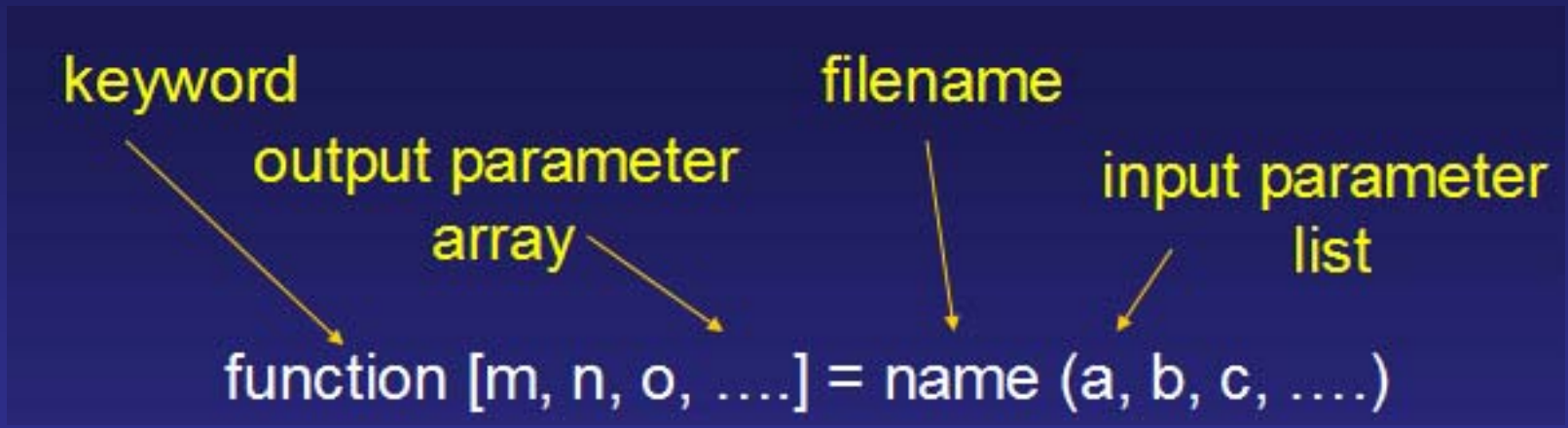
MatLab Programming – Functions



What have we learned last time

- Conditionals : if, else, elseif
- Conditionals: switch, case
- Loops: for
- Loops: while
- Nesting

- Finding trajectory numerically from equation of motion

What are Functions?

Functions are reusable building blocks for more complex programs

MatLab support functions of different types but two are particularly important:

- Named functions defined via m-files
- Anonymous function defined via function handles

