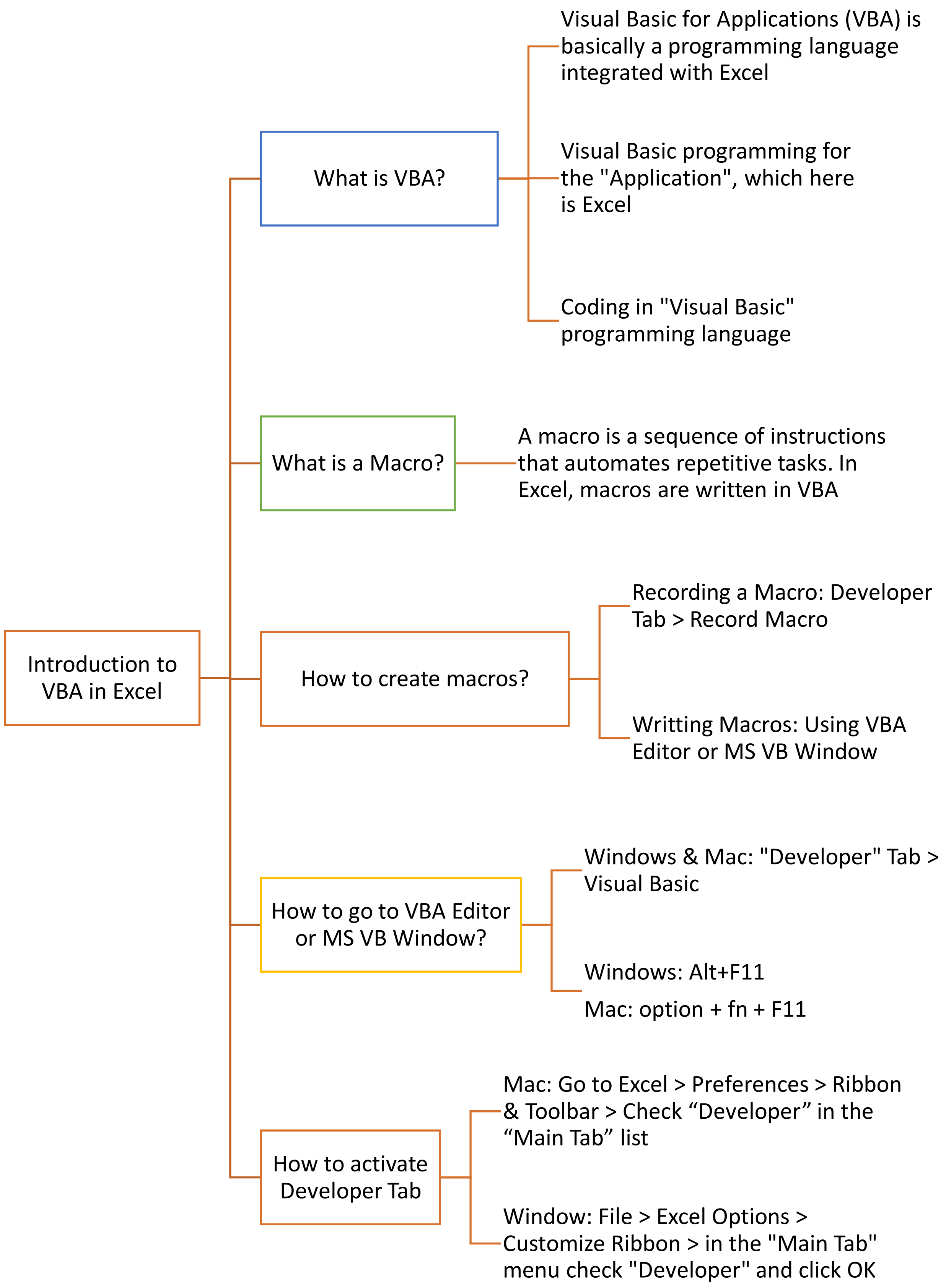
Introduction to VBA in Excel

Hamed Ghadimi

#VBAinExcelSnaps

What is VBA in Excel?



