

# Variable Scope





## ***Public (aka. Global)***

**Use Public inside Module1.bas**

Global variables are defined in standard modules (see following paragraph). A global can be accessed (read or written to) anywhere in the entire project. However, erroneous side-effects can occur if one part of a program accesses a variable when it shouldn’t. To solve this problem it is possible to limit the access scope of a variable to specific parts of a project only (see Module and Local below). An analogy with the ID pass above would be to limit certain personnel to various parts of the building.

## ***Module***

**Use Private inside General / Declarations**

Visual Basic has three types of modules: form modules (.frm), standard modules (.bas), and class modules (.cls). A form module contains a ‘general declarations’ section, subroutines for all events used, and any programmer-defined subroutines. A standard module, like the form module, also contains a ‘general declarations’ section. It is also used to hold any public subroutines which will be made accessible to the whole application. Finally, class modules are used to write code for new objects.

Variables declared as module wide can only be accessed by subroutines *within* the current module (i.e. any subroutine on the same form). Subroutines used in other routines are barred access from such variables. If a variable does need to be accessed across multiple modules then it will need to be defined as public.

## ***Local***

**Use Dim inside required subroutine**

Local variables are limited in scope to the subroutine in which they are declared. Any other subroutine in the project will be denied access. Locals should be used for variables which are to be used in the current procedure or function and then discarded. For example, a variable used in a FOR loop should be made local where possible.

**NOTE:** When a subroutine terminates any variables local to that subroutine will be destroyed and the memory made available for other purposes. This means that if you need a varaible to keep a value in between calls to the subroutine then you cannot use a standard local. A solution is to use a static variable ([further details](http://docs.google.com/data_struct.html#Static%20Variables)). These will still only be accessable to the subroutine in which they are declared, but their value will persist across calls to the subroutine. To test this try writing a small subroutine with a normal local variable, change the value, print the value on the screen, call the subroutine again and print out the value again. The second value will be 0. Now try the same thing but use a static variable instead. You should now see that the value of the variable on the second subroutine call is the same as the first, its value has been preserved.

 [Menu](http://docs.google.com/dvbhome.html)