Defining a named function via m-file

Consider a very simple function

```
function c=add(a,b)
c=a+b;
```

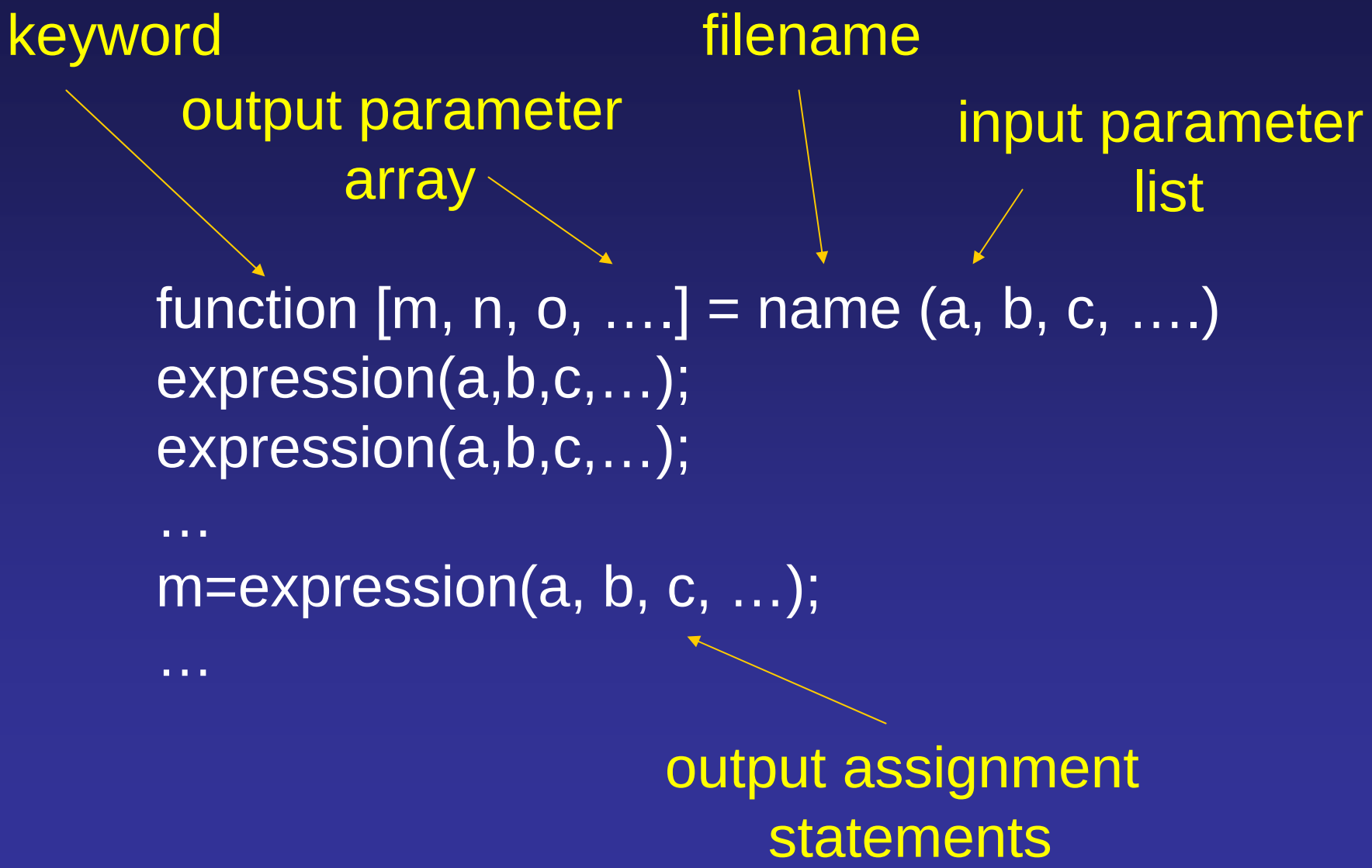
This function is save as an m-file, add.m, in the working directory

Calling this function from workspace:

```
add(2,3)

ans = 5
```

General structure of a function



A slightly more complex example

A basic statistics function that sums two input Arrays and gets its mean, standard dev, and number of elements:

```
function [avg, stdev, num] = sum_stat (X, Y)
Z=X+Y;
avg = mean(Z);
stdev = std(Z);
num = length(Z);
```

If we run:

```
>> sum_stat(1..3, 2..4)
```

What happened?

```
ans = 5
```

Functions: Assigning output parameters

Ok, try this:

```
>> [avg stdev num]=sum_stat(1:3, 2:4)

avg =
     5
stdev =
     2
num =
     3
```

If you don't define output parameters, function just returns the first one! MatLab automatically create these variables as you run the function.

Functions: significance of names of parameters

```
>> [a b c]=sum_stat(1:3, 2:4)

a =
     5
b =
     2
c =
     3
```

The original output parameter names defined in the function is of NO significance. New variable names are defined as the function is called!
The ORDER of the parameters is key!!!!

Functions: Input parameters

Some input parameters can of course be variables

```
>> X=1:3;
>> Y=2:4;
>> [avg stdev num]=sum_stat(X,Y)
avg =
     5
stdev =
     2
num =
     3
```

Functions: Name of input parameter list

The names for the variables in a function's input parameter list are also dummys

```
>> M=1:3;
>> N=2:4;
>> [avg stdev num]=sum_stat(M,N)

avg =
     5
stdev =
     2
num =
     3
```

What is scope?