- Introduction to VBA in Excel
  - What is VBA?
    - Visual Basic for Applications (VBA) is basically a programming language integrated with Excel
    - Visual Basic programming for the "Application", which here is Excel
    - Coding in "Visual Basic" programming language
  - What is a Macro?
    - A macro is a sequence of instructions that automates repetitive tasks. In Excel, macros are written in VBA
  - How to create macros?
    - Recording a Macro: Developer Tab > Record Macro
    - Writting Macros: Using VBA Editor or MS VB Window
  - How to go to VBA Editor or MS VB Window?
    - Windows & Mac: "Developer" Tab > Visual Basic
    - Windows: Alt+F11

      Mac: option + fn + F11
  - How to activate Developer Tab
    - Mac: Go to Excel > Preferences > Ribbon & Toolbar > Check “Developer” in the “Main Tab” list
    - Window: File > Excel Options > Customize Ribbon > in the "Main Tab" menu check "Developer" and click OK

VBA Editor

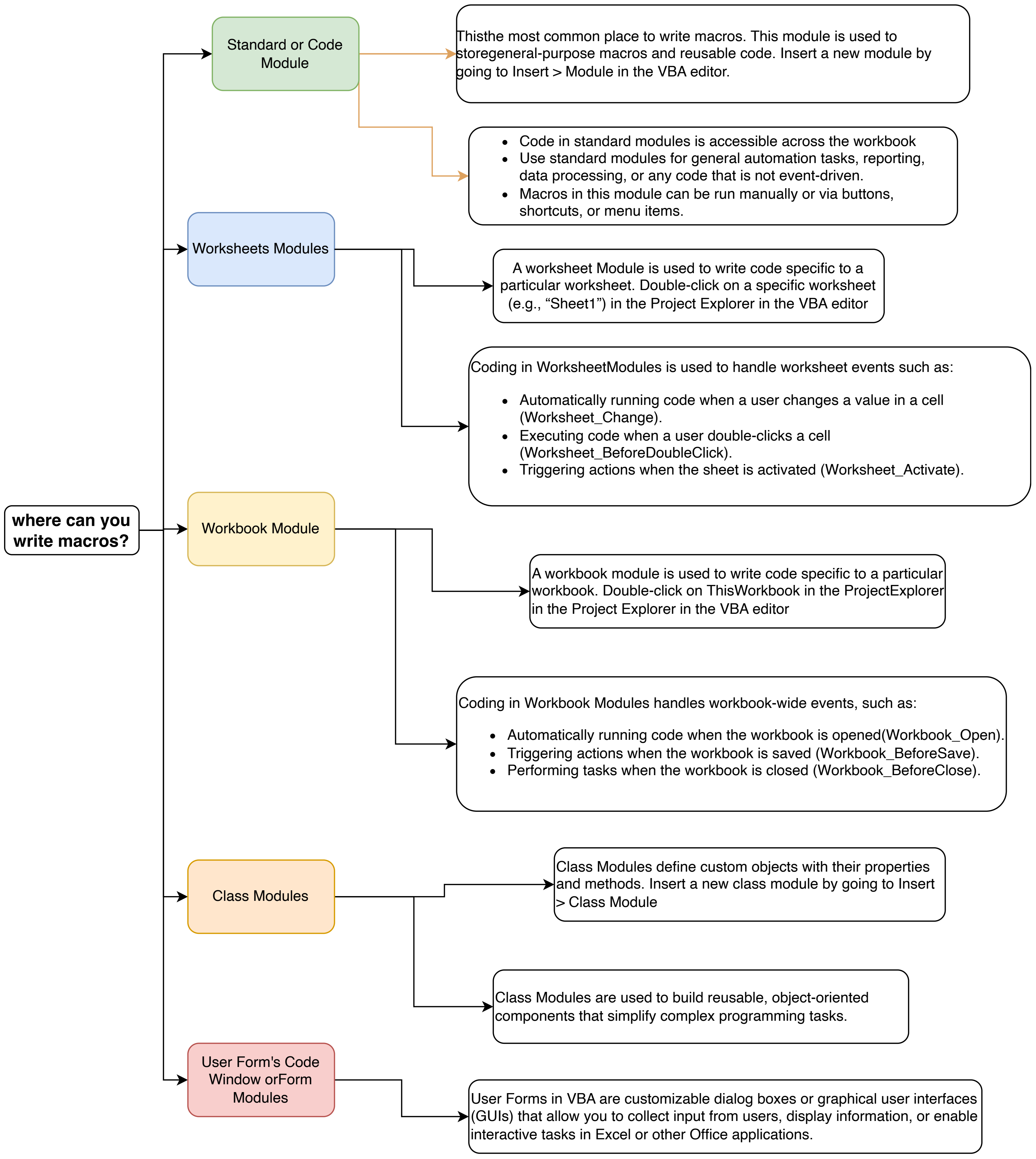
- Main Section of VBA Editor Window
  - Menu & Toolbars
    - Menus: File, Edit, View, Insert, Format, Debug, Run, & Tools
    - Object Browser
      - Is a reference library which is used to explore and understand the various objects, methods, properties, and functions available in VBA for Excel
  - Project Explorer
    - It is like the table of contents for your VBA code. It lists all the VBA Projects (Excel workbooks or add-ins) that are open, as well as the sheets and modules in them
    - You Always have only one VBA Editor Window for all the open workbooks
  - Properties Window
    - Is used to set properties or characteristics for objects like worksheets or workbooks
  - Code Window
    - This is where you actually write, edit, and view the code that makes everything work.
    - Each object (worksheet, module, etc.) has its own Code Window
  - Locals Window (Not Visible by Default)
    - The Locals Window shows you the current state of all variables within a running procedure.
  - Watch Window (Not Visible by Default)
    - This window helps you monitor certain variables as your code runs, which is helpful for debugging. It lets you track how values change step-by-step.
  - Immediate Window (Not Visible by Default)
    - This Window is like a quick test area

