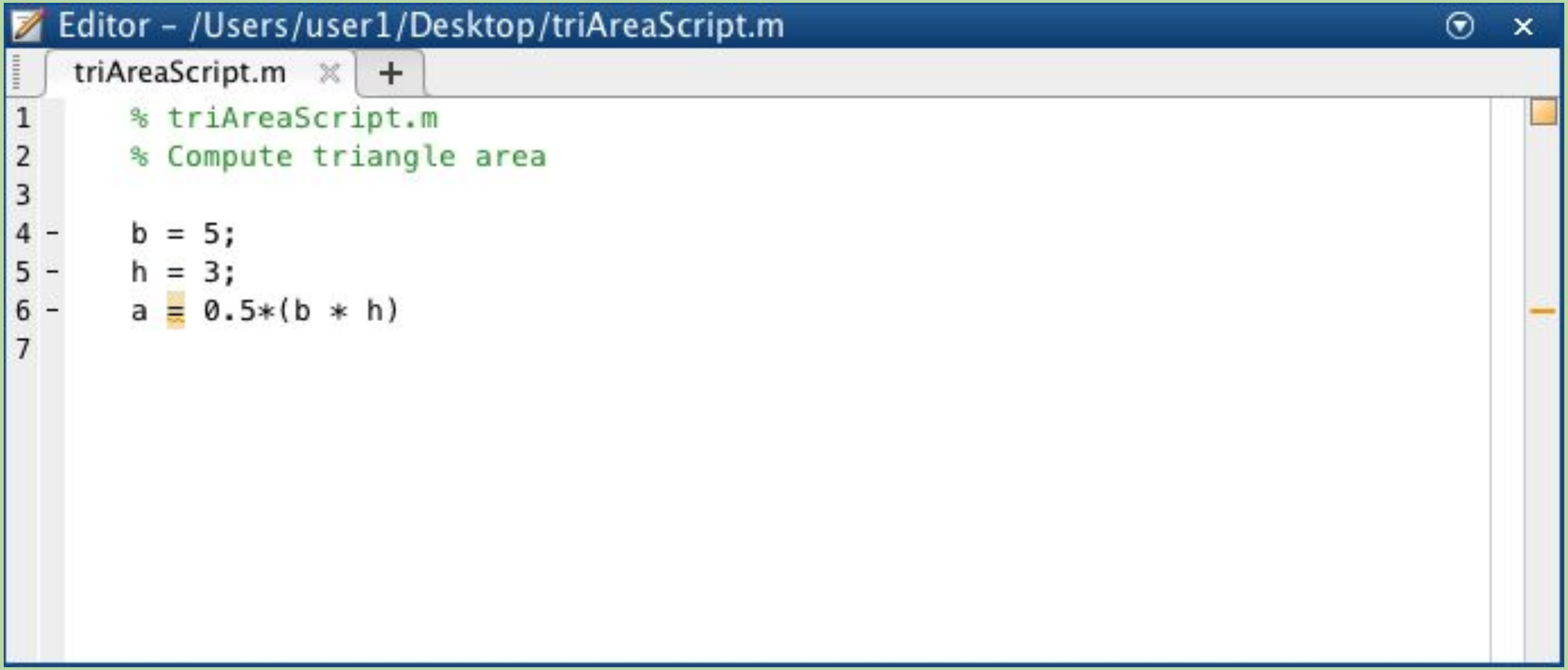
Mixing Integers and Doubles

- Arithmetic results from mixing integer classes with class double retain integer type – this can cause all sorts of problems
    - Fractional parts are rounded!
    - Results that are too large are converted to class **intmax**
    - Results that are too small are converted to class **intmin**
    - For **uint8**, **intmax** is 2^8-1=255 and **intmin** is zero.

```
>> uint8(16) + 1000.67
ans = 255
>> class(ans)
ans = uint8
```

Programs

Example Script (triAreaScript.m)



