L11. User-Defined Functions

Input parameters
Local Variables
Output Values

Why?

1. Elevates reasoning by hiding details.

2. Facilitates 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

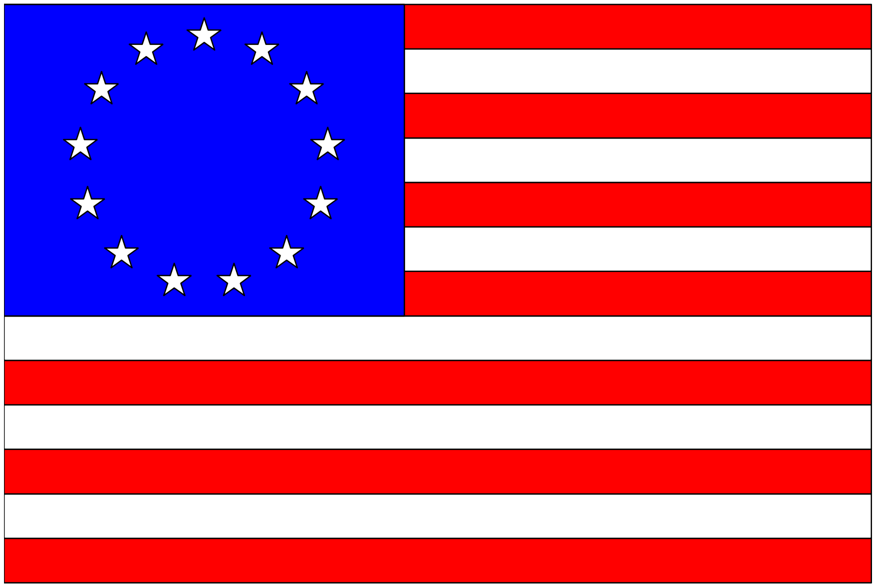
$ax^2 + bx + c = 0$

Elevates Reasoning

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



Facilitates Top-Down Design

1. Focus on how to draw the flag given just a <u>specification</u> of what the functions DrawRect and DrawStar do.

2. Figure out how to <u>implement</u> 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'.
```

To Implement a Function…

You write the code so that the function works. I.e., code that "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);
```

Not to worry. You will understand this 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 the Next 10 years :

You write software that makes extensive use of

`EPerimeter(a,b)`

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

Software Management

After 10 years :

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.

Example 1. MySqrt(A)

Recall that we can approximate square roots through the process of ractangle averaging

```
L = A; W = A/L;
L = (L+W)/2; W = A/L;
L = (L+W)/2; W = A/L;
```

etc

Package this Idea…

```
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

A User-Defined Function…

```
function s = MySqrt(A)
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

A Function Begins with a Header

```
function s = MySqrt(A)
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

A Function Has a Name