How to record a Macro?

- Macro Recording
  - What is it used for?
    - This feature is used to automate repetitive tasks, it allows you to capture actions (such as formatting, calculations, or data manipulation) in the form of VBA code. This recorded code, or macro, can then be replayed to perform the same steps automatically, saving time and reducing repetitive work.
  - How to record a Macro
    - Developer Tab > Click on "Record Macro" to open "Record Macro" Window
    - Assign a name to your Macro, create a shortcut key to run your Macro and add Description to explain what your Macro does
    - Store Macro in:
      - This Workbook (Macro-Enabled Excel File): Saves the macro in the current workbook only. You need to save this Excel file in .xlsm format.
      - New Workbook (Macro-Enabled Excel File): Creates a new .xlsm workbook to store the macro
      - Personal Macro Workbook (PERSONAL.XLSB): Saves the macro in a hidden workbook that opens every time you start Excel, so you can use the macro across different workbooks
  - Run Macro
    - Developer Tab > Macros > Select the Macro and Click Run
    - Use the short key assigned to Macro
    - Create a Button and assign the Macro to it
  - Limitations of Macro Recording
    - The macro recorder captures actions exactly as they were performed, including specific cell references. This can make the macro less adaptable if data sizes or structures change.
    - The recorder can’t capture complex logic, such as looping through rows or applying actions conditionally
    - Recorded macros may not be optimized for performance. Once recorded, macros cannot be modified through the recorder. They can be slow, especially with large data sets, because the code generated by recording is often inefficient

Note: The Personal Macro Workbook (PERSONAL.XLSB) is created on your computer the first time you record a macro and choose to store it there. Essentially, this is a hidden workbook dedicated to storing macros that you want available at all times. It opens automatically whenever Excel is launched, allowing you to access your saved macros in any workbook without needing to re-create them. Plus, by storing your macros here, you don’t have to worry about saving each file as a macro-enabled workbook (.xlsm).

Where Can you Write macros?



Thisthe most common place to write macros. This module is used to storegeneral-purpose macros and reusable code. Insert a new module by going to Insert > Module in the VBA editor.

- Code in standard modules is accessible across the workbook
- Use standard modules for general automation tasks, reporting, data processing, or any code that is not event-driven.
- Macros in this module can be run manually or via buttons, shortcuts, or menu items.

A worksheet Module is used to write code specific to a particular worksheet. Double-click on a specific worksheet (e.g., “Sheet1”) in the Project Explorer in the VBA editor

Coding in WorksheetModules is used to handle worksheet events such as:

- Automatically running code when a user changes a value in a cell (Worksheet_Change).
- Executing code when a user double-clicks a cell (Worksheet_BeforeDoubleClick).
- Triggering actions when the sheet is activated (Worksheet_Activate).

A workbook module is used to write code specific to a particular workbook. Double-click on ThisWorkbook in the ProjectExplorer in the Project Explorer in the VBA editor

Coding in Workbook Modules handles workbook-wide events, such as:

- Automatically running code when the workbook is opened(Workbook_Open).
- Triggering actions when the workbook is saved (Workbook_BeforeSave).
- Performing tasks when the workbook is closed (Workbook_BeforeClose).

