MATLAB PROGRAMMING TECHNIQUES

Pony Lai

support@terasoft.com.tw

Course Outline

- Writing Functions
- Structuring Code

Advanced MATLAB® Programming Techniques

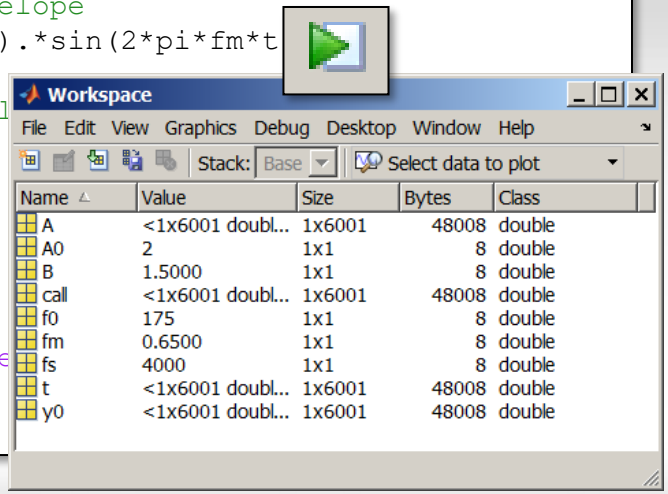
Writing Functions

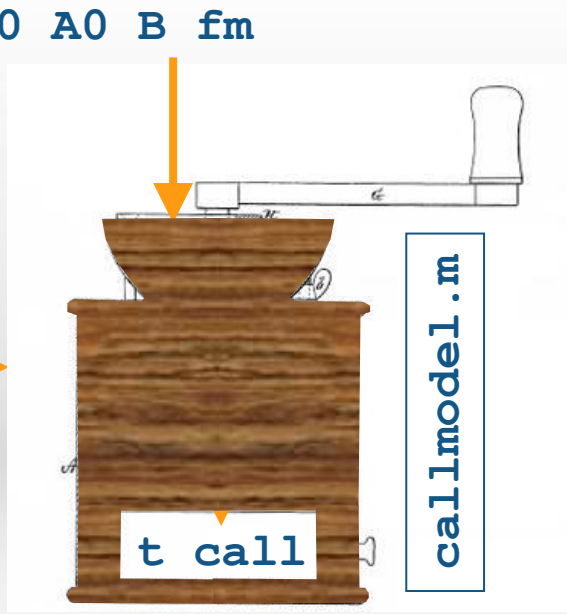
Section Outline

- Creating functions
- Calling functions
- Workspaces
- Path and precedence

Increasing Automation

```
% Create the time base for the signal.
fs = 4000;
t = 0:(1/fs):1.5;

% Set the fundamental frequency of the call.
f0 = 175;

% Create the harmonics.
y0 = sin(2*pi*f0*t) + ...
     sin(2*pi*2*f0*t) + sin(2*pi*3*f0*t);

% Set the additional parameters in the model.
A0 = 2; % Initial amplitude.
B  = 1.5; % Amplitude decay rate.
fm = 0.65; % Frequency of the modulating envelope.
% Create the envelope
A  = A0*exp(-B*t).*sin(2*pi*fm*t

% Create the call
call = A.*y0;

% Plot the model
figure
plot(t,call)
xlabel('Time')
ylabel('Amplitude
title('{\bf Blue
soundsc(call,fs)
```





