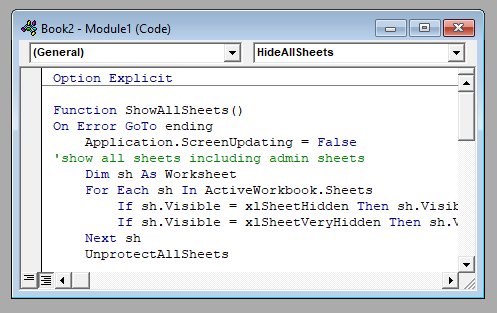
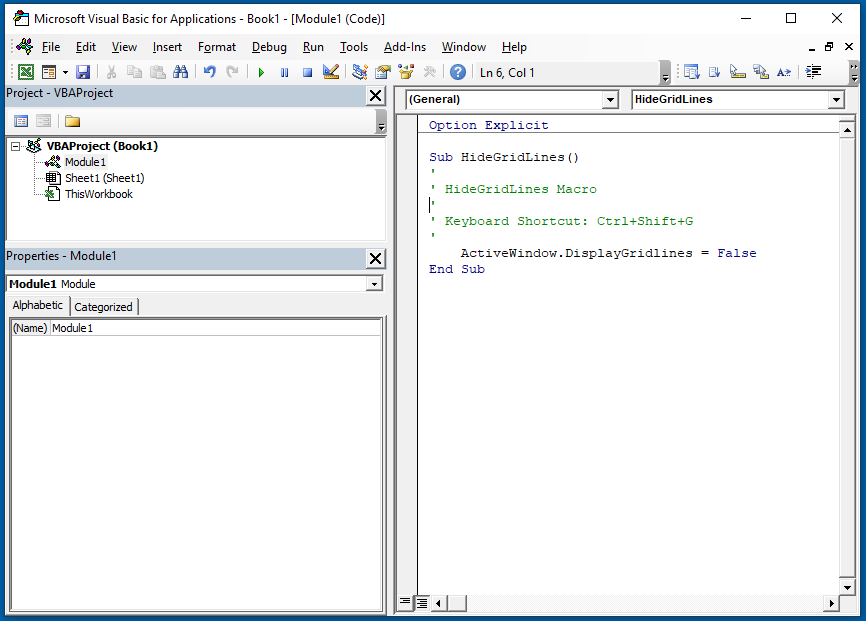
**1. What are modules in VBA and describe in detail the importance of creating a module?**

A VBA module is used to store any VBA code that you have written in the VBE (Visual Basic Editor).



The modules are contained within a VBA Project and when the file is saved – be it an Excel workbook, Word document or Access database, the module or modules are saved within that file – that file is essentially the parent application of the module.



Modules can also be exported out of the parent file and saved as their own individual files.  This is useful when you want to re-use code in a different file, and therefore perhaps import that module into a new file.

**2. What is Class Module and what is the difference between a Class Module and a Module?**

In a standard module, you can store procedures and functions. You can also store declarations, variables, and constants in modules. As we can spend most of your VBA career using only modules and can get what your users need.

Classes are used to create your own objects. Built-in objects in MS Excel are things like Workbooks, Worksheets, Sheets, etc. Built-in objects in MS Project are things like Subprojects, Tasks, Calendars, Resources, Assignments, etc.

In a class, we can build our own object with as many properties as we need. Then we can call the class to instantiate your object in your module. This is done similarly to how you call the built-in objects and their properties.

**3. What are the Procedures? What is a Function Procedure and a Property Procedure?**

Visual Basic offers different types of procedures to execute small sections of coding in applications. The various procedures are elucidated in details in this section. Visual Basic programs can be broken into smaller logical components called Procedures. Procedures are useful for condensing repeated operations such as the frequently used calculations, text and control manipulation etc.

A Function procedure is a series of Visual Basic statements enclosed by the Function and End Function statements. The Function procedure performs a task and then returns control to the calling code. When it returns control, it also returns a value to the calling code.

Each time the procedure is called, its statements run, starting with the first executable statement after the Function statement and ending with the first End Function, Exit Function, or Return statement encountered.

You can define a Function procedure in a module, class, or structure. It is Public by default, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it.

A Function procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

A property procedure is a series of Visual Basic statements that manipulate a custom property on a module, class, or structure. Property procedures are also known as *property accessors*.

Visual Basic provides for the following property procedures:

* A Get procedure returns the value of a property. It is called when you access the property in an expression.
* A Set procedure sets a property to a value, including an object reference. It is called when you assign a value to the property.

You usually define property procedures in pairs, using the Get and Set statements, but you can define either procedure alone if the property is read-only ([Get Statement](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/get-statement)) or write-only ([Set Statement](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/set-statement)).

You can omit the Get and Set procedure when using an auto-implemented property. For more information, see [Auto-Implemented Properties](https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/procedures/auto-implemented-properties).