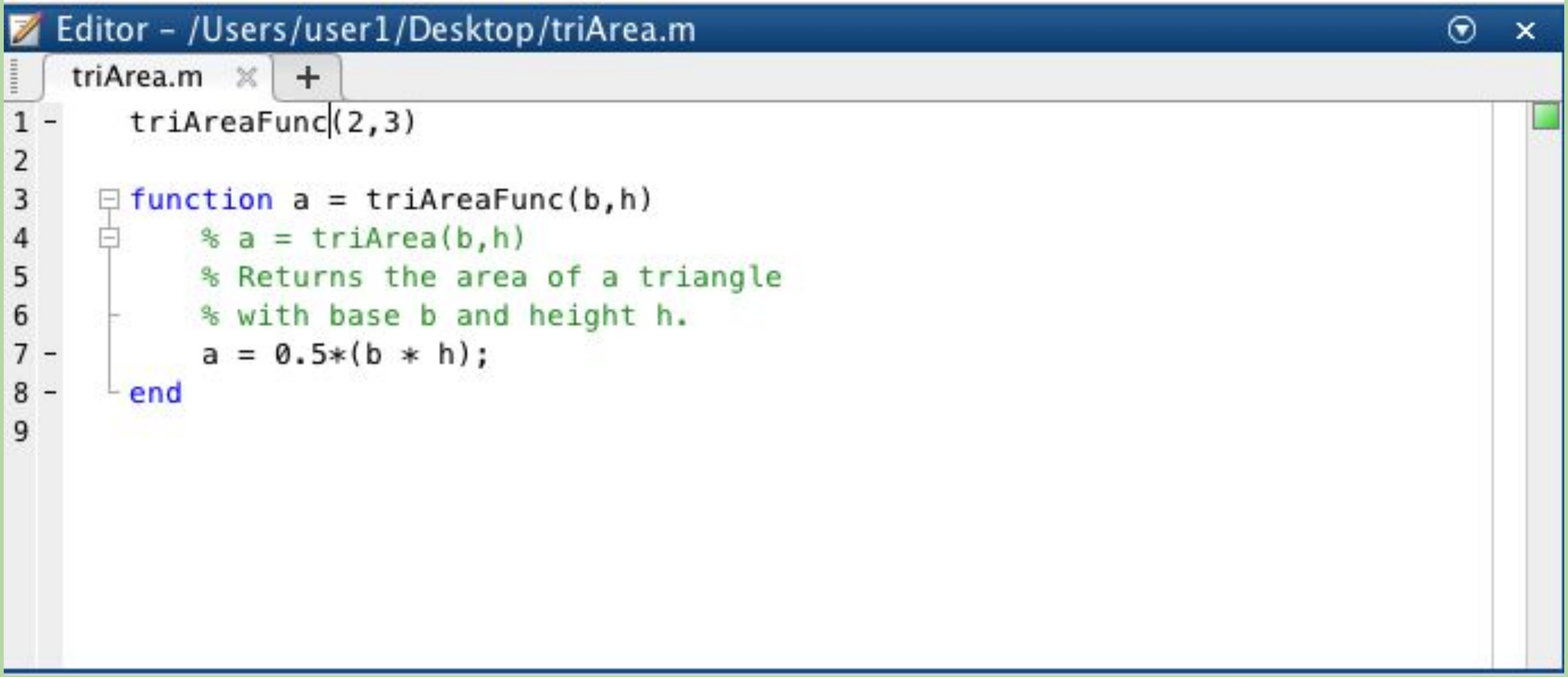
```
% triAreaScript.m
% Compute triangle area

b = 5;
h = 3;
a = 0.5*(b * h)

```

Scripts

- MATLAB allows one to store a sequence of commands (programs!) as a **script** or a **function**.
- You can edit them in the editor window
- Scripts
  - Behave exactly as if you ‘cut and paste’ from them to the command line. They share the Global Workspace and do not allow one to pass any arguments to them
  - This is both very useful and very inconvenient
  - It allows one to ‘work incrementally’ on a solution, with full access to the scripts variables
  - However, scripts can also be <u>very dangerous</u> due to the fact they all share the same Workspace

FUNCTIONS

Example Function (triArea.m)



```
triAreaFunc(2,3)

function a = triAreaFunc(b,h)
    % a = triArea(b,h)
    % Returns the area of a triangle
    % with base b and height h.
    a = 0.5*(b * h);
end

```

- Functions are denoted with the keyword “function” and are closed with the keyword “end”

Functions (file based!)