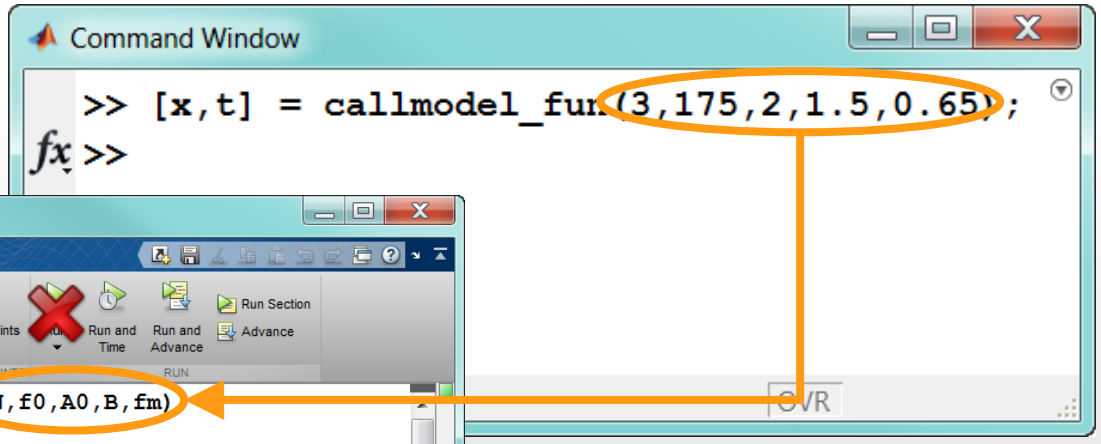
Creating a Function

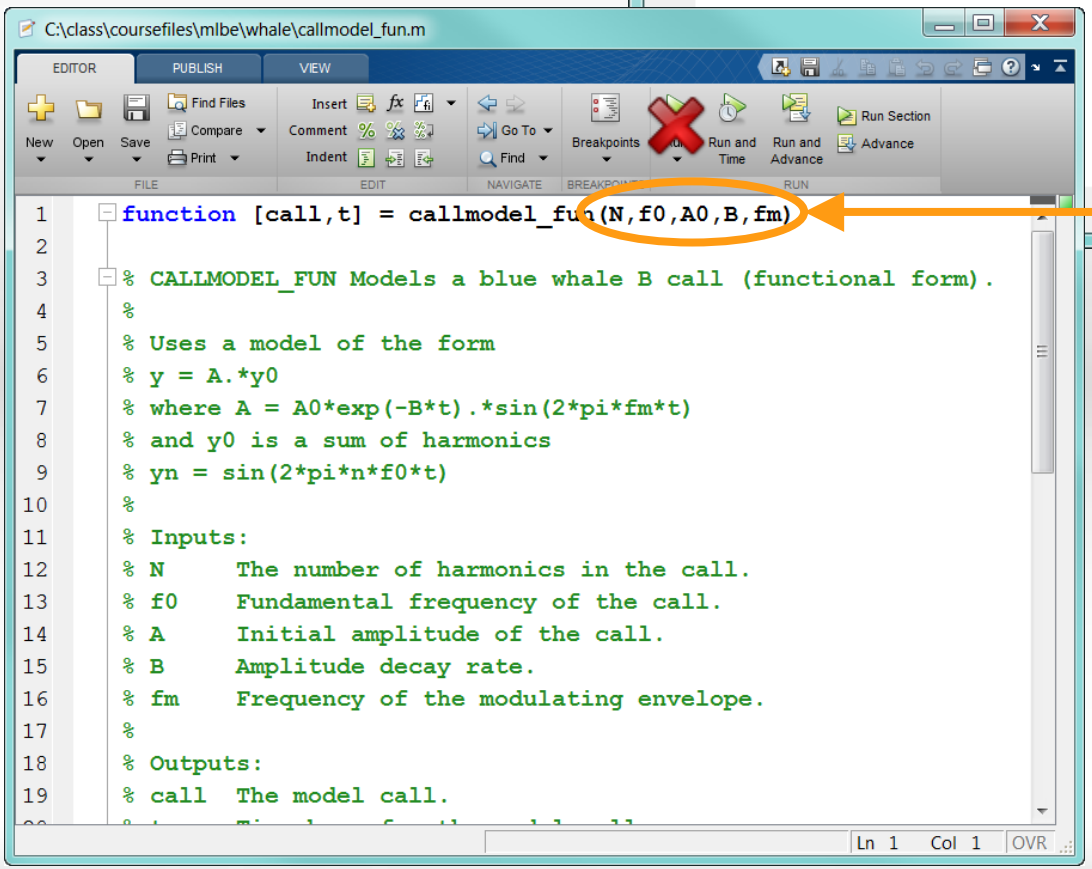
Function declaration: Keyword | Output arguments | Function name | Input arguments

```
1  function [call,t] = callmodel_fun(N,f0,A0,B,fm)
2
3  % CALLMODEL_FUN Models a blue whale B call (func
4  %
5  % Uses a model of the form y = A.*y0
6  % where A0 = A*exp(-B*t).*sin(2*pi*fm*t)
7  % and y0 is a sum of harmonics
8  % yn = sin(2*pi*n*f0*t)
```