```
function s = MySqrt(A)
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

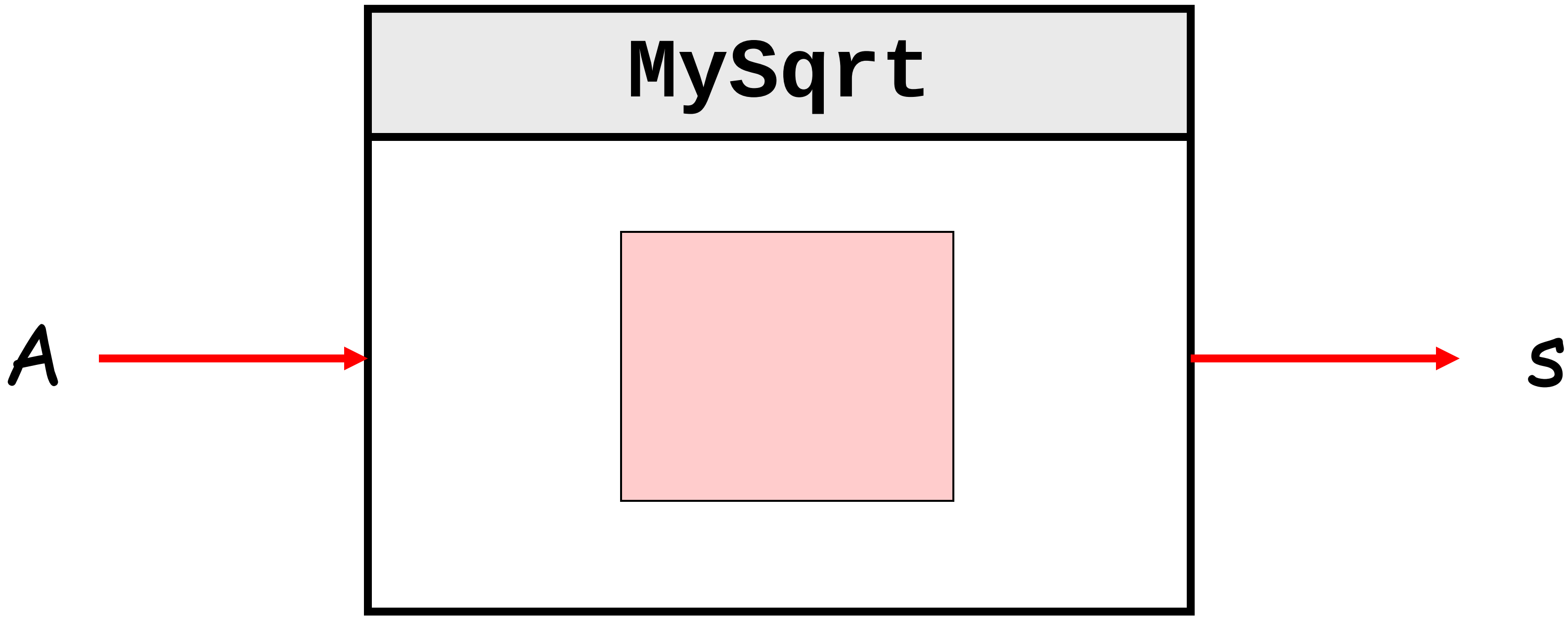
Input Arguments

```
function s = MySqrt( A )