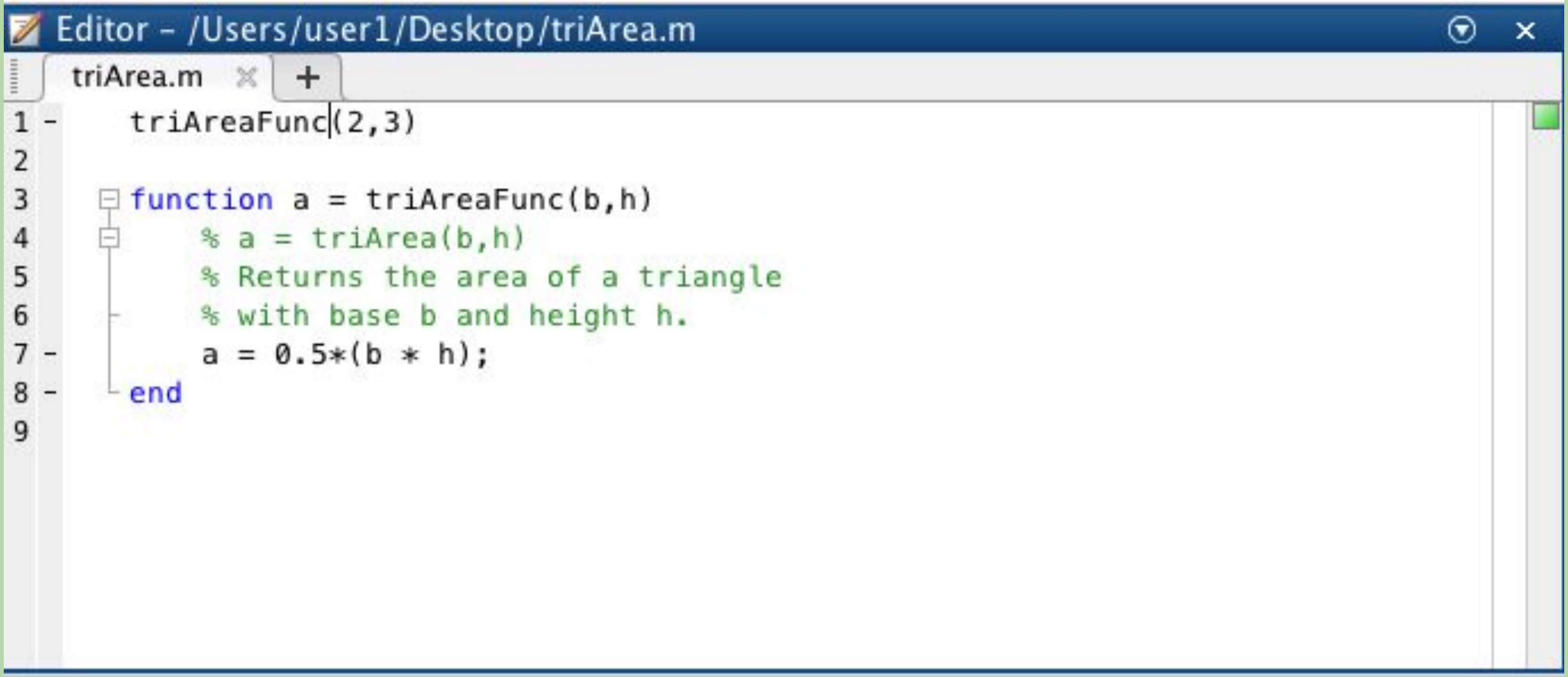
- Functions
  - Allow you to pass arguments to them and have a private (local) Workspace
  - Functions execute as if they have their own copy of MATLAB with its behavior (value it returns) determined by the values of the arguments passed to it
  - Functions, unlike scripts, allow one to easily build complex programs from smaller programs
  - We ♥ functions
  - **NOTE:** If a line does not end in a semicolon, the output of that line will be printed in the console window
    - Useful for debugging, but it can cause too much output to appear

Example Function (triArea.m)

```
function a = triAreaFunc(b,h)
   % Returns the area of a triangle
   % with base b and height h.
   a = 0.5*(b * h);
end
```

The first line in a function specifies the value(s) it will return (it’s outputs), the function name, and it’s arguments (it’s inputs)

Example Function (triArea.m)



```
triAreaFunc(2,3)

function a = triAreaFunc(b,h)
    % a = triArea(b,h)
    % Returns the area of a triangle
    % with base b and height h.
    a = 0.5*(b * h);
end

```

- **Note**: We store the return value in the variable to which we assign the function - this is important!