Class Modules define custom objects with their properties and methods. Insert a new class module by going to Insert > Class Module

Class Modules are used to build reusable, object-oriented components that simplify complex programming tasks.

User Forms in VBA are customizable dialog boxes or graphical user interfaces (GUIs) that allow you to collect input from users, display information, or enable interactive tasks in Excel or other Office applications.

What is an Object in VBA?

What is an Object?

- What is an Object?
  - In VBA, an Object is essentially a "thing" that you can manipulate through your code. These "things" are components of Excel, like workbooks, worksheets, charts, or ranges. Application, Workbooks, Worksheets, and Range are the common Objects we use while coding.
- Applications of Objects in VBA
  - **Interacting with Excel Components:** Objects allow VBA to interact with Excel, letting you automate tasks like formatting a range, copying data, or generating charts.
  - **Building Flexible Code:** Objects make your code modular and adaptable. For example, you can write a macro that works with any workbook by referencing objects dynamically instead of hardcoding values.
- Objects have three primary aspects:
  - **Properties:** Properties describe the characteristics or attributes of an object. Each Object has its specific characteristics.
    Example: For a Range object representing a cell, properties include:
    • Value (the value of the cell) > `Range("A1").Value = "Hello"`
    • Font (the font style and size) > `Range("A1").Font.Bold = True`
  - **Methods:** Methods are the actions or operations that an object can perform. Methods allow you to perform actions on objects.
    Example: For a Workbook object, methods include:
    • Save (saves the workbook) > `Workbooks("Report.xlsx").Save`
    • Close (closes the workbook) > `Workbooks("Report.xlsx").Close`
  - **Event:** Events are triggers that occur when something happens to an object or specific triggers that objects respond to. Events enable dynamic behavior in your VBA code by responding to user actions or system processes.
    Example: A Worksheet object has events such as:
    • Worksheet_Change (triggered when a cell's value is changed) >

```
Private Sub Worksheet_Change(ByVal Target As Range)
       MsgBox "You changed the value in " & Target.Address
End Sub
```