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

Output Arguments

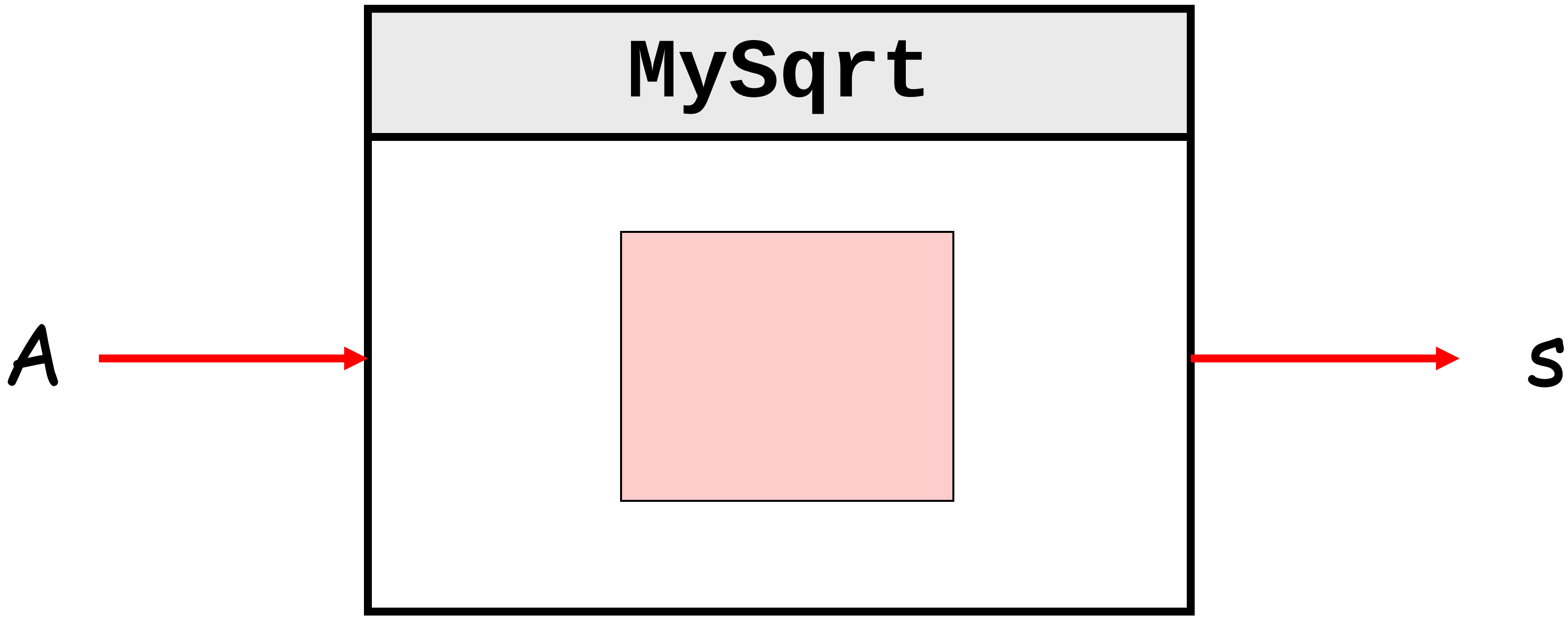
```
function s = MySqrt(A)
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