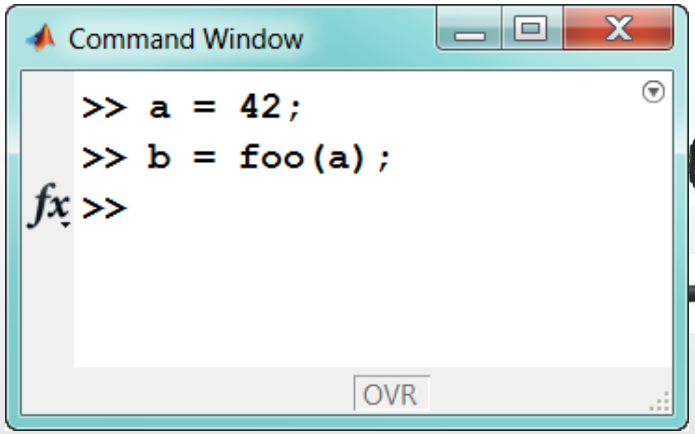
Calling a Function

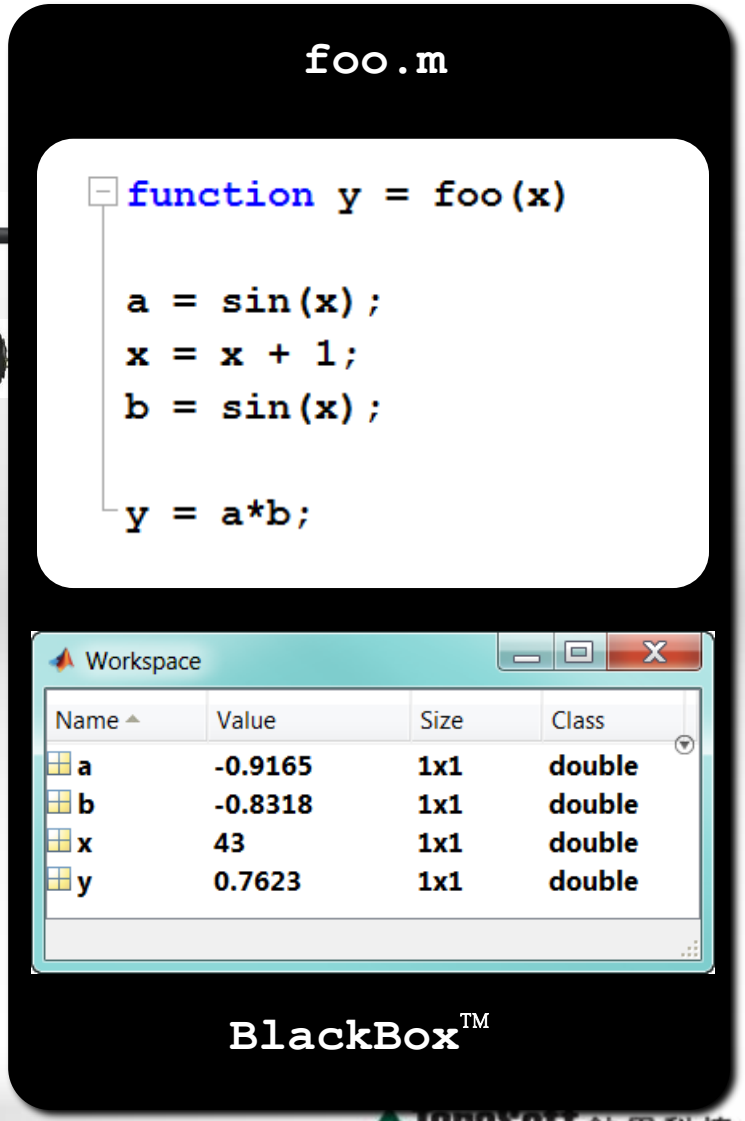
callmodel_fun.m

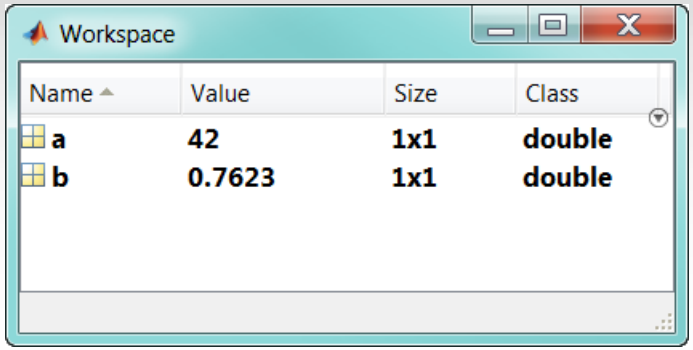




Workspaces







Calling Precedence

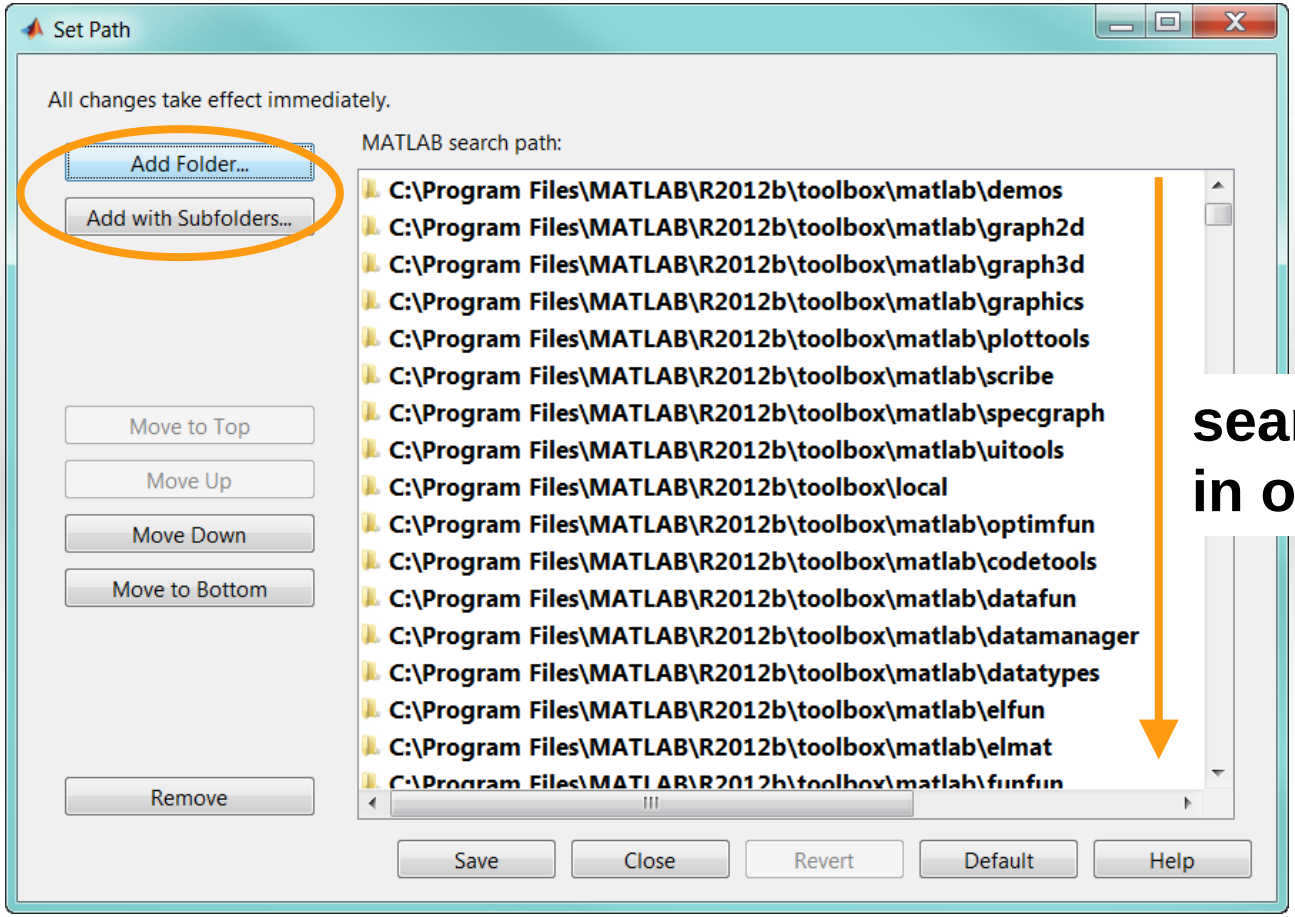
```
>> whale
```

1. Variable
2. Nested function
3. Subfunction
4. Private function
5. Class constructor
6. Overloaded method
7. File in the current directory
8. File on the path