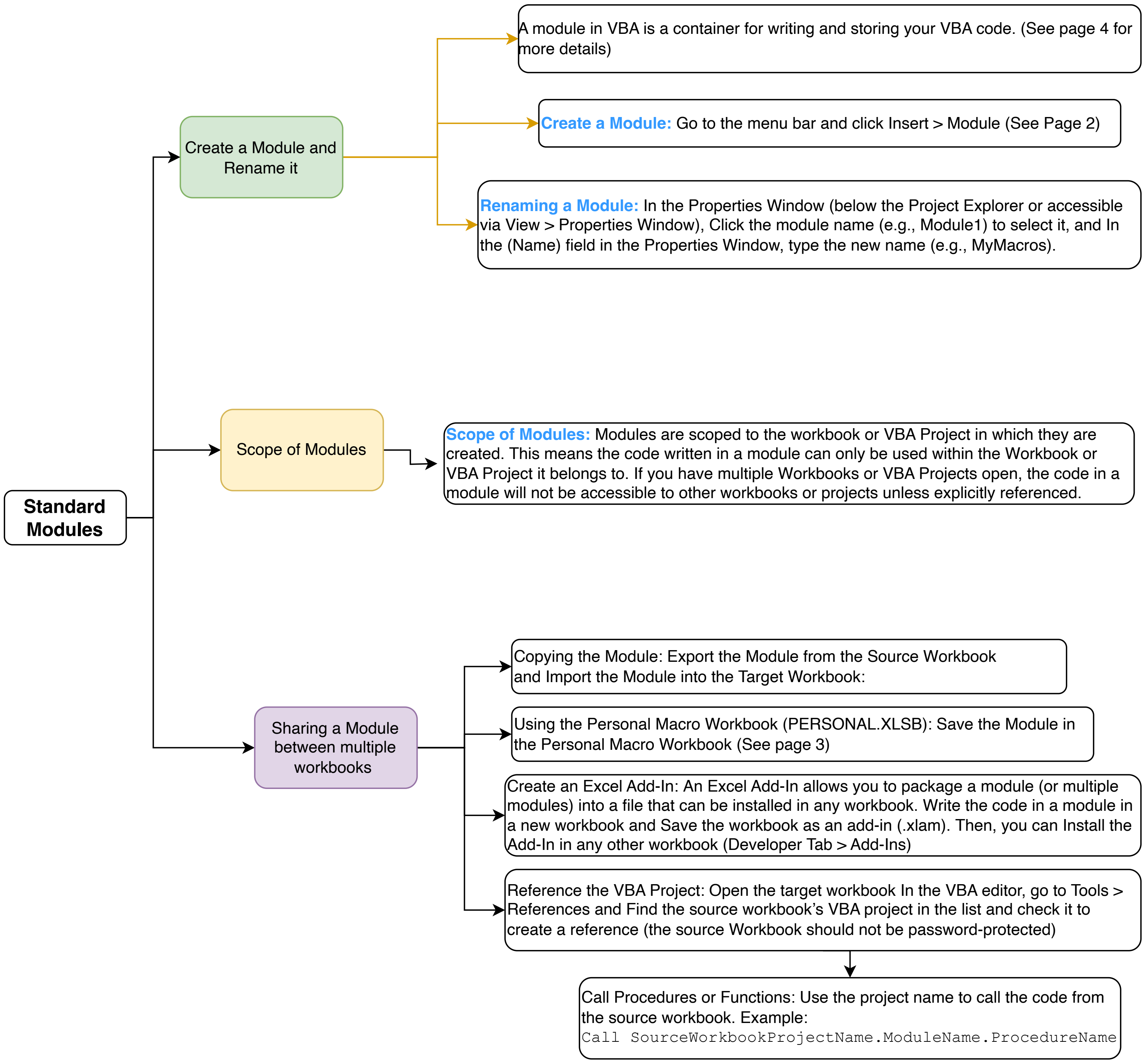
- Hierarchy of Objects
  - Objects in Excel are organized in a hierarchical structure, called the **Object Model**. The hierarchy starts from the top-level Excel application and narrows down to specific components like cells/ranges. This structure ensures you can access any part of Excel logically.
  - Common Object Hierarchy:
    1. Application: Represents the Excel application itself.
    Example: `Application.ActiveWorkbook` > refers to the currently active workbook.
    2. Workbook: Represents an Excel workbook.
    Example: `Workbooks("Budget.xlsx")` > refers to a specific workbook named "Budget.xlsx"
    3. Worksheet: Represents a sheet within a workbook.
    Example: `Workbooks("Budget.xlsx").Worksheets("Sheet1")` > accesses a specific worksheet (Sheet1) of Budget.xlsx
    4. Range: Represents a cell or group of cells within a worksheet.
    Example: `Workbooks("Budget.xlsx").Worksheets("Sheet1").Range("A1")` > refers to cell A1 in Sheet1 of Budget.xlsx
- How to Work with Objects in VBA
  - Use Existing Objects → Referencing Objects: Use dot notation to navigate the object hierarchy and access specific objects, properties, or methods.
  - Create New Objects → Creating and Managing Objects: New Objects can be created as variables → Existing types: You can create a new object as a Workbook, etc.
    Example: Create a new workbook:

```
Dim MyWB As Workbook
Set MyWB = Workbooks.Add
```

To release an object reference:

```
Set MyWB = Nothing
```

Standard or Code Module


Standard Modules
Create a Module and Rename it
Scope of Modules
Sharing a Module between multiple workbooks

A module in VBA is a container for writing and storing your VBA code. (See page 4 for more details)

Create a Module: Go to the menu bar and click Insert > Module (See Page 2)

Renaming a Module: In the Properties Window (below the Project Explorer or accessible via View > Properties Window), Click the module name (e.g., Module1) to select it, and In the (Name) field in the Properties Window, type the new name (e.g., MyMacros).

Scope of Modules: Modules are scoped to the workbook or VBA Project in which they are created. This means the code written in a module can only be used within the Workbook or VBA Project it belongs to. If you have multiple Workbooks or VBA Projects open, the code in a module will not be accessible to other workbooks or projects unless explicitly referenced.

Copying the Module: Export the Module from the Source Workbook and Import the Module into the Target Workbook:

Using the Personal Macro Workbook (PERSONAL.XLSB): Save the Module in the Personal Macro Workbook (See page 3)

Create an Excel Add-In: An Excel Add-In allows you to package a module (or multiple modules) into a file that can be installed in any workbook. Write the code in a module in a new workbook and Save the workbook as an add-in (.xlam). Then, you can Install the Add-In in any other workbook (Developer Tab > Add-Ins)

Reference the VBA Project: Open the target workbook In the VBA editor, go to Tools > References and Find the source workbook's VBA project in the list and check it to create a reference (the source Workbook should not be password-protected)

Call Procedures or Functions: Use the project name to call the code from the source workbook. Example:

```
Call SourceWorkbookProjectName.ModuleName.ProcedureName
```

Procedures

Procedures

Introduction to Procedure

A procedure is a block of code that performs a specific task. Procedures allow you to organize your code into manageable sections, make it reusable, and avoid repetition.