You can define properties in classes, structures, and modules. Properties are Public by default, which means you can call them from anywhere in your application that can access the property's container.

**4. What is a sub-procedure and what are all the parts of a sub-procedure and when are they used?**

A Sub procedure is a series of Visual Basic statements enclosed by the Sub and End Sub statements. The Sub procedure performs a task and then returns control to the calling code, but it does not return a value to the calling code.

**Parts**

* attributelist

Optional. See [Attribute List](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/attribute-list).

* Partial

Optional. Indicates definition of a partial method. See [Partial Methods](https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/procedures/partial-methods).

* accessmodifier

Optional. Can be one of the following:

* + [Public](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/public)
  + [Protected](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/protected)
  + [Friend](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/friend)
  + [Private](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/private)
  + [Protected Friend](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/protected-friend)
  + [Private Protected](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/private-protected)

See [Access levels in Visual Basic](https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/declared-elements/access-levels).

* proceduremodifiers

Optional. Can be one of the following:

* + [Overloads](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/overloads)
  + [Overrides](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/overrides)
  + [Overridable](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/overridable)
  + [NotOverridable](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/notoverridable)
  + [MustOverride](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/mustoverride)
  + MustOverride Overrides
  + NotOverridable Overrides
* Shared

Optional. See [Shared](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/shared).

* Shadows

Optional. See [Shadows](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/shadows).

* Async

Optional. See [Async](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/modifiers/async).

* name

Required. Name of the procedure. See [Declared Element Names](https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/declared-elements/declared-element-names). To create a constructor procedure for a class, set the name of a Sub procedure to the New keyword. For more information, see [Object Lifetime: How Objects Are Created and Destroyed](https://learn.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/objects-and-classes/object-lifetime-how-objects-are-created-and-destroyed).

* typeparamlist

Optional. List of type parameters for a generic procedure. See [Type List](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/type-list).

* parameterlist

Optional. List of local variable names representing the parameters of this procedure. See [Parameter List](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/parameter-list).

* Implements

Optional. Indicates that this procedure implements one or more Sub procedures, each one defined in an interface implemented by this procedure's containing class or structure. See [Implements Statement](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/implements-statement).

* implementslist

Required if Implements is supplied. List of Sub procedures being implemented.

implementedprocedure [ , implementedprocedure ... ]

Each implementedprocedure has the following syntax and parts:

interface.definedname

| **Part** | **Description** |
| --- | --- |
| interface | Required. Name of an interface implemented by this procedure's containing class or structure. |
| definedname | Required. Name by which the procedure is defined in interface. |

* Handles

Optional. Indicates that this procedure can handle one or more specific events. See [Handles](https://learn.microsoft.com/en-us/dotnet/visual-basic/language-reference/statements/handles-clause).

* eventlist

Required if Handles is supplied. List of events this procedure handles.

eventspecifier [ , eventspecifier ... ]

Each eventspecifier has the following syntax and parts:

eventvariable.event

| **Part** | **Description** |
| --- | --- |
| eventvariable | Required. Object variable declared with the data type of the class or structure that raises the event. |
| event | Required. Name of the event this procedure handles. |

* statements

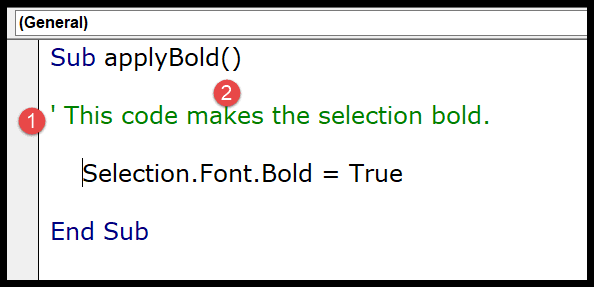
Optional. Block of statements to run within this procedure.

* End Sub

Terminates the definition of this procedure.

**5.How do you add comments in a VBA code? How do you add multiple lines of comments in a VBA code?**

A VBA COMMENT is a green line of text that helps you to describe the written code. In simple words, a comment is a line of text which is not a code and VBA ignores it while executing the code. It’s a good practice (I’d say one of the best) to add comments in your VBA codes.



## Add a Comment in a VBA Code

1. First, **click on the line** where you want to insert the comment.
2. After that, **type an APOSTROPHE** using your keyboard key.
3. Next, **type the comment** that you want to add to the code.
4. In the end, **hit enter** to move to the new line and the comment will turn green.

The moment you do this the entire line of the code will turn green which means that line is comment now.

## Enter a Multi-Line VBA Comment

There could be a situation where you need to enter a comment in multiple lines, like a block of the comments.

**But here is**one thing which you need to note down, every line of comment needs to start with an apostrophe, so if you want to add multiple lines of comments every line should have an APOSTROPHE.

The easiest way is to select all the lines and then use the comment button from the toolbar or you can also add an APOSTROPHE at the starting of each line.

The moment you click the comment button it will convert all the lines into a multi-line comment block.