1. whale.bi
2. whale.mexw32
3. whale.mdl
4. whale.p
5. whale.m

The MATLAB® Path

```
>> pathtool
```

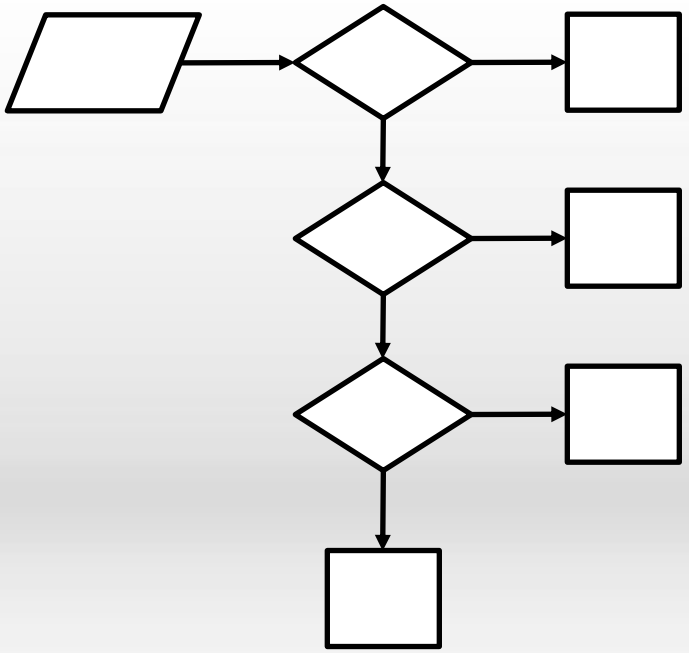


Advanced MATLAB® Programming Techniques

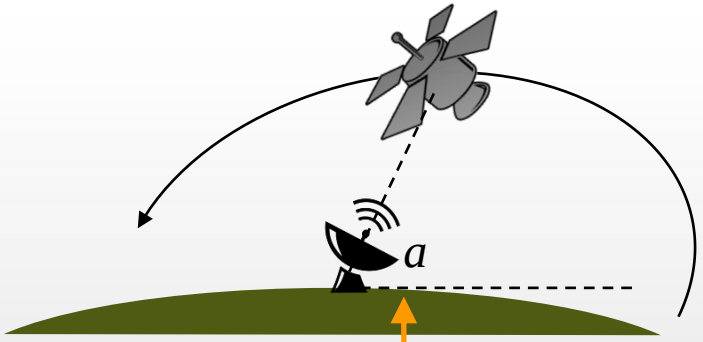
Structuring Code

Section Outline

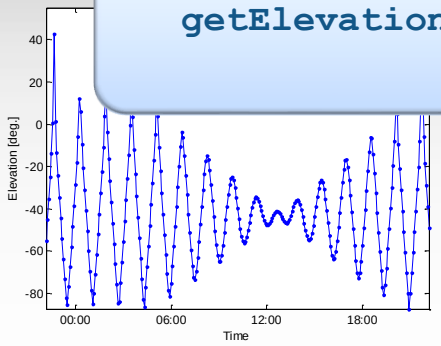
- Private functions
- Subfunctions
- Nested functions
- Function handles
- Anonymous functions
- Precedence rules
- Comparison of function types