Think of MySqrt as a Factory



= Our method for approximating sqrt(A)

Hidden Inner Workings



Can use **`MySqrt`** w/o knowing how it works.

Practical Matters

The code sits in a separate file.

MySqrt.m

```
function s = MySqrt(A)
L = A; W = A/L;
for k=1:10
    L = (L+W)/2; W = A/L;
end
s = (L+W)/2;
```

Practical Matters

The .m file has the same name as the function.

Thus, in MySqrt.m you will find an implementation of MySqrt.

Practical Matters

The first non-comment in the file must be the function header statement.

E.g.,

```
function s = MySqrt(A)
```

Syntax

```
function            =       (           )
```

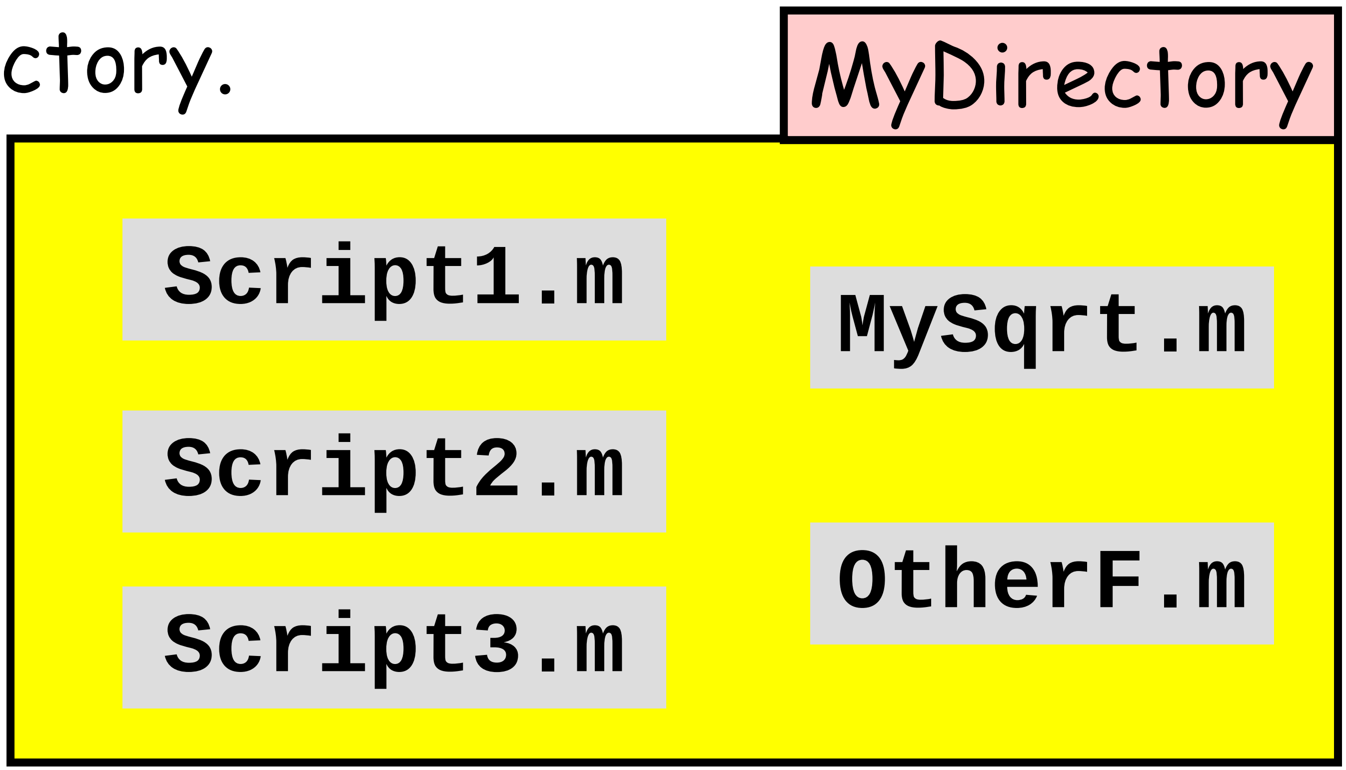
Name. Same rules as variable names

List of input parameters.

List of output parameters.

Practical Matters

For now*, scripts and other Functions that reference **MySqrt** must be in the same directory.



*The **path** function gives greater flexibility. More later.

Using MySqrt

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

Understanding Function Calls

There is a substitution mechanism.

Local variables are used to carry out the computations.

Script

```
a = 1
b = f(2)
c = 3
```

function

```
function y = f(x)
z = 2*x
y = z+1
```

Let's execute the script line-by-line and see what happens during the call to f.

Script

```
a = 1
b = f(2)
c = 3
```

function

```
function y = f(x)
z = 2*x
y = z+1
```

x, y, z serve as local variables during the process. x is referred to as an input parameter.

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

Green dot tells us what the computer is currently doing.

Control passes to the function.

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

x: 2

The input value is assigned to x

Control passes to the function.

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

x: 2

The input value is assigned to x

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

x: 2

z: 4

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

x: 2

z: 4

y: 5

The last command is executed

Control passes back to the calling program

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

b: 5

After the
the value is
passed back,
the call to the
function ends and
the local variables
disappear.

```
a = 1
b = f(2)
c = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

a: 1

b: 5

c: 3

Repeat to Stress the distinction between local variables and variables in the calling program.

Script

```
z = 1
x = f(2)
y = 3
```

function

```
function y = f(x)
z = 2*x
y = z+1
```

Let's execute the script line-by-line and see what happens during the call to f.

```
z = 1
x = f(2)
y = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

```
z: 1
```

Green dot tells us what the computer does next.