What is scope? Scope is the range of validity

```
function [a b]=testscope(c, d)
x=1;
c=c+x;
a=c;
d=d+x+1;
b=d;
```

If we run:

```
>> [a b]=testscope(1, 2)
a =
     2
b =
     4
```

Scope of function & main window

What if we run:

```
>> x=10;
>> [a b]=testscope(1, 2)

a =
     2
b =
     4
```

Note that output for a, b is not 11 and 13 but 2 and 4. That is because the variable, x, in the main window which is equal to 10 is different from the x in the function which is 1.

Scope Rule

(1) Variables defined locally in each section of the program are independent even if they have the same name!

(2) The variables in a functions input and output parameter lists are considered to be local variables defined ONLY inside the function.

(3) Values are “passed” into the variables in the input parameter list at the start of a function and the values are passed from the parameters in the output parameter list

Understanding scope further

```
>> c=1;
>> d=2;
>> [a b]=testscope(c, d)

a =
     2
b =
     4
>> c
c =
     1
>> d
d =
     2
```

Note that c and d inside the function are changed but the c and d in the main program are unaffected by the function's operation.

What if we want a function to change the value of its input parameters?

```
>> c=1;
>> d=1;
>> [c b]=testscope(c, d)

c =
     2
b =
     3
>> c
c =
     2
```

To accomplish this, all you have to do is to put the same variable in the input AND output parameter lists

Debugging a MatLab Function

Important concepts in debugging MatLab functions

nargin – returns the number of input arguments
nargout – returns the number of output arguements

A debuggable version of sum_stat:

```
function [avg, stdev, num] = sum_stat2 (X, Y)
if nargin==0, X = 1:4; Y=2:4; end
Z=X+Y;
avg = mean(Z);
stdev = std(Z);
num = length(Z);
```

Runtime error in a MatLab Function

What if we run?

```
>> M=1:4;
>> N=2:4;
>> [avg stdev num]=sum_stat(M,N)
??? Error using ==> plus
Matrix dimensions must agree.
Error in ==> sum_stat at 2
Z=X+Y;
```

Recall function:

```
function [avg, stdev, num] = sum_stat (X, Y)
Z=X+Y;
avg = mean(Z);
stdev = std(Z);
num = length(Z);
```

Debugging a MatLab Function

Important concepts in debugging MatLab functions

nargin – returns the number of input arguments
nargout – returns the number of output arguements

A debuggable version of sum_stat:

```
function [avg, stdev, num] = sum_stat2 (X, Y)
if nargin==0, X = 1:4; Y=2:4; end
Z=X+Y;
avg = mean(Z);
stdev = std(Z);
num = length(Z);
```

Runtime error in a MatLab Function

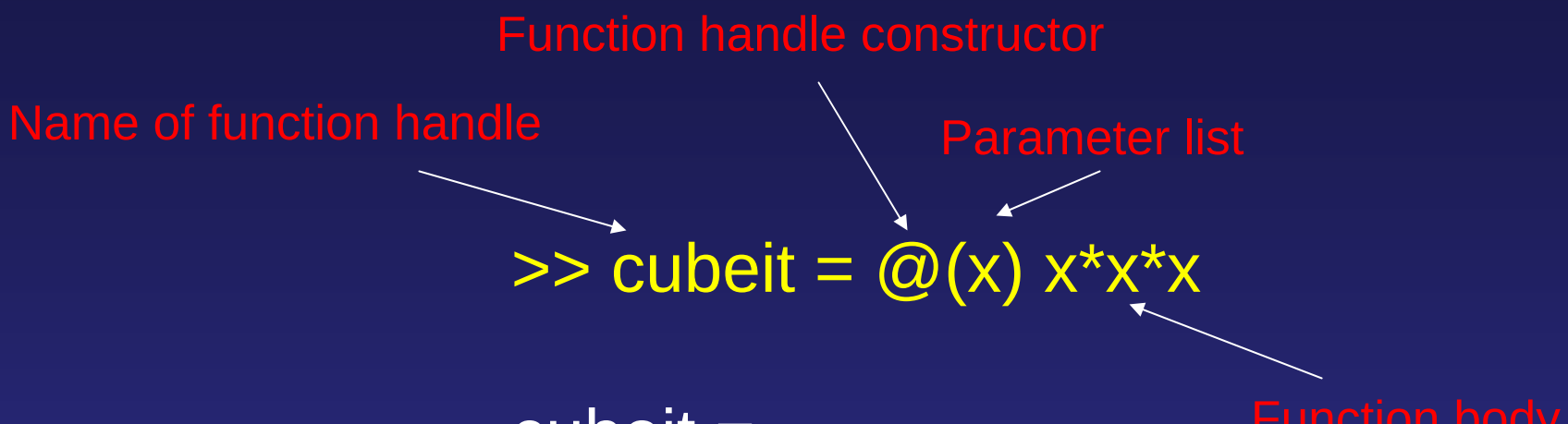
What if we run?

```
>> M=1:4;
>> N=2:4;
>> [avg stdev num]=sum_stat(M,N)
??? Error using ==> plus
Matrix dimensions must agree.
Error in ==> sum_stat at 2
Z=X+Y;
```