Course Example: Satellite Tracking



getElevations



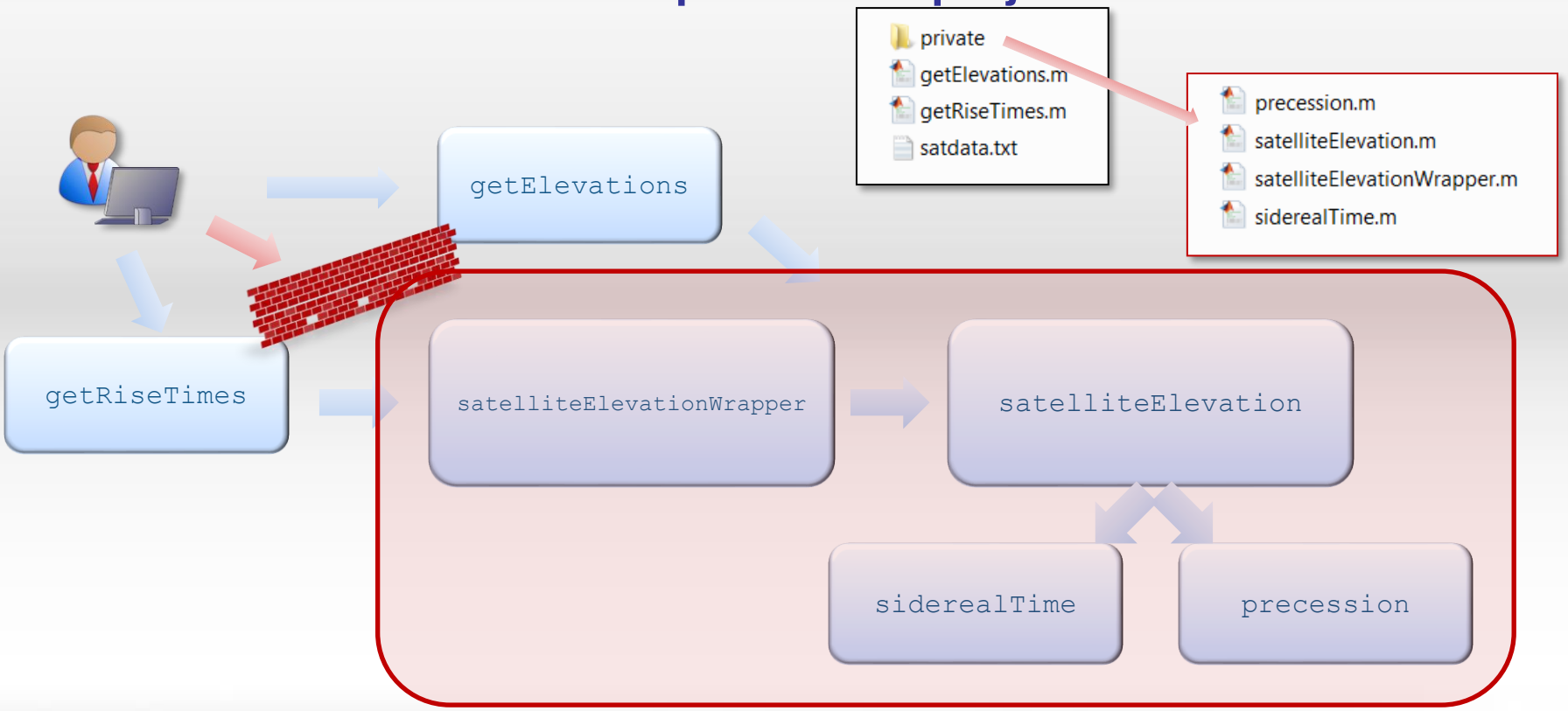
satelliteElevation

siderealTime

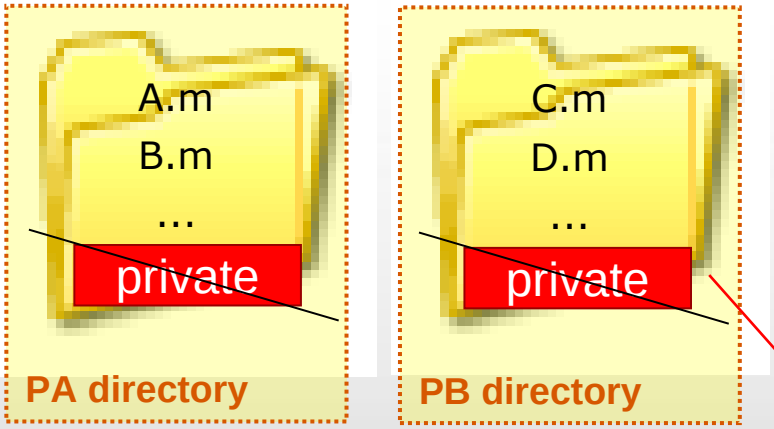
precession

Private Functions

- Function files in a folder named private
- Accessible only from within this and the parent folder
- Use case: make function specific to a project



Private Functions