How to Create a Procedure

Create a standard module and either type the procedure manually or use the Insert menu: click “Procedure”, enter a name, choose the type (Sub, Function, or Property), select the scope (Public or Private), and click OK.

Naming Rules for Procedures

- Must start with a letter (A-Z or a-z)
- Can contain letters, numbers, and underscores (_)
- Cannot contain spaces or special characters (@, #, $, %, &, *, etc.)
- Must be unique within a VBA project
- Can be up to 255 characters long (shorter names are recommended for readability)

Note: Treat these rules as general naming conventions in VBA, whether you’re naming modules, variables, procedures, or other elements.

Types of Procedure

Subroutines (Sub): A subroutine, or "Sub" procedure, is a block of VBA code that performs a series of actions but does not return a value. Subs can accept arguments (optional or required) and can be executed using the Run button in the VBA editor, a keyboard shortcut, or by calling it in another procedure.

```
Sub ProcedureName(Optional arguments As DataType)
    ' Your code here
End Sub
```

Functions: A Function is a procedure that performs a task or a calculation and returns a value. Functions are often used to calculate and return results and can be used within other procedures. Function procedures can be used to create custom Excel formulas to extend Excel’s functionality

```
Function FunctionName(arguments As DataType) As ReturnType

    ' Your code here
    FunctionName = Result
End Function
```

Property Procedures: Property procedures allow you to define custom properties for objects. These properties can be read-only, write-only, or read/write. These procedures are used mainly in class modules to define object properties and are useful for advanced programming and object-oriented design. Types of "Property Procedures" are:

1. Property Let: Assigns a value to a property.
2. Property Get: Retrieves the value of a property.
3. Property Set: Assigns an object reference to a property.

Scope of Procedure in Standard Module

The scope of a procedure determines where it can be accessed. Procedures in a worksheet or workbook module are tied to specific events and cannot be called directly from other modules.

Private Procedures: Accessible only within the module in which they are defined. "Private Procedures" are defined using the "Private" keyword. Example:

```
Private Sub SecretMessage()
    MsgBox "This is private!"
End Sub
```

Public Procedures: These procedures are accessible from anywhere within the workbook or even from other workbooks (if referenced). Public Procedures are defined using the "Public" keyword (or no keyword, as "Public" is the default for modules). Example:

```
Public Sub HelloWorld()
    MsgBox "Hello, World!"
End Sub
```

Procedure Arguments

Required Arguments: Must be provided when the procedure is called.

```
Sub GreetUser(name As String)
    MsgBox "Hello, " & name
End Sub
```