Functions are Flexible

Arguments mean we can apply the function to all sorts triangles

```
a1 = triAreaFunc(1,5)
a1 =
    2.5000
a2 = triAreaFunc(2,10)
a2 =
    10
a3 = triAreaFunc(3,6)
a3 =
     9
```

Remarks

```
function a = triAreaFunc(b,h)
   % Returns the area of a triangle
   % with base b and height h.
   a = 0.5*(b * h);
end
```

The comments (lines that start with %) immediately after the first line are displayed when help or doc is invoked on the function name

```
>> help triArea
  Returns the area of a triangle with
  base b and height h.
```

Writing Function Headers

```
function a = triAreaFunc(b,h)
   % Returns the area of a triangle
   % with base b and height h.
   a = 0.5*(b * h);
end
```

In CS4 you must use function headers of this form:

1) function statement must be on first line

2) Following comment lines must concisely describe (declare) what the function does (and NOT how it does it).

Returning more than one value

- Just add variables to the list of values to be returned
- They will be returned in the order given

```
function [r1, r2] = myQuadRoots(a,b,c)
   % Returns the roots r1 and r2 of the
   % quadratic equation defined by ax^2+bx+c.
   % Assumes a is nonzero.
```

Returning two values

```
function [r1, r2] = myQuadRoots(a,b,c)
   % Returns the roots r1 and r2 of the
   quadratic
   % equation defined by ax^2+bx+c.
   % Assumes a is nonzero.

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

Returning two values

Let’s find the roots of x^2+3x+2.

```
>> [x1, x2] = myQuadRoots(1, 3, 2)
x1 =
    -1
x2 =
    -2
```

Check:

x^2+3x+2 = (x+1)(x+2)

iClicker Question 3.1

```
function [r1, r2] = myQuadRoots(a,b,c)
disc = b^2-4*a*c;
r1 = (-b+sqrt(disc))/2*a;
r2 = (-b-sqrt(disc))/2*a;
```

After executing

```
>> clear all
>> [x1, x2] = myQuadRoots(1,0,-9)
```

What are the values x1 and x2?

A) undefined **B) x1 = -3, x2 = 3**

C) x1 = 3, x2 = -3 **D) I don’t know**

iClicker Question 3.1

```
function [r1, r2] = myQuadRoots(a,b,c)
disc = b^2-4*a*c;
r1 = (-b+sqrt(disc))/2*a;
r2 = (-b-sqrt(disc))/2*a;
```

After executing

```
>> clear all
>> [x1, x2] = myQuadRoots(1,0,-9)
```

What are the values x1 and x2?

A) undefined **B) x1 = -3, x2 = 3**

C) x1 = 3, x2 = -3 **D) I don't know**

iClicker Question 3.2

```
function [r1, r2] = myQuadRoots(a,b,c)
disc = b^2-4*a*c;
r1 = (-b+sqrt(disc))/2*a;
r2 = (-b-sqrt(disc))/2*a;
```

After executing

```
>> clear all
>> [x2, x1] = myQuadRoots(1,0,-9)
```

What are the values x1 and x2?

A) undefined **B) x1 = -3, x2 = 3**

C) x1 = 3, x2 = -3 **D) I don’t know**

iClicker Question 3.2

```
function [r1, r2] = myQuadRoots(a,b,c)
disc = b^2-4*a*c;
r1 = (-b+sqrt(disc))/2*a;
r2 = (-b-sqrt(disc))/2*a;
```

After executing

```
>> clear all
>> [x2, x1] = myQuadRoots(1,0,-9)
```

What are the values x1 and x2?

A) undefined **B) x1 = -3, x2 = 3**

C) x1 = 3, x2 = -3 **D) I don’t know**

iClicker Question 3.3

```
function [r1, r2] = myQuadRoots(a,b,c)
disc = b^2-4*a*c;
r1 = (-b+sqrt(disc))/2*a;
r2 = (-b-sqrt(disc))/2*a;
```

After executing