```
>> cd([matlabroot '/work/examples'])
>> edit AA.m
```

```
function AA(x)
C(x);
```

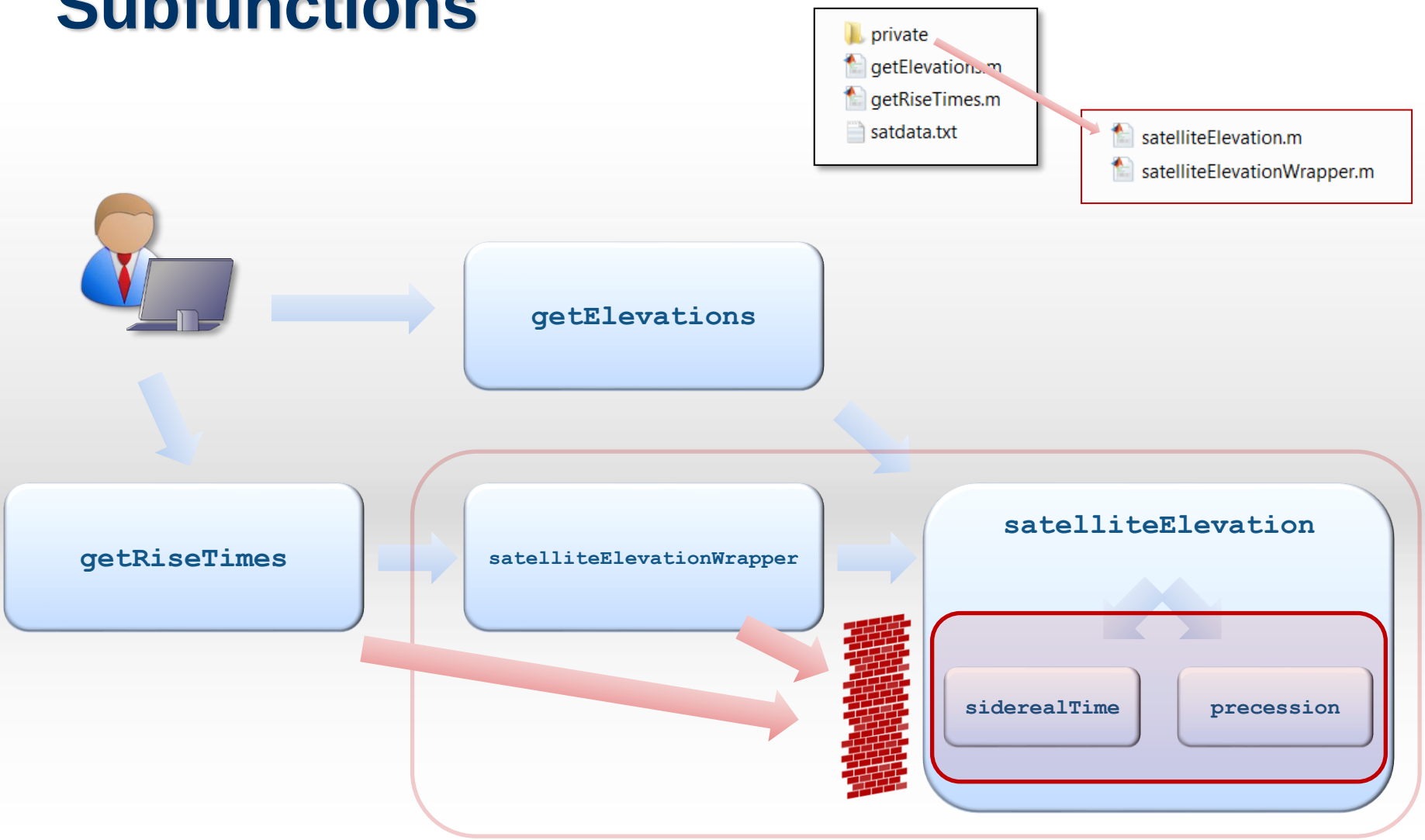
Subfunctions

- Several functions in one file
- Keyword `function` used as delimiter
- First function accessible from outside world
- Others accessible only from within the same file
- Use case: hide internal utility functions

