Persistent Variables

• Normally when a function finishes its task, all of the variables used within the function go out of scope.
• The local variables have no legitimate reason for existing any more as the function no longer needs them.
• Typically the memory allocated to these variables is freed for reuse.
• The next time the function is called, a new local workspace is created with new local variables.
• There is no connection between any newly created local variables and the ones previously destroyed.

Persistent Variables

• However, there are occasions where you want the value of a variable within a function to persist and be "remembered" from function call to function call.
• We can declare a variable as one to be remembered between function calls with the `persistent` statement.

Example

```matlab
% RUNNINGAVERAGE: A function that maintains a running average of values entered by the user.
% Usage: runningAverage('reset')
% initialise the average.
% av = runningAverage(x)
% enter a new value into the running average.
% Arguments: x - The new value to be entered.
% Returns: av - The running average of the values entered so far.
% Author: PK
% Date: August 2009
```
Example (cont.)

```matlab
function av = runningAverage(x)
    % The number of values entered so far - declared persistent.
    persistent n;
    % The sum of values entered so far - declared persistent.
    persistent sumOfX;
    if x == 'reset' % Initialise the persistent variables.
        n = 0;
        sumOfX = 0;
        av = 0;
    else % A data value has been added.
        n = n + 1;
        sumOfX = sumOfX + x;
        av = sumOfX / n; % Update the running average.
    end
```

Calling the function

```matlab
>> runningAverage('reset')
an = 0
>> runningAverage(5)
an = 5
>> runningAverage(10)
an = 7.5000
>> runningAverage(3)
an = 6
>> runningAverage('reset')
an = 0
>> runningAverage(8)
an = 8
```

Example (cont.)

- Information about the running average remains between successive function calls of runningAverage.
- When using persistent variables, you typically need to provide some mechanism for initialising and resetting them.

Global variables

- So far we have been concerned with information hiding to prevent unwanted interaction between functions.
- There are (very rare) occasions when Matlab’s pass-by-value approach for communicating values to functions becomes cumbersome.
- There are two main reasons why this might be the case:
  - The data object to be shared is so large that the time and memory cost in creating a copy may be too large. For example, there may only be enough memory to store one copy of the data object.
  - There may be some data that needs to be used by a very large number of functions. Communicating this data via an argument or specifying the value in each one of these functions may be excessively cumbersome.
Examples

• For example, gravitational acceleration is 9.81 ms\(^{-2}\). Rather than defining a variable:

\[
G = 9.81;
\]

in every function that uses this value, you could declare it as a global variable.

• Variables are declared to be global with the `global` statement.

• For example:

```matlab
global G;  % Declare G to be global.
G = 9.81;  % Then specify its value
% (in that order).
```

• Declaring a variable to be global means that it is stored in the global workspace and not in a local workspace.

• Any function wanting to use this global variable must also include the statement:

```matlab
global G;     % This tells Matlab to use the global variable called G and not create a % new local variable.
```

• This statement must be placed at the beginning of the function code before any attempt to use the variable G is made.

The very big danger

• If any function accidentally (through programming error) corrupts the value of G, the resulting mess spreads through the rest of your code.

• Always avoid global variables where possible.

• Reserve global variables for cases where the data object is so huge that passing it as an argument becomes difficult.

• If possible only share global variables between a function and subfunctions within the same file. Note however, even passing matrices of size 1000x1000 is no problem.

• Global variables should be fully capitalised to signify their status.

Example

```matlab
% FALLINGVELOCITY: A function to calculate the velocity of an object falling under the influence of gravity.
% Usage: vel = fallingVelocity(t)
% Arguments: t - The time in seconds.
% Returns: vel - The velocity of the object.
% Author: QL
% Date: August 2009

function vel = fallingVelocity(t)

global G;
vel = G*t;
end % function fallingVelocity
```
Another way of avoiding global variables

• One way of avoiding global variables for values that are kept constant is to define a function that sets the value for you.
• For example, in the case of the gravitational constant G, you would define a function called G in a file called G.m as follows:

```matlab
function value = G
    value = 9.81;
end %function G
```

Now if you have a piece of code such as

```matlab
vel = G * t;
```
• The G now refers to a function. MATLAB will
  – invoke the function G.m,
  – receive back the value 9.81, and then
  – multiply that by t.
• Note you still have the potential problem of shadowing should you inadvertently create a variable also called G.