```
>> clear all
>> [x1, x2] = myQuadRoots(1,0,-9)
```

What is the value of disc?

A) disc = 36

B) disc = -9

C) disc = 9

D) undefined

iClicker Question 3.3

```
function [r1, r2] = myQuadRoots(a,b,c)
disc = b^2-4*a*c;
r1 = (-b+sqrt(disc))/2*a;
r2 = (-b-sqrt(disc))/2*a;
```

After executing

```
>> clear all
>> [x1, x2] = myQuadRoots(1,0,-9)
```

What is the value of disc?

A) disc = 36 **B) disc = -9**

C) disc = 9 **D) undefined**

iClicker Question 3.4

```
% triAreaScript.m - Computes triangle area
b = 5;
h = 3;
a = 0.5*(b * h);
```

After executing

```
>> clear all
>> triAreaScript
```

What are the value of b,h and a?

A) undefined **B) b,h undefined, a = 7.5**

C) b=5, h=3, a=7.5 **D) I don’t know**

iClicker Question 3.4

```
% triAreaScript.m - Computes triangle area
b = 5;
h = 3;
a = 0.5*(b * h);
```

After executing

```
>> clear all
>> triAreaScript
```

What are the value of b,h and a?

A) undefined **B) b,h undefined, a = 7.5**

C) b=5, h=3, a=7.5 **D) I don't know**

Variables in Functions

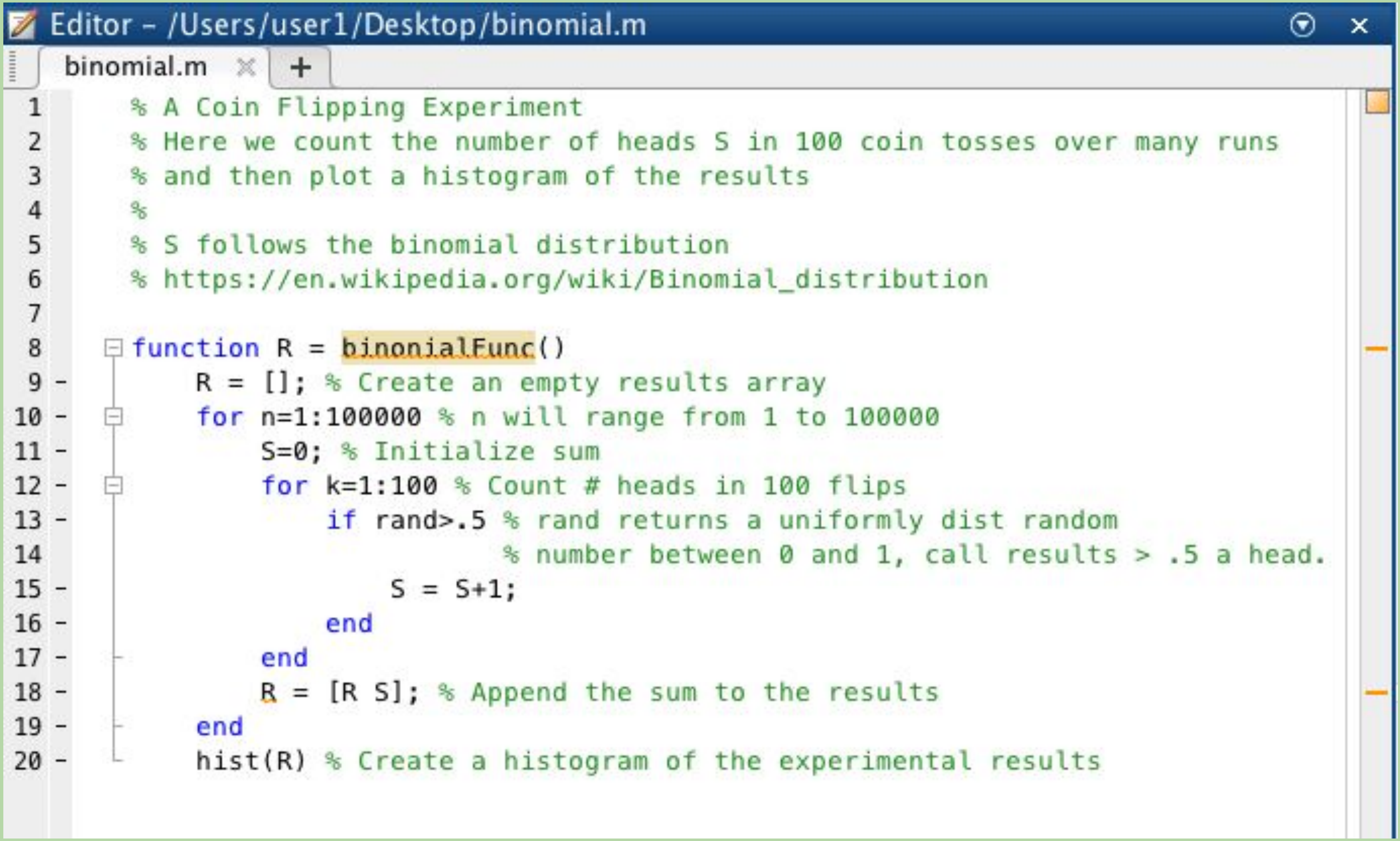
- Functions usually can only “see” values passed to them
- Therefore it is usually enough to look at function’s header to understand what it does
- This also limits unintended consequences and leads to clearer code

Variables in functions are local