Optional when using only subfunctions

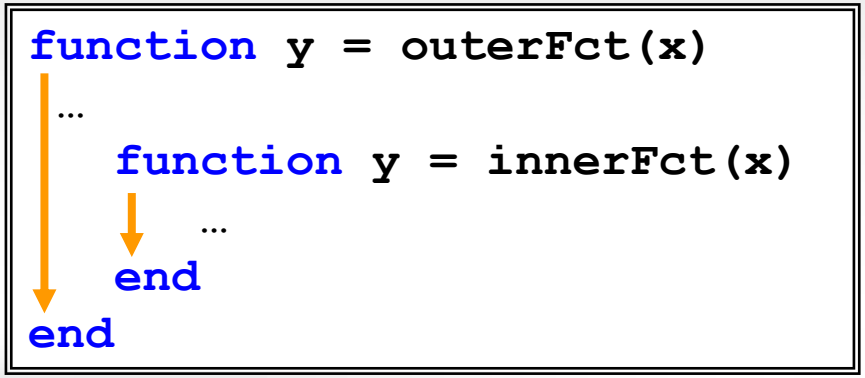
```
function y = primaryFct(x)
...
end
function y = subFct1(x)
...
end
function y = subFct2(x)
...
end
```

Subfunctions

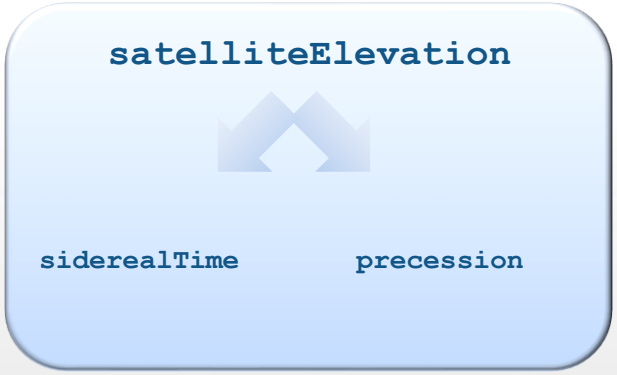


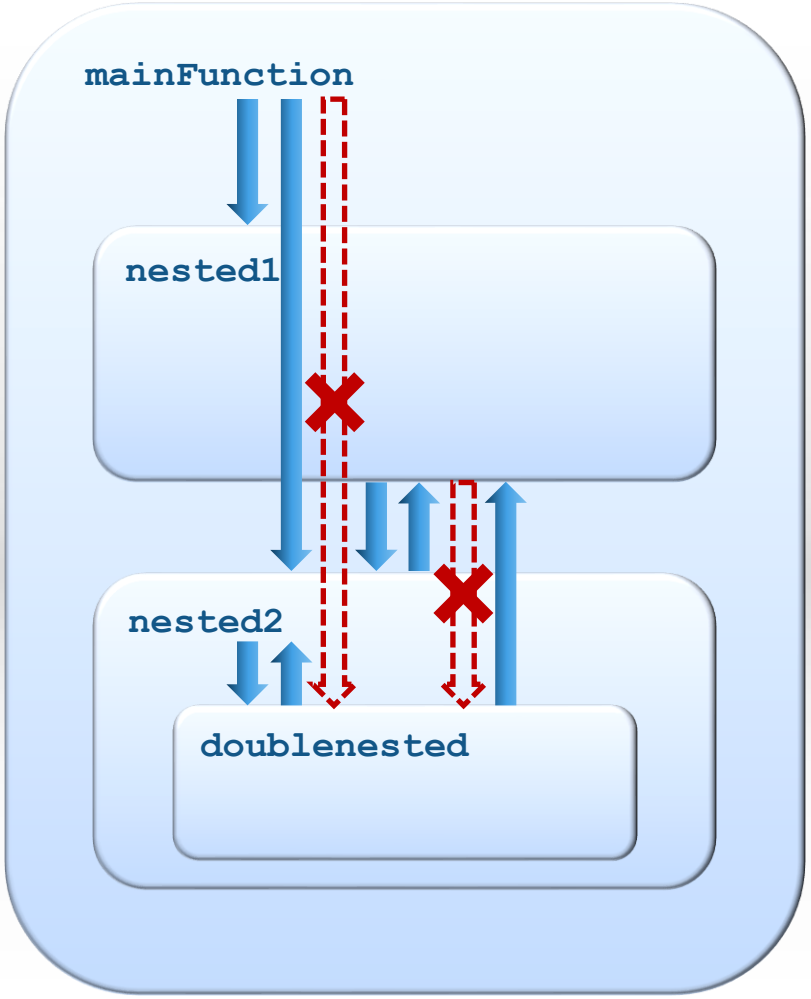
Nested Functions

- Functions nested inside other functions
- Mark their extent using `function` and `end`
- Can be called
  - From level immediately above
  - From function at same level within same parent function
  - From a nested function at any lower level
- Access to superior function workspaces
- Have their own workspace