Recall function:

```
function [avg, stdev, num] = sum_stat (X, Y)
Z=X+Y;
avg = mean(Z);
stdev = std(Z);
num = length(Z);
```

Anonymous Functions?




```
cubeit =
    @(x) x*x*x
>> cubeit(3)

ans =
    27
```

Handles are special kind of variable that identify the start of a chuck of code in memory

More complex anonymous functions

```
>> sum_and_sqr = @(x,y) (x+y)*(x+y)

sum_and_sqr =
    @(x,y) (x+y)*(x+y)

>> sum_and_sqr(2,3)

ans =
    25
```

Communication with anonymous function

```
>> a = 3;
>> a_times_it = @(x) a*x;
>> a_times_it(2)

ans =
     6

>> a = 2;
>> a_times_it(2)

ans =
     6
```

Note that:

1. Anonymous function definition can see variables defined on the desktop
2. Once the value of the variable is captured in the function definition future changes of the variable are ignored

3. Avoid using too much anonymous functions

Why do I tell you about anonymous functions?

Consider the MatLab function “quad” which does this:

quad(func, a, b)

$$y = \int_a^b f(x)dx$$

What is func? Func is a function handle that we have talked about.

We can create func by either through m-file function or anonymous function

Using function handle to pass functions I

Define m-file:

```
function y=fun1(x)
y=x.*x;
```

Now we are ready to use quad:

```
>> quad(@fun1, 0, 1)

ans =
    0.3333
```

@ associate the name of a function to a function handle

Using function handle to pass functions II

```
>> fun2 = @(x) x;
>> quad(fun2, 0, 1)

ans =

    0.5000
```

This is a quicker way. Note that fun2 is already a function handle, i.e. no @ in quad

What happen if we want to change the parameters of a function?

$$y = \int_0^1 (ax + b)dx$$

What happen if we want to change a & b at will without defining a new function handle every time?

```
>> g = @(a,b) quad(@(x) a*x+b, 0, 1);
>> g(1, 0)
ans =
    0.5000
>> g(0, 1)
ans =
```

Sub-Functions

```
function y=subfunctions(x,n)
    switch n
        case 1
            y=square(x);
        case 2
            y=cube(x);
        otherwise
            y=x;
    end
end
function y=square(x)
    y=x.*x;
end
```

```
function y=cube(x)
    y=x.*x;
    y=y.*x;
end
```

Note:

1. The first function is the main function

2. The scope of the variables in the main and subfunctions are all local

Global Variables

```
function y=subfunctions3(x,n)
   switch n
       case 1
           y=square(x);
       otherwise
           y=x;
    end
end
function y=square(x)
global a;
    y=x.*x;
    y=a*y;
end
```

```
>> global a;
>> a=1
a =
     1
>> subfunctions3(3,1)
ans =
     9
>> a=2
a =
     2
>> subfunctions3(3,1)
ans =
    18
```