```
function a = triAreaBad(h)
   % Returns the area of a triangle with
   % base b and height h.
   a = 0.5*(b .* h);
```

```
>> b=10; triAreaBad(10)
Undefined function or variable 'b'.
Error in triAreaBad (line 4)
a = 0.5*(b .* h);
```

Example: Binomial.m



```
% A Coin Flipping Experiment
% Here we count the number of heads S in 100 coin tosses over many runs
% and then plot a histogram of the results
%
% S follows the binomial distribution
% https://en.wikipedia.org/wiki/Binomial_distribution

function R = binonialFunc()
    R = []; % Create an empty results array
    for n=1:100000 % n will range from 1 to 100000
        S=0; % Initialize sum
        for k=1:100 % Count # heads in 100 flips
            if rand>.5 % rand returns a uniformly dist random
                       % number between 0 and 1, call results > .5 a head.
                S = S+1;
            end
        end
        R = [R S]; % Append the sum to the results
    end
    hist(R) % Create a histogram of the experimental results
```

Naming Scripts and Functions

- Same as for variables
- Use specific names (for example, *findRoots* instead of *doCalc*)
- Matlab is case-sensitive: *result* and *RESULT* are different, avoid using both!
- **Do not use names of built-in functions**

Search Order

- During evaluation of a variable, script or function, MATLAB first looks in the **current Workspace** and **current directory** and then searches `path` directories in order

```
>> path
   /Users/Dan/Documents/MATLAB
   /Applications/MATLAB_R2014b.app/toolbox/matlab/demos
   /Applications/MATLAB_R2014b.app/toolbox/matlab/graph
 2d
   /Applications/MATLAB_R2014b.app/toolbox/matlab/graph
 3d
   /Applications/MATLAB_R2014b.app/toolbox/matlab/graph
 ics
   …
```

Redefinition

- Incorrect **current folder** and accidental **redefinition** of built-ins is a very common mistake
- Use of the **`which`** command can help

```
>> which pi
built-in (/Applications/MATLAB_R2014b.app/toolbox/matlab/elmat/pi)
```

Redefinition

```
>> cos = 1;
>> cos(.1)   % oops
Subscript indices must either be real positive
integers or logicals.
>> cos(1)    % eek!
ans = 1

pi = 3; % iffy, but ok, if you don’t use pi as pi

i = 101; % ok, if you don’t use i as sqrt(-1)
```

Anonymous functions

- Sometimes the same calculation is used in many places inside a function

```
A1 = b*h/2;
A2 = b1*h2/2;
A2 = b1*h3/2;
A4 = b*h0/2;
```

Anonymous Functions

□ Matlab allows you to define a function using a single expression inline, e.g.,

```
areaT = @(b,h) b*h/2
A1 = areaT(b,h);
A2 = areaT(b1,h2);
A2 = areaT(b1,h3);
A4 = areaT(b,h0);
```

Passing, Redefining Functions

The @ operator is also used to refer to a function’s memory location, use it when passing functions and when reassigning functions, e.g.

```
someFun = @otherFun
higherOrderFun(3, @sin)
```