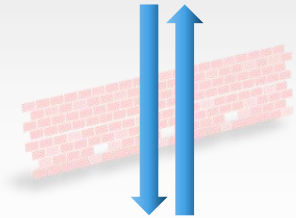


Nested Functions





main function data



nested function

Example: Nested Functions

```
function T = tax(income)

adjusted_income = max(income - 6000, 0);

T = compute_tax;


    function t = compute_tax

        t = 0.28*adjusted_income;

    end

end
```

Scope of a Variable

Using Subfunction

```
function [A,B] = sub_scope(x,y)

A = subfun1(x);
B = subfun2(y);

function v = subfun1(u)
v = rand(u,1);

function v = subfun2(u)
v = randn(u,1);
```

separate workspaces

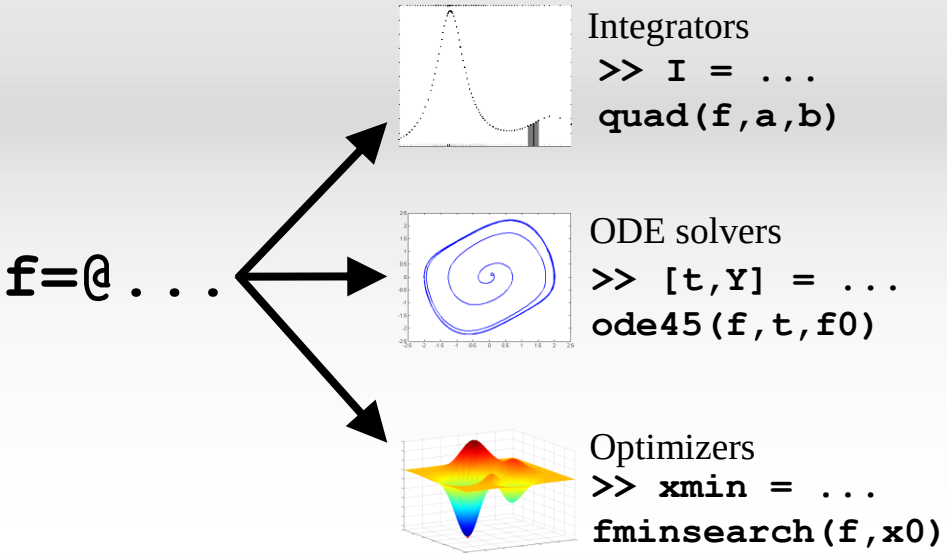
Using Nested Function

```
function T = tax(income)

adj_income = ...
max(income - 6000, 0);
T = compute_tax;

    function t = compute_tax
        t = 0.28*adj_income;
    end
end
```

shared workspaces

Function Handles

- Special MATLAB data type
- Create a variable for calling a function.
- Use case 1: Flexible/dynamic function calls
- Use case 2: Extending the visibility of a function
- Use case 3: Changing the function interface



Creating and Using Function Handles

- **Syntax**

```
fhandle = @functionname
```

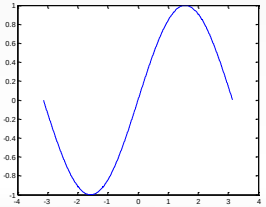
- **Example**

create → `fhandle = @sin;`

use → `fhandle(arg1, arg2, ...);`

```
function plot_fhandle(fhandle, data)
plot(data, fhandle(data));
```

```
>> plot_fhandle(@sin, -pi:0.01:pi)
```



Example:
access to subfunction using function handle

main function

sub-function

```
function myFhandle = myFunction(myInput)
SomeOtherValue = 7;
myFhandle = @mySubFunction;

disp(['7 times ' num2str(myInput) ' is ' ...
    num2str(myFhandle(myInput, SomeOtherValue))]);

function myReturnValue = mySubFunction(x, y)
myReturnValue = x.*y;
```

```
>> myTimes = myFunction(7)
>> myTimes(8,2)
```

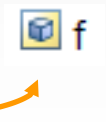
Anonymous Functions

- Wrapper to slightly change the function (interface)
- Write @ followed by list of arguments and function call
- No function name
- No file necessary

```
>> f = @myfun;
```

```
function y = myfun(a,b,c)
y = a*(b-sin(c));
```

```
>> f = @(a,b,c) a*(b-sin(c));
```



Comparison of Function Types

Aspect	Private	Sub	Nested	Anonymous
File	Yes	Yes	Yes	No
Workspace	Separate	Separate	Shared	Depends
Access	Files in `private` and parent folder	Within file	See page 2-6	Via function handle variable
Typical use	Project specific functionality	Hide utilities	Share application data	Change of function interface

Q & A