Control passes to the function.

```
z = 1
x = f(2)
y = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

z: 1

x: 2

The input value is assigned to x

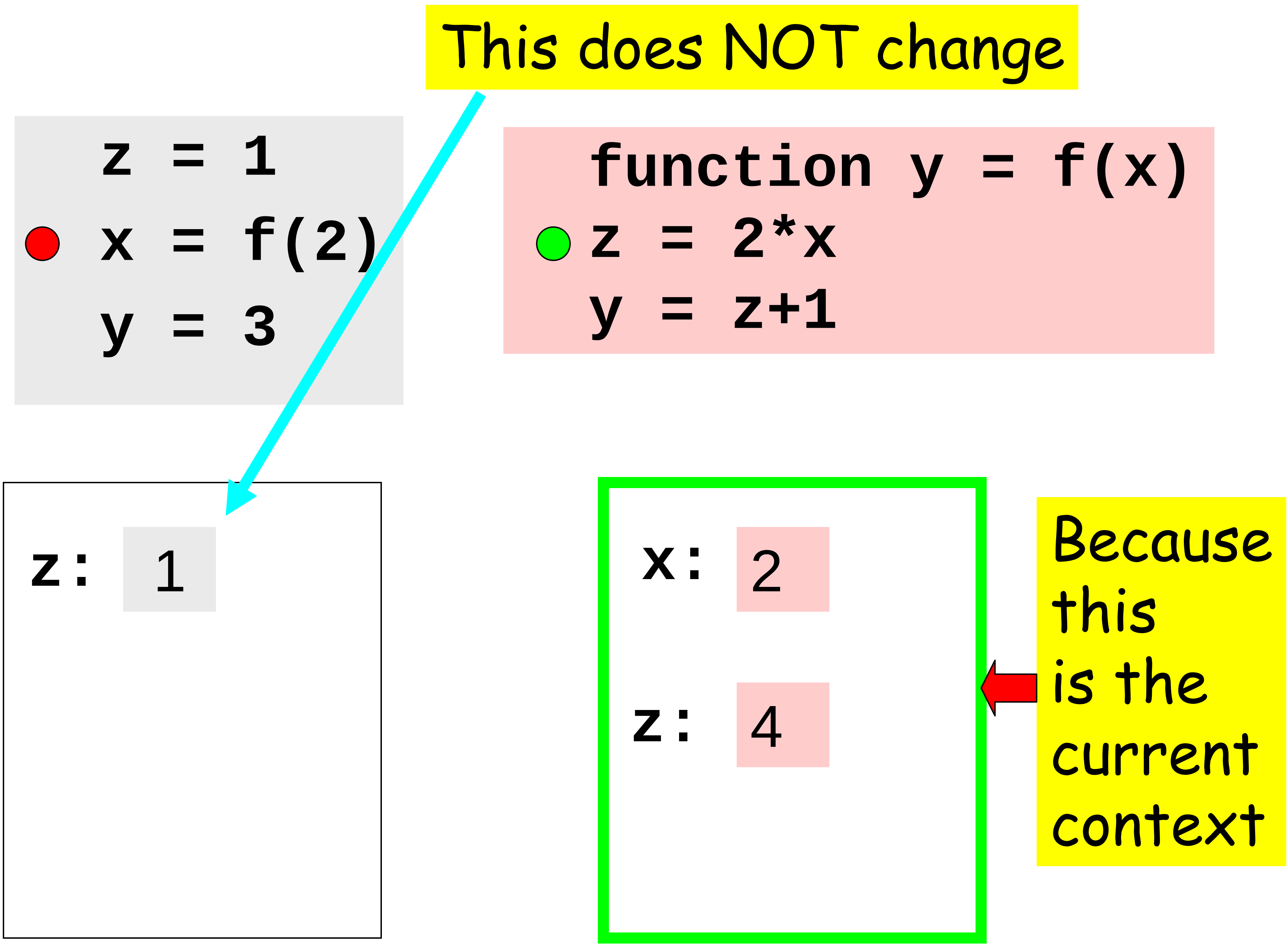
```
z = 1
x = f(2)
y = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

z: 1

x: 2

z: 4


This does NOT change
z = 1
x = f(2)
y = 3
function y = f(x)
z = 2*x
y = z+1
z: 1
x: 2
z: 4
Because this is the current context

```
  z = 1
● x = f(2)
  y = 3
```

```
  function y = f(x)
  z = 2*x
● y = z+1
```

z: 1

x: 2

z: 4

y: 5

The last command is executed

Control passes back to the calling program

```
  z = 1
● x = f(2)
  y = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

```
z: 1

x: 5
```

After the the value is passed back, the function "shuts down"

```
z = 1
x = f(2)
y = 3
```

```
function y = f(x)
z = 2*x
y = z+1
```

z: 1

x: 5

y: 3

Question Time

```
x = 1;
x = f(x+1);
y = x+1
```

```
function y = f(x)
x = x+1;
y = x+1;
```

What is the output?

A. 1 B. 2 C. 3 D. 4 E. 5

Question Time

```
x = 1;
x = f(x+1);
y = x+1
```

```
function y = f(x)
x = x+1;
y = x+1;
```

What is the output?

A. 1 B. 2 C. 3 D. 4 E. 5

Back to MySqrt

```
   function s = MySqrt(A)
% A is a positive real number
% and s is an approximation
% to its square root.
```

The specification is given in the form of comments just after the header statement.

Back to MySqrt

```
   function s = MySqrt(A)
% A is a positive real number
% and s is an approximation
% to its square root.
```

It must be clear, complete, and concise.

Back to MySqrt

```
   function s = MySqrt(A)
% A is a positive real number
% and s is an approximation
% to its square root.
```

— ∞ If ever you write a function with no specification!!!