Call:
```
GreetUser "Hamed"
```

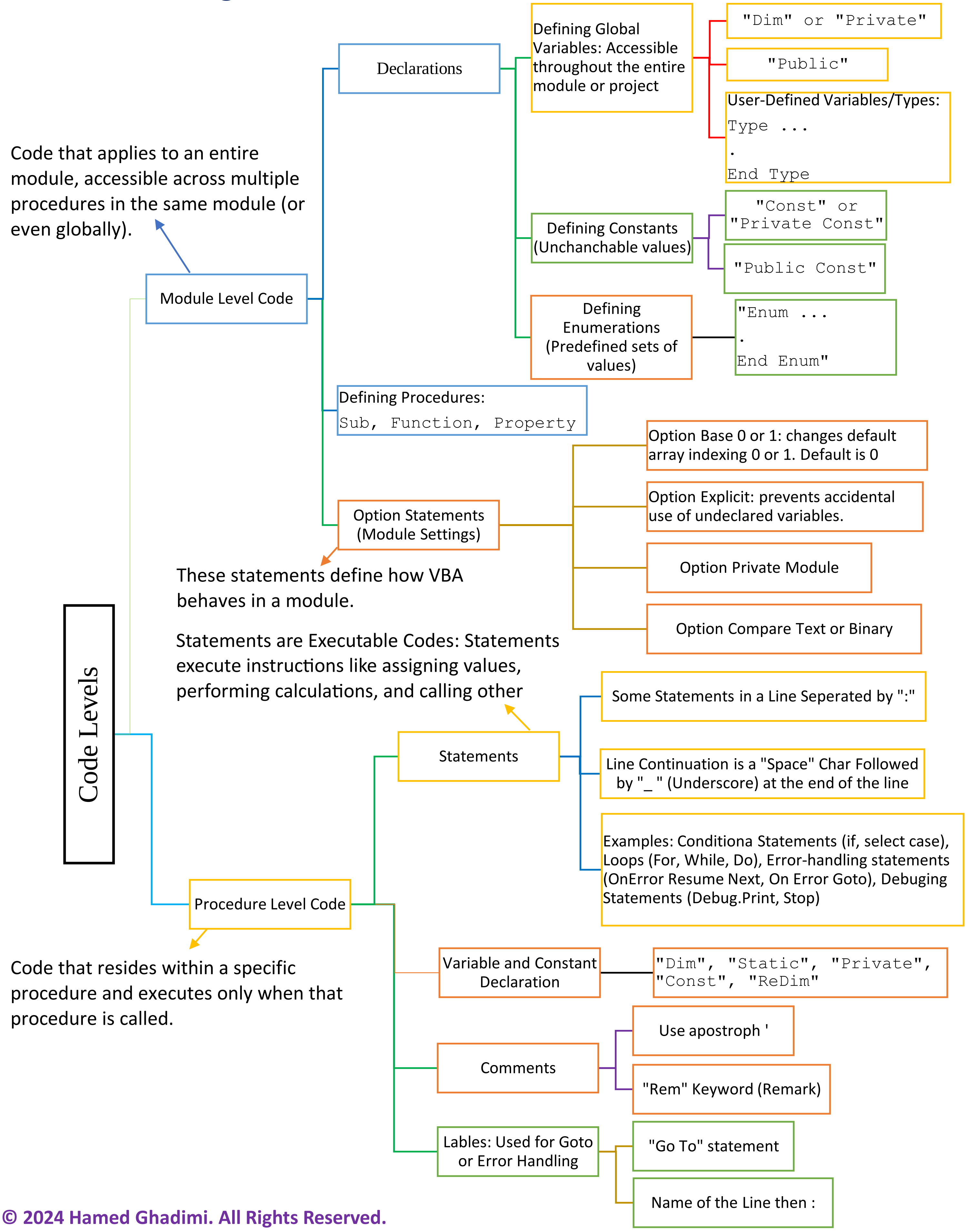
Optional Arguments: Do not need to be provided; you can define a default value.

```
Sub GreetUser(Optional name As String = "Guest")
    MsgBox "Hello, " & name
End Sub
```

ByRef (default): The argument is passed by reference, meaning changes affect the original variable.
ByVal: The argument is passed by value, meaning changes do not affect the original variable.

```
Sub DoubleValue(ByRef x As Integer)
    x = x * 2
End Sub
```

VBA Code Categorization


Code Levels
Module Level Code
Declarations
Defining Global Variables: Accessible throughout the entire module or project
"Dim" or "Private"
"Public"
User-Defined Variables/Types:
Type ...
.
End Type
Defining Constants (Unchanchable values)
"Const" or "Private Const"
"Public Const"
Defining Enumerations (Predefined sets of values)
"Enum ...
.
End Enum"
Defining Procedures:
Sub, Function, Property
Option Statements (Module Settings)
Option Base 0 or 1: changes default array indexing 0 or 1. Default is 0
Option Explicit: prevents accidental use of undeclared variables.
Option Private Module
Option Compare Text or Binary
Procedure Level Code
Statements
Some Statements in a Line Seperated by ":"
Line Continuation is a "Space" Char Followed by "_ " (Underscore) at the end of the line
Examples: Conditiona Statements (if, select case), Loops (For, While, Do), Error-handling statements (OnError Resume Next, On Error Goto), Debuging Statements (Debug.Print, Stop)
Variable and Constant Declaration
"Dim", "Static", "Private", "Const", "ReDim"
Comments
Use apostroph '
"Rem" Keyword (Remark)
Lables: Used for Goto or Error Handling
"Go To" statement
Name of the Line then :

Code that applies to an entire module, accessible across multiple procedures in the same module (or even globally).

These statements define how VBA behaves in a module.

Statements are Executable Codes: Statements execute instructions like assigning values, performing calculations, and calling other

Code that resides within a specific procedure and executes only when that